# Summary of TDT4265 Computer Vision

Morten Fyhn Amundsen

June 5, 2016

## Contents

# 1 Intensity transformations

## 1.1 Basic functions

Image intensity levels are defined in the range $[0, L-1]$. 0 is black, $L$ is white.

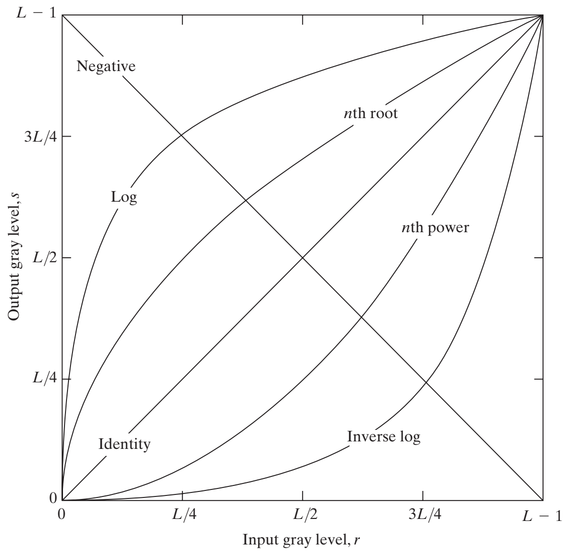


Figure 1: Some basic grey-level transformations

### 1.1.1 Negative

Inverts the intensity levels of the image (white becomes black and so on). Enhances details in dark regions.

$$s = L - 1 - r \tag{1}$$

### 1.1.2 Log transformation

Used to expand dark areas and compress bright areas, i.e. give more shadow detail. Good for displaying images with very large dynamic range.

$$s = c \log(1 + r), \quad r \geq 0 \tag{2}$$

### 1.1.3 Gamma transformation

Can compress blacks or whites depending on the gamma value. Similar to the log transformation, but more versatile. Many devices have an inherent gamma transform response that can be compensated for by applying the opposite gamma transform. Can be used to improve detail by darkening a washed-out image or brightening a too dark image.

$$s = cr^{\gamma} \tag{3}$$

### 1.1.4 Piecewise linear transformation

Use a piecewise linear function as the transformation (as opposed to the smooth transformations in Figure 1). Many uses, depending on the function:

- Contrast stretching: Increase the dynamic range of the grey levels, similar to a a sigmoid function. (Can also be done with a smooth function.)
- Grey-level slicing: Highlight a range of grey-levels. Either leave other values as they are or darken all other values.
- Bit-plane slicing: Consider e.g. an 8-bit image to be made of 8 bit 'planes'. Extracting one or some of these is called bit-plane slicing. The most significant plane can be extracted to form a binary black or white image, for instance.

## 1.2 Histogram processing

### 1.2.1 Histogram equalization

The cumulative probability density (CPD) of an image is

$$F(r) = \int_0^r p_r(w)\,\mathrm{d}w, \tag{4}$$

which leads to the transformation

$$s = T(r) = (L-1)F(r). \tag{5}$$

$p_r(w)$ is the probability for a pixel to be of intensity $w$. This transformation makes the histogram flat/uniform, which improves contrast by utilizing darks and lights equally much, rather than compressing the image into midtones.

### 1.2.2 Histogram matching

Sometimes you don't want a flat histogram, but a specific, nonflat distribution. Histogram matching is the method of transforming an image so that its histogram matches any given distribution.

### 1.2.3 Local histogram processing

The previous two methods are global, but sometimes a local method is better. This is done by iterating through pixels, looking at and changing the histogram of its neighborhood, and updating the pixel value according to the altered local histogram.

# 2 Spatial filtering

## 2.1 Fundamentals of spatial filtering

### 2.1.1 Mechanics of spatial filtering

A spatial filter consists of a neighborhood and an operation performed on the pixels in the neighborhood. A *linear filter* requires a linear operation, such as summation. Otherwise, it is a *nonlinear filter.*

### 2.1.2 Spatial correlation and convolution

Neighborhood operations where each output pixel is a weighted sum of neighbors to the corresponding input pixel. The weights are stored in a matrix called filter, mask or kernel. Correlation and convolution are the same except for the fact that the filter is rotated 180° for convolution.

If the input image is a discrete unit impulse (all zeros except a single '1' value), the output of convolution is the filter itself at the position of the unit impulse. Correlation has the same result, but the output is rotated 180 degrees. Convolution is commutative and associative:

$$f \star g = g \star f \tag{6}$$
$$f \star (g \star h) = (f \star g) \star h \tag{7}$$

However, correlation is *not.*

Because of the similarity of the operations, often only (8) is implemented, and the filter is rotated beforehand for convolution.

**Correlation**

$$w(x,y) * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{8}$$

**Convolution**

$$w(x,y) \star f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t) \tag{9}$$

## 2.2 Smoothing spatial filters

Used for blurring and noise reduction. Blurring can e.g. remove small, unwanted details before object detection.

Smoothing linear filters take an average of pixels under the mask. Also called averaging or lowpass filters. A typical filter is

$$\frac{1}{9} \cdot \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}. \tag{10}$$

Such filters where all coefficients are equal are called box filters. Another possibility is a weighted average such as

$$\frac{1}{16} \cdot \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}, \tag{11}$$

which can smooth an image with less blurring than a box filter.

### 2.2.1 Median filter

A nonlinear filter, good for removing e.g. salt-and-pepper noise. Does not blur sharp edges. Method:

1. Order pixels in the neighborhood by intensity.
2. Replace center pixel with median intensity.

## 2.3 Sharpening spatial filters

Just like smoothing/averaging is analogous to integration, sharpening can be done by differentiation. Some requirements for a sharpening filter based on the first derivative are

- zero in constant areas,
- nonzero at the onset and end of an intensity ramp, and
- nonzero during an intensity ramp.

Requirements for a secord-order derivative filter are the same, with the difference that it must be zero during ramps. The one-dimensional definitions

$$\begin{aligned} \frac{\partial f}{\partial x} &= f(x+1) - f(x) \\ \frac{\partial^2 f}{\partial x^2} &= f(x+1) + f(x-1) - 2f(x) \end{aligned} \tag{12}$$

satisfy these requirements.

### 2.3.1 Image sharpening with the Laplacian

For 2D images, sharpening filters can be based on the Laplacian

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \tag{13}$$

Some common filters are:

$$\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} \quad \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & -8 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \quad \begin{array}{|c|c|c|}\hline 0 & -1 & 0 \\ \hline -1 & 4 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} \quad \begin{array}{|c|c|c|}\hline -1 & -1 & -1 \\ \hline -1 & 8 & -1 \\ \hline -1 & -1 & -1 \\ \hline \end{array} \tag{14}$$

The sharpened image is obtained by taking the sum of the original and the filtered image:

$$g(x,y) = f(x,y) + c[\nabla^2 f(x,y)] \tag{15}$$

where $c$ is $-1$ if the center element of the filter is negative, and 1 otherwise.

### 2.3.2 Unsharp masking and highboost filtering

Unsharp masking sort of goes in the opposite direction of Laplacian sharpening: First blur the image, and subtract the blurred from the original to get a an unsharp mask $g_{\text{mask}}(x,y) = f(x,y) - \overline{f}(x,y)$. Finally add the mask to the original:

$$g(x,y) = f(x,y) + k \cdot g_{\text{mask}}(x,y) \tag{16}$$

Unsharp masking is strictly speaking only for $k = 1$. With $k > 1$, we have highboost filtering, and you can also lessen the effect with $k < 1$. The result is a sharpened image, because you remove unsharpness, which increases corner contrast.

### 2.3.3 Image sharpening with the Gradient

Section 5.2.4 defines the gradient. Its magnitude

$$M(x,y) = \sqrt{g_x^2 + g_y^2} \approx \left|g_x\right| + \left|g_y\right| \tag{17}$$

forms a gradient magnitude image. An example of masks that approximate this is the Sobel operator (82).

## 2.4 Combining spatial enhancement methods

In the real world, you need to use several methods to solve a task. One example is to first sharpen the image, then enhance the dynamic range.

# 3 Frequency filtering

$$\mathrm{e}^{\mathrm{i}\theta} = \cos\theta + \mathrm{i}\sin\theta \tag{18}$$

## 3.1 Fourier transformation

### 3.1.1 1D continuous Fourier transform (CFT)

The following equations define the forward and backward CFTs:

$$F(u) = \int_{-\infty}^{\infty} f(x)\mathrm{e}^{-2\pi ux}\,\mathrm{d}x \tag{19}$$

$$f(x) = \int_{-\infty}^{\infty} F(u)\mathrm{e}^{\mathrm{i}2\pi ux}\,\mathrm{d}u \tag{20}$$

$$\tag{21}$$

In many fields $f(x)$ is a function of time, but in image processing it is usually a function that maps from spatial position in an image to image intensity. Then $F(u)$ is a complex function of frequency, with a magnitude and phase (usually only magnitude is displayed).

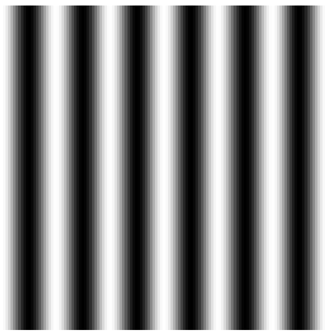
### 3.1.2 2D discrete Fourier transform (DFT)

Digital images have discrete values, and are 2D, so we need a 2D DFT to convert them to the frequency domain. For an image of size $M \times N$, it is

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\mathrm{e}^{-\mathrm{i}2\pi\left(\frac{ux}{N}+\frac{vy}{N}\right)} \tag{22}$$

and the inverse transform is

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)\mathrm{e}^{\mathrm{i}2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}. \tag{23}$$


(a) Intensity image

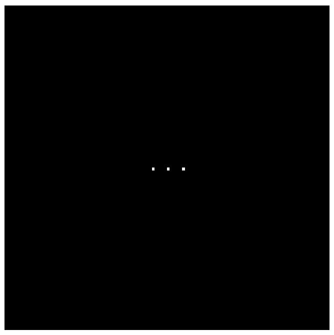

(b) Frequency image

Figure 2: An image and its Fourier transform

## 3.2 Convolution

**Convolution theorem** Convolution in one domain is equal to multiplication in the other:

$$\begin{aligned} h(x) \star f(x) &\iff H(x) \cdot F(x) \\ h(x) \cdot f(x) &\iff H(x) \star F(x) \end{aligned} \tag{24}$$

## 3.3 The sampling theorem

If samples are taken at a rate over twice the highest frequency of a function, it can be recreated without loss of information. This limit is the sampling theorem:

$$\frac{1}{\Delta T} > 2\mu_{\max} \tag{25}$$

## 3.4 Basics of frequency domain filtering

Filtering of an image $F(u, v)$ in the frequency domain with a filter $H(u, v)$ is done by

$$g(x, y) = \mathcal{F}^{-1}[H(u, v)F(u, v)] \tag{26}$$

where $H$ is a matrix of equal size to the image. (The multiplication $HF$ is done elementwise.)

**Shifting** To obtain a centered Fourier transform, multiply the image by $(-1)^{x+y}$ before transforming. Then $F(0, 0)$ will be at the center. Alternatively swap the quadrants after transforming.

## 3.5 Frequency domain smoothing

### 3.5.1 Ideal low-pass filter (ILPF)

An ILPF is a filter with no attenuation for frequencies below a threshold (the *cutoff frequency*), and full attenuation for all frequencies above:

$$H(u, v) = \begin{cases} 1 & \text{if } D(u, v) \leq D_0 \\ 0 & \text{if } D(u, v) > D_0 \end{cases} \tag{27}$$

The spatial representation of the ILPF is the sinc function (in 2D, that means any cross-section through the origin is a sinc function).

The convolution theorem (24) then reveals why the ringing effect occurs: Multiplying with an ILPF in the frequency domain is equivalent to convolving with a sinc filter (Figure 3) in the spatial domain.

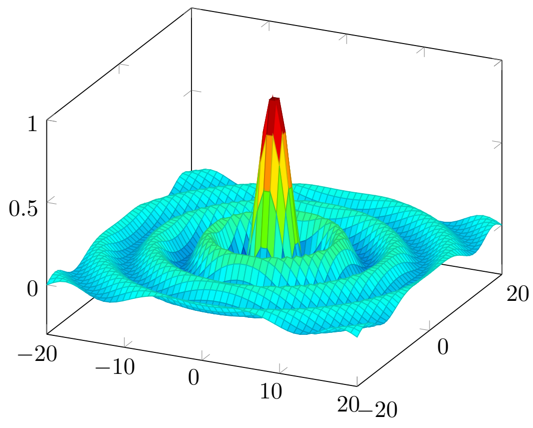


Figure 3: The sinc function

### 3.5.2 Butterworth low-pass filter (BLPF)

Defined as

$$H(u, v) = \frac{1}{1 + [D(u, v)/D_0]^{2n}} \tag{28}$$

where $n$ is the order and $D(u, v)$ is the distance in the frequency domain between $(u, v)$ and the center. This gives a smooth cutoff, which reduces ringing (depending on order). Order 1 has no ringing, 2 very little, and higher orders may have visible ringing. (A BLPF with $n = \infty$ is equal to an ILPF.)

### 3.5.3 Gaussian low-pass filter (GLPF)

Given in 2D as

$$H(u, v) = \mathrm{e}^{-\frac{D^2(u,v)}{2D_0}}. \tag{29}$$

The inverse Fourier of this is also a Gaussian function, so it will have no ringing. It's pretty nice.

### 3.5.4 Some examples of lowpass filtering

- Smoothing digitized text to fill gaps and improve legibility, especially for machine processing.
- Cosmetic processing, such as softening skin on images of people.
- Removing artifacts, such as noise and scan lines.
- However, smoothing is mostly used for preprocessing.

## 3.6 Frequency domain sharpening

Done with high-pass filters, which are pretty much the opposite of low-pass filters:

$$H_{\text{highpass}}(u,v) = 1 - H_{\text{lowpass}}(u,v) \tag{30}$$

### 3.6.1 Ideal high-pass filter (IHPF)

Defined as

$$H(u,v) = \begin{cases} 0 & \text{if } D(u,v) \le D_0 \\ 1 & \text{if } D(u,v) > D_0 \end{cases} \tag{31}$$

### 3.6.2 Butterworth high-pass filter (BHPF)

$$H(u,v) = \frac{1}{1 + [D_0/D(u,v)]^{2n}} \tag{32}$$

### 3.6.3 Gaussian high-pass filter (GHPF)

$$H(u,v) = 1 - \mathrm{e}^{-\frac{D^2(u,v)}{2D_0^2}} \tag{33}$$

### 3.6.4 Frequency domain Laplacian filter

The Laplacian works the same way in the frequency domain as in the spatial domain. With

$$H(u,v) = -4\pi^2(u^2+v^2) \tag{34}$$

the Laplacian image is

$$\nabla^2 f(x,y) = \mathcal{F}^{-1}\{H(u,v)F(u,v)\} \tag{35}$$

and the enhanced image is

$$g(x,y) = f(x,y) + c\nabla^2 f(x,y) \tag{36}$$

### 3.6.5 Unsharp masking, highboost filtering, high-frequency emphasis filtering

Like in the spatial domain (16), unsharp masking and highboost filtering can be be done in the frequency domain:

$$\begin{aligned} g(x,y) &= \mathcal{F}^{-1}\left\{[1 + k\cdot[1 - H_{\text{LP}}(u,v)]]\,F(u,v)\right\} \\ &= \mathcal{F}^{-1}\{[1 + k\cdot H_{\text{HP}}(u,v)]F(u,v)\} \end{aligned} \tag{37}$$

Remember (30). This leads to the high-frequency emphasis filter

$$g(x,y) = \mathcal{F}^{-1}\{[k_1 + k_2\cdot H_{\text{HP}}(u,v)]F(u,v)\} \tag{38}$$

where $k_1 \ge 0$ is the DC term, and $k_2 \ge 0$ sets the contribution of high frequencies. By setting a nonzero DC term, the low frequency grey levels is not lost. High-frequency emphasis filtering followed by histogram equalization can enhance clarity and detail significantly, and is useful for e.g. X-ray images.

## 3.7 Selective filters

These filters only affect parts of the frequency spectrum.

### 3.7.1 Bandreject and bandpass

Filters that remove all frequencies inside or outside a given range. Like normal filters, these can be created as ideal, Butterworth, or Gaussian filters.

### 3.7.2 Notch filters

These filters reject (or pass) frequencies at chosen locations in the frequency rectangle. A common example is removing periodic noise, such as moiré.

# 4 Morphological image processing

Morphology is image manipulation via set theory. It's pretty cool. Sort of like spatial filtering, but with non-linear operations.

## 4.1 Preliminaries

In morphology, images are represented as sets. Usually images are binary, and the set of an image is the set of 1-valued pixels, defined on $\mathbb{Z}^2$. An extension in $\mathbb{Z}^3$ to greyscale exists, and also to color and so on.

### 4.1.1 Some set theory

- The absolute complement of a set $A$ is
$$A^c = \{w | w \notin A\}, \tag{39}$$
which is all elements outside of $A$.
- The relative complement between two sets $A$ and $B$ is
$$A \setminus B = \{w | w \in A, w \notin B\}, \tag{40}$$
which is all elements that are *only* in $A$.
- The reflection (180 degree rotation) of a set is
$$\hat{B} = \{w | w = -b \text{ for } b \in B\}. \tag{41}$$
- The translation of a set $A$ by a vector $z = (z_1, z_2)$ is
$$(B)_z = \{c | c = b + z \text{ for } b \in B\} \tag{42}$$

### 4.1.2 Structuring elements (SEs)

An SE is a small subimage or subset that is used to find properties of the image. They are usually symmetric and with their center defined in the middle.

## 4.2 Erosion and dilation

### 4.2.1 Erosion

The erosion of $A$ by $B$ is defined as
$$A \ominus B = \{z | (B)_z \subseteq A\}, \tag{43}$$
which is the set of all points where $B$ is completely inside $A$. ($A$ is an image and $B$ is an SE.) The operation shrinks objects, and removes anything smaller/thinner than the SE, such as thin lines.

### 4.2.2 Dilation

The dilation of $A$ by $B$ is defined as
$$A \oplus B = \left\{z | (\hat{B})_z \cap A \neq \varnothing\right\} \tag{44}$$
which is the set of all points where $\hat{B}$ (the reflection of $B$) overlaps with $A$ by at least one element. This operation grows objects, and can fill gaps and small holes.

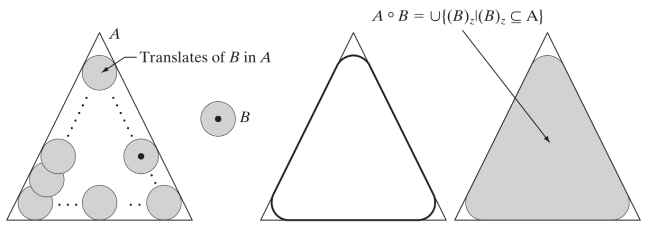
### 4.2.3 Duality

Erosion and dilation are dual operations:
$$\begin{aligned}(A \ominus B)^c &= A^c \oplus \hat{B} \\ (A \oplus B)^c &= A^c \ominus \hat{B}\end{aligned} \tag{45}$$
This is useful for symmetric SEs ($\hat{B} = B$), because then we can erode $A$ by instead dilating the background $A^c$.

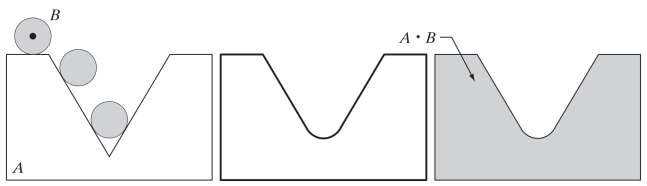
## 4.3 Opening and closing

*Opening* is defined as
$$A \circ B = (A \ominus B) \oplus B \tag{46a}$$
$$= \bigcup \{(B)_z | (B)_z \subseteq A\} \tag{46b}$$
It is analogous to the area defined by moving a ball (or other shape) along the inner boundary of a set, as in Figure 4. Equation (46b) shows that opening can be expressed as "all translates of $B$ that fit inside $A$". Used for e.g. removing thin connections between components and removing noise pixels.

*Closing* is defined as
$$A \bullet B = (A \oplus B) \ominus B, \tag{47}$$
which is analogous to the area defined by moving e.g. a ball along the outer boundary of a set, as in Figure 4. Used for e.g. filling holes.



(a) Opening



(b) Closing

Figure 4: Analogies to the opening and closing operations

Note that both opening and closing are idempotent operations.

**Combining opening and closing** Opening followed by closing can remove noise, both specks of object pixels on the background and background pixels where there should be object. However, some connectivity may be lost.

## 4.4 The hit-or-miss transformation

A basic tool for shape detection. Let $B = (B_1, B_2)$, $B_1$ be an object, and $B_2$ its background. Then the matches of

$B$ in $A$ is

$$A \circledast B = (A \ominus B_1) \cap (A^c \ominus B_2). \tag{48}$$

The set $(A \ominus B_1)$ are all locations where the object fits in, and $(A^c \ominus B_2)$ are all locations where the background fits. Locations where both fit are locations of all shapes in $A$ that perfectly match $B$. We assume that each object we want to find is separate from the other objects, and therefore each has a complete local background at least one pixel wide.

## 4.5 Some basic morphological algorithms

Now we can finally use this stuff.

### 4.5.1 Boundary extraction

Get the boundary by subracting the eroded image from the original.

$$\beta(A) = A - (A \ominus B) \tag{49}$$

The erosion gives a "smaller" version of $A$, and subtracting that from the image leaves only the border.

### 4.5.2 Hole filling

Same principle as in Section 4.5.3 below.

$$X_k = (X_{k-1} \oplus B) \cap A^c \tag{50}$$

The seed pixel must be in a hole. It will grow until it fills the hole. Stop when no further change happens.

### 4.5.3 Extraction of connected components

Done by iteration:

$$X_k = (X_{k-1} \oplus B) \cap A \tag{51}$$

Choose a starting pixel (seed) $X_0$. Dilate $X_0$, but only keep the pixels that are in the original. Continue dilating and only keeping pixels present in $A$ until convergence.

### 4.5.4 Convex hull

A set is convex if any line segment between two points in the set lies fully within the set. The *convex hull* $H$ of a set $S$ is the smallest convex set containing $S$. The difference $H - S$ is the *convex decifiency* of $S$. It can be computed by iterating

$$X_k^i = (X_{k-1} \circledast B^i) \cup A, \quad i = 1, 2, 3, 4, \quad k = 1, 2, \ldots \tag{52}$$

with $X_0^i = A$ and $B^i$ being as shown in Figure 5. For each SE, let $D^i = X_k^i$ when its iteration converges. Then the convex hull is

$$C(A) = \bigcup_{i=4}^{4} D^i \tag{53}$$

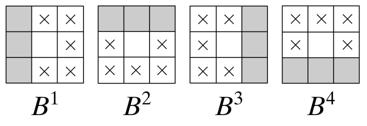


Figure 5: SEs for convex hull extraction

### 4.5.5 Thinning

Used e.g. for skeletonization and to thin edge detector output. It is defined in terms of the hit-or-miss transformation as

$$A \otimes B = A \cap (A \circledast B)^c \tag{54}$$

with a sequence of SEs

$$\{B\} = \{B^1, \ldots, B^n\} \tag{55}$$

where $B^i$ is a rotated $B^{i-1}$. Then just keep applying hit-or-miss with $B$ repeatedly, until continuing has no effect.

One pass of thinning is written as

$$A \otimes \{B\} = ((\ldots((A \otimes B^1) \otimes B^2) \ldots) \otimes B^n) \tag{56}$$

### 4.5.6 Thickening

Thickening is the dual of thinning:

$$A \odot B = A \cup (A \circledast B) \tag{57}$$

### 4.5.7 Skeletons

The skeleton is

$$S(A) = \bigcup_{k=0}^{K} (A \ominus kB) - (A \ominus kB) \circ B \tag{58}$$

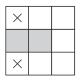
where $B$ is an SE and $(A \ominus kB)$ indicates $k$ successive erosions of $A$. $K$ is the number of erosions possible before resulting in the empty set.

### 4.5.8 Pruning

Skeletons often have unwanted "spurs". Assuming the spurs are sufficiently small, pruning can remove them. We perform thinning with SEs designed to only detect end points.

$$X_1 = A \otimes \{B\} \tag{59}$$

The SEs used are shown in Figure 6.



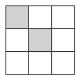


Figure 6: SEs for pruning

#### 4.5.9 Morphological reconstruction

Uses two images, a marker $F$ with starting points, and a mask $G$ that constrains the transformation.

**Geodesic dilation** Geodesic dilation is the intersection of the mask and the dilated marker. One pass is

$$D_G^{(1)}(F) = (F \oplus B) \cap G. \tag{60}$$

Several passes, denoted $D_G^{(n)}$, is just repeating the operation $n$ times.

**Geodesic erosion** Geodesic erosion is the union of the mask and the eroded marker. One pass is

$$E_G^{(1)}(F) = (F \ominus B) \cup G. \tag{61}$$

**Morphological reconstruction by dilation** Reconstruction by dilation is just repeating (60) until more passes have to effect:

$$R_G^D(F) = D_G^{(k)}(F) \tag{62}$$

**Morphological reconstruction by erosion** Same as by dilation, but using erosion:

$$R_G^E(F) = E_G^{(k)}(F) \tag{63}$$

**Opening by reconstruction** With normal opening (Section 4.3), erosion removes small things and dilation restores the original size/shape. Opening by reconstruction restores the precise original shape, and is done as reconstruction by dilation of $F$, from the erosion of size $n$ of $F$:

$$O_R^{(n)}(F) = R_F^D\left[(F \ominus nB)\right] \tag{64}$$

where $(F \ominus nb)$ indicates $n$ erosions.

The SE for erosion $B$ determines the shapes we want to identify. The erosion itself leaves us with markers for all matching objects (because they are not removed by the erosion). Then, the image $F$ is used as a mask, and reconstruction is done by dilating the markers, subject to the mask. That way, all objects marked will be completely filled, and the result is removing all unmarked objects.

**Filling holes** With morphological reconstruction we can fill holes automatically, without knowing where they are. Create a marker image $F$ of zeros, except for the border, where it is $1 - I$. Then

$$H = \left[R_{I^c}^D(F)\right]^c \tag{65}$$

is $I$ with all holes filled.

**Border clearing** Morphological reconstruction can also be used to remove objects connected to the borders of an image. Create a marker image $F$ that is equal to the input image $I$ at the border, and 0 elsewhere. We can extract all border-touching objects by the morphological reconstruction $R_I^D(F)$, and subtract it from the original to get an image without border objects $X$:

$$X = I - R_I^D(F) \tag{66}$$

#### 4.5.10 Summary

We have now defined the following morphological operations:

- Basic set operations:
  - Translation
  - Reflection
  - Complement
  - Difference
- Basic morphological operations:
  - Erosion
  - Dilation
  - Opening
  - Closing
- Complex morphological operations:
  - Hit-or-miss transform
  - Boundary extraction
  - Hole filling
  - Connected components
  - Convex hull
  - Thinning
  - Thickening
  - Skeletons
  - Pruning
- Morphological reconstruction operations:
  - Geodesic dilation

- Geodesic erosion
- Morphological reconstruction by dilation
- Morphological reconstruction by erosion
- Opening by reconstruction
- Closing by reconstruction
- Hole filling
- Border clearing

## 4.6 Grey-scale morphology

We can dilate, erode, open, and close grey-scale images too! Now we use images $f(x, y)$ and SEs $b(x, y)$. The SEs can now be either flat (uniform) or nonflat. We stick to flat SEs for simplicity.

### 4.6.1 Erosion and dilation

Erosion of $f$ by a flat SE $b$ is the minimum value of the image in the region covered by the SE:

$$\left[f \ominus b\right](x, y) = \min_{(s,t)\in b} \left\{f(x + s, y + t)\right\}. \tag{67}$$

Dilation is similarly

$$\left[f \oplus b\right](x, y) = \max_{(s,t)\in b} \left\{f(x - s, y - t)\right\}, \tag{68}$$

which uses $\hat{b} = b(-x, -y)$.

### 4.6.2 Opening and closing

This is mathematically just as before:

$$f \circ b = (f \ominus b) \oplus b \tag{69a}$$

$$f \bullet b = (f \oplus b) \ominus b \tag{69b}$$

Looking at the image as a topographic surface, opening is pushing the SE up from below, clipping all peaks where it doesn't fit, and closing is pushing the SE down from above, clipping all valleys where it doesn't fit. See Figure 7. This means that opening removes small, bright details, and closing removes small, dark details.

### 4.6.3 Some basic greyscale morphological algorithms

**Smoothing** The "small bright/dark details" mentioned in Section 4.6.2 might be noise. This means we can remove this type of noise by opening and then closing. The size of the SE then sets a threshold for what is image and what is noise.

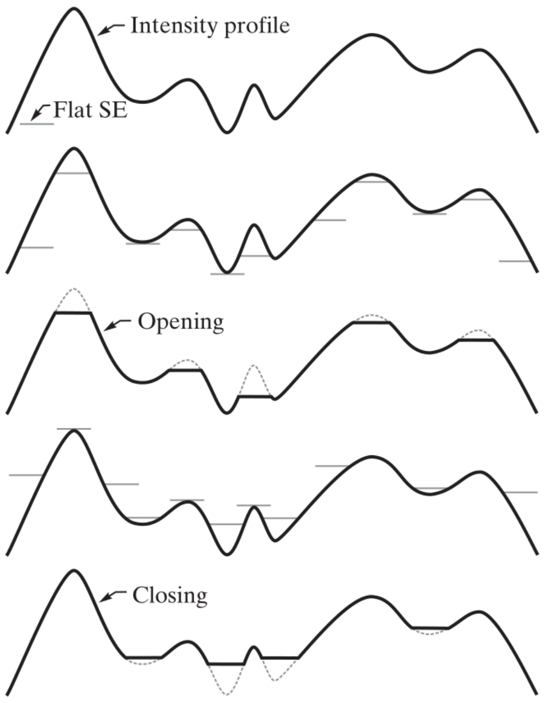


Figure 7: Analogy to opening and closing greyscale images

**Morphological gradient** The morphological gradient is

$$g = (f \oplus b) - (f \ominus b). \tag{70}$$

The dilation thickens objects in the image, and the erosion shrinks them. Differencing these emphasises edges and removes homogenous areas, which gives a gradient-like effect.

**Top-hat and bottom-hat transformations** Subtracting the opening of an image from the image itself defined the top-hat transformation

$$T_{\text{hat}}(f) = f - (f \circ b). \tag{71}$$

The bottom-hat transformation is defined similarly:

$$B_{\text{hat}}(f) = (f \bullet b) - f \tag{72}$$

Top-hat can isolate light objects on a dark background, and bottom-hat does the opposite. This is a type of segmentation, and works well even with uneven lightning.

Opening with an SE that is too large to fit in the objects you want to segment will remove all such objects and leave only an approximation of the background, including any uneven lightning. If you subtract the approximate background from the original, the unevenness will be reduced, and now thresholding methods can be used with success.

**Granulometry** (Assuming light objects on a dark background.) Open the image with gradually larger SEs, and compute the sum of pixel values. As the SEs grow larger than the objects, they disappear, and the pixel sum decreases. Plotting pixel sum against SE size indicates by peaks in the plot the predominant sizes of objects in the image. This can be used to simply get the size distribution of objects, but also to identify defect objects (sizewise).

**Textural segmentation** (Assuming light objects on a dark background.) Textural segmentation is to divide an image into regions based on their texture contents. If you have an image with some coarse and some fine texture, you can close it with an SE large enough to remove the fine objects. Then open it with an SE large enough to fill the gaps between the coarse objects. Now coarse areas are dark, and fine areas are bright, and the boundary can be extracted by e.g. a morph. gradient.

#### 4.6.4 Greyscale morphological reconstruction

Let $f$ be the marker, and $g$ be the mask. The basic operations are similar to their black-and-white counterparts.

- Geodesic dilation:

$$D_g^{(1)}(f) = (f \oplus b) \wedge g \tag{73}$$

- Geodesic erosion:

$$E_g^{(1)}(f) = (f \ominus b) \vee g \tag{74}$$

- Reconstruction by dilation:

$$R_g^D(f) = D_g^{(k)}(f) \tag{75}$$

- Reconstruction by erosion:

$$R_g^E(f) = E_g^{(k)}(f) \tag{76}$$

- Opening by reconstruction:

$$O_R^{(n)}(f) = R_f^D\left[(f \ominus nb)\right] \tag{77}$$

- Closing by reconstruction:

$$C_R^{(n)}(f) = R_f^E\left[(f \oplus nb)\right] \tag{78}$$

**Top-hat by reconstruction** This method consists of subtracting an image's opening by reconstruction from the image.

# 5 Image segmentation

Segmentation divides an image into the regions or objects it consists of. Most algorithms here are based on intensity discontinuity and similarity.

## 5.1 Fundamentals

We pretty much just divide an image $R$ into $n$ regions $R_1 \ldots R_n$, and require the following to hold:

- The union of all regions is equal to the whole image.
- Each region is a connected set.
- Each region is disjoint from the others.
- Some predicate[1] is true for each region separately.
- The predicate is false for the union of any adjacent regions.

Segmentation is either edge-based or region-based.

## 5.2 Point, line, and edge detection

These methods detect sharp, local changes in intensity. First- and second-order derivatives are used, and can detect intensity ramps, isolated points, and steps. Simple spatial filters can realize 1st and 2nd derivatives.

#### 5.2.1 Point detection

Points can be detected by a 2nd derivative mask and thresholding. (If the absolute value of the output of a 2nd derivative mask is above a threshold, that pixel is marked as a point.) A suitable mask for this is

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | −8 | 1 |
| 1 | 1 | 1 |

. (79)

#### 5.2.2 Line detection

The same mask (79) can be used. However, this mask gives a double line effect—negative values on one side of the line and positive on the other. This can be avoided by only keeping the positive-valued pixels. Note that lines must be thin compared to the mask in order to be detected as lines. Fat lines should be treated as regions.

(79) is independent of line direction. Other masks can be used to detect lines in specified directions.

[1]A predicate can be e.g. the that the intensity of each pixel is in a certain range.

### 5.2.3 Edge models

Edges can be steps (sudden change), ramps (gradual change), or roofs (gradual increase followed by gradual decrease).

The magnitude of the 1st derivative can indicate an edge, and the sign of the 2nd derivative reveals if you are on the bright or dark side of it. Because this is done with derivatives, noisy edges are problematic.

### 5.2.4 Basic edge detection

The image gradient is

$$\nabla f = \text{grad}(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}. \tag{80}$$

The magnitude is the value of the rate of change in the direction of the gradient (steepness of the ramp). The direction of $\nabla f$ is

$$\alpha(x, y) = \arctan\left(\frac{g_y}{g_x}\right). \tag{81}$$

This is the direction of the steepest ascent, and is therefore orthogonal to the edge.

A often used and pretty good gradient mask is the Sobel operator

$$\begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \text{ and } \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}. \tag{82}$$

These two masks can detect vertical and horizontal edges, respectively. The same operator but with ones instead of twos is called the Prewitt operator. However, the ±2-elements give some smoothing, which is good as derivatives are sensitive to noise. Sobel and Prewitt masks can also be rotated 45° to detect diagonal edges.

To maintain connectivity of edges and highlight the strongest edges, you must smoothe the image first, then extract edges, then threshold it.

### 5.2.5 More advanced edge detection

These methods worry more about noise and the nature of edges.

**The Marr-Hildreth edge detector (Laplacian of Gaussian)** This method uses masks of variant scales to detect edges of variant sharpness. The filter is the Laplacian of the Gaussian (LoG):

$$\nabla^2 G(x, y) = \left[\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{83}$$

This formula can be used to create LoG masks of any size. The Gaussian part $G$ blurs the image without ringing. The Laplacian part $\nabla^2$ is used to detect edges because it is isotropic[2], as opposed to the 1st derivative. In reality, Marr-Hildreth edge detection is done by

1. Filter image with a Gaussian lowpass filter.
2. Compute the Laplacian of the image.
3. Find the zero-crossings.

Note that it is usually better to use a small positive threshold for zero-crossing. This removes a lot of "noisy" edges. Also note that the LoG can be approximated with a difference of Gaussian (DoG) operator.

**The Canny edge detector** The Canny edge detector is an attempt to

1. find all edges without false positives,
2. locate edge points near the true edge, and
3. return edges only one pixel wide.

This is fairly well achieved by the following:

1. Smooth the image with a Gaussian filter.
2. Compute gradient magnitude and gradient angle images (using e.g. Sobel).
3. Apply nonmaxima suppression to the gradient magnitude image.
4. Detect and link edges with double thresholding and connectivity analysis.
5. (Optional.) Use edge thinning to make sure all edges are one pixel wide.

Canny is generally better than Marr-Hildreth/LoG, but more complex and computationally heavy.

**Edge linking and boundary detection** Done with local processing, regional processing, or global processing with the Hough transform.

## 5.3 Corner detection

The dominant method for corner detection is the Harris corner detector.

[2]Invariant to rotation.

### 5.3.1 Harris corner detector

Define a box of some size to move around the image and look for corners inside of. For each location, calculate gradients in $x$ and $y$ directions. In a flat area, the gradients will all be close to zero. Edges and corners give different non-zero responses. This can be quantified by

$$M = \sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{84}$$

where $I_x$ and $I_y$ are gradients in the $x$ and $y$ directions. $W$ is the window we are looking for corners in. We can then use the Harris corner response function

$$R = \det(M) - \alpha \operatorname{Trace}(M)^2 = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2 \tag{85}$$

to assume features:

| Response function | Eigenvalues | Feature found |
|---|---|---|
| $R > 0$ | Both large | Corner |
| $R < 0$ | One large | Edge |
| $\lvert R\rvert$ small | Both small | Flat area |

## 5.4 Thresholding

Thresholding is an attempt to separate the image into object points and background points. It can be done globally (same threshold value everywhere) or variably (different value in different areas). Sometimes multiple thresholding is done, to separate the image into more than two parts.

Thresholding is made harder by noise, and uneven illumination/reflectance. Remedies are direct correction of shading pattern, top-hat transformation (Paragraph 4.6.3), and variable thresholding.

### 5.4.1 Basic global thresholding

Easy, but we need a way to select the threshold automatically:

1. Select initial threshold $T$.
2. Segment image with $T$.
3. Compute mean intensities $m_1$ and $m_2$ of the two groups.
4. Compute new threshold $T = \frac{1}{2}(m_1 + m_2)$.
5. Repeat until convergence.

This works well when there is a clear valley in the histogram.

### 5.4.2 Otsu's method

Otsu's method is based on maximizing between-class variance:

1. Compute normalized histogram $p_0 \dots p_{L-1}$.
2. Compute cumulative sums $P_1(0) \dots P_1(L-1)$.
3. Compute cumulative means $m(0) \dots m(L-1)$.
4. Compute global intensity mean $m_G$.
5. Compute between-class variance $\sigma_B^2(0) \dots \sigma_B^2(L-1)$.
6. Obtain Otsu threshold $k^* = \operatorname{argmax}_k \sigma_B^2(k)$.
7. Obtain separability measure $\eta^*$.

### 5.4.3 Improving global thresholding with smoothing

Even with Otsu's, noise can make thresholding fail completely. Smoothing the image first can often make it succeed again, although heavy smoothing can distort the edges.

### 5.4.4 Improving global thresholding with edges

In some cases, such as with noisy images where the object is very small compared to the background, edge detection-based thresholding is useful.

1. Compute an edge image with some method from Section 5.2.
2. Set a threshold $T$.
3. Threshold the edge image.
4. Compute a histogram of only the pixels in the original that are 1-valued in the edge image.
5. Segment this histogram globally, with e.g. Otsu.

### 5.4.5 Multiple thresholds

Sometimes you want to segment an image into three groups of intensity values, which is done with two thresholds. (More than two thresholds usually consider other properties than intensity.) A modification of Otsu exists for dual thresholds.

### 5.4.6 Variable thresholding

**Image partitioning** Partition the image into smaller images, and threshold each part separately. Sufficiently small parts should make, say, uneven illumination appear fairly even within the part.

**Thresholding based on local properties** Threshold based on a predicate of the neighborhood. If a function $Q$ of pixels near the current is true, mark that pixel a 1.

**Moving average** Good for stuff such as images of text with uneven illumination. As long as the text stands out from the local background, it's marked a 1. Global thresholding will fail in such situations.

#### 5.4.7 Multivariable thresholding

With color images, we might want to do thresholding based on more variables than greyscale intensity. An example is the "distance" from a pixel in the image to a given color, such as only keeping pixels that are sufficiently red.

## 5.5 Region-based segmentation

Instead of finding edges or doing thresholding, let's just find the regions directly.

#### 5.5.1 Region growing

Start with a "seed" of points, and grow these into regions:

1. Erode connected components of the seeds to single pixels.
2. Compute a predicate image with ones for all pixels that satisfy the predicate.
3. Append to the seed points all ones from the predicate image that are 8-connected to the seed.
4. Label each disjoint region.

#### 5.5.2 Region splitting and merging

Based on splitting the image into sections and merging them to satisfy the list in Section 5.1. One method is to divide the image into quadrants, and for each quadrant that fails a predicate, divide it further. Afterwards, adjacent regions that fulfill the predicate are merged.

#### 5.5.3 $k$-means clustering

$k$-means clustering is a method that can be used to segment an image into $k$ categories based on some vector of quantitative features (such as color levels). The algorithm is

1. In the space of all possible feature vectors, initialize the centers of $k$ categories randomly.
2. Assign each pixel to the nearest category.
3. Based on the pixels assigned to each category, compute the centroid of their feature vectors as the new category centroids.
4. Repeat until convergence.

This method

## 5.6 Morphological watersheds

We look at the image in a topographic way, letting intensity levels represent heights. Then "fill" the image by rising water from below. Notice which areas fill separately, and where the water connects when the level is high enough.

As the water fills, build pixel-wide dams to keep the basins from connecting. These dams define the segmentation boundaries between regions. Noise and irregularities lead to oversegmentation. You can preprocess to generate markers, which are the only allowable starting points for the water filling, in order to reduce this.

## 5.7 Using motion for segmentation

It can be useful to detect regions by looking at movement.

#### 5.7.1 Spatial techniques

A very simple method is to look at pixel-level differences: Make a difference image of ones for all pixels that have changed significantly. To suppress noise, you can form 4- or 8-connected regions in the difference image and ignore too small regions.

Another method is accumulative differences: Increment a counter for a given pixel when a significant change occurs there.

For these methods to work, we need a good reference image. The best is one with only stationary elements. If you cannot capture a stationary image directly, it can be constructed from several images.

# 6 Representation and description

The results of image segmentation (Section 5) can be represented and described in certain ways to be useful for computers. For instance, a region can be represented by its boundary, and the boundary described by its lenght.

## 6.1 Representation

Can be external (the boundary) or internal (the pixels within).

### 6.1.1 Boundary following

Produces an ordered sequence of points from a *binary* region $R$. The following algorithm is also called the *Moore boundary tracking algorithm*:

1. Start at $b_0$, the uppermost, leftmost point in $R$. Let $c_0 \notin R$ be the west neighbor of $b_0$. Examine all 8 neighbors of $b_0$ starting at $c_0$ moving clockwise. Let $b_1$ be the first neighbor that is in $R$, and $c_1 \notin R$ the one immediately before $b_1$.
2. Let $b = b_1$ and $c = c_1$.
3. Examine the neighbors $n_1, \ldots, n_8$ of $b$ starting at $c$. Let the first neighbor in $R$ be $n_k$.
4. Let $b = n_k$ and $c = n_{k-1}$.
5. Repeat Steps 3 and 4 until $b = b_0$ *and* the next boundary point found is $b_1$. The sequence of $b$ points is the set of ordered boundary points.

The output is the outer boundary of the region. The inner boundary can be found by extracting the "hole" and running it on that.

### 6.1.2 Chain codes

Chain codes represent a boundary by a sequence of line segments of specified length and directon. Usually 4- or 8-directional (number of possible directions). Direction is coded by numbers, as shown in Figure 8. The boundary is downsampled with a sampling grid first, to reduce the length of the sequence.

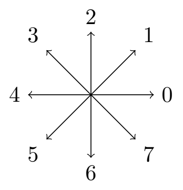


Figure 8: Example of direction numbers for an 8-directional chain code

The code depends on the starting point, but is a circular sequence that can be normalized by letting the starting point be the point that gives the lowest integer value of the sequence. The direction values can also be normalized by using the first difference instead of the absolute direction number.

### 6.1.3 Minimum-perimeter polygons (MPPs)

The goal is to encode the general shape using as few straight segments as possible.

One method is to overlay a coarser grid on the boundary, mark the cells visited by the boundary, and finding the shortest path trough these cells. The hard part is to automate the grid size choice. The algorithm itself is pretty complex stuff.

### 6.1.4 Other polygonal approaches

**Merging** This method merges points along a line until the squared sum of errors between the boundary and the line is too large. When the threshold is reached, the procedure repeats with a new line.

**Splitting** Start with a line, and find the point that fits the line least. Add that point to the perimeter polygon (now a triangle), and continue until good enough.

### 6.1.5 Boundary segments

It can be useful to separate the boundary into segments. This can be done by finding the convex hull of the shape, and cutting the boundary when it moves in and out of convex deficiencies. See Section 4.5.4.

For most images, noise etc. will create many small, meaningless convex deficiencies. These can be removed/reduced by smoothing the region, averaging pixels along the boundary, or by using a polygonal approximation.

### 6.1.6 Skeletons

Skeletonizing reduces a shape to a graph, which represents the structural shape of the region. Can be done by thinning (Section 4.5.5), but that doesn't ensure connectivity.

The *medial axis transformatian* can be used instead: You have a region $R$ with border $B$. For each point $p \in R$, find its closest neighbor in $B$. If $p$ has more than one such neighbor, it is part of the medial axis. The medial axis then defines the skeleton, which is guaranteed connected. This is computationally heavy to do directly, and there are some more optimized algorithms around.

## 6.2 Boundary descriptors

### 6.2.1 Simple descriptors

- Length: Number of pixels along boundary. For a chain-coded curve: $n_{\text{vertical}} + n_{\text{horizontal}} + \sqrt{2} \cdot n_{\text{diagonal}}$.
- Diameter: $\max_{i,j} D(p_i, p_j)$.

### 6.2.2 Shape numbers

The shape number of a boundary is the first difference of its chain code, rotated to minimize the integer value. The number of digits of the code defines the order $n$ of the shape number (which is even for a closed boundary).

### 6.2.3 Fourier descriptors

A boundary can be described as a sequence of $K$ points where each point is represented by a complex number:

$$s(k) = x(k) + \mathrm{i}y(k), \quad k = 0, \dots, K-1 \tag{86}$$

The DFT and inverse DFT of this is

$$\begin{aligned} a(u) &= \sum_{k=0}^{K-1} s(k) \mathrm{e}^{-\mathrm{i}2\pi \frac{uk}{K}} \\ s(k) &= \frac{1}{K} \sum_{u=0}^{K-1} a(u) \mathrm{e}^{\mathrm{i}2\pi \frac{uk}{K}}. \end{aligned} \tag{87}$$

However, if we take the DFT and recreate $s(k)$ with only the first $P$ coefficients, we can get a decent approximation:

$$\hat{s}(k) = \frac{1}{P} \sum_{u=0}^{P-1} a(u) \mathrm{e}^{\mathrm{i}2\pi \frac{uk}{P}}. \tag{88}$$

This is a sort of low-pass filtering. If you simplify to 4 descriptors, you get an ellipse, and with 2 a circle.

### 6.2.4 Statistical moments

Take a boundary segment and rotate it so that the line between its ends is horizontal. The resulting histogram $p(v_i)$ can be described by statistical moments. The mean is

$$m = \sum_{i=0}^{A-1} v_i p(v_i) \tag{89}$$

and the $n$th moment is

$$\mu_n(v) = \sum_{i=0}^{A-1} (v_i - m)^n p(v_i). \tag{90}$$

This essentially describes a boundary as 1D functions.

## 6.3 Regional descriptors

Regional descriptors are (obviously) used to describe regions rather than boundaries.

### 6.3.1 Some simple descriptors

- Area $A$
- Perimeter $P$
- Compactness $P^2/A$
- Circularity ratio $R_c = 4\pi A/P^2$
- Mean and median intensity
- Min and max intensity
- Number of pixels over/under mean

### 6.3.2 Topological descriptors

Topology is the study of properties unaffected by deformation. Examples are:

- The number of holes, $H$.
- The number of connected components, $C$.
- The *Euler number*, $E = C - H$.

We can also extract e.g. the largest connected component of an image.

### 6.3.3 Texture

Textures can be e.g. smooth, coarse, or periodic.

**Statistical approaches** The variance $\sigma^2(z) = \mu_2(z)$ (see (90)) is useful, and can be used to define smoothness

$$R(z) = 1 - \frac{1}{1 + \sigma^2(z)} \tag{91}$$

which is 0 for uniform areas and nearly 1 for your acne-ridden face.

The 1st (standard deviation), 3rd (histogram skewness), and 4th (relative flatness) moments are also used, and sometimes even higher ones.

Another metric is the *co-occurence matrix* $\boldsymbol{G} \in \mathbb{Z}^{n \times n}$ for $n$ possible intensity levels. (Group intensities if $\boldsymbol{G}$ becomes too large.) It is based on a relative position operator, which can be for instance "one pixel to the right". Then, if there are, say, 3 occurences of intensity 2 immediately to the right of intensity 6, we get $\boldsymbol{G}_{6,2} = 3$. Then further methods can be used to characterize the co-occurence matrix.

**Spectral approaches** Three Fourier spectrum features are particularly useful:

- Prominent peaks give the direction of patterns.

- Location of peaks give the spatial period of patterns.
- Filtering out periodic features leaves nonperiodic elements which can be analysed statistically

To simplify detection and interpretation, you can represent the spectrum in polar coordinates as a function $S(r, \theta)$.

### 6.3.4 Moment invariants

The 2D moment of order $(p + q)$ of an $M \times N$ image is

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x, y) \tag{92}$$

and the centralized moment is

$$\mu_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (x - \bar{x})^p (y - \bar{y})^q f(x, y) \tag{93}$$

where

$$\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}}. \tag{94}$$

Then, the *normalized central moments* are

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \quad \text{with} \quad \gamma = \frac{p + q}{2} + 1 \tag{95}$$

Based on this, seven moments invariant to translation, scaling, mirroring, and rotation can be formulated. These are the *invariant moments*.

## 6.4 Use of principal components for description

Principal component analysis is a method of describing data by maximizing variance. The axis that maximizes variance if you project all data points onto it, is the first principal component (PC). That is, the first PC is the axis along which the data is most spread out. All subsequent PCs must be orthogonal to the all other. With many dimensions, you can find $N$ PCs, and describe the data in only those axes. This minimizes the euclidean norm error given the number of axes used.

Each axis is associated with an eigenvalue, and these are nonincreasing from the $n$th to the $(n + 1)$th PC. Use this to find which PCs to throw away when you want to reduce dimensionality without damaging the data too much.

**Method** A pixel in an image can be represented as a vector. For an RGB image, this can be $\boldsymbol{x} = (x_R, x_G, x_B)$. If you gather more data than RGB, the vector will be larger. The mean of all pixel vectors is

$$\boldsymbol{m_x} = E\{\boldsymbol{x}\} = \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{x}_k \tag{96}$$

and the covariance is

$$\boldsymbol{C_x} = E\left\{(\boldsymbol{x} - \boldsymbol{m_x})(\boldsymbol{x} - \boldsymbol{m_x})^{\mathrm{T}}\right\} = \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{x}_k \boldsymbol{x}_k^{\mathrm{T}} - \boldsymbol{m_x} \boldsymbol{m_x}^{\mathrm{T}}. \tag{97}$$

We can approximate $\boldsymbol{x}$ by calculating the eigenvectors of $\boldsymbol{C_x}$, placing them in a matrix $\boldsymbol{A}$, transforming

$$\boldsymbol{y} = \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{m_x}), \tag{98}$$

and finally transforming back using a modified $\boldsymbol{A}$ that only includes the eigenvectors for the $k$ largest eigenvalues:

$$\hat{\boldsymbol{x}} = \boldsymbol{A}_k^{\mathrm{T}} \boldsymbol{y} + \boldsymbol{m_x} \tag{99}$$

Then, $\hat{\boldsymbol{x}} \in \mathbb{R}^k$.

## 6.5 Feature parameter fitting using the Hough transform

The Hough transform can be used to detect and parametrize features such a lines found in an image. It requires an edge image, which can be obtained with e.g. the Canny operator (Section 5.2.5). For each edge pixel in the edge image, it loops over parameters for the model, and increments a value for each matching parametrization. The parametrizations with the "most votes" will then match features in the image.

A line can be parametrized by two parameters $a$ and $b$

$$y = mx + b, \tag{100}$$

but this gives unbounded $a$, and therefore an unbounded search space of possible parametrizations. Instead, we use the polar parametrization

$$\rho = x \cos\theta + y \sin\theta, \tag{101}$$

where $\rho$ is the shortest distance from the origin to the perpendicular line, and $\theta$ is the angle of the perpendicular, as seen in Figure 9. Now the parameters are bounded. We can search for any parametrization of a shape, but more parameters require more computation.

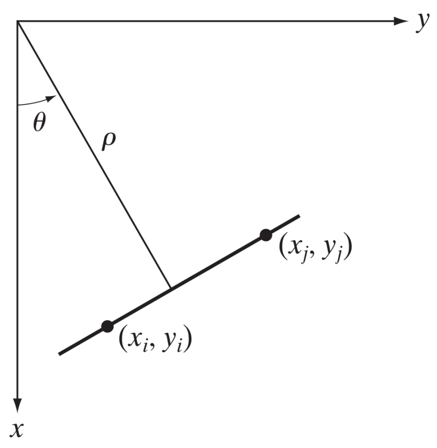


Figure 9: Polar parametrization of a line

## 6.6 Local feature description with SIFT

SIFT (scale-invariant feature transform) is an algorithm that detects and describes local features in an image. A rough outline is

1. Construct a scale-space: Gaussian-blur the image using increasing $\sigma$ to create a stack of images, ranging from sharp to super blurry. To approximate the LoG, take the difference of each adjacent pair of images. This gives a 3D space in $(x, y, \sigma)$.
2. Search for local extrema: Max and min values as seen relative to its neighbors in all directions $(x, y, \sigma)$.
3. Keypoint localization: Use a Taylor series to better find the location of the local extrema. Remove low-contrast keypoints and edge keypoints.
4. Orientation: Calculate gradient magnitudes in a scale-dependent neighborhood, and put into a gradient histogram. Use this to assign an orientation to the keypoint.
5. Keypoint descriptor: Use a 16 by 16 neighborhood, divide into 16 blocks, create an 8 bin orientation histogram for each block. You get 128 bin values, which becomes the feature vector of the keypoint.
6. Matching: Use vector distances to determine matches between descriptors in different images.

# 7 Object recognition

Object recognition is the task of finding a specific object within an image.

## 7.1 Patterns and pattern classes

Patterns are arrangements of descriptors (or "features"). Pattern recognition is assigning observed patterns to a pattern class. Usually patterns are represented as vectors (quantitative), strings, or trees (structural). When using pattern vectors, the choice of descriptors is super important.

## 7.2 Recognition based on decision-theoretic methods

Based on decision functions. For pattern classes $\omega_1, \ldots, \omega_W$, we need functions $d_1(\boldsymbol{x}), \ldots, d_W(\boldsymbol{x})$, so that if $\boldsymbol{x}$ belongs in class $\omega_i$, then $d_i(\boldsymbol{x})$ has the largest value. The decision boundary between two classes are values of $\boldsymbol{x}$ where two decision functions have the same value.

### 7.2.1 Matching

**Minimum distance matching** This method matches candidate vectors $\boldsymbol{x}$ to prototype vectors $\boldsymbol{m}$ that represent each class. A candidate belongs to the class with the nearest prototype ("nearness" measured with some vector norm).

**Correlation matching** Using a template, you can shift it around on an image to find out where it fits best (in the sense of maximum correlation). This can easily be normalized for intensity changes, but not so easily for size and rotation.

### 7.2.2 Optimum statistical classifiers

We can also formulate a classification that is optimal in the sense of minimizing classification errors.

### 7.2.3 Neural networks

Neural networks can be used to classify images and image components. More on this somewhere else.

## 7.3 Feature-based object recognition

This method depends on a lower-level method to extract features and interest points from images. These can then be matched between images to determine the location of an object. Usually your input is an image of the object alone, and an image of the object in a scene. The output is the location and orientation of the object within the scene. SIFT in Section 6.6 is one way to extract feature descriptors.

#### 7.3.1 RANSAC

RANSAC (random sample consensus) is a method of outlier detection that can be used when matching feature vectors between images, to remove false matches. It goes like this:

1. Repeat $N$ times:
   (a) Choose a subset $s \in S$ of points and fit your model to them.
   (b) See how many points in $S$ match the model.
2. Choose the best fitting model as your result.

Here, $S$ are all your data points. $s$ is a subset large enough to define your model (2 for a line, 8 for a fundamental matrix, etc.). $N$ is a predetermined number of iterations defined to guarantee a certain probability of finding a good model.

RANSAC can do robust estimation (accurate estimation despite many outliers). However, it cannot guarantee with finite $N$ that it finds a good model, it can only guarantee it to a specified probability. It needs very many iterations for sets with many outliers, but variations exist to solve this. The number of iterations necessary to guarantee a good model with probability $p$, given outlier ratio $\epsilon$ is

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)}. \tag{102}$$

## 8 Epipolar geometry

With one camera, you know the direction to each pixel/object in an image, but not its distance. With two cameras and epipolar geometry, this can be determined.

In Figure 10, both cameras see the object $\boldsymbol{X}$. The line $\boldsymbol{O}_L$–$\boldsymbol{X}$ from the left camera to $\boldsymbol{X}$, is seen as the epipolar line $\boldsymbol{e}_R$–$\boldsymbol{x}_R$ by the right camera, marked in red. (Correspondingly, $\boldsymbol{x}_L$–$\boldsymbol{e}_L$ is an epipolar line for the left camera.) This means that $\boldsymbol{X}$ must lie on the red epipolar line, but we don't know where (this is the fundamental ambiguity). This provides a constraint that can be used to create a model describing the geometric transformation between the two image planes.

### 8.1 Essential matrix

The epipolar constraint from the previous section can be formulated as

$$\boldsymbol{y}_1^{\mathrm{T}} \boldsymbol{E} \boldsymbol{y}_0 = 0 \tag{103}$$

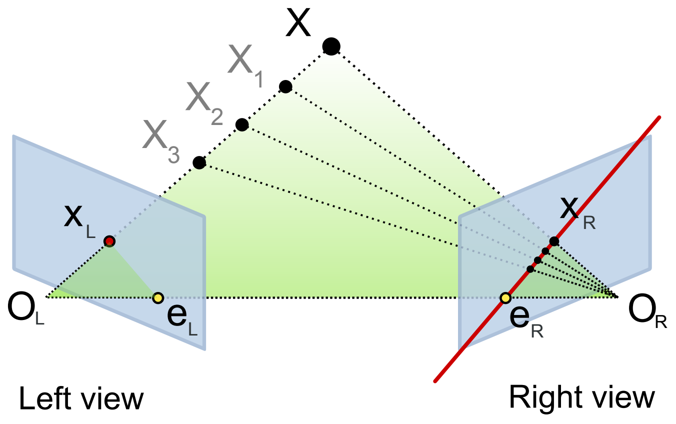


Figure 10: Epipolar geometry

where $\boldsymbol{y}_1$ and $\boldsymbol{y}_0$ are normalized image coordinates and $\boldsymbol{E}$ is the essential matrix. This relation holds if the coordinates correspond to the same point in 3D space. Thus, the essential matrix relates points in global coordinates.

### 8.2 Fundamental matrix

The fundamental matrix $\boldsymbol{F}$ does the same as the essential matrix, but relates points given in image pixel coordinates, such as $\boldsymbol{x}_L$ and $\boldsymbol{x}_R$ in Figure 10. A similar relation holds for image coordinates that describe the same point in space:

$$\boldsymbol{x}_1^{\mathrm{T}} \boldsymbol{F} \boldsymbol{x}_0 = 0 \tag{104}$$

### 8.3 Structure from motion (SfM)

SfM is a method for using data from many 2D images to create a 3D map. It's based on finding features in all the images taken (a huge amount is necessary). Match features between images to estimate camera location, and then pixel distances.

## 9 Optical flow

Optical flow is the task of describing how pixels move in an image, thus describing motion of surfaces or objects in an image. Mathematically, you calculate the motion at each pixel position between two frames at times $t$ and $t + \Delta t$.

## 9.1 Constraints

We usually assume constant brightness between frames

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{105}$$

where $I(\cdot)$ is image intensity. We also assume small motion

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, z) + \tfrac{\partial I}{\partial x}\Delta x + \tfrac{\partial I}{\partial y}\Delta y + \tfrac{\partial I}{\partial t}\Delta t. \tag{106}$$

This gives

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0. \tag{107}$$

Determining optical flow can be done in several ways. Sections 9.2 and 9.3 describe differential methods.

**Aperture problem** The aperture problem comes from the fact that what a camera sees is only a small part of the world. Motion inside of this frame may appear different than it really is, if the moving object is not contained in the frame.

## 9.2 Lucas–Kanade (LK) method

LK tracks interest points in a scene from one frame to the next. Can use Harris corner points as the interest points. It assumes constant flow in a neighborhood of the pixel examined, and solves the optical flow equations by least squares. Using a neighborhood reduces ambiguities and improves noise performance. But being purely local, it cannot determine flow in the interior of uniform regions.

## 9.3 Horn–Schunck (HS) method

A method to estimate optical flow globally. It introduces a global smoothness constraint to solve the aperture problem. It minimizes distortion in flow globally, and prefers solutions that are smoother.

## 9.4 Kanade–Lucas-Tomasi (KLT) feature tracker

KLT is a computationally fast feature extraction method that is useful for tracking in realtime video.