Università degli studi di Trento

Signal, Image and Video

Prof. Rosani

Academic Year 2024/2025

Contents

Chapter 1

Signals and Systems

Signals

A signal is a function that conveys information about a given phenomenon. It can be seen as the variation a given domain of some physical quantity. The same information can be associated to different signals that represent it.

Signals can be represented as functions of one or more variables. An audio signal is a function of time x(t), describing the intensity of a sound at a given instant. An image is a function of space $I(x, y)$, describing the luminance and/or color of a given point in space. A video is a function of space and time $V(x, y, t)$, describing the luminance and/or color of a given point in space at a given instant.

Systems

A system is a generic equipment, hw or sw, that modifies the signal in some way. More in general, the signal can be smoothed, attenuated, amplified, cut, stored, etc.: each operation requires a system that performs it. A system can be represented as a composite function that maps a signal into another

$$x(\cdot) \rightarrow SYSTEM \rightarrow y(\cdot) = \mathcal{F}(x(\cdot)) \quad .$$

It would be useful to express $\mathcal{F}(x(\cdot))$ in analytic form, in order to be able to predict the response of a system to any given input. This turns out to be difficult or even impossible in general, except for a special class of systems, called linear and time-invariant.

1.1 Linear Time-Invariant (LTI) systems

A system is linear if it fulfills the superposition property.

$$\begin{aligned} \forall x_i(t) : \tilde{S}(x_i(t)) &= y_i(t) \\ \forall x_j(t) : \tilde{S}(x_j(t)) &= y_j(t) \\ \tilde{S}(\alpha x_i(t) + \beta x_j(t)) &= \alpha y_i(t) + \beta y_j(t); \quad \forall \alpha, \beta \end{aligned}$$

A system is time-invariant if it fulfills the time-shift property.

$$\forall x(t) : \tilde{S}(x(t)) = y(t)$$
$$\tilde{S}(x(t-T)) = y(t-T)$$

A system that fulfills both properties is called LTI. Given an LTI system, we can calculate the response to any input signal if we know the response of the system to a single special function called unit impulse $\delta(t)$.
To reach this result we need to follow some steps:

- Approximate the input signal with series of rectangular waves R(t) of fixed duration ΔT and unit area.
- Observing the response of the system to R(t), call it $h_R(t)$. Now, since the system is LTI, then the response to a weighted sum of inputs R(t) is the weighted sum of the corresponding outputs $h_R(t)$. A translation of R(t) by any arbitrary interval T produces the corresponding output $h_R(t)$ translated by the same time interval $h_R(t-T)$.

$$x_R(t) = \sum_k x(k\Delta T) \cdot R(t - k\Delta T) \cdot \Delta T \approx x(t)$$

- We improve the approximations of step 1 and 2 by sending $\Delta t \to 0$. In the limit, R(t) becomes a function with null duration and infinite amplitude, but still unit area. This function is just theoretical and is called unit impulse (Dirac delta). It is represented with a vertical arrow. $\lim_{\Delta t \to 0} R(t) = \delta(t)$ accordingly, in the limit $h_R(T)$ becomes the response to a unit impulse, called impulse response $h(t)$. $x(t)$ becomes an infinite series of consecutive impulses, and should be written in integral form. The same holds for $y_R(T)$.

1.2 Convolution Theorem

Given an LTI system with impulse response $h(t)$, the response to a generic input signal $x(t)$ will be the **convolution integral**:

$$y(t) = \int_{-\infty}^{+\infty} x(\tau)h(t-\tau)d\tau = x(t) * h(t)$$

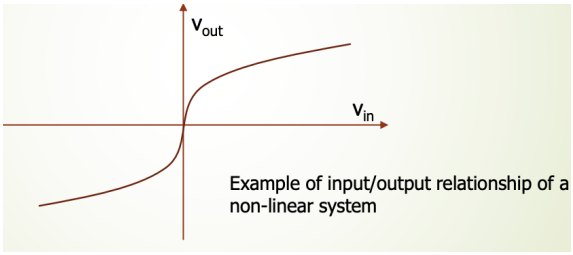
This means that, for an LTI system, it is sufficient to know the impulse response to analytically calculate the response to any input signal.

1.3 Not LTI systems

If the system is not LTI, impulse response exists, but it does not describe the behavior of the system for a general input. Convolution theorem does not hold. We can just represent the behavior of the system at a specific point in the domain: the so-called Input/Output Relationship.

I/O Relationship

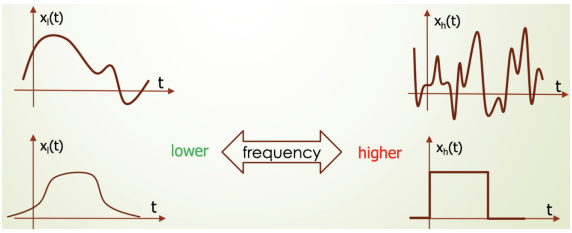
The input/output relationship determines the output value for a given input value. It is not a function of time or space, but it simply expresses the behavior of the system at a specific point in the input domain (time, space, ...). For an LTI system, the input/output relationship exists and it is simply a straight line.



1.4 Frequency

In physics, the concept of frequency is typically connected to periodic events (e.g., the oscillations of a pendulum). In signals the concept is somewhat larger. Higher frequencies are associated to signals that:

- Present more variations per domain unit → similar to periodicity
- Present steeper transitions → less intuitive



The previous definition is qualitative. In many cases we need to precisely calculate the frequency content of a signal.

1.4.1 Fourier Transform

Fourier demonstrated that a signal can be decomposed in an infinite series of sinusoidal waves, with varying amplitude, frequency and phase. For any given frequency, the FT provides:

- amplitude → how much that frequency is present in the signal
- phase → which is the offset of that frequency in the signal

To represent both, we need a complex function.

To calculate how much a given frequency f_0 is present in a signal $x(t)$, we need to "correlate" the signal with the sinusoid at f_0:

- multiply the signal by a complex sinusoid at frequency f_0
- integrate over the whole domain to obtain the **Fourier Transform:**

$$X(f_0) = \int_{-\infty}^{+\infty} x(t)e^{-j2\pi f_0 t}dt$$

$X(f_0)$ is a complex number with $|X(f_0)|$ that is modulus (intensity of frequency f_0 in $x(t)$, $argX(f_0)$ that is phase (offset of frequency f_0 in $x(t)$.

Extending the above computation to the whole frequency domain we obtain the forward transform. It is a continuous function of f, returning all the (possibly infinite) frequency components of $x(t)$. It is also possible to reconstruct the signal $x(t)$ from its frequency representation (**inverse transform**). We need to sum up all the frequency components, each one with the amplitude and phase given by the corresponding FT value:

$$x(t) = \int_{-\infty}^{+\infty} X(f)e^{j2\pi ft}df$$

`Example: Sinusoids`

If we apply the above definition to a sinusoidal signal, it's clear that only the frequency corresponding to the input will "correlate", all the other will output a null value. Accordingly, the transform of a sinusoidal signal at frequency f_0 will be an impulse in f_0 and zero otherwise. The difference between a cosine and a sine will just a phase shift (give by j):

$$F(\cos(2\pi f_0 t)) = \frac{1}{2}[\delta(f - f_0) + \delta(f + f_0)]$$
$$F(\sin(2\pi f_0 t)) = \frac{1}{2j}[\delta(f - f_0) - \delta(f + f_0)]$$

Due to the mathematical model of the transform, the frequency representation of real signals presents a symmetry, then there is always a matching "negative frequency" counterpart (with no physical meaning).

`Example: Impulse vs Costants`

An impuse signal (Dirca delta) contains all the frequencies with the same amplitude and shift. Accordingly, its transform is a constant in the frequency domain.

Conversely, the transform of a constant is an impulse frequency. This is more intuitive, if we think to a constant as a sinusoid at frequency zero.

$$\mathcal{F}(\delta(t)) = 1$$
$$\mathcal{F}(1) = \delta(f)$$

An impulse is the steepest possible variation: as such, it contains all possible frequencies. This also explain why impulses are so representative of a system (convolution theorem): they excite all frequencies responses.

`Example: Rectangles` Another function we came across is the rectangle $R(t)$, let see its Fourier counterpart:

$$\mathcal{F}(R_T(t)) = T\frac{sin(\pi fT)}{\pi fT} \triangleq sinc(fT)$$

where $R : T(t)$ is a rectangular function of duration T and unit amplitude, centered on $t = 0$. Also in this case the steep transitions produce infinite frequency components, but with a damping factor proportional to $\frac{1}{T}$.

1.4.2 FT vs LTI Systems response

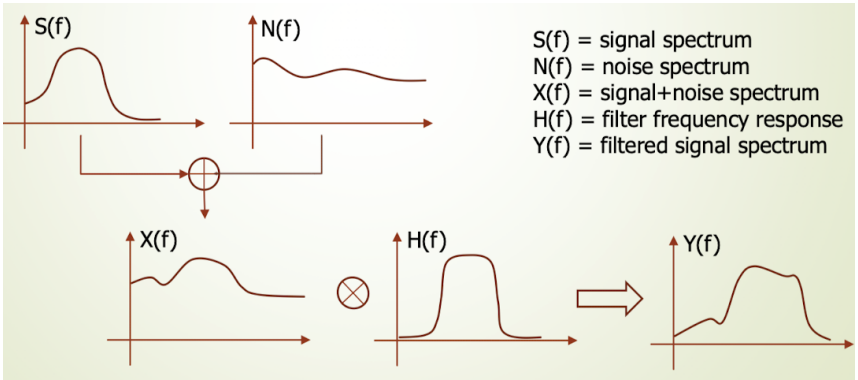
We can calculate the response to an LTI systems provided that we know its impulse response. This operation (convolution) could be difficult to calculate. In the Fourier domain the concolution becomes a simple product:

$$\mathcal{F}(x(t) * h(t)) = \mathcal{F}(x(t)) \cdot \mathcal{F}(h(t)) = X(f) \cdot H(f) = Y(f)$$

The transform of the impulse response $h(t)$ is called **frequency response** $H(f)$ of the LTI system, and it completely determines the behavior of the system. It's true ONLY for LTI systems.

1.5 Linear filters

If we have a sound immersed in noise and we need to design a system to reduce noise while preserving the sound.
Typically signals have different frequency spectra. Linear filters modify frequency spectrum components. The filter in the figure is called bandpass (BPF). It cuts



lower and higher frequencies, and leaves almost unchanged the intermediate ones. We can place the lower/upper cuts as to preserve the wanted signal and remove as much as possible the unwanted signal (noise). In thee frequency domain (Fourier)

its behavior is: $Y(f) = X(f) \cdot H(f)$.
In the time domain is much harder: $y(t) = x(t) * h(t)$, where $h(t)$ is the inverse transform of $H(t)$, typically, a sort of *sinc* function.

Exactly the same way it is possible to define various types of filters useful for different applications:

- LOW-PASS FILTER (LPF): cuts higher frequencies leaving lower ones unchanged
- HIGH-PASS FILTER (HPF): cuts lower frequencies leaving higher ones unchanged
- ALL-PASS FILTER: leaves signal unchanged
- NOTCH FILTER: cuts a given part of the spectrum from the signal leaving the rest unchanged

Typical usage:

- Noise removal (LPF, BPF)
- Interference removal (BPF)
- Signal separation (BPF)
- Highlighting variations (HPF)

Resume

- Signals convey information, systems are used to manipulate them.
- LTI systems are particularly easy to handle, they response can be calculated based on impulse response and convolution theorem.
- Non-LTI systems can only be studied in terms of input/output relationship (on a fixed time instant).
- Signals can be decomposed in sinusoids, thanks to the Fourier Transform: this allows understanding their frequency content.
- Fourier transform is reversible: I can easily switch from time to frequency domain and vice versa without loosing information.
- In the Fourier domain it is much easier to understand the behavior of systems (filters), but this applies only to LTI systems.

Chapter 2

From analog to digital signal

2.1 Analog vs Digital

Analog signals

Signals acquired by sensors are typically in analog form. Continuous in time/space and in amplitude. This kind of signals cannot be managed directly by a computer, it's necessary to convert them in binary format.

Digital signals

Digital signals present several advantages with respect to their analog counterparts: easier processing, transmission, storage (compressed and stored on data repositories) and interface/interoperability.

2.2 Analog to digital conversion (ADC)

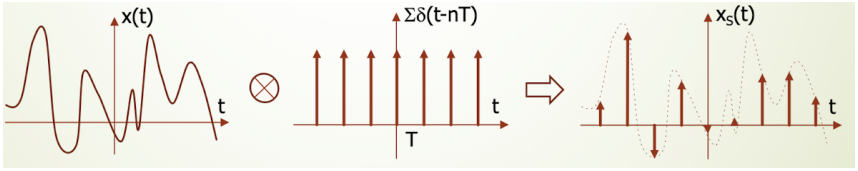
An analog-to-digital converter is an equipment that transforms an analog signal into a numerical one. It performs three main operations, in sequence:

1. Discretization in time (space) → sampling
2. Discretization in amplitude → quantization
3. Numerical representation of samples → coding

2.2.1 Ideal sampling, Nyquist theorem and aliasing

Sampling can be thought as the operation of extracting a series of instantaneous values at specific points in the original signal. Ideally, this operation can be implemented by multiplying the signal by a sequence of equally-spaced unit impulses. Each impulse captures the amplitude of the signal at a specific point and associates it to the area of the impulse itself.

$$x_s(t) = x(t) \cdot \sum_{n=-\infty}^{+\infty} \delta(t - nT) = \sum_{n=-\infty}^{+\infty} x(nT) \cdot \delta(t - nT)$$



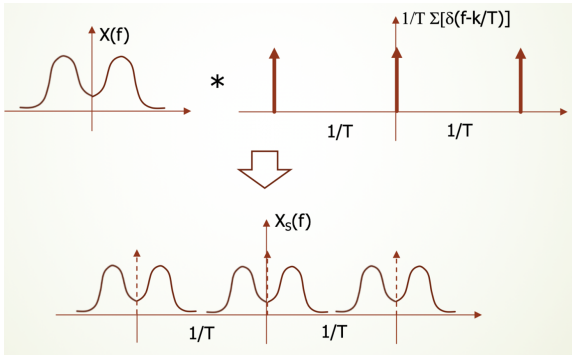
The distance between two consecutive impulses T is called **sampling interval**. More commonly, we use the inverse of T, which express the **sampling frequency** $f_s = \frac{1}{T}$. The sampling frequency cannot be completely arbitrary and independent of the signal characteristics. To derive the minimum sampling frequency, we need to look at the signal in the Fourier domain. To this purpose we should introduce the FT of an impulse series:

$$\mathcal{F}\left(\sum_{n=-\infty}^{+\infty}\delta(t-nT)\right) = \frac{1}{T}\sum_{k=-\infty}^{+\infty}\delta\left(f-\frac{k}{T}\right)$$

Then the frequency spectrum of the sampled signal $x_S(t)$ will be:

$$\mathcal{F}\left(x(t)\cdot\sum_{n=-\infty}^{+\infty}\delta(t-nT)\right) = \frac{1}{T}X(f) * \sum_{k=-\infty}^{+\infty}\delta\left(f-\frac{k}{T}\right)$$

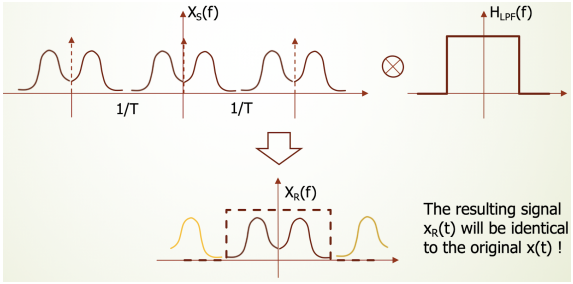
NOTE: The convolution of a signal with an impulse simply translates the signal at the impulse location.
Graphical form of the result:



We can observe that $X_s(f)$ is a series of replicas of the spectrum of the originl signal, placed at a distance $\frac{1}{T}$ one from each other. If the situation is the one depicted in the previous image, we can easily recover the original signal from the sampled one with a LPF.

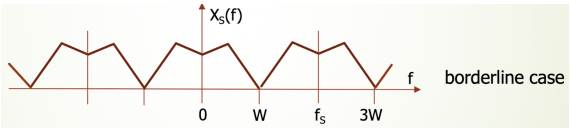
Aliasing

In order to be able to recover the continuous signal, the sampling frequency should be high enough to prevent replicas to overlap each other. If this happens the replicas will mix, making it impossible to reconstruct the original signal. This effect is called aliasing.



Aliasing introduces unwanted high frequencies into the signal (lowpass components of the replica overlap to high frequency components of original spectrum). This reflects in spurious high frequencies in reconstructed signal.

To avoid aliasing, the sampling frequency must be set as to prevent overlapping of replicas. This means that the second replica must be placed at a distance $\frac{1}{T}$ equal or grater than twice the maximum frequency present in the signal (the so-called signal bandwidth). Nyquist criterion:



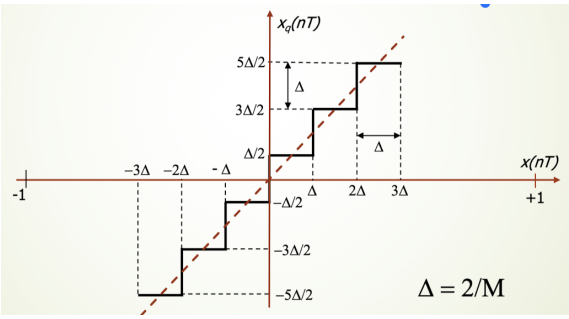
$$f_s = \frac{1}{T} \geq 2W$$

The process described is ideal. In the real world, it is impossible to generate impulses. The sampling signal will be some kind of rectangular function. The system takes the name of **sample&hold**, as it acts as a filter that gets the amplitude of the continuous signal at the sampling instant and retains it for the sampling period $T = \frac{1}{f_s}$. This results in a deformation of the signal spectrum that requires an **equalization**.

2.2.2 Quantization, quantization noise

In the sampled signal, each sample has still continuous amplitude values, and therefore could not be expressed in numerical way. It is necessary to discretize also the amplitude à quantization. A quantizer is a non-linear component that associates the continuous input samples to a pre-defined set of possible values, called **quantization levels**.
The association between and input value (continue) is associated to an output value (discrete) is performed according to a minimum distance criterion.
NB: This operation always introduces an error, called quantization error or quantization noise (distance between original and quantized samples).

A uniform quantizer subdivides the input range in M equal ranges associated to M equidistant levels. It could be described by a stair function with equal steps.



Encoding

The last step of the analog-to-digital conversion is the encoding of the quantized samples. Since at this point we have a finite number of discrete levels, we can code them with a finite number of bits. Given M quantization levels, we will need $log2M$ bits, or the larger nearest integer is M is not a power of 2. Typically we will chose M as a power of 2: $M = 2B$.
At this point, the continuous signal $x(t)$ has been converted to a discrete sequence $x(n)$, where each element of the sequence is represented by a B-bit binary code.

Quantization error

If we take a single step of the stair function, we can characterize the error function as a sawtooth wave:

$$x(nT) = x_q(nT) + e_q(nT)$$

Where e_q is called quantization error and behaves like a random noise.

Quantization noise

Although the quantization error is systematic, the input signal can be considered a random variable. If we assume the input signal to be equally distributed, the error will inherit the same statistical properties, then: e_q is a random variable equally distributed in $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$.
We can calculate the power of the quantization noise as the mean square value of the error $\overline{e^{q^2}}$. Since the expectation of e_q is zero, $\overline{e^{q^2}} = \sigma_q^2$, therefore:

$$\sigma_q^2 = \int_{-\Delta/2}^{\Delta/2} e_q^2 f(e_q) e_q de_q = \frac{1}{\Delta} \int_{-\Delta/2}^{\Delta/2} e_q^2 de_q = \frac{1}{\Delta} \frac{e_q^3}{3} |_{-\Delta/2}^{\Delta/2} = \frac{\Delta^2}{12}$$

If we compare the power of the noise with the power of the signal, we have an estimate of how good the noisy signal is. This measure is called SNR (**signal to noise ratio**), and is typically measured in logarithmic scale (dB, or decibel).

$$SNR = 10 \cdot log_{10}(2^{2B}) \approx 6B$$

Every bit added to the quantizier increases the SNR of 6 dBs.

2.2.3 Bitrate

Once the signal is sampled, quantized and coded, it generates a stream of bits. For time signals it is useful to calculate the number of bits generated in a time unit (1 sec): this value is called bitrate.
A signal of bandwidth W, sampled at the Nyquist frequency and quantized as to achieve around $Q\,dB$ SNR, will generate a stream at bitrate:

$$bitrate = 2W \left\lfloor \frac{Q}{6} \right\rfloor$$

For instance, an audio signal with bandwidth $20KHz$ and desired quality $50dB$ will generate a bitrate: $20 \cdot 10^3 \cdot 8 = 160Kbps$.

2.3 Digital systems

To deal with numerical signals, we need numerical systems. A numerical system is a system that accepts numerical signals in input and produces numerical signals in output. In principle, we can converted an analog system into a numerical one: sampling the impulse response, working in the Fourier domain and applying discrete inverse transform. But, we typically need an infinite number of samples in the impulse response (IIR). It is possible to cut the response with a windowing (à aliasing). What we obtain is the so-called FIR (**finite impulse response)** filter.

2.3.1 Discrete convolution

Once we have calculated the discrete impulse response $h_{DW}(n)$, we can calculate the output through a discrete convolution. It's the discrete version of the analog convolution:

$$y(n) = \sum_{k=-M/2}^{M/2} x(n-k)h_{DW}(k)$$

It requires M product-sum operations per sample.

```
int input[N], output[N]        // input and output vectors
int imp_resp[M]                // impulse response
int hl = floor(M/2)            // half length of filter
load(input), load(imp_resp)    // load input data and kernel
for i in hl ... N-hl-1 {       // scan input vector w/o borders
    tmp = 0                    // init var
    for j in -hl ... hl {      // scan filter
        tmp += (input[i+j] * imp_resp[j+hl])   // filter
        }
    output[i] = tmp            // write result to output vector
   }
```

2.3.2 Discrete signals in the frequency domain (DFT)

Also the Fourier frequency representation has a discrete counterpart, called Discrete Fourier Transform (DFT). Given a finite sequence of samples, the DFT converts it into a finite sequence of samples (complex coefficients) with the same length. Each DFT sample is associated to a complex sinusoid.
Given a discrete input signal $\{x_n\}$ of length N: $\{x_n\} = x_1, x_2, ..., x_n$. Its DFT is the discrete sequence $\{X_k\}$ of equal length N:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{j2\pi}{N}kn} \quad \text{DFT}$$

The exponential terms represent complex sinusoids (Eulero) at multiples k of a basis frequency:

- k=0 $\rightarrow$ frequency zero (DC-term)
- k=1 $\rightarrow$ basis frequency (1 cycle over N samples)
- ...
- k=$K \rightarrow K-th$ frequency (k cycles over N samples)
- ...
- k=$N-1 \rightarrow$ highest frequency ($N-1$ cycles over N samples)

Each sinusoid is sampled in N equally spaced points (aliasing occurs at higher harmonics).

Inverse DFT (IDFT) and Fast transform

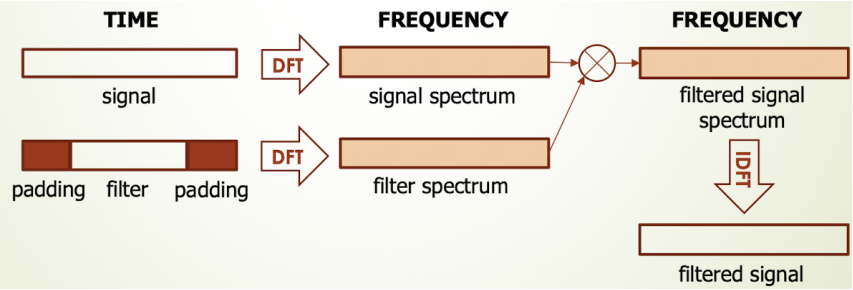
Given the DFT coefficients, it is possible to reconstruct the original sequence by calculating the weighted sum of the basis functions:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{\frac{j2\pi}{N}kn} \quad IDFT$$

In general, direct and inverse DFT require N product-sum operations per sample, with an overall complexity $O(N^2)$. There exists a fast algorithm called Fast Fourier Transform FFT that provides the same result with $O(N \cdot logN)$ complexity. FFT is largely diffused and implemented in most signal processing and math software libraries.

2.3.3 Filtering in digital frequency domain

Exactly the same way we can work in the Fourier domain with continuous signals, we can do it with discrete signals. Discrete time convolution becomes a product in DFT domain. In the discrete domain we just have to multiply numerical samples. In order to do that, we must ensure that sequences have the same length (**zero-padding** is applied to the shortest sequence).



Resume

- Digital signals are more robust and easier to manipulate
- Analog to digital conversion requires sampling (discretization in time), and quantization (discretization in amplitude)
- Sampling does not loose information, if it is performed correctly (sampling theorem), otherwise it produces artifacts (aliasing)
- Quantization always looses information, but such loss can be limited to an acceptable level by using enough bits
- To handle numeric signals we need numerical systems: as for the analog case, for LTI systems we can define a numerical impulse response, and a numerical convolution
- An equivalent of the frequency representation also exists for digital signals (DFT), with similar properties and use of the analog counterpart

Chapter 3

From 1D to m-D signals

3.1 From 1-D to m-D signals (images)

Some signals require more than one dimension to be expressed:

- Classical pictorial images require 2 dimensions $\rightarrow i(x, y)$
- Volumetric data (e.g., CAT, NMR) require 3 dimensions $\rightarrow d(x, y, z)$
- Volumetric data (e.g., CAT, NMR) require 3 dimensions $\rightarrow d(x, y, z)$
- Videos require a mixed space-time 3D domain $\rightarrow v(x, y, t)$
- Moving point-clouds require 4 dimensions $\rightarrow c(x, y, z, t)$

Some elements of the theory we've just seen need to be adapted to deal with these more complex domains.

An image is typically generated by an acquisition device, which translates a m-D physical stimulus into a 2D signal. The stimulus can refer to any physical quantity. An appropriate sensor is required to convert that specific physical phenomenon into an electrical signal.

3.1.1 Acquisition process

Independently of the physical phenomenon to measure, the acquisition process is more or less similar, and it is made of:

- An image formation system: projects the desired snapshot of the m-D reference world into the 2D image plane.
- A sensor: translates the physical quantity to be measured within the image plane into an electrical signal
- A recorder: stores the acquired signal into some physical device

Depending on the physical nature of the signal to be acquired we will have different hardware components. E.g., optical sensor will be sensitive to visible light radiations, thermal sensors will be sensitive to infrared radiations, etc.

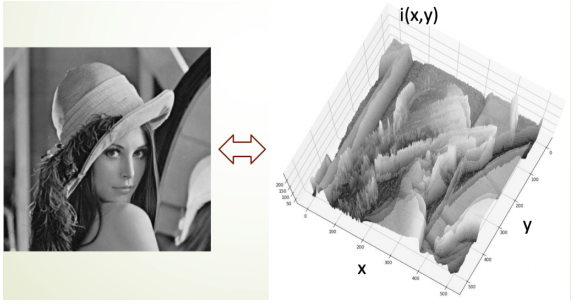
Acquisition vs human sensing

Humans are able to perceive a subset of possible physical stimula according to their

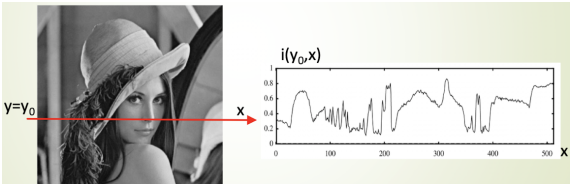
(limited) senses, machines can do more.
E.g., we can perceive images into the visible range of wavelengths, but we are unable to perceive infrared or ultraviolet.
Human sensing mechanisms are quite similar to artificial ones.

Image signals vs math

Let's consider a grey level picture, it can be seen as a continuous signal $i(x, y)$, whose amplitude is given by the luminance value at coordinates x, y.
An image can be also seen as a surface in the 2D space.



A simpler and sometimes useful way to represent an image signal is to extract scan lines (intensity profiles over a row or column).



3.1.2 Mathematical representation in space and frequency domains

Also in this case, frequency representation is often useful. Fourier transform straightforwardly extends to the 2D domain:

$$I(f_x, f_y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} i(x, y) e^{-j2\pi(f_x x + f_y y)}\, dx\, dy \quad \text{Direct 2D FT}$$

$$i(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} I(f_x, f_y) e^{j2\pi(f_x x + f_y y)}\, df_x\, df_y \quad \text{Inverse 2D FT}$$

NOTE: We are implicitly saying that the 2D signal is represented by a combination of horizontal (f_x) and vertical (f_y) frequency components.

Spectrum

The 2D spectrum makes evident the low-pass nature of the image. This characteristic is very common in natural images, and is associated to concepts such as smoothness, correlation, slow variation

3.2 Multi-dimensional analog systems

To process multi-dimensional signals we need multi-dimensional systems. In the 2D case: $f(i(x,y)) = \hat{i}(x,y)$.
Linearity and domain invariancy are again important properties. A 2D system is called LSI (linear, space-invariant) if it fulfills both superposition and shift properties:

$$\forall i(x,y) : f(i(x,y)) = \hat{i}(x,y)$$
$$\forall j(x,y) : f(j(x,y)) = \hat{j}(x,y)$$
$$f(\alpha i(x,y) + \beta j(x,y)) = \alpha\hat{i}(x,y) + \beta\hat{j}(x,y); \forall \alpha, \beta \quad \text{superposition}$$

$$\forall i(x,y) : f(i(x,y)) = \hat{i}(x,y)$$
$$f(i(x - x_0, y - y_0)) = \hat{i}(x - x_0, y - y_0) \quad \text{shift invariance}$$

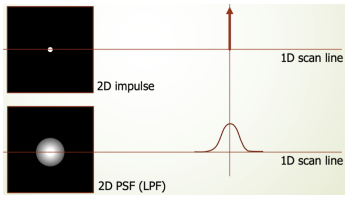
3.2.1 Extension of impulse response and convolution

Exactly the same way a 1D LTI system outputs the convolution between the input and its impulse response, an m-D system responds with a m-D convolution. Skipping the proof (analogous to the 1D case), in 2D we have:

$$\hat{i}(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} i(\lambda,\mu) \cdot h(x-\lambda, y-\mu)\, \partial\lambda\, \partial\mu = i(x,y) * h(x,y)$$

$h(x,y)$ is the 2D impulse response, i.e., the way our system reacts to a 2D impulse in input. Since a 2D impulse can be though as a point in space, and the system typically expands that point in some way, the 2D impulse response takes the name of **Point Spread Function (PSF)**.

`Example PSF`
We can see an impulse in space as a white dot on a black background (black being the absence of signal). PSF is the 2D output when an impulse is input. Its shape depends on the system characteristics.



3.3 Analog to digital conversion

Also in the case of image signals, if we want to manipulate them with a computer we need to convert them in binary form. we have to perform analog to digital (AD) conversion.

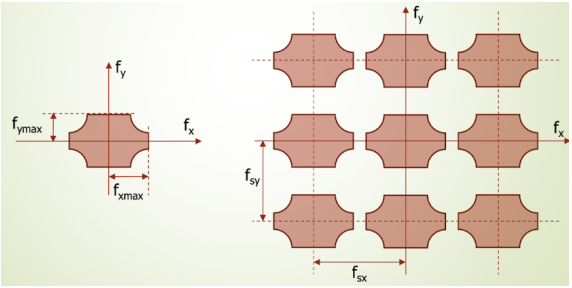
3.3.1 Sampling in 2D

Images are theoretically unbound in resolution, you can zoom and zoom and every time you'll find new details.

The maximum frequency in a captured image is not a property of the image itself, but it is enforced by the acquisition system. Acquisition systems (e.g., photo cameras) act as low-pass filters. We can determine the Nyquist frequency by Fourier transforming the continuous-space captured image. Since we are in 2D, we'll get two cut-off frequencies, on horizontal and vertical directions, respectively (f_{xmax}, f_{ymax}). To avoid aliasing, Nyquist criterion should be applied to both:
- $f_{sx} \geq 2 \cdot f_{xmax}$
- $f_{sy} \geq 2 \cdot f_{ymax}$

NOTE: Although this may lead to different sampling frequencies in the two directions, typically we will use the same (the maximum one) in order to achieve a square grid.

Without repeating all the theory, ideal sampling will generate replicas of the spectrum in the two frequency directions. If we fulfill the Nyquist criterion, replicas will not overlap.



Also in this case, if the sampling frequency is too low (under-sampling) **aliasing** will appear.

Pixels

After sampling, the image is converted into a discrete 2D collection of samples over an ordered grid (a NxM matrix). Each sample is called pixel, from the acronym of the words "PICture" "ELement". When we look at digital images we don't perceive pixels because they are very close to each other (the human eye acts as a LPF), but if we zoom it, pixels appear.

Aspect ratio

The aspect ratio is the ratio between horizontal and vertical dimensions of an image ($X : Y$). Some typical aspect ratio values are: 4:3 (old TV sets SDTV, VGA), 16:9 (newer TV sets HDTV, widescreen), 1.85:1 or 2.39:1 (cinema). Larger aspect ratios provide more "immersive" experience.

Image Size vs image Resolution

There is often a confusion between size and resolution of images:
- Size tells the number of pixels it is made of (rows x columns).

- Resolution refers instead to the number of pixels per unit space.
Clearly, if the same area is acquired with a sensor with more pixels, also the resolution will be higher. Resolution should be appropriately set depending on the application we target, for instance if I have to detect an object on an image, I need a resolution that provides a minimum given number of points for that object. If I have to print something on a given surface, I need a resolution that prevents perceiving the individual pixels (pixelation).

Image Size vs image Quality

Another common confusion is between image size and quality. Although having an insufficient number of pixels (with respect to the application) may hinder quality, having lots of pixels does not necessarily mean getting high-quality images. There are other important factors that determine quality:
- Optical components: quality and size of lenses and mechanics are fundamental (blurring, distortion)
- Sensor: quality and size of the sensor (CCD or CMOS) is also very important (low-pass effect, noise).

3.3.2 Quantization of visual signals

Pixel quantization

Up to this point, pixel are represented by continuous values in a given range. In order to complete the numerical conversion, we have to quantize them to achieve discrete amplitude values. In principle, quantization of pixel is completely equivalent to quantization of every other signal. In practice, we have to deal with the perceptual impact of the quantization on images.

Uniform quantization

Let consider a standard uniform quantizer at B bit per pixel (bpp), applied to a greyscale image. As we have seen in part 2, the SNR of our image will be $6B\ dB$. At $4bpp$ we get a $24dB$ image, which can be considered a low but nearly acceptable signal quality.

Contouring is due to the sensitivity of the HVS (human visual system) to contrasts. To avoid it we must be sure that transitions among quantized greylevels are below our perception threshold. If we double the number of bpp, thus having 256 grey levels, we are pretty sure that no contouring will be visible. Incidentally, $8bpp$ means 1 byte per pixel (perfect for memory alignment in computers) and around $50dB$ SNR (high-quality).

Contrast quantization

The **Weber's law** says that the HVS is unable to perceive more than 50 contrast variations. More precisely, Weber says that the HVS is not able to distinguish

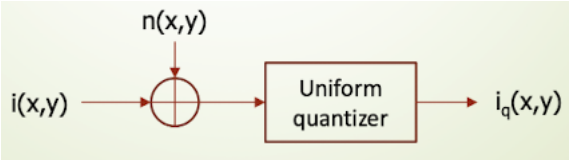
greylevel variations that are below 2%, or:

$$\frac{l_{fg} - l_{bg}}{l_{bg}} < 0.02$$

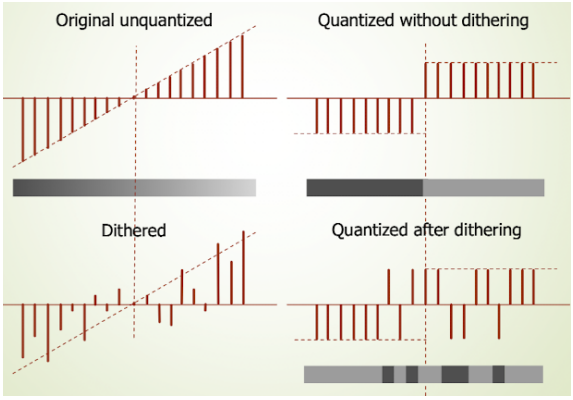
This means around 50 perceivable contrast levels (6 bits). To quantize the contrast it is possible to apply a nonlinear transform (log) followed by uniform quantization.

Dithering

And if we want to go further? For instance quantize at 3-4 bpp without introducing contouring?
The answer is again 'yes', but we can't use a simple quantizer anymore.
The idea is to fool the HVS with a 'trick'. The technique is known as dithering: we add a pseudo-random noise $n(x, y)$ (**dither**) to the image before quantizing it. The noise causes pixels nearby a greylevel transition to randomly switch their value with the neighboring one. Human eyes act as a lowpass filter, mixing the two values and virtually producing intermediate greylevels across the transition.



Let see what it happens across two quantization levels on a scanline:



Halftoning

And in the limit case that we have just 1 bpp?
It is still possible to represent the image with acceptable perceptual quality. Again we use dithering, but increasing the resolution and applying 1 bit quantization (black&white). It is possible to use random noise or deterministic patterns. It is the process used in printing (e.g., laser printing, offset printing).

3.3.3 Color representations

A peculiar characteristic of image signals is color. Color are represented with RGB vector. In the human eye there are multiple neural receptors:
- Rods: are sensitive to monochromatic light (b/w)
- Cones: divided in 3 types, sensitive to different wavelengths (color)

Given a color $C(\lambda)$, the way we perceive it's a function of the 3 cones' responses to the stimulus

$$\alpha_i(C) = \int_{\lambda_{min}}^{\lambda_{max}} S_i(\lambda)C(\lambda)d\lambda \ ; \ i \in [01, 2]$$

To synthesize a color that provides the same responses $\alpha_i(C)$ we need 3 sources $P_k(\lambda)$ such that:

$$\alpha_i(C) = \int_{\lambda_{min}}^{\lambda_{max}} S_i(\lambda) \left[\sum_{k=0}^{2} \beta_k P_k(\lambda) \right] d\lambda =$$

$$\sum_{k=0}^{2} \left[\beta_k^2 \int_{\lambda_{min}}^{\lambda_{max}} S_i(\lambda)P_k(\lambda)d\lambda \right] = \sum_{k=0}^{2} \beta_k \alpha_{i,k}$$

where $\alpha_{i,k}$ is the response of the i-th cone to the k-th source.

We can therefore theoretically define an unlimited number of color spaces varying the 3 primary sources P_k. Not all the spaces will be useful, however. Some color space may be physically unfeasible. Some color spaces may be incomplete (to generate some colors we need negative components β_k) or overcomplete (some licit mixtures produce colors not perceived by the HVS). Some color spaces are less suitable for a given application (e.g., to measure color distances).

Red, Green, Blue color space (CIE 1931) can be represented as a cube, where the main axes represent the intensity of the three primary colors (R = 700 nm, G = 546.1 nm, B = 435.8 nm). Along main axes we have intensity variations of pure RGB colors. Along main diagonal we have greylevels (black and white at vertices). Opposite vertices represent complementary colors CMY (cyan, yellow, magenta).

Based on the RGB color space, we can define 2 physically realizable models for color synthesis:

- Additive synthesis of colors (RGB): start from black, mix primaries to achieve the desired color. Typical of TVs, projectors, monitors.
- Subtractive synthesis of colors (CMY): start from white, mix primaries to achieve the desired color. Typical of printers

Color quantization

Two possibilities:
- Quantize colors in 3D space (vector quantization): perhaps more efficient in terms of bpp but complex

- Quantize color components: simple extension of what we've already seen, straightforwardly applied to color components
Typically the second solution is preferred. To preserve the byte alignment each component is quantized at $8pbb$, for a total of 24 bpp. This means about 16 millions of colors (often referred to as true color).
NB. This is well beyond the perceptual capabilities of our visual system

Color conversion

There are dozens of color spaces tailored to different applications. Typically the conversion between color spaces is just a linear transformation (a matrix operation).
`Example 1: RGB to YCbCr space, used in JPEG standard`

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

`Example 2: RGB to YUV space, used in analog TV transmission (NTSC)`

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

3.4 Multi-dimensional digital systems

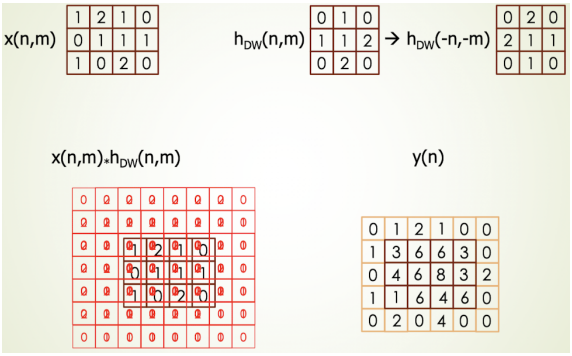
Now that we finally have a digital image, let see how to extend the concept of filtering in space and frequency. Let start from filtering in space. Let $i(x, y)$ be an analog image and $h(x, y)$ the PSF of a linear, space-invariant system. We apply AD conversion to both image and PSF to obtain their digital counterparts $i(n, m)$ and $h(n, m)$. Since $h(n, m)$ has typically infinite extension, we have also to window it in $[K, L]$ to obtain the finite-length discrete PSF $h_{DW}(n, m)$. Extending the concept already seen for 1D signals, the filtered image will be given by the 2D discrete convolution:

$$i_F(n, m) = i(n, m) * h_{DW}(n, m) = \sum_{k=-K/2}^{K/2} \sum_{l=-L/2}^{L/2} i(n-k, m-l) h_{DW}(k, l)$$

3.4.1 Discrete 2D convolution

Complexity and separability

Looking at the pseudo-code, it is easy to see that the 2D discrete convolution has quadratic complexity with respect to the PSF size. Sometimes, the complexity can be reduced by applying the separability property. A filter is separable if
$h(m, n) = h_1(1, m) * h_2(n, 1)$, then $i(n, m) = h(m, n) * h_1(1, m) * h_2(n, 1)$
i.e. a cascade of two 1D convolutions (linear complexity).



3.4.2 2D DFT and frequency domain digital filtering

To see how filtering works in the frequency domain, we have first to extend the frequency representation to discrete 2D domain (DFT 2D). Also in this case, the extension is rather straightforward. The basis functions become 2D:

$$e^{-\frac{j2\pi}{N}kn} \rightarrow e^{-j2\pi\left(\frac{km}{N}+\frac{lm}{M}\right)}$$

where (n, m) and (k, l) are space and frequency indexes, respectively.

NB1. The above basis functions suggest that the 2D frequency representation can be split into 2 orthogonal components (vertical and horizontal), diagonal components derive from their combination.
NB2. Again, we will get a number of coefficients equal to the number of samples (and a transformed image of the same size of the original one).

Since basis functions are 2D we can refer to them as basis images. To transform an NxM image, we need NxM basis images. Each basis image has a size NxM.

As for the 1D case, we can define the direct and inverse DFT as:

$$I(k,l) = \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} i(n,m)\cdot e^{-j2\pi\left(\frac{kn}{N}+\frac{lm}{M}\right)} \quad \text{DFT 2D}$$

$$i(n,m) = \sum_{k=0}^{N-1}\sum_{l=0}^{M-1} I(k,l)\cdot e^{j2\pi\left(\frac{kn}{N}+\frac{lm}{M}\right)} \quad \text{IDFT 2D}$$

- $0, 0 \rightarrow$ continuous wave component (DC-term)
- $0, i \rightarrow$ i-th pure vertical frequency (increasing with i)
- $j, 0 \rightarrow$ j-th pure horizontal frequency (increasing with j)
- $i, j \rightarrow$ mixed frequencies(increasing with i, j)

NB. Also in this case, FFT can be used to speed up the transformation, with complexity $O(NlogN)$.

The transformed domain is discrete and periodic (sampling). Conventionally, we place the "zero" frequency in the center.

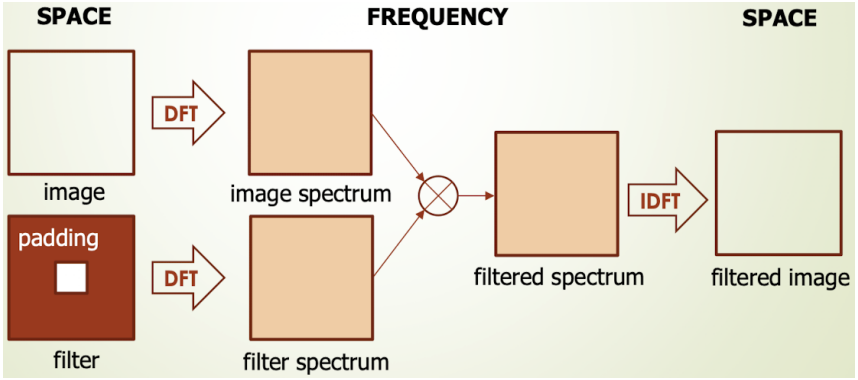


Image filtering in DFT domain

It is a trivial extension of the method we have introduced for 1D discrete signals. Apply DFT to 2D image and zero-padded PSF. Multiply the frequency spectra and apply IDFT to the result.

Resume

- Images are signals in multiple dimensions (2 or more)
- Their acquisition relies on sensing (mimicking human senses)
- 1-D signals and systems theory can be straightforwardly extended to m-D (representation in space and frequency, system response)
- Also A/D conversion is readily extended, but visual effects of sampling and quantization need to be addressed appropriately (it's important to understand how HVS works and adapt to it)
- Color is a peculiar feature of images and needs special care. Various representations are available for different scopes.
- Filtering in 2-D discrete domain, both in the spatial and frequency domains, is a direct extension of 1-D case

Chapter 4

Low-level image processing

The main purpose of low-level processing is to improve the image by correcting some acquisition problems, removing noise, improving the perceptual quality, etc.
There are two families of approaches:
- Pre-processing: operators that work on a single pixel (pixelwise).
- Filtering: operators that work over a given spatial region around the processed pixel.

4.1 Pre-processing

Pre-processing refers to a set of low-level algorithms applied to remove some artifacts (e.g., geometrical distortion) and improve the visual appearance of the image (e.g., enhancing contrast). Pre-processing takes in input an image and provides in output another (modified) image.
Three common operations:

1. Geometrical distortion correction
2. Color distortion correction
3. Histogram manipulation

4.1.1 Calibration and geometrical/color correction

Geometrical distorsion

Geometrical distortion is a phenomenon caused by the geometry of the lenses used in image formation. It is typically associated to a particular type of lens (e.g., a fish-eye), but it can be also due to defects in construction.

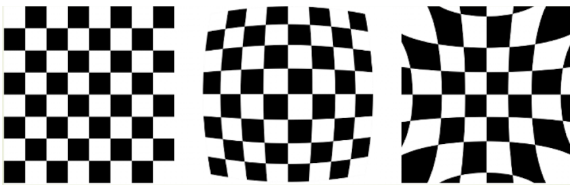
To remove the distortion, we first need to measure it. This process is called **calibration**, and concerns in estimating a 2D distortion function based on a parametric geometrical model of the distortion and a number of matching points to determine the model parameters. The model establishes a relation between (unknown) posi-

tions of real objects in the world and relevant image-plane coordinates

$$\begin{cases} x_d = x + \delta_x(x, y) \\ y_d = y + \delta_y(x, y) \end{cases}$$

where δx, δy are modeled as polynomial functions of x_d,y_d and take into account various phenomena (radial distortion, decentering distortion, prism distortion). The model has to be adapted to the specific camera by estimating the coefficients of the polynomial functions (10 or more).

To determine the model parameters, we have to establish a sufficient number of known terms and then solve a system. Usually, the number of known terms exceeds the number of unknowns to that we can solve it via numerical optimization to avoid ill conditioning problems. Known terms are determined by using a calibration image with a clear correspondence of points, for instance a chessboard.



Once the model is know, we can invert it to correct the image (map distorted coordinates to undistorted ones → dewarping). Blurring appears because we have to apply interpolation. When we dewarp the image, the new coordinates do not match the image grid anymore, then we have to interpolate the values on the grid based on their nearest neighbors (in some areas may be sparse).

Color distorsion

Systems may introduce non-linear alterations of the color components, thus modifying color perception. This may happen both in acquisition, processing and rendering, thus, across the whole processing chain we may have several modifications. As a result, the image we see on a monitor or on a printed picture will show completely different colors with respect to the original one. Also in this case, measuring the phenomenon is fundamental to correct it. Again, **calibration** uses models and a set of known values to determine the model parameters.

A simple yet effective correction mechanism used in cameras is the so-called white balancing. If colors are calibrated, neutral colors (greylevels, and in particular white), should appear as such. We therefore frame a white image, and manipulate the color mix (through appropriate curves) until the acquired image 'looks white'. The same color mix is then applied to all the other colors.
NB. This also corrects problems due to bad illumination

4.1.2 Histogram calculation and manipulation

We introduce now a set of simple operations that modify the image based on the relevant histogram. These operations are also called **pointwise operations**, in the

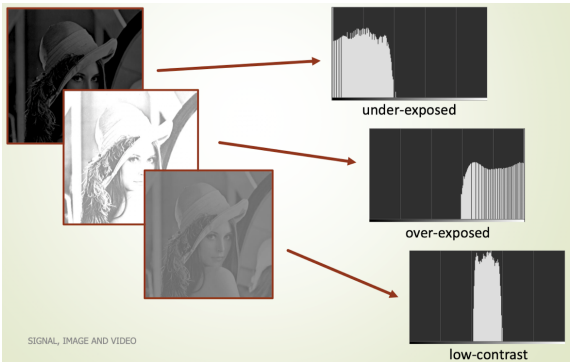
sense that they act on single image points (pixels) by changing their value according to a given function. Histogram manipulations are typically non-linear transformations, therefore, they should be studied based on the input-output relationship. We need first to introduce image histograms and more in general, image statistics.

We define the image histogram $h(p)$ as the frequency of occurrence of the intensity values associated to image pixels. If the image is quantized at b bpp (thus having 2^b possible intensity levels), its histogram will be made of 2^b bins. The p-th bin will be calculated as the number of pixels that assume the value p, normalized by the image size. Since the number of bins is low as compared to the image size, the histogram approximates quite accurately the PDF (probability density function) of the image

$$h(p) = \frac{\sum_{x=1}^{N}\sum_{y=1}^{M}\begin{cases}1 & \text{if } i(x,y)=p\\ 0 & \text{otherwise}\end{cases}}{N \cdot M}$$

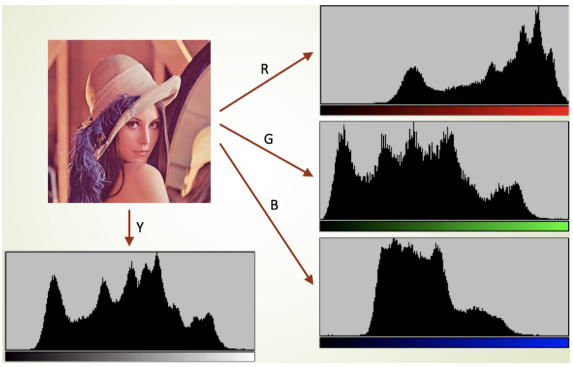
NB. For color images we usually calculate the separate R,G,B histograms, even if they provides an incomplete information on actual image colors.

Histograms provide an immediate feeling about some characteristics/defects of the image.



Histogram modes represent the peaks (or pseudo-gaussian components) of the histogram. Bimodal histograms may represent images with well-separated background and foreground).

`Example`



Starting from the histogram we could easily calculate:

- Cumulative histogram (distribution function):

$$Ch(\alpha) = Pr\{i < \alpha\} = \sum_{i=0}^{\alpha} h(i)$$

- Image expectation (mean value):

$$\mu = \frac{1}{N \cdot M} \sum_{x=1}^{N} \sum_{y=1}^{M} i(x,y) = \sum_{i=0}^{2^b-1} i \cdot h(i)$$

- Image variance:

$$\sigma^2 = \frac{1}{N \cdot M} \sum_{x=1}^{N} \sum_{y=1}^{M} [i(x,y) - \mu]^2 = \sum_{i=0}^{2^b-1} (i-\mu)^2 \cdot h(i)$$

Now we can easily define some simple operators based on the modification of the histogram:

- Histogram stretching
- Histogram clipping
- Histogram thresholding
- Histogram equalization
- Histogram inversion (image negative)

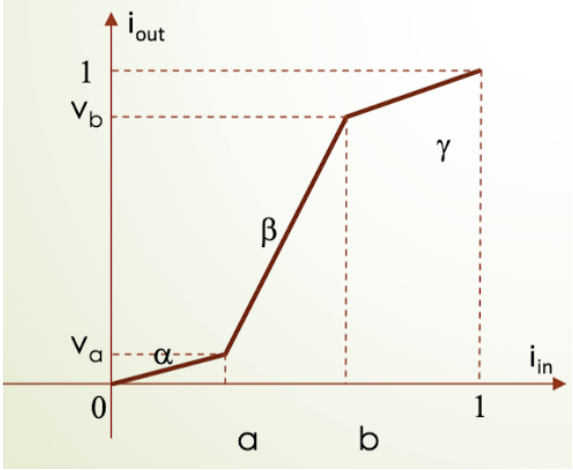
The general idea is to apply a (non-linear) input-output relationship to the original image histogram.

Histogram stretching

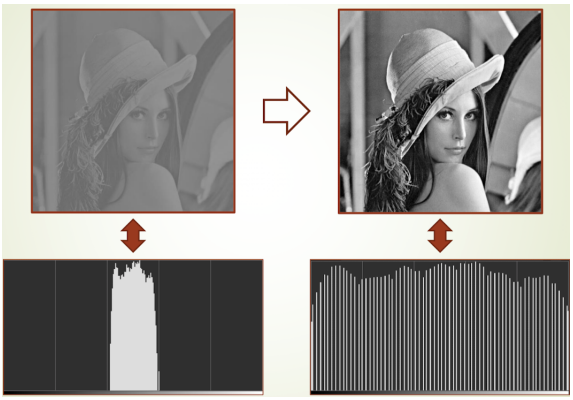
The goal is to increase the contrast of the image within a given range of grey levels. Since we want to keep the total number of levels unchanged, this implies shrinking the image dynamics outside the selected range. We can use a piecewise-linear input-output relationship.

$$i_{out} = \begin{cases} a i_{in} & \text{for } 0 \le i_{in} < a \\ \beta(i_{in} - a) + v_a & \text{for } a \le i_{in} < b \\ \gamma(i_{in} - b) + v_b & \text{for } b \le i_{in} \le 1 \end{cases}$$

- $\alpha, \gamma < 1 \rightarrow$ compression
- $\beta > 1 \rightarrow$ expansion



`Example:`



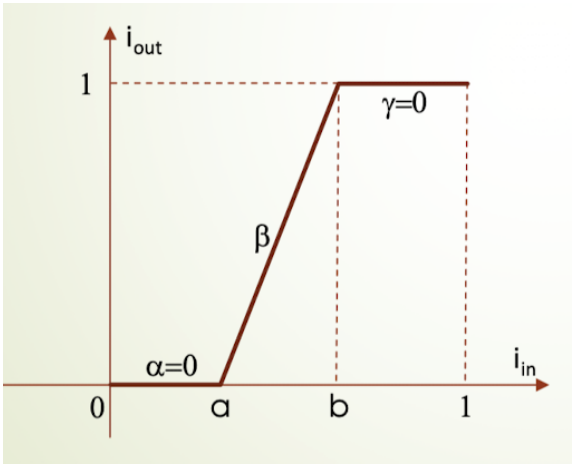
Histogram clipping

In the particular case that $\alpha = \gamma = 0$ we have a clipping effect:

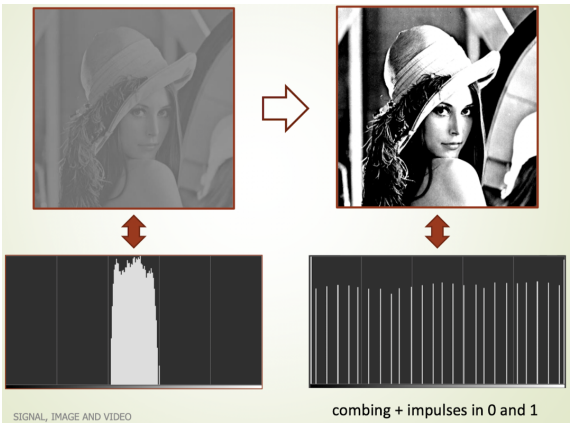
- All pixels with intensity $i_{in} < a$ become black.
- All pixels with intensity $i_{in} > b$ become white.
- The full dynamic range is available for intensities between 'a' and 'b'.

$$i_{out} = \begin{cases} 0 & \text{for } 0 \leq i_{in} < a \\ \beta(i_{in} - a) & \text{for } a \leq i_{in} < b \\ 1 & \text{for } b \leq i_{in} \leq 1 \end{cases}$$

$-\ \alpha, \gamma = 0 \rightarrow$ clipping
$-\ \beta > 1 \rightarrow$ expansion



`Example`

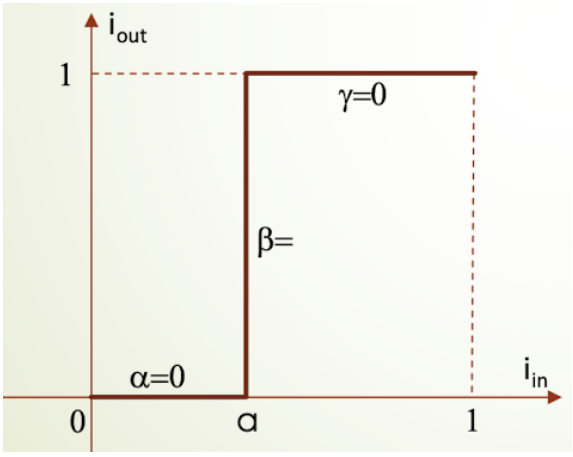


Histogram tresholding

If we now collapse a and b into a single point, the transformation curve becomes a step function.

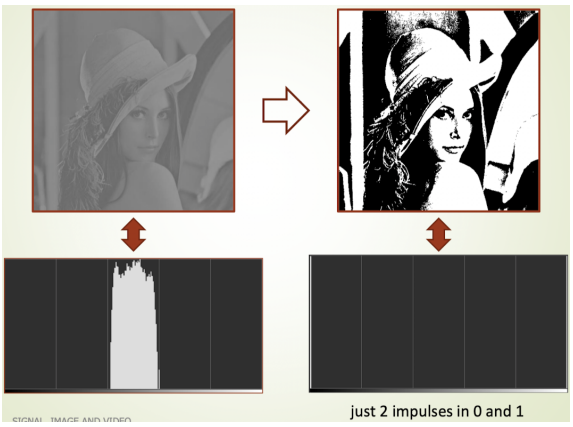
- All pixels with $i_{in<a}$ become black
- All pixels with $i_{in>a}$ become white

$$i_{out} = \begin{cases} 0 & \text{for } 0 \leq i_{in} < a \\ 1 & \text{for } a \leq i_{in} \leq 1 \end{cases}$$



- $\alpha, \gamma = 0 \rightarrow$ clipping
- $\beta = 1 \rightarrow$ transition

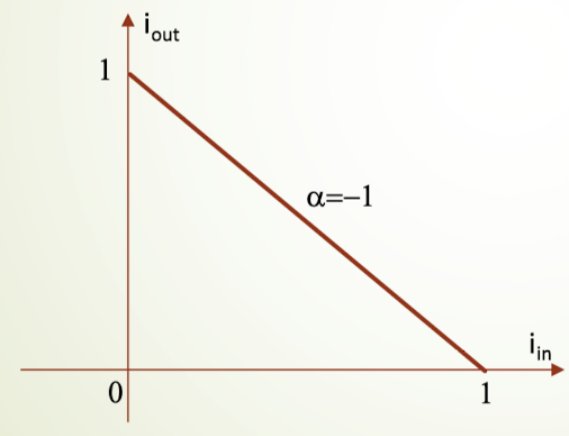
`Example`



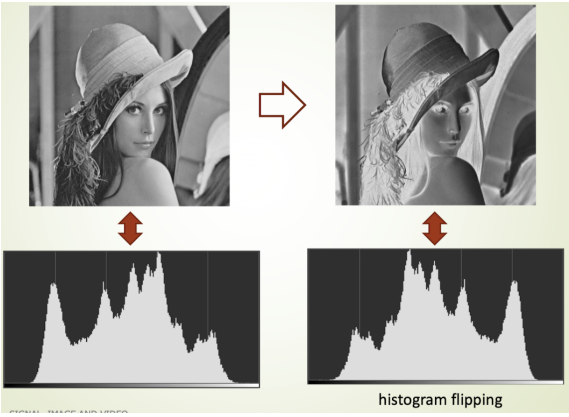
Histogram inversion

To obtain a photographic negative from the positive image, we need to reverse all pixel intensities. The transformation curve will be a linear function with negative slant.

$i_{out} = 1 - i_{in}$



`Example`

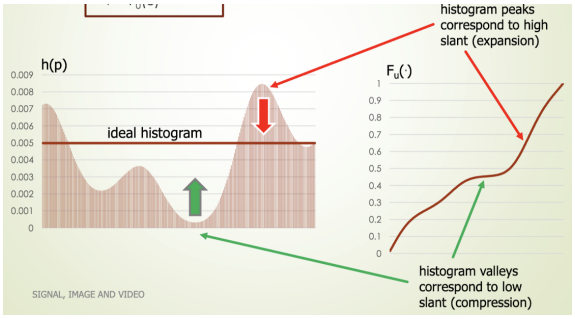


Histogram equalization

Histogram equalization is a bit more complex. The objective is to map the greylevels as to achieve an image with flat (equalized) histogram. The theoretical idea behind this operation comes from information theory: a random variable with uniform distribution in associated to maximum entropy, i.e., maximum information content. From a practical viewpoint, to achieve this goal we apply a known theorem of statistics.
Let u be a random variable with generic pdf $f_u(u)$.
Let $F_u(\alpha) = Pr\{u < \alpha\}$ be the statistical distribution of u.
Then, $v = F_u(u)$ is a uniformly distributed random variable.

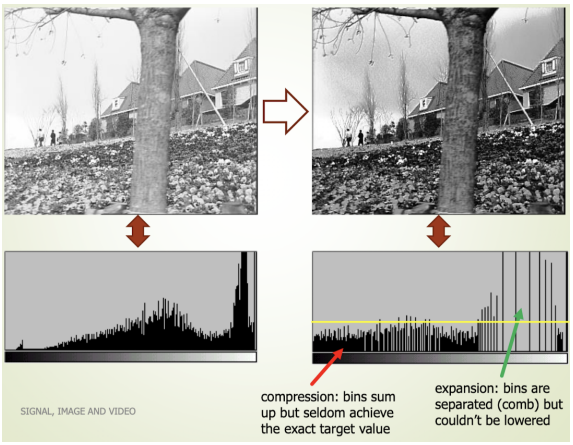
In the discrete domain we have:

$$P_U(U) \iff h(p)$$
$$F_U(\alpha) \iff C_h(\alpha)$$
$$v = F_U(U)$$



NB: $F_U(\cdot)$ takes the meaning of a transformation function (it hasn't the meaning of a statistical distribution anymore).

`Example`



4.2 Linear filters (FIR, IIR)

Filtering refers to a set of low-level algorithms applied to image with the purpose of, for instance: reducing various types of noise, restoring a damaged image, resampling/interpolating, enhancing/diminishing some particular characteristics. Filtering takes in input an image and provides in output another (filtered) image. We'll introduce 2 main types of filters: Linear and Non-linear.

Linear filters can be implemented by convolving the digital image with the discrete PSF of the filter. In fact, this is feasible only if the PSF is finite in space. Based on this, we distinguish two types of linear filters:

- FIR filters (FIR stands for "Finite Impulse Response"): are the most common, may be implemented through discrete convolution
- IIR filters (IIR stands for "Infinite Impulse Response"): may be implemented through auto-regression or in the frequency domain

Linear filters can only cut (attenuate) frequencies that are already present in the input signal. Depending on the frequency ranges we cut, we distinguish different types of filters:

- LPF (low-pass filters) – cut higher frequencies leaving unaltered lower ones. The result will be a smoother (blurred) image, with weaker contours and lower detail, but also cleaner (less noisy).

- HPF (high-pass filters) – cut lower frequencies leaving unaltered higher ones. The result will be a zero-mean image where contours and details come out, together with high-frequency noise.
- BPF (band-pass filters) – are the cascade of the above two: both lower and higher frequencies are cut, thus making contours and details emerge, but smoother and cleaner.
- BSF (band-stop or notch filters) – are the inverse of BPFs: only intermediate frequencies are cut, resulting in a sharper image.

4.2.1 Convolution kernels and relevant functions (LPF, HPF, BPF)

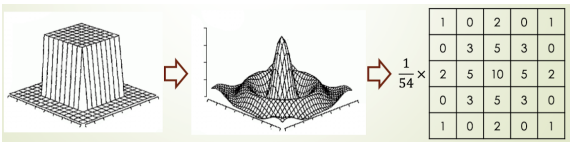
To achieve the desired behavior of the filter, we should define an appropriate convolution kernel (mask) We can define:

- Size/shape (rows x columns): determine the spatial extension of the filter, i.e., the pixels that are used to calculate the convolution for a single point (the reference point of the mask)
- Coefficients: represent the PSF of the filter and determine its behavior in the frequency domain

The procedure to define a kernel starts typically from the frequency domain:

1. Define the desired frequency response (frequencies you want to cut)
2. Come back to space with inverse FT and calculate the PSF
3. Sample the PSF to achieve a discrete representation
4. Apply windowing to achieve a finite spatial extension

`Example:`
For instance, suppose we want to design an LPF. The desired frequency response is represented on the left (upper cut frequency could be arbitrarily placed). We apply the inverse transform, obtaining a 2D PSF (in the middle). We sample it, obtaining a (theoretically) infinite number of coefficients. Due to PSF symmetry, we apply a square windowing, e.g., 5x5 (right). We normalize the coefficients to avoid shifting the mean value.



Moving average LPF

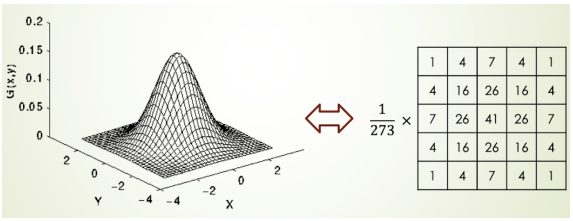
The simplest LPF is the moving average filter. Coefficients are all equal: it is like averaging the pixels in the window. The only thing one could change is the mask size. Increasing the size we obtain a stronger LP effect (lower cut). It is a separable filter and it is the strongest possible LP filter for a given kernel size.

Gaussian low-pass

In GLP, the coefficients are shaped as a Gaussian distribution

$$h_{GLP}(n,m) = \frac{1}{2\pi\sigma^2} e^{-\frac{n^2+m^2}{2\sigma^2}}$$

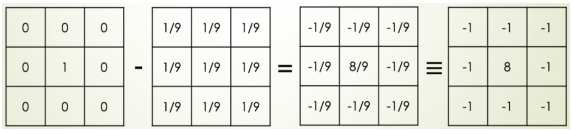
The larger is σ, the stronger the LP effect, below an axample for $\sigma = 1$.



N.B. Not with standing the size of the filter, the impact is lower, as central pixels are weighted more than peripheral ones.

High-pass filters

An HPF can be thought as the difference between an all-pass and a low-pass. The PSF of an all-pass filter is simply an impulse equal to the input one. The LPF should have the upper cut frequency where we need the lower cut frequency to appear. The resulting kernel will be the difference between a mask with a 1 in the center and the PSF of the LPF calculated as before.
`Example (based on a moving average LPF)`



N.B. There is no need to normalize as HPFs have zero sum by definition.

Sharpening filters

We could also use filters to magnify certain frequency ranges.
`Example: crispening filter`

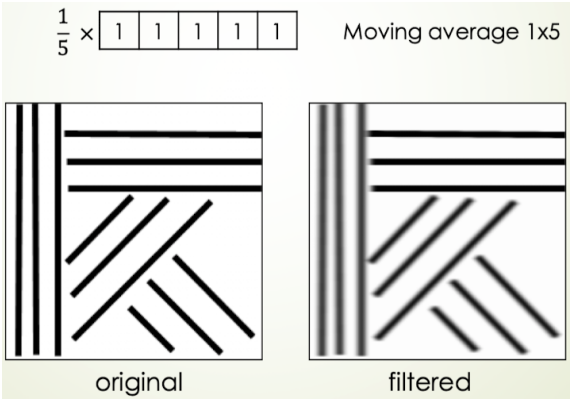
It is given by the following mask:

-1	-1	-1
-1	9	-1
-1	-1	-1

It looks like an HPF, but the sum is 1, therefore it doesn't cut lower frequencies, instead it enhances contrasts.

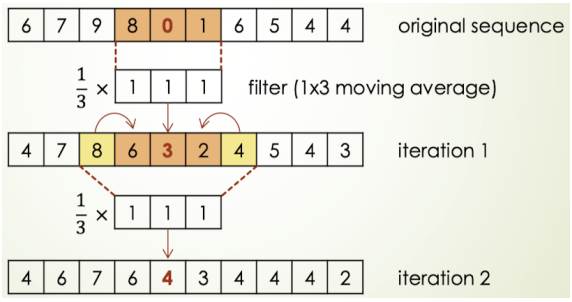
Directional filters

Modifying the shape and/or the coefficients of the kernel, we can design filters that act differently in different directions. For instance, if we want to smooth only vertical contours, we'll have to design a filter that works only in the horizontal direction.



Iterative filters

FIR filters can be iterated. This is due to the fact that FIR filters are approximations of the desired behavior (we don't really cut the frequencies as expected). Iterating the filter we virtually enlarge the spatial extension of the kernel.



N.B. The effect tends to vanish asymptotically across iterations.

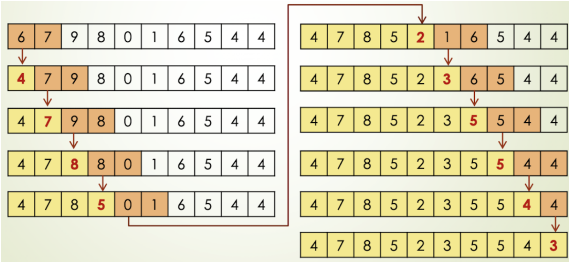
IIR filters

If we were able to use kernels with infinite extension (IIR), there wouldn't be need to iterate (indeed, it would be useless). Unfortunately, this would imply infinite computation (unfeasible with regular convolution). A possible solution is to introduce auto-regressive filtering. In auto-regressive filtering, previously computed sample values are used to filter successive samples. In this way, the filter virtually extends to infinite in a single iteration. It is a recursive operation.
N.B. It is anyway an approximation, it also depends on the scan direction.

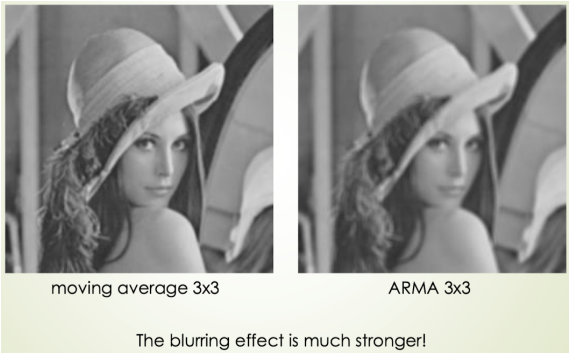
4.2.2 Auto-regressive filtering

Let see what happens if we apply the same kernel used in iterative filtering (moving average 1x3) in auto-regressive mode. This filter is called auto-regressive moving

average (ARMA). It can be applied with any filter size.



`Example ARMA filter`



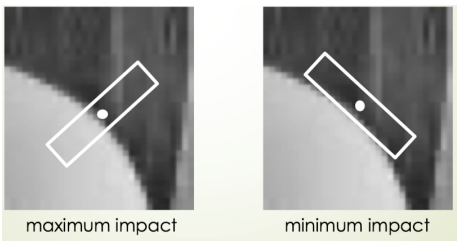
4.3 Non-linear filters

Non-linear filters cannot be described in terms of PSF. Either superposition or shift properties are not fulfilled. Quite complex to design in hardware, they turn out to be easily implemented in software (an advantage of digital processing).

4.3.1 Adaptive filters (edge-preserving)

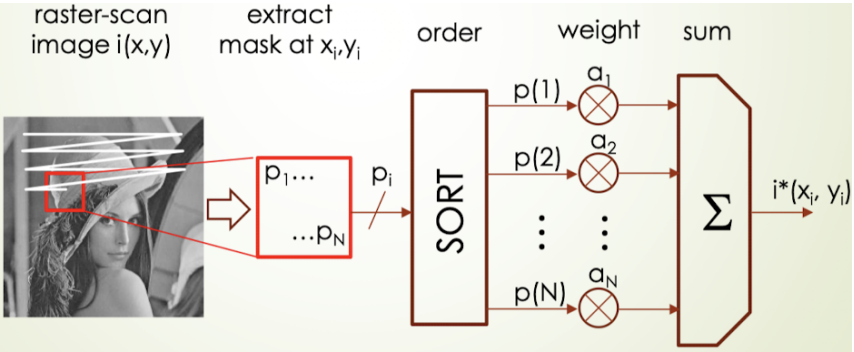
We have seen that low-pass filters are typically adopted to reduce noise in images. Nevertheless, a drawback of LPFs is that they also produce blurring and smoothing of contours. A possible idea to reduce this problem is to filter in a selective way, adapting the filter to local structures. The filter becomes space-variant, then, it couldn't be implemented with a simple convolution.

`Example`
A simple example is a linear low-pass whose angle of application changes according to the possible presence of contours. At each point, the filter scans all possible angles and selects the one that minimizes some contrast measure (e.g., local variance). LP filtering is applied in that direction, thus reducing noise without affecting the image structures.



4.3.2 Rank-order filters (ROF)

Rank-order filters are a large class of filters that are characterized by the introduction of a sorting operation within the filter.
The general scheme of a ROF is shown below: Note that the processing is very



similar to a FIR, the main difference being the introduction of the ordering block. The ordering causes pixels to be weighted according to their value instead of their position. This operation is of course highly non-linear. As for FIR filters, the result depends on: size/shape of the mask and values of the coefficients.
We'll introduce 3 simple examples: Min, Max, and Median (assume increasing order):

- Min: $a_i = 0 \quad \forall i \neq 0; \quad a_0 = 1$
- Max: $a_i = 0 \quad \forall i \neq N; \quad a_N = 1$
- Median: $a_i = 0 \quad \forall i \neq \lfloor \frac{N}{2} \rfloor + 1; \quad a_{\lfloor \frac{N}{2} \rfloor + 1} = 1$

Min and Max filters

Min/Max filters eliminate bright/dark outliers, respectively. Small details that are above/below the most frequent values in the mask are substituted by the lower/higher level present in the mask. No new grey levels are introduced by the filter, however the image tends to become darker/brighter. The outlier is eliminated if it is smaller than the mask size. It should not affect the first/last element of the ordered sequence.

Median filter

Despite its simplicity, median filter is very popular. Since it selects the median value (central value of the ordered array) it may eliminate both high and low outliers. In this case, the maximum size of the outlier is half the mask size. More precisely, it could be demonstrated that a median filter of size N leaves unchanged constant sequences of length equal to or larger than $\frac{(N+1)}{2}$. Again, it doesn't introduce new grey levels. Differently from min/max filters, median preserves the mean value.
Effect:
It acts approximately as an edge-preserving low-pass, but can be more precisely considered a regularization filter. It works very well with impulsive noise (speckle).

ROF complexity

ROF filters have higher complexity as compared to FIR. They perform exactly the same operations (weighted sum over a moving window) + sorting. Sorting algorithms have usually quadratic complexity but can be reduced to $O(N \cdot logN)$ for reasonably long sequences (quick sort). Alternatively, it is possible to keep a local histogram that is updated at each shift of the mask.

Separability

ROF filters are not separable. Nevertheless, median is sometimes applied by rows and columns to save computation $\rightarrow$ the result is similar but not equal.

4.4 Operations with multiple images

Sometimes it is useful to combine images in some way. The combination may involve different operators:

- Mathematical operators: e.g., sum, difference, product
- Logical operators: e.g., AND, OR, XOR
- Masking operators: e.g., mask-based composition, chroma-key

Algebraic operators To apply algebraic operators to 2+ images it is first necessary to align them and resize. Once we have the desired pixel-to-pixel correspondence, we simply apply the desired operation.
`Example:` difference operators can be used to detect variations among images.

Logical operators Exactly the same way we can apply logical (binary) operators. Typically it is more useful for binary ($1bpp$) images or working on bitplanes.
`Example:` XOR operator to detect local difference.

Mask-based operators We can also combine images using masks. A mask is a third image (typically binary) that is used to decide where the pixels should originate from (and possibly how) at each point. It is possible to do stitching, composition, etc. `Example:` chroma-key. A particular color is used to determine the mask, every pixel that gets that particular color is replaced with another image (e.g. green screen).

Resume

- Low-level image processing concerns the filters that are used to improve image characteristics, removing acquisition artifacts
- With pointwise operators, we can partially correct geometrical and color artifacts, as well as bad camera regulations
- Linear image filters with appropriate convolution kernels are used to remove noise and modify the frequency content of the image
- Non-linear filters can also be used to remove specific types of noise or to make more evident some specific image features
- Non-linear filters are much easier to implement in the digital domain
- We can combine images together using algebraic or Boolean operators, in order to compose them or to highlight some joint characteristics

Chapter 5

Middle-level image processing

The purpose of intermediate-level operations is to convert the image into more meaningful (semantically richer) representations. For instance, starting from a pixel-level representation we want to describe the image in terms of contained "objects". Each object could be described by its shape, color, texture, etc.
At this level, we are still not able to define what an object is, but just its appearance. Middle-level processing produces descriptions that are easier to use for high-level systems. Based on descriptors, high-level could go further in semantic interpretation, trying to make sense of the image content.

Image descriptions We'll introduce four main types of descriptors:

- Contours: boundaries of objects
- Regions: shapes associated to objects
- Textures: arrangement of colors on the object surface
- Structures: groups of objects with spatial relationships

The four descriptors are somewhat complementary

- Contours enclose regions
- Regions contain textures
- Groups of regions/contours create structures

The techniques used to extract such descriptors from images, however, are quite different.

Process of extracting image descriptors We distinguish 3 main phases:

- Detection: it is the action of revealing the desired descriptor from the image (e.g., detecting edge points to produce an edge-map).
- Representation: it is the action of associating an appropriate description to each detected primitive (e.g., representing a chain of edge point as a 2D curve, or contour).

- Feature extraction: it is the action of associating a set of quantitative parameters to each detected primitive (e.g., representing a contour in terms of length, shape, closeness, curvature, etc.).

Each phase can be implemented in different ways (algorithms), with relevant pros and cons.

5.1 Contours

The first descriptors we introduce are image contours. The expressivity of contours in representing the nature of objects is rather evident.

We distinguish 3 phases of the contour extraction process:

- Edge detection: we convert an image (grayscale or color) into a binary map of edges.
- Contour extraction: we scan the edge points that have been detected at the previous step to create connected chains of contours (curves).
- Feature extraction: we associate a set of parameters to each contour chain.

5.1.1 Edge detection

Edges are characterized by steep luminance/color variations that are present in an image in the presence of object borders. Since edges are associated to steep variations of the image function, they can be detected by analyzing spatial high-frequencies. Typical approaches are therefore based on the use of gradient filters.

Since we are working in 2D, we can calculate the directional derivative $D_{\vec{r}}\, i(x,y)$ along any unit vector $\vec{r}$ of direction θ.

$$D_{\vec{r}}\, i(x,y) = \nabla i(x,y) \cdot \vec{r} = \frac{\partial i(x,y)}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial i(x,y)}{\partial y}\frac{\partial y}{\partial r} = i_x \cos\theta + i_y \sin\theta$$

The derivative will be maximum when θ is perpendicular to the contour. In such point therefore we will have:

$$\frac{\partial}{\partial\theta} D_{\vec{r}}(i(x,y)) = -i_x \sin\theta + i_y \cos\theta = 0$$

Calling θ_n the direction corresponding to the maximum, we get:

$$\begin{cases} \theta_n = \arctan\left(\frac{i_y}{i_x}\right) \\ |\nabla i(x,y)| = \sqrt{i_x^2 + i_y^2} \end{cases}$$

where θ_n is the contour direction and $|\nabla i(x,y)|$ is the contour intensity.

Quadratic operators Therefore, to detect edge points we could simply:

1. Calculate the partial derivatives along two orthogonal directions (typically x, y)
2. Calculate the module of the resulting vector (ix, iy)
3. Threshold the result to select local maxima

These methods are called gradient operators or quadratic operators. To implement the partial derivatives we use directional FIR filters.
`Example`

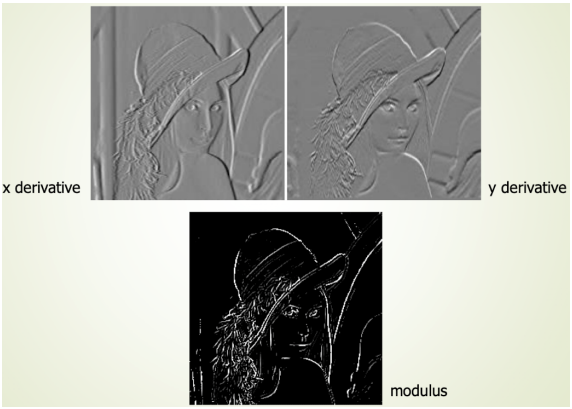
$$\text{Prewitt filters} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

$$\text{Sobel filters} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Denoising It should be noted that the gradient tends to emphasize also the noise present in the image, detection of false edge points. The Prewitt and Sobel operators try to reduce this effect by applying an LPF 'hidden' in the filter. For instance, Sobel operator is a separable kernel.

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

It performs a HPF along y direction and a simple Gaussian LPF along the orthogonal (x) direction. The LPF does not affect the detector performance, while removing some noise.
`Example Sobel edge detector`



Gradient operators

PROS:

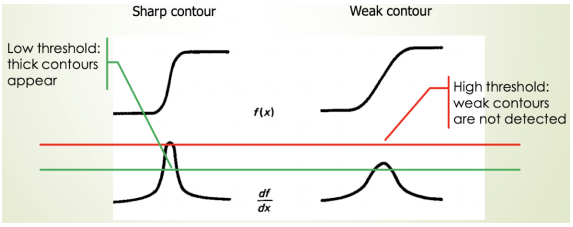
- Easy to implement
- Limited complexity (two $3x3$ FIR filters)
- Relatively accurate if image is not too noisy

CONS:

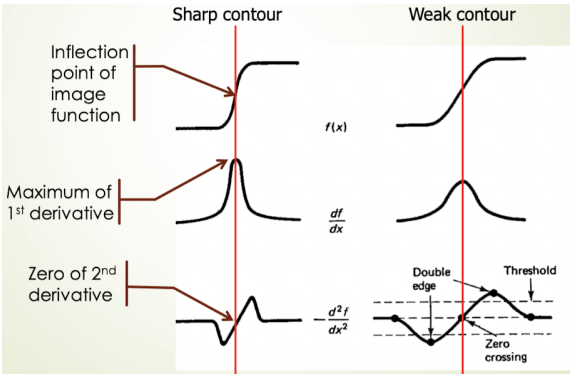
- Hardly detect weak contours
- Disconnected and thick contours (depends on thresholding)
- Imprecise localization

5.1.2 Derivatives

Problem with first derivative The problem with gradient methods is that, when the contrast variation is not steep enough, the peak generated by the derivative is low and spread. When we apply thresholding, we risk to miss it or, vice versa, to get too many points across the contour.



The problem can be solved by using 2nd derivative (Laplacian). In this case, however, we have to detect the zero crossings (which are unique) instead of local maxima.



5.1.3 LoG operator

Unfortunately, the second derivative is even more sensitive to noise. This means that we'll get a large number of zero crossings. To avoid this problem, the solution is to

perform an intensive denoising before applying the Laplacian operator. Denoising can be achieved by a strong LPF. Notice that this does not compromise the detection, as zero crossings survive the low-pass filtering. The cascade of Gaussian lowpass and Laplacian takes the name of **LoG filter** (Laplacian of Gaussian) and was proposed by Marr and Hildreth.

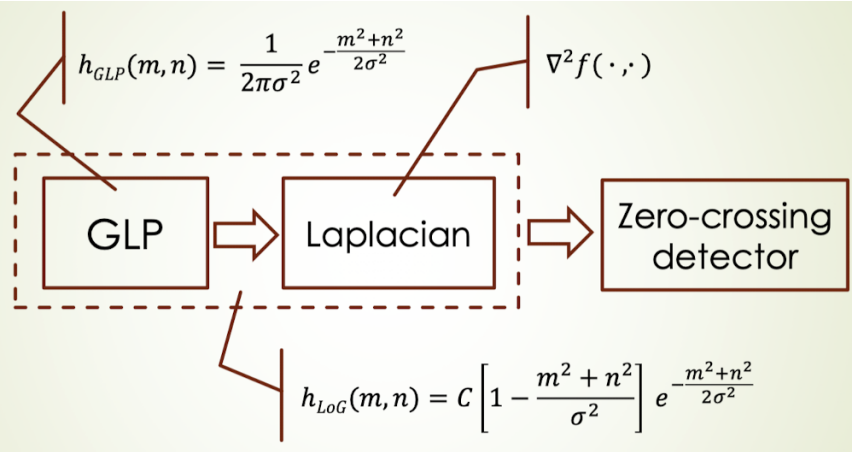


Figure 5.1: LoG-based edge detection

The cascade of GLP and Laplacian (both linear operators, can be fused into a single Laplacian of Gaussian filter.

The LoG filter, also called Mexican-hat function due to its shape, typically requires very high kernel size ($31x31$ or more). This makes the computational load of Marr-Hildreth filter quite high as compared to Sobel or other gradient-based operators. As to the binarization, only significant zeros are detected, based on the thresholded 1^{st} derivative.

Laplacian operators

PROS:

- Good localization
- Thin contours
- Good continuity of lines
- Quite resistant to noise

CONS:

- Computationally expensive
- Tends to round every shape (due to the intensive GLP filtering)
- Tends to create small artifacts

5.1.4 Canny edge detector

John Canny proposed to address the problem of optimal edge detection in a formal way. An "optimal" detector should provide:

- low misdetection (catch as many real edges as possible)
- good localization (identify the center of the edge)
- low rate of false edges (edges should be marked only once and noise should not create false edges)

To this end Canny proposed a method based on the optimization of a functional, defined as the sum of 4 exponential terms. In actual implementations, the above optimization is approximated by a more traditional sequence of filters.
The detector follows 5 steps:

1. Apply a Gaussian filter to remove noise (typically, a $5x5$ FIR kernel).
2. Calculate the gradients along x and y (similar to Sobel), and associate to each point an intensity and an angle (quantized to 4 directions).
3. Perform non-maximum suppression, a kind of thinning where the edge strength is compared to the neighbors along the edge direction and only the point with larger gradient is selected.
4. Apply double thresholding: pixels are marked as strong edges, weak edges or suppressed according to a lower and an upper threshold.
5. Edge tracking by hysteresis: weak edges are preserved only if they are 8-connected to at least a strong one.

PROS:

- Very good localization
- Thin contours
- High-continuity of lines, also for weak contours
- Highly resistant to noise

CONS:

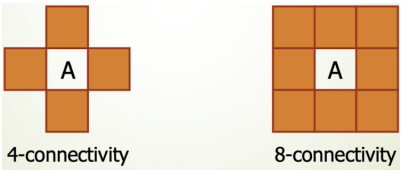
- Several parameters/thresholds
- Relatively complex

5.1.5 Contour representation & description

Edge detectors produce in output binary images. This is not yet a true contour representation. We still need to process the edge image and extract the contour chains, in the form of 2D curves. To this purpose, we need appropriate algorithms to follow sequences of connected contour points. The most known algorithm is the so-called **Freeman chain code**.

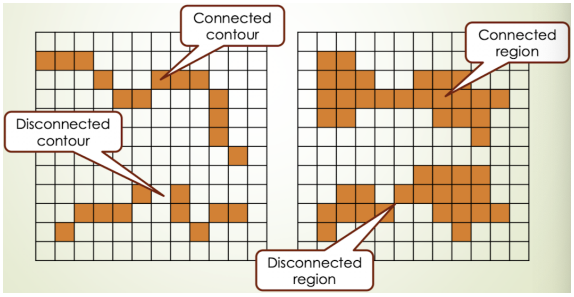
Connectivity

Let's first define a standard rule on the concept of connectivity Given a pixel A, we define:



- 4-connected to A the neighbors in horizontal and vertical direction.
- 8-connected to A the neighbors in horizontal, vertical and diagonal direction

We apply the 8-connectivity rule to contours and the 4-connectivity rule to regions. In this way we avoid ambiguities in defining the border to a region.

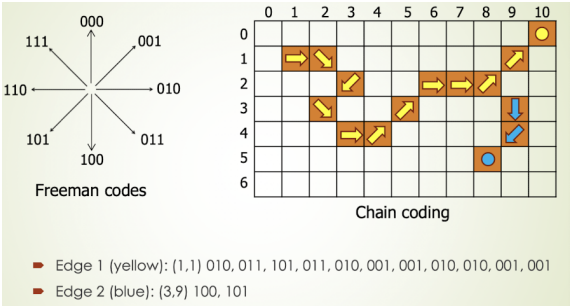


5.1.6 Freeman chain code

The Freeman chain code is an algorithm that allows extracting a contour for an 8-connected chain of edge points.
The idea is very simple:

- From an edge point I can move to the following one (if any) along 8 possible directions (the 8-connected pixels)
- Each direction can be associated to a code (8 directions $\rightarrow$ 3 bits).
- Starting from an edge point I can represent a chain of connected edge points as a sequence of direction codes.
- The representation will require $3xN$ bits, where N is the length of the contour (in pixels).

`Example`



N.B. In case of branches, the direction corresponding to the lowest code is followed first.

5.1.7 Thinning

Thickness is a potential problem when following a contour. There are ambiguities when following a path across the contour chain. Contours are fragmented, replicated, and/or follow convoluted patterns.

If we use an edge detector that generates thick edges, it is necessary to thin the contours before applying chain code. There are numerous thinning algorithms that have been proposed.

5.1.8 Other representations

As an alternative to chain code, there are representations that provide approximate descriptions of the curve. Typical examples are approximations based on piecewise linear functions and piecewise polynomial functions. In these cases, the representation is more compact (only a subset of points has to be stored) BUT the reconstructed curve does not perfectly match the original one. Linear and spline interpolations provide vector representations of the contours, largely used in computer graphics and CAD (see, e.g., splines).

Piecewise linear interpolation

Piecewise linear interpolation can be achieved trough a simple recursive algorithm:

1. Given a curve from point A to point B, approximate it with a straight segment AB.
2. Call C the point on AB at maximum distance from the original curve.
3. If the distance from C to AB exceeds a given threshold TH, replace the segment AB with the polygonal ACB.
4. Recursively apply steps 2 and 3 on AC and CB.

Spline interpolation

Splines (piecewise polynomial interpolations) are a bit more complex, but provide more graceful approximations. The idea is to approximate each segment with a curve, defined by a parametric polynomial of a given degree (typically cubic). The major problem is to find the best fitting curves: it is no more sufficient to determine the cutting points, but we need also the curve parameters. A possibility is to control the local curvature by imposing the $N - th$ order derivatives at the nodes.

5.1.9 Contour features

Once we have extracted the contours, we have converted the input image into a set of curves. A further simplification is to associate to each curve a set of features that synthesize the characteristics of the curve itself. The objective of this last processing phase is to define a minimal set of parameters that provide a meaningful description of the image content, with reference to a target application.
Each feature can be described by numerical or Boolean values.

5.2 Regions

In image regions the expressivity of the descriptor in representing the nature of objects is rather evident.
As for contours, we distinguish 3 phases:

- Region detection or segmentation: we convert an image (grayscale or color) into a map of homogeneous regions
- Region representation: we describe each region in the segmented image as a 2D shape
- Feature extraction: we associate a set of parameters to each region

5.2.1 Segmentation

The segmentation should satisfy a set of rules:

1. Intersection: Segmented regions must not intersect each other.

$$R_i \cap R_j = \emptyset \quad \forall i \neq j$$

2. Union: The union of segmented regions must cover the whole image.

$$\bigcup \{R_i\} = I$$

3. Connection: Region points must be 4-connected.

$$\forall x_i, x_j \in R \Rightarrow \exists \text{ path } X(n) \text{ from } x_i \text{ to } x_j \text{ such that } X(n) \in R \quad \forall n$$

4. Homogeneity: Region pixels must obey some given homogeneity rules. Given a set of homogeneity parameters $H_k(R_i)$, $k = 1, \ldots, K$, and the relevant thresholds T_k,

$$H_k(R_i) < T_k \quad \forall R_i, k$$

It is difficult to define a quality of a segmentation. It is an ill-posed problem, there is no unique solution. It is even difficult to define a benchmark.
Classical segmentation problems include:

- Over-segmentation: a region which is clearly homogeneous is split into two or more segments.

- Under-segmentation: a single region contains multiple distinct objects or object and background.

Typically, the result of a segmentation shows both problems.

There are two basic approaches:

- Top-Down: the segmentation algorithm starts from large unstructured regions (in the limit, the whole image), and proceeds by progressive subdivisions.
- Bottom-up: the segmentation starts from small elementary regions (in the limit, single pixels), and proceeds by progressive aggregations.

Top-down approaches are typically faster but less accurate than bottom-up. Sometimes, both approaches are jointly applied in the same technique

Segmentation based on clustering

A simple method to segment an image is to perform a clustering according to some homogeneity criterion. For instance, we can cluster together pixels that show similar color. Clustering is an unsupervised classification approach and can be achieved through various algorithms.

K-Means One of the most popular clustering algorithms is the K-means. We have to define a-priori the number of clusters K, then the algorithm automatically generates the K clusters and associates every image pixel to one of the clusters. This process associates each pixel to a cluster, but does not produce a segmentation (the connection rule is not satisfied). A labeling phase is required to obtain a real segmentation.

Split & Merge segmentation It is a more sophisticated technique based on a cascade of two methods:

- A top-down process called SPLIT: the image is recursively decomposed into blocks of decreasing dimension, until each block satisfies the requested homogeneity.
- A bottom-up process called MERGE: pairs of blocks generated by the split are progressively fused together if their merge satisfies the requested homogeneity.

The split consists in a recursive dichotomic subdivision of the image in blocks The algorithm is very simple:

1. Define a homogeneity rule $H(R) \rightarrow [T, F]$
2. Initialize R_{curr} to contain the whole image
3. If $H(R_{curr}) = TRUE$ then STOP
4. else
    - split R_{curr} in 4 equal regions $R_i, i = 1, ..., 4$ (dichotomic split in x, y dimensions)

- for i in $1, ..., 4$ go to step 3 with $R_{curr} = R_i$

The recursion creates a tree-structured subdivision of the image. It generates a particular tree with 4 sons per node, called **quadtree**. The quadtree is no more suitable as a data structure, as it does not provide the proximity information.

Merge: the RAG

A graph structure is used: the Region Adjacency Graph (RAG). The RAG is composed of nodes and links:

- Nodes represent the segmented regions R_i
- Links are pointers to pairs of nodes associated to adjacent regions
- To each link L_k is associated a cost C_k, corresponding to the loss of homogeneity caused by merging the two pointed regions

The merge algorithm is iterative:

1. Select the link L_{min} at minimum cost C_{min}
2. If $H(R_i \cup R_j) = FALSE$ then STOP $\quad$ (R_i, R_j = regions pointed by L_{min})
3. else
    - merge regions R_i, R_j to form region R_{ij}
    - update the links to R_i and R_j to point R_{ij}
    - recalculate the costs of the updated links
    - go to 1

Region growing segmentation It is the typical bottom-up segmentation: starting from a set of atomic elements (single pixels or small, highly homogeneous agglomerates), the segmentation proceeds by progressively merging all the image pixels around them.

The algorithm is based on two important steps: the selection of the starting points (seeds) and the growing process.

The selection of seeds is important to achieve a good result. If there are relevant objects with no seeds, there is a risk to miss the associated regions in the segmented image (under-segmentation). If there are many seeds on a single relevant object, there is a risk to split it into several regions (over-segmentation). If the seed is close to the object boundary, there is a risk to initialize the growing from a value that does not characterize either the object or the background.

Typical strategies to select seeds are:

- Calculate the gradient of the image and select pixels in the centroids of minimum gradient areas
- Pre-segment the image with a simple clustering (using a low threshold), and use the clusters as atomic regions

Growing process

The growing process consists in iteratively checking 4-connected pixels to a region

and merge them if they are homogeneous. At some point, regions may get in contact each-other. Merging regions is more impactful, then additional criteria may be used to control that process, for instance:

- Criterion 1: $\frac{\omega}{p_m} > \theta_1$
  - ω = number of boundary pixels with contrast below a given threshold
  - p_m = perimeter length of smallest region
- Criterion 2: $\frac{\omega}{l} > \theta_2$
  - l = length of boundary between regions

Criterion 1 $\rightarrow$ merge regions that have low contrast along their boundary, provided that at least one is very small
Criterion 2 $\rightarrow$ merge regions that have low contrast along their boundary, provided that their boundary is short enough

- PROS
  - Good quality of the segmentation
  - Possibility of defining custom rules according to the problem
  - Rather easy to implement (at least for baseline algorithm)
- CONS
  - High-computation
  - Many empirical rules, many parameters/thresholds (difficult to use)
  - Some under- and over-segmentation problems (depending on seeding and merging rules)

Watershed segmentation

The idea of watershed segmentation comes from geology: a watershed is a separation line that divides catchment basins. In images, the watershed is built starting from the image gradient. Higher gradient values (typically corresponding to edges) become barriers between regions. Once the watersheds is defined, the internal basins are virtually flooded starting from a set of local minima (markers). If there is more than one marker per basin, over-segmentation occurs and a successive merge phase is required.

Around these general principles, many variants have been proposed with different procedures for selecting markers and performing the flooding.

Gradient The first step consists in defining the basins: typically, this is achieved by applying a high-pass filter.

Flooding After that, markers are defined and flooding may start. Typically, the first phase procedures over-segmentation. Merging with appropriate rules may solve the problem.

- PROS
  - Good quality of the segmentation
  - Computationally effective
  - Rather easy to implement (at least for baseline algorithm)
- CONS
  - Some under- and over-segmentation problems (depending on marker definition and strength of contours)
  - Not suitable for textured regions (appropriate modifications have been proposed)
  - Sensitive to noise (pre-processing may be applied to limit the problem)

5.2.2 Region Representation & Description

Segmentation algorithms produce in output labeled images. This is not yet a true region representation. We still need to process the segmented image and extract each region as a 2D shape. The region should be then represented in a convenient format. Finally, we could associate a set of parameters to each region to effectively describe its geometrical characteristics.

Region representation: Bounding Box

As a first step, each region R_i can be extracted from the segmented image by cropping the minimum-size rectangular area that contains it. Then, the cropped area can be binarized, setting to 0 the background pixels and to 1 the region pixels.

$$u(m,n) = \begin{cases} 1 & \text{if } (m,n) \in R_i \\ 0 & \text{otherwise} \end{cases}$$

NB. The cost of this representation can be reduced by using appropriate techniques for binary image encoding.

Region representation: Contours

As an alternative, a region can be represented by its contour. Region boundaries are extracted from the segmented image (or from every single region). It is not an edge detection, we simply have to follow the region boundary. The contour points obtained can be represented with any of the approaches previously defined for edge images. Note that each region has a closed contour by definition. A region may have "holes", in this case we'll have internal contours that will be interpreted as areas to be subtracted.

`Example`

- Boundary length P_i: number of pixels at the boundary of region R_i (including possible holes)
- Region area A_i: number of pixels included in the region R_i
- Number of holes NH_i: holes count
- Symmetries (vertical S_v, horizontal S_h): yes/no
- Centroid $B_i = (B_x, B_y)$: center of mass of region R_i

$$B_x = \frac{1}{A}\sum_{i \in R_i} x_i; \quad B_y = \frac{1}{A}\sum_{i \in R_i} y_i$$

- Shape factor α_i:

$$\alpha_i = \frac{R_{min}}{R_{max}}; \quad R_{min}, R_{max} = \text{min, max distance from centroid}$$

Projections A 2D shape R_i can be associated to a set of 1D functions $g_{Ri}(s, \theta)$, representing the projections of R_i over an axis oriented along θ. The projection is simply an histogram: each entry$g_{Ri}(s, \theta)$ is the count of the pixels of R_i that cross the perpendicular to the axis in s_0.

Moments Image moments provide a set of significant and easily computed parameters to The general equation that represents a moment of order (p,q) is:

$$m_{pq} = \sum_x \sum_y x^p y^q f(x, y); \quad p, q = 0, 1, 2, ... \text{ where } f(x, y) = \begin{cases} 1 & \text{if } (x, y) \in R_i \\ 0 & \text{otherwise} \end{cases}$$

Examples of parameters deriving from low-order moments are:

- $m_{00} \rightarrow$ region area
- $m_{10}/m_{00} \rightarrow B_x$ (coordinate x of region centroid)
- $m_{01}/m_{00} \rightarrow B_y$ (coordinate y of region centroid)

$m_{p,q}$ depends on the position R_i on the image (translation) as well as its rotation and scale. This may be a drawback: imagine you have to recognize an object matching a given template on an image. Moments won't match if the object appears at a different position or if it is rotated or scaled.
In order to remove this problem, it is possible to define moments that are invariant to a given geometrical transformation

- Moments invariant to translation (central moments):

$$H_{pq} = \sum_x \sum_y (x - B_x)^p (y - B_y)^q f(x, y)$$

- Moments invariant to translation and scaling:

$$N_{pq} = \frac{H_{pq}}{(H_{00})^y} \quad \text{with} \quad y = 1 + (p+q)/2$$

- Moments invariant to rotation are more complex. M.K. Hu provided the formulas for low-order moments invariant to translation, scale and rotation. The first three are the following:

$$I_1 = \eta_{20} + \eta_{02}; \quad I_2 = (\eta_{20} + \eta_{02})^2 + 4\eta_{11}^2; \quad I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$

5.3 Textures

Textures complement contours and shapes in revealing (or hiding) the nature of an object.
A texture is the visual appearance of a surface. It can be thought as a spatial arrangement of colors (pattern) showing some kind of regularity. In the simplest case, it can be a uniform color. It is a complementary feature that, associated to a shape (contour or region), completes the description of an object.

5.3.1 Texture analysis

We typically don't "detect" textures, but we rather analyze them to:

- define parameters that can identify a given texture (e.g., to evaluate texture homogeneity in segmentation)
- detect texture irregularities (e.g., finding anomalies in textured surfaces for visual inspection purposes)
- classify textures (e.g., in object detection, when the texture is a characteristic feature of a given object)
- define models to synthesize similar textures (e.g., in computer graphics and virtual reality)

Many algorithms have been proposed for texture analysis. We can broadly classify them in two categories:

- Statistical models: come from the statistical theory and consider the texture as the realization of a stochastic process
- Structural models: are more deterministic and consider the texture as a composition/replication of structures according to some rules

For each of these categories there are various methods.

Statistical models

Textures are typically associated to the concept of repetition. Although they are not periodic signals in the strict sense, they show significant regularity. Artificial textures (e.g., a tissue) are more regular, while natural textures (e.g., wood or stones) tend to be more stochastic. A texture can be thought as a spatial replication of a 2D basic element called texel: both the texel and the way it is replicated are ruled by some statistical properties. Statistical methods aim at revealing such properties, through the use of various measures, such as:

- Autocorrelation function and associated parameters
- Transformed-domain parameters
- Multi-dimensional histograms and associated parameters

A texel is the minimum-size patch that contains the texture information (similar to a period of a periodic signal). It is not easy to define or univocal, it may have irregular shape. Its shape and size give information about texture characteristics.

Autocorrelation function (ACF) ACF is a statistical measure that provides information on the memory of a process (self-similarity at increasing shift factors). It can be estimated as follows (under ergodicity assumption):

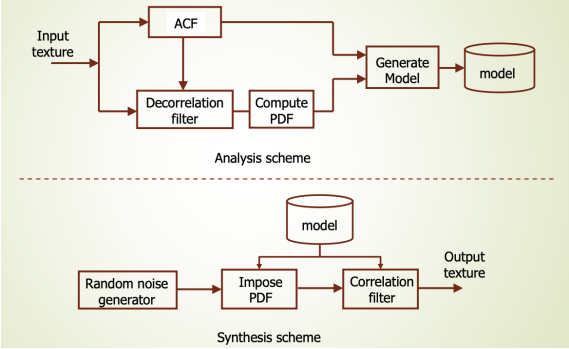
$$m_2(k,l) = \sum_m \sum_n u(m,n) \cdot u(m+k, n+l)$$

or, in its normalized form:

$$r(k,l) = \frac{m_2(k,l)}{m_2(0,0)}$$

- Directional memory: m_x, m_y such that $r(m_x, 0) = r(0, m_y) = 0.5$
- Mean values: $\overline{m} = \sum_m \sum_n m \cdot r(m,n)$
  $\overline{n} = \sum_m \sum_n n \cdot r(m,n)$
- Moments: $M_{pq} = \sum_m \sum_n (m-\overline{m})^p (n-\overline{n})^q r(m,n)$
  Particularly used are low-order moments:
  - $M_{20}, M_{02} \rightarrow$ profile spreads
  - $M_{11} \rightarrow$ cross-correlation
  - $M_{22} \rightarrow$ second-degree spreads

ACF and the relevant parameters are easy and fast to compute However, they are not unique. Different textures may have ACF parameters very close to each other. ACF can be used as a simple method for texture synthesis and analysis.



Transform-based methods Transforms (e.g., DFT but not only) provide information about the energy distribution of an image in the frequency domain. Looking at transform coefficients I can perceive if an image contains higher or lower frequencies, as well as their dominant directions. This is very much related to the textures present in the image, in fact, textures produce frequency peaks in specific zones of the transform. To extract a representative set of features it is therefore sufficient to subdivide the transformed domain in cells, and calculate the energy of each cell (sum of coefficients modulus). Only a small percentage of the cells will provide significant values These cells are characteristic of that texture.

MPEG-7 is an ISO standard introducing image descriptors to be used for content-based storage and retrieval. Textures are a significant features to describe an image. The standard suggests the use of Fourier-based descriptors. The concept is exactly the one we've just described.

Local Binary Patterns (LBP) A popular yet simple approach to describe textures is to use the so-called local binary patterns. LBPs analyze the local neighborhood of pixels in small texture cells. Several variants have been proposed, we define the base technique. The algorithm is made of 5 steps:

1. Divide the texture into cells (e.g. 16x16 pixels)
2. For each pixel in a cell, scan its 8 neighbors in clockwise direction
3. If the center pixel is greater than the neighbor write 0, otherwise write 1 (each pixel will be associated to an 8-bit word)
4. Compute the normalized histogram of the resulting words over the cell
5. Concatenate the histograms of all cells

The resulting descriptors are rather heavy (each cell generates a 256 bins histogram) and difficult to be interpreted. They are typically used as input to statistical classifiers (SVM,neural networks, etc.). Variants have been proposed to simplify the descriptors, especially to achieve smaller dimension histograms: dLBP uses only the 4-connected neighbors, thus forming 4-bit words and 16 bin histograms. Uniform LBP consider separately the 'uniform patterns', i.e., the patterns that show no more than 2 transitions $(1-0, 0-1)$.

2D histograms Since textures show pseudo-periodic behavior, they are likely to show repetitive patterns. These patterns can be identified by looking at the joint probability of pixels to take a given pair of values at a given distance and direction.. Such joint probability can be estimated as:

$$P(r, \theta, l_a, l_b) = \frac{\#\text{pairs } u_1, u_2 \text{ s.t. } d_\theta(u_1, u_2) = r \cup |u_1| = l_a \cup |u_2| = l_b}{\#\text{pairs } u_1, u_2 \text{ s.t. } d_\theta(u_1, u_2) = r}$$

The distribution of the above probability over all possible pairs l_a, l_b gives a 2D probability density function (PDF). $f_{r,\theta}(l_a, l_b)$. It is a discrete PDF.

In principle, we can calculate a matrix $f_{r,\theta}(l_a, l_b)$ for each distance r and each angle θ. Each matrix will have a dimension NxN, where N is the number of possible values associated to a pixel.
We can reduce these numbers by quantizing the parameters:

- Reducing the number of angles (e.g., every 45 degrees)
- Reducing the number of distances (limit the range, introduce a step)
- Reducing the levels (e.g., quantize to 3 bits, thus, 8x8 matrices)

In any case, the matrices will be very sparse. Only few significant values, according to the texture characteristics. Sometimes, entire matrices are irrelevant (equally distributed). We can therefore extract the few significant parameters (typically, some kind of moments of the PDFs) and use those parameters instead of the entire matrices.

Structural models

In structural models, the idea is to identify the texels and their spatial organization in a more deterministic way. In practice, we need to identify some primitives and a set of rules to replicate and position them in space. The texture primitives are 2D patterns in the form of pixel patches or geometrical structures. The rules can be strictly deterministic (a given periodicity) or pseudorandom (e.g., stochastic perturbations of a deterministic rule). Such descriptions are easily converted into some kind of formal language, e.g., a grammatic. An interesting property of structural models is that they can readily used for texture synthesis (just applying rules to primitives).

In structural models a stochastic behavior may be achieved by introducing random perturbations in the rules. For instance, the positioning may be within a given range, with random variations distributed in some predefined way.

Deterministic vs Stochastic rules When dealing with real patterns, this is not easy as it seems, as it may cause problems of alignment. Sophisticated placement rules may take into account additional constraints, such as, e.g., the minimization of boundary distortions.

Resume

- Middle-level processing consists in extracting descriptors from images, for further use in recognition and classification problems
- We've introduced 3 types of descriptors : contours (borders of objects), regions (uniform areas probably belonging to an object), textures (patterns describing the surface of a region)
- Each of these features needs to be detected, represented and associated to a set of features. These operations require appropriate algorithms.
- Algorithms are in continuous evolution (no ultimate solutions exist, also because in general the 'optimal target' is not defined)
- At the end of this process, we pass from a low-level representation of the image (pixel domain) to a middle-level representation (feature domain) $\rightarrow$ less data, more significant, at a higher abstraction level.

Chapter 6

Video Signal processing

6.1 Video signal

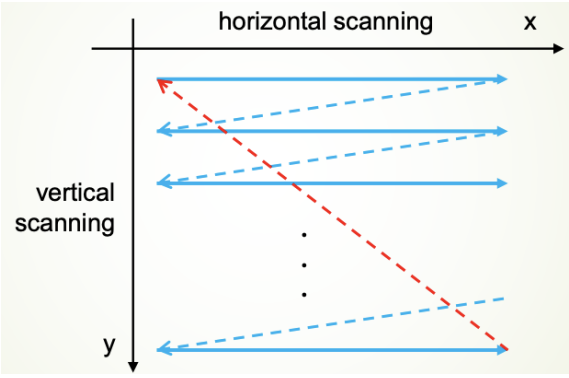
Video signals are multi-dimensional signals characterized by the presence of both space and time variables $I(x, y, t)$. We can define video signals as images varying in time (moving pictures, as opposed to still pictures). Moving pictures are typically sequences of still pictures captured at sufficiently short and regular time intervals. As such, every single image in the sequence (called **frame**) can be processed with the techniques introduced so far for static images. Nevertheless, video shows some peculiar characteristics to be exploited with dedicated processing techniques.

6.2 Analog vs. digital video

Typical examples of analog video signal are the old TV broadcasting standards (e.g., PAL or NTSC standards). Even in the analog case, however, we cannot have a continuum of images in time, we need to sample in the time direction. Each single image at time t_i is called video frame. We need to acquire enough frames per second (FPS) such that our eyes cannot perceive the discontinuity in movement (flickering).

6.2.1 Analog Video

Analog video is indeed a 1D signal obtained by scanning a timevarying 2D image along a given trajectory. Raster scanning: left to right, top to bottom. Once a



frame is completed, scanning restarts from the top-left position.
Note that a signal like this is continuous only along x, since both the number of lines and the frame scanning rate are discrete.

Interlacing

In order to limit the frame rate without causing perceptually annoying artifacts, interlacing can be applied. With interlacing, odd and even lines are transmitted alternatively. The frame rate is virtually doubled: 50 half-frames (fields) per sec require the same bandwidth of 25 full-frames per sec (FPS).
Clearly, content can change across successive half-frames. Combining 2 half-frames into a single one causes artifacts (combing).

`Example of analog video: PAL` PAL (Phase Alternate Line) system was used in Italy (and part of Europe) until the introduction of Digital Video Broadcasting.

Progressive video

Opposed to interlaced video, progressive video produces a full frame at each time sampling. It is typically used in digital transmission.
For instance, Full HD generates frames $1920x1080$ (0,92 Mp) at 25 or 50 FPS in progressive mode (also called $1080p$). In order to deal with huge data rate, compression is needed.
NB. It is important to distinguish between frame rate and refresh rate

- Frame rate: refers to the number of frames per second that are coded in the video signal (real frames).
- Refresh rate: refers to the number of times a picture is drawn by the monitor in a second.

Often the refresh rate is higher than the frame rate (es. $100Hz$ TVs), so that the same frame is drawn several times, to reduce flickering.

6.2.2 Digital video

Digital video can be obtained in two ways:

- A/D conversion of an analog video (frame grabbers). The signal is acquired with an analog camera and then digitized with an A/D converter (sampling + quantization).
- Direct digital acquisition (digital cameras). The signal is directly sampled during acquisition (using a matrix sensor), quantization is anyway needed

From a theoretical viewpoint, the sampling process should take into account both spatial and temporal frequencies. Spatial frequencies: already discussed in image sampling. Temporal frequencies: are related to the intensity of the variations in a small spatial area Δs of the frame along time.
N.B. There is a complex relationship between spatial and temporal frequencies (changes in time depend on local contents).

Temporal sampling The frame rate is not decided as a function of temporal frequencies, while to avoid motion artifacts (jagging, flickering). Below 12 FPS our eyes perceive separate images. Above such rate, we start having the motion illusion, but annoying effects may still appear (illumination is also relevant).

Digital video format Frame sampling, instead, follows the same rules of still images. The resulting digital video will be a discrete signal $I(x, y, ti)$, where:

- t_i: is the temporal index of the $i - th$ frame of the sequence, in a possibly infinite range
- x, y: are the spatial coordinates of frame pixels, in the range $1...N$, $1...M$ (NxM = frame resolution)
- $I(\cdot)$: is the value associated to the (x, y)-pixel of frame at time t_i, and it is typically a vector of color components (e.g., RGB or YUV)

Video processing Since video is a sequence of still pictures, in principle image processing techniques can be straightforwardly applied. Note however that this may be inefficient, because:

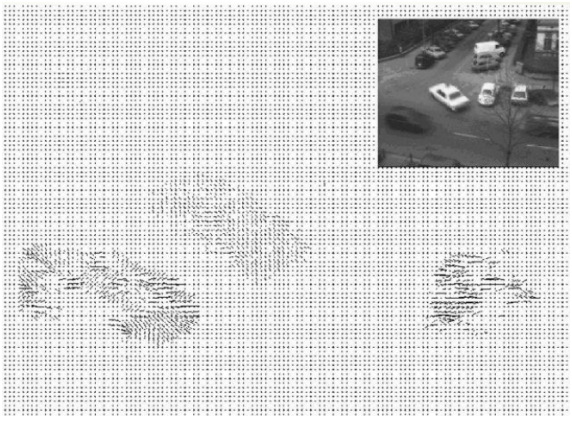
1. Video is a much richer source of information, then, I can use the temporal information to improve the processing. E.g.: if I want to remove noise and there is no motion, I could filter in the temporal direction, where the only changes are due to noise
2. Video has peculiar problems that need specific solutions. E.g.: motion blur or interlacing artifacts are not present in still pictures and require adequate techniques.

Video processing techniques largely use motion information to fully exploit the temporal information. Then, we'll start introducing motion analysis.

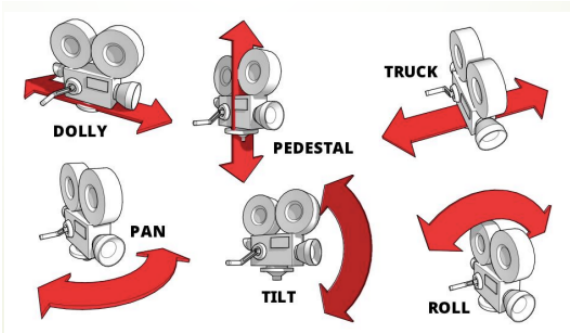
6.3 Motion analysis

If the frame rate is high enough, the motion is limited, then, the difference between a pair of adjacent frames is very small.
The objective of motion analysis is to understand how objects move along a video sequence. Ideally, we would like to estimate the speed and direction of each point in the scene.

Needle diagram We can represent the motion with a vector field. Graphically, we can associate to each point a segment oriented along the motion direction, whose length is proportional to the speed. The resulting chart is called needle diagram.



Real and apparent motion In fact, understanding the real motion is not so simple. Our observation represents just the relative motion between the scene and the observer (camera motion). We cannot know if the camera or the scene moves (or both): for instance, if the camera makes a 'pan' in a given direction, we perceive it as a translation of the scene in the opposite direction. Perceived movement also depends on relative position (e.g., distance) and motion direction of objects.



6.3.1 Optical flow

If no additional information is available, we can just detect the apparent motion, i.e., the relative motion between observer and scene. We define the optical flow as the distribution of apparent velocities of brightness patterns in an image (a vector field).
We need to introduce some working hypotheses:

- Uniform Illumination: illumination is spatially and temporally uniform (at least in a short temporal range)
- No occlusion: occlusion of one object by another, as well as uncovering of the background are neglected

Optical flow is an ill-posed problem Given the previous hypotheses, we can assume that each point $I(x, y, t)$ at time t has a corresponding point $I(x + \Delta x, y + \Delta y, t + \Delta t)$ at time $t + \Delta t$.

Assuming small movement $(\Delta x, \Delta y)$, we can approximate (Taylor):

$$I(x + \Delta x, y + \Delta y, t + \Delta t) \approx I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t$$

Simplifying and dividing each term by Δt we have:

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

Where (V_x, V_y) are the components of the optical flow in (x, y), and the other terms are the directional derivatives of the frame along x, y, t.
NB. We have 2 unknowns and 1 equation $\rightarrow$ infinite solutions.

Aperture problem Previous equations lead also to a problem known as aperture. The motion direction of a straight contour is ambiguous, since the component parallel to the line cannot be inferred from the visual input.

The derivative in the direction of the contour is null. This means that we can determine only the motion component in the direction orthogonal to the edge. We need to add constraints (e.g., jointly analyzing the motion of different parts of the object, introducing regularization constraints).

6.3.2 Block-based motion estimation

As we have seen, to solve the optical flow equation we need to add constraints. There are various methods to estimate the optical flow. Each method uses different (regularization) constraints. We introduce here one of the simplest approaches, called blockbased motion estimation (BBME). BBME introduces a further hypothesis: the translation model. We only observe the motion components laying on a plane that is parallel to the camera plane. Motion components perpendicular to the camera plane, camera zoom, and object rotations are not considered (they eventually come out from the joint observation of local motion vectors).

Baseline BBME consists in subdividing the image into arbitrarilyshaped blocks, and finding a correspondence among them in a pair of successive frames. A block can be a single pixel, a small patch, or even a region. Typically, regular blocks of small dimensions are adopted. Single pixels are noisy and subject to aperture problems. Large areas may contain different objects with incoherent motion.
We assume that the motion of small blocks is coherent.

- Spatial coherence: Neighboring points typically belong to the same surface and hence have similar motion.
- Temporal coherence: the content of a block is strictly correlated with what came before and what will come next.

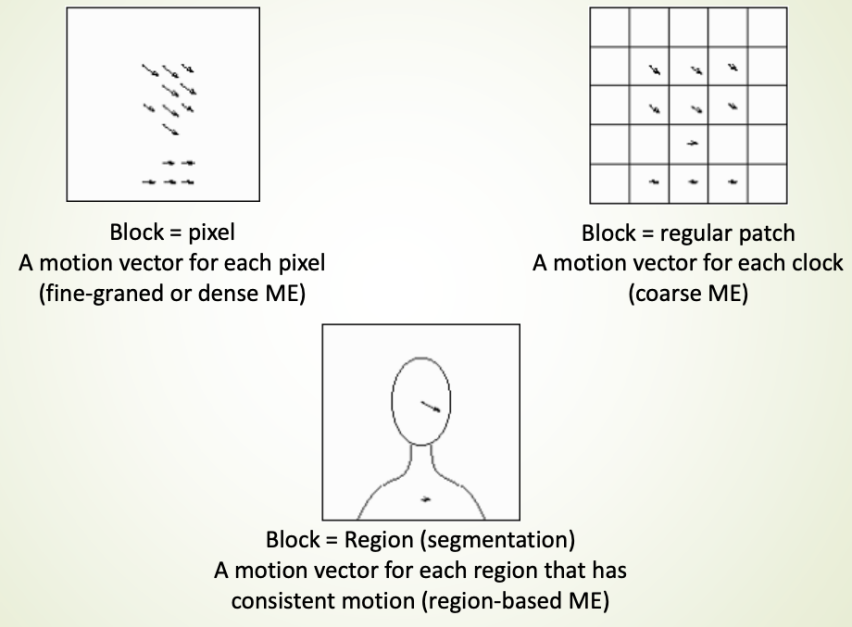


Figure 6.1: BBME: Choice of blocks

Notation The frame $I(x, y, t_1)$, for which we want to calculate the motion field, is called **target frame**.
The frame $I(x, y, t_2)$ that we use as a reference, is called **anchor frame**

$$\begin{cases} \text{if } t_1 < t_2 \rightarrow \text{ backward motion estimation} \\ \text{if } t_1 > t_2 \rightarrow \text{ forward motion estimation} \end{cases}$$

The displacement $(,)$ of the block B_{xy} centered in (x, y) is called **motion vector**.
The ensemble $W_1(x, y)$ of all motion vectors related to a given anchor frame $I(x, y, t_1)$ is called **motion field**.

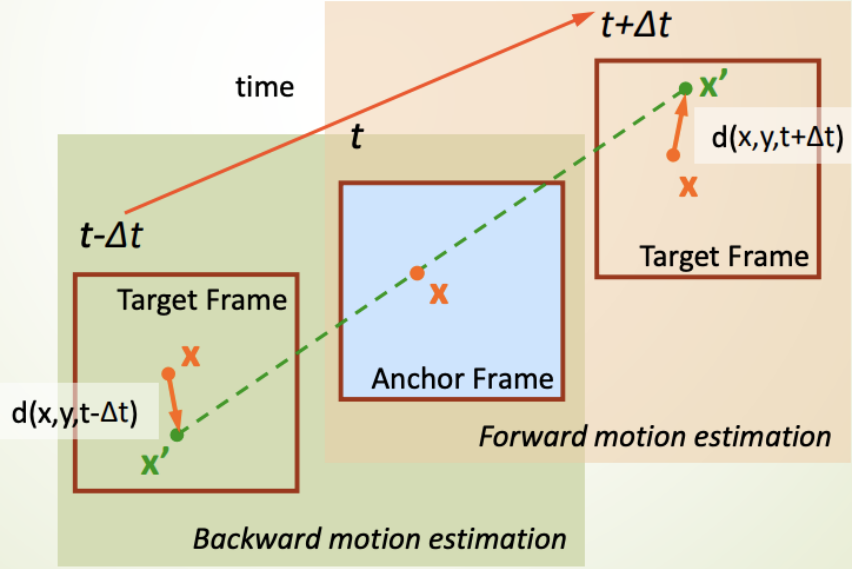
Forward vs Backward ME Two possibilities:

- Select as reference frame a past one and analyze the motion in the time direction (forward motion estimation)
- Select as reference frame a future one and analyze the motion in the inverse time direction (backward motion estimation)

Criteria To estimate the motion field, we need to calculate the displacement of each block. The best estimate of the displacement is given by the translation that minimizes the difference between anchor and target block (maximum similarity). Such difference is called DFD: **Displaced Frame Difference**. It is typically calculated as the p-norm distance of the two blocks.

BBME: Search methods

We have defined a 'cost function' (the DFD) and the 'goal' (find the MV that minimizes the cost function for a given block). We still have to define a strategy



(algorithm) to reach the goal. It is an optimization problem!
Several algorithms have been proposed, we'll introduce three well-known approaches:

- Exhaustive search: it is a brute-force method, all the possible solutions in a given solution space are explored
- Gradient-based search: relies on the assumption that the cost function has a single global minimum and that some 'locality criterion' holds
- Hierarchical search: relies on the assumption that the cost function is self-consistent at different scale factors (again, locality criterion holds)

FROM SLIDE 34

6.3.3 Global motion estimation