Computer Vision

By

Alessia Pivotto

Lorenzo Orsingher

Handouts of the course 2023/2024

Contents

Chapter 1

Images and Videos

1.1 Definitions

1.1.1 Computer Vision

Is the science and technology of machines that see, where“see” in this case means that the machine is able to extract information from an image that is necessary to solve some task. The image data can take many forms, such as video sequences, views from multiple cameras, or multidimensional data from a medical scanner.

Some examples of applications of computer vision include systems for controlling processes, like an industrial robot or an autonomous vehicle, detecting events as for visual surveillance or people counting, or again organizing information for example for indexing databases of images and image sequences, even modeling objects or environments like in industrial inspection, medical image analysis or topographical modeling and interaction as the input to a device for computer-human interaction.

1.1.2 Acquisition

It refers to the process of transferring a portion of the real 3D world onto a 2D surface bringing a continuous-parameter real world into a discrete-parameter one. The acquisition process is the first step in the processing chain and it is the transformation of

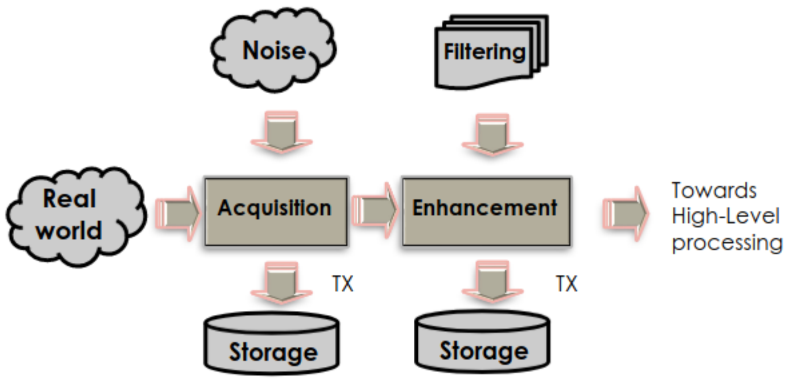


Figure 1.1: The processing chain(The first part is more about signal and video, in this course we'll focus mainly on the second half)

a physical signal into an electrical one, by means of a sensor. Where the sensor is a device that responds to a physical stimulus (light, heat, pressure, etc.) and produces an electrical signal. *PS: The representation is in a standard format.*

1.1.3 Digital Images

A digital image is a representation of a two-dimensional image as a finite set of digital values, called picture elements or pixels. The digital image contains a fixed number of rows and columns of pixels and so is a collection of coordinates. Pixels are the smallest individual element in an image, a single pixel represent a projection of a portion of the real world, holding quantized values that represent the brightness of a given color at any specific point. Pixels can be **Grayscale** or **Color**. Grayscale images are represented by a single component, typically 8bit, while color images are represented by three components, typically 24bits.

1.1.4 Sampling

Is the process of converting a continuous signal into a discrete signal. This is because the ”real world” is a continuous function, and computers are digital. Analog video is a 1-D continuous function where one spatial dimension is mapped onto time by means of a scanning process, while digital video is instead sampled in a three dimensions (2D spatial and 1D temporal).

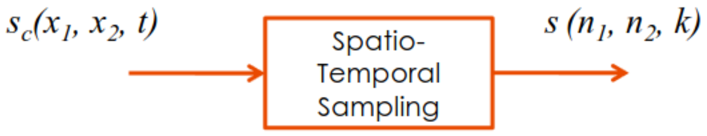


Figure 1.2: Sampling

In a Nutshell:

- Continuous signal

$$s_c(x_1, x_2)$$

- Spatial rectangular sampling

$$x_1 = n_1 \triangle x_1, x_2 = n_2 \triangle x_2$$

- So

$$s(n_1, n_2) = s_c(n_1 \triangle x_1, n_2 \triangle x_2)$$

- Once we have the digital format, we can manipulate data and apply filters, change colors, store and transmit.

For what concerns handling images we need only to know that pixels are numbered starting from the top left corner (so the top left corner is the origin, arrow down for the rows and right for the columns). The value of a pixel in a certain position is defined as $I(r, c)$, where r is the row index and c the column one. (Start at 0 or 1, depends on you

:D) Monochrome images have values normalized in the range $[0, 1]$, where 0 is black, 1 is white and the intensity is called *grey level*. While color pictures have 3 channels (RGB) and the values are normalized in the same way.

1.2 Color

But what is color? It's the attribute the human visual system associates to objects, more scientifically it's a mathematical relationship that combines different wavelengths. It's important to check whether something we see is what we expect, to recognize objects or to distinguish similar things. For example, a white car, that for us is obviously white, for a computer is a combination of red, green and blue, moreover it has some black points for the wires, some point gray due to the street, etc. For this reason let's talk about color perception. The human eye is like a camera with a focal length of about 20mm, where the iris controls the amount of light by adjusting the size of the pupil. The perception of color is possible through cones in the fovea that has around 100M receptors. Cones have peak responses on three main wavelengths, red (700nm), green (546.1nm) and blue (435.8nm).

But going back to the main topic, data can be processed locally but even transmitted remotely or archived on a storage unit. The problem is that images and videos require a lot of bandwidth, so we need to compress them with a codec. A codec is a device or computer program for encoding or decoding a digital data stream or signal in the compressed domain. It stands for coder-decoder, and it's a way to reduce the dimension of the file (e.g JPEG, MPEG, DIVX).

NB: Compression is lossy, so reduces quality (we lose some information) and can even introduces visual artifacts (such as blocking, blurring, chromatic aberrations(plus some noise from the sensor)). However, the loss is not a problem because the human visual system is not perfect. NB1: Exist even lossless compression, but it's not used much. NB2: Processing is typically done in the uncompressed domain. NB3: Raw images are vector of pixels.

1.2.1 Additive color model

The additive color model is a method to create color by mixing the primary colors.

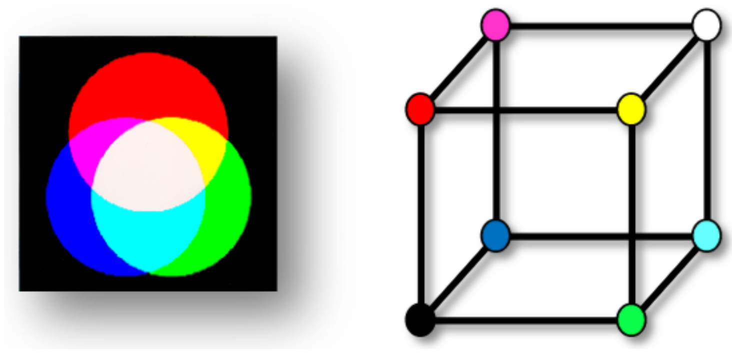


Figure 1.3: Additive Color Model

Colored beams are projected onto a black surface, then overlap so the human eye receives the stimula without generating interference, mixing the components and perceiving the resulting color. Starting from the primary colors RGB we can obtain:

- R+G = Yellow
- R+B = Magenta
- B+G = Cyan
- R+G+B = White

NB:Subtractive color is the inverse process.

About the way colors combined, we can have:

- Black: RGB(0,0,0)
- Green: RGB(0,1,0)
- Yellow: RGB(1,1,0)
- White: RGB(1,1,1)
- Gray: RGB(0.5,0.5,0.5)

Indeed, looking at the image above we can identify the gray scale along the diagonal connecting the black corner with the white corner.

NB: If we separate the three components and generate single images we notice that components are correlated, this means that the three version in grey-scale of the starting image carry almost the same amount of information.

In RGB we have a major response in the green component, while red and blue are less relevant. In addition, human eye is more sensitive to luminance and contrast variations than color so maybe a different representation would be more effective. Indeed, more effective is the YCbCr color space, where Y, the luminance is separated from Cb and Cr that are the chrominance. *PS: YCbCr is a generalization of YUV, just a matter of conversion matrices. (TODO: Downsampling)*
Another color space is the HSV, where H is the hue (color), S is the saturation (brightness) and V is the value (intensity).

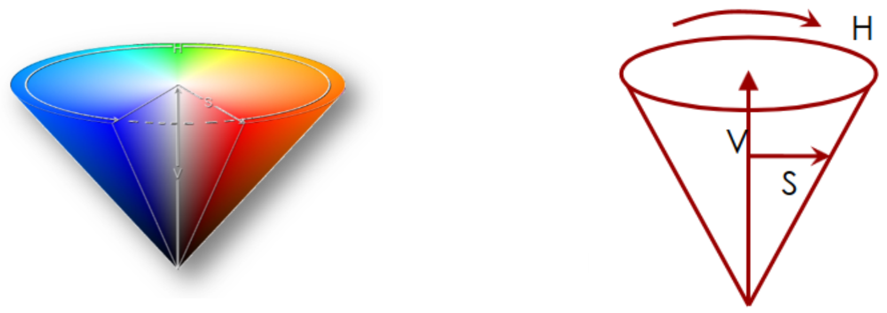


Figure 1.4: HSV Color Space

Summing up just a little bit:

- RGB is used in general for visualization, in displays each pixel in composed by three phosphors (CRT) or LEDs (LCD);
- YUV is suitable for compression since we are less sensitive to chrominance variations and U and V can be downsampled;
- HSV is robust for computer graphics and image analysis.

1.2.2 Bayer Pattern

In the acquisition phase, light is captured by the CCD (Charge Coupled Device) that is an array of cells. Each cell is a photosensitive element that converts light into an electrical signal. The Bayer pattern is a color filter array for arranging RGB color filters on a square grid of photosensors. The best solution would be to have devices with 3 different CCDs and to correctly exploit the human eye response there would be three types of photosensors of color filters 50% green, 25% red and 25% blue. *NB: Green sensors are defined as luminance-sensitive elements, while the red and blue ones are defined as chrominance-sensitive.*

1.2.3 Quantization

Like in the mono-dimensional case, signals need to be quantized. That implies the definition of a number of levels to define our signal. Typically, the range 0-1 is quantized using 8bpp but even other representations with 10-12 bpp are available. But why 8bpp? well, 8bpp represent 256 levels, which is fine for the human eye, that can distinguish 100 levels of grey. Indeed, if we quantize with less that 6bpp (64 levels, minimum to ensure smooth pictures), then false contours will appear $\Rightarrow$ contouring.

1.2.4 Limits of the 2D

Still images provide a reliable information about static scenes. We lose motion information like temporal evolution of the scene, rapid changes, dynamics of motion (qualitative and quantitative) and even how subjects/objects relate one to each other. Moreover, analyzing a video provides a more consistent representation of the scene. It's closer to what humans do every day.

1.2.5 Video

A video is a sequence of 2D images that represent a projection of moving 3D scene onto the video camera image plane. PS: It's expected that adjacent frames are strongly correlated. For what concerns the resolution, the images are up to 50mp, while in video is typically lower, 4k is 3840x2160=8.3mp, 8k is 7680x4320=33.2mp and full HD is around 2mp. The reasons are that video can last hours, so we need to store a lot of data, and the human eye is less sensitive to resolution in motion. Just move on a little bit, once the image is acquired, what are the most relevant features we could be interested in? Maybe color and its distribution or presence of edges and contours. . . And again, if the analysis concerns a video, instead of an image, what is the added value? Consistency of the features over time, evolution of the scene and objects displacement and, of course, objects that may enter or exit the scene.
So in the end, what makes a scene complex to analyze and process? The presence of multiple objects, occlusions, shadows, noise, motion, illumination changes, clutter and non-rigid objects. TODO: vedi se aggiungere gli esempi.

1.2.6 Histogram

The histogram is a simple way to describe the color distribution of a picture, indeed, it can be seen as a probability density function and it represents the occurrence of all colors on a graph. In other words it's a statistical representation of the pixel values. For an MxN image

$$hist(p) = \frac{I(x,y)}{MxN}$$

Where $I(x,y)$ is the intensity of the pixel at position (x,y) and MxN is the total number of pixels in the image.
The equation holds for one component, so in case of more components, a histogram can be obtained from each of them. So, what kind of information can we obtain from the histogram?

- Is the image dark or bright?

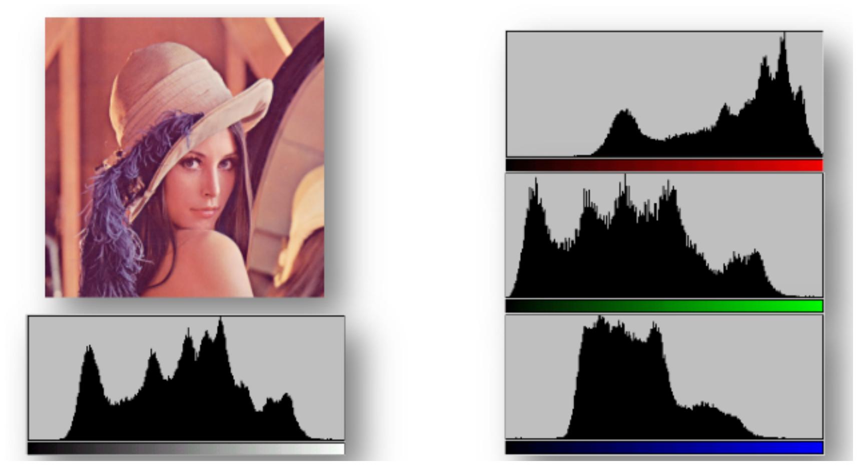


Figure 1.5: Histograms of Lana

- Are colors distributed equally?
- Are there dominant colors?
- Are there colors missing?

Some applications:

- Is the illumination of a scene correct for environmental monitoring?
- How has the background changed in the past 2 hours?
- Can I use the histogram to determine whether the moving object I'm observing is A or B?

In general it can be seen as a "signature" to be applied in different domains.

1.2.7 Operations

Stretching:

A simple operation to change the dynamic range of the image by stretching the histogram and increasing the contrast. Usually apply a piecewise linear function.

Equalization:

Working on the statistics of the pixels it is possible to improve the quality of an image.

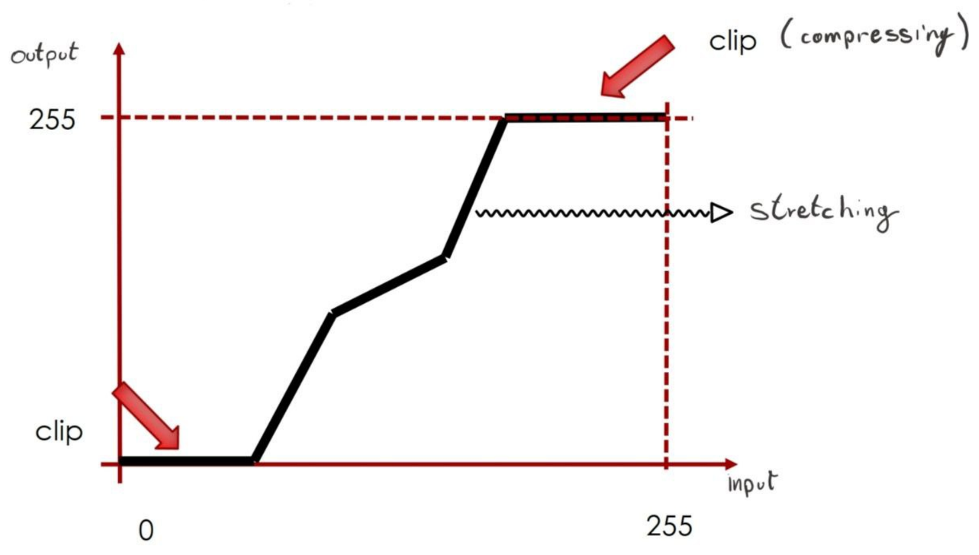


Figure 1.6: Stretching

Ideally we'd like to obtain a FLAT histogram, and to do so we have to compute the cumulative histogram (equivalent to CDF).

$$CHist_I(p) = \sum_{k=0}^{p} hist(k)$$

$$hist_eq(p) = \frac{CHist(p) - CHist_{min}}{M \times N - 1} \times 255$$

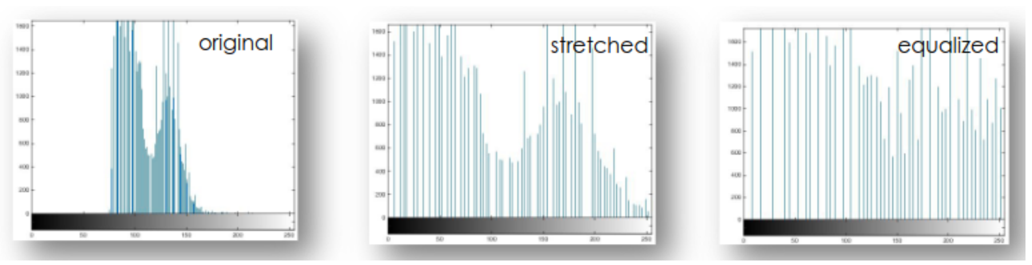


Figure 1.7: Different operations on the histogram

1.3 Edge extraction

We perceive objects through color (appearance model) and shape. Shape is the boundary between the object and the rest of the picture, and it helps us recognizing things. A sharp change in image brightness can be seen as an edge.
PS: Environmental conditions, such as cast shadows or surface orientation, can generate changes. Moreover edges are not always meaningful, for example in textured areas or in the presence of noise.
Steps:

- Determine intensity, and possibly the direction of an edge for each pixel location through gradient or Laplacian;
- Find a threshold and binarize.

NB: The first step is the most difficult one. Different tools are available but we'll focus here on the gradient-based algorithms.

Edge extraction by gradient analysis

As for mono-dimensional signals, the goal is to find maxima and minima but in this case, we also have a direction. This means finding a gradient along a line r oriented in the direction θ.

$$\frac{\partial f}{\partial r} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial r} = f_x \cos(\theta) + f_y \sin(\theta)$$

The edge extraction process consists of computing the 1st order derivative in two orthogonal directions, f_1 and f_2. To each of them we associate an amplitude. Going a little more into the concrete, for edge detection typically we use FIR filters, that are linear and shift-invariant. The most common are the Prewitt, Roberts and Sobel operators.
For the first two we have to choose a convolution mask and apply it to the picture. The convolution is computed for both masks and a threshold is chosen to highlight only the strongest edges.

Roberts operator: $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ **Prewitt operator:** $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ $\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

NB: The Prewitt operator is preferred because it is isotropic $\Rightarrow$ you know where is the center.

For the Sobel operator, we have to apply two masks, one for each orthogonal direction. The mask is a 3x3 matrix and the gradient is computed for each point. Then we have to threshold the result.

Sobel operator: $D_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ $D_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

The convolution with the FIR masks is performed similarly to the 1D convolution. Take the mask, rotate, slide from left to right and associate to the central point the value of the convolution. (formule ancora da mettere, non avevo voglia sorry)

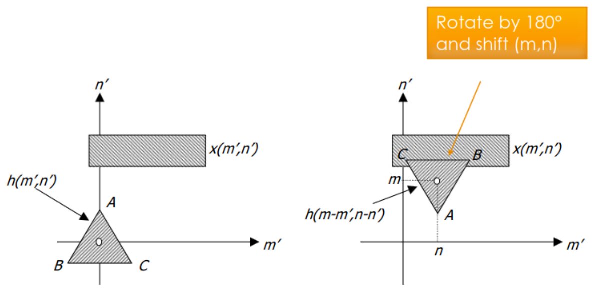


Figure 1.8: How to (convolution edition)

1.4 Filters

1.4.1 Low-pass filtering

The easiest way is to average the values in the sliding window. The window is a matrix of size $m \times n$ and the result is the average of the values in the window.

$$I_{LP}(x,y) = \frac{1}{mn} \sum_{x=-a}^{a} \sum_{y=-b}^{b} I(x,y)$$

Preserve low-frequency components and remove high-frequency ones. It's a weighted sum of low high frequency functions. Cut the high frequency components above the threshold and smooth the image. Average filtering $\Rightarrow$ average of the pixels in the window. Bigger the filter, more the smoothing (the difference is smaller). It helps denoising the signal.

1.4.2 Gaussian filtering

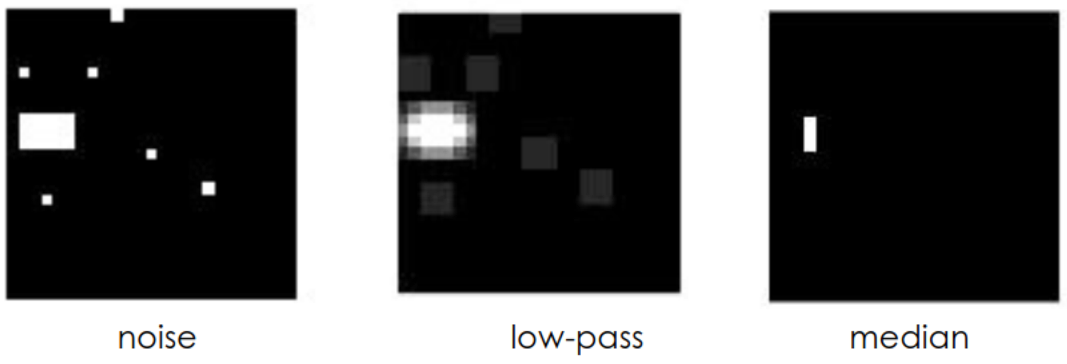
Gaussian filtering is a low-pass filter that is used to remove the high-frequency components of the image. It is a 2D convolution filter that is used to blur the image and remove noise. Gaussian try to reconstruct a gaussian function, so it's isotropic and the mask is not flipped. The mask is a 2D Gaussian usually centered in the center of the filter mask, set big enough to make sure values in the mask are integer. Using a Gaussian as a mask is convenient as in the Fourier domain it is still a Gaussian, it is isotropic (simmetric) and no need to flip the mask (no rotation in convolution).

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

the bigger the better. The central pixels are more considered, far pixels have a lower impact.

1.4.3 Median filtering

Median filtering is a nonlinear method used to remove noise from an image. It is a spatial domain method that replaces the pixel value with the median value of the



neighborhood pixels. It is particularly effective in removing salt and pepper noise. It is a good method to use when the image is corrupted by impulse noise. In case the noise is zero-mean, smoothing is fine. But when you have spikes, LP filtering blurs also the noise. Indeed, LP spread the dot, so it generates artifact. The noise is less visible but larger. Median avoid this issue, remove noise, no linear function, no convolution, sorting operation for every shift, filter ranks the values of the filter from bigger to smaller and store the median value in the central pixel. Bigger could introduce artifacts, so it's better to use a small filter. The noise that remains is because the size of the mask is not big enough to cover the noise.

1.5 Morphology

Morphology refers to the shape of a region. The goal is to check whether a certain shape fits into another, check whether a picture has holes of a certain size, remove areas smaller than a threshold and with certain shape, etc. *PS:It refers to non-linear filtering.* It uses a structuring element, a known arbitrary shape, and the operations are performed by sliding the structuring element over the image. The structuring element has an origin (typically central point, but not necessarily) and the operation is performed by comparing the structuring element with the image. There are four main operations: Erosion, Dilation, Opening and Closing.
Erosion and Dilation are self-explanatory, the first one reduces the area of a shape, the second one enlarges it. Instead, opening and closing are a combination of erosion and

dilation. Opening gets rid of small portions of the image close to the boundaries of relevant areas, while closing fills holes and makes region boundaries smoother. *Concerning structuring elements, depending on the type of shape we want to edit, the right element must be chosen.*

1.5.1 Dilation

Dilation is a morphological operation that is used to enlarge the boundaries of regions of foreground pixels in an image. It is used to merge the objects in the image. The structuring element is a binary mask that defines the neighborhood of the pixel. The dilation of the image is obtained by sliding the structuring element over the image and placing the origin of the structuring element at the center of the pixel. More formally, the dilation of an image B by a structuring element S is given by:

$$B \oplus S = \cup_{b \in B} S_b$$

Sweep the structuring element on the whole image, as the origin of the structuring element touches a "1" of the image all pixels of the structuring element are OR'ed to the output image. *PS: OR/union $\Rightarrow$ every time that the structuring element touches a 1, the output is 1, so it enlarges the shape.*

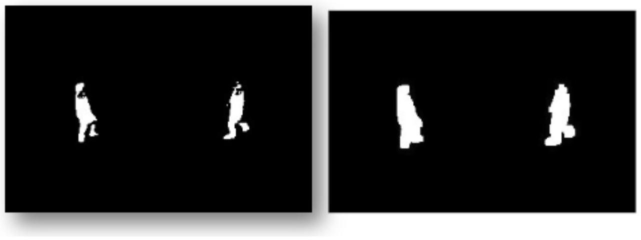


Figure 1.9: Dilation

1.5.2 Erosion

Erosion is a morphological operation that is used to reduce the boundaries of regions of foreground pixels in an image. It is used to separate the objects in an image. Sweep the structuring element on the whole image, at each position where every 1-pixel of the structuring element covers a 1-pixel of the binary image the binary image corresponding to the origin is OR'ed with the output image. More formally, the erosion of an image B by a structuring element S is given by:

$$B \ominus S = \{b | b + S \in B \quad \forall s \in S\}$$

Erosion of A by B can be understood as the locus of points reached by the center of B when B moves inside A.

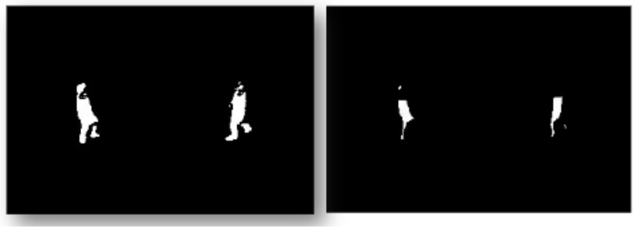


Figure 1.10: Erosion

PS: se prima bastava che il centro toccasse un 1, ora TUTTI devono toccare un 1.

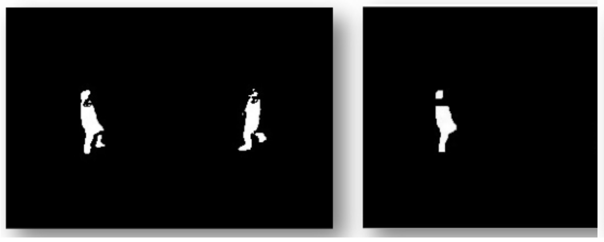
1.5.3 Closing and Opening

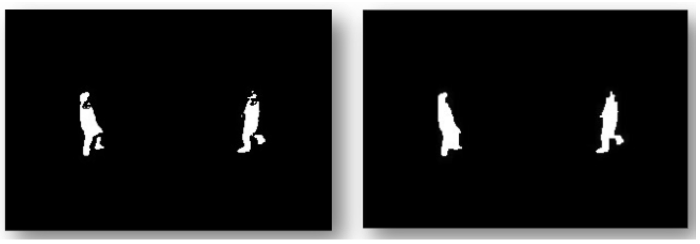
- Closing ⇒ first dilatate and then erode;

  Non-contiguous regions are firs merged, then the boundaries are refined.

- Opening ⇒ first erode and then dilatate;

  First small areas are removed, then residual information is refined.


(a) Opening


(b) Closing

This operations are in general non-reversible, so the result depends on the order of the operations.

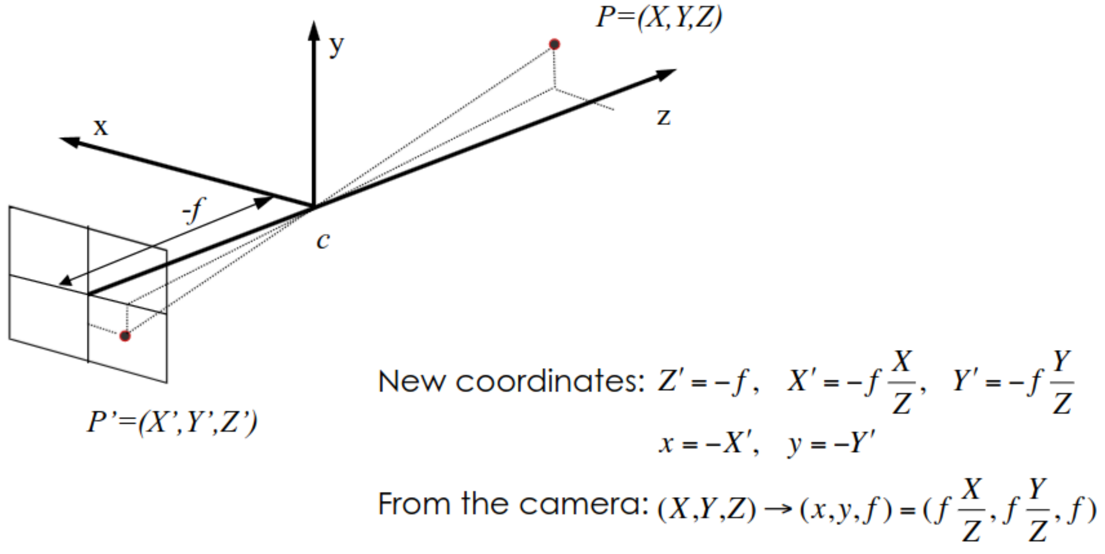
Chapter 2

Models

Let's begin the chapter with a question. Why do we need models? Well, they represent a good approximation of the real world and allow describing events through a parametric representation (parameters can be extracted and used for processing). In the context of computer vision, models are used to describe the geometry of the world, the appearance of objects, and the imaging process. In this chapter, we will discuss the most common models used in computer vision. We will start with the pinhole camera model,then we will see the perspective projection model and finally, we will discuss the appearance model.

2.1 The Pinhole Camera Model

The pinhole camera model is the simplest model used to describe the imaging process. It is based on the principle that light travels in straight lines. The model is composed of a pinhole, a plane, and an image plane. The pinhole is a small hole in the plane, and the image plane is placed behind the pinhole. The light rays coming from the scene pass through the pinhole and project the scene onto the image plane.
PS: If the object is far, it appears small in the image. If the object is close, it appears large in the image.



New coordinates: $Z' = -f, \quad X' = -f\dfrac{X}{Z}, \quad Y' = -f\dfrac{Y}{Z}$

$x = -X', \quad y = -Y'$

From the camera: $(X,Y,Z) \rightarrow (x,y,f) = (f\dfrac{X}{Z}, f\dfrac{Y}{Z}, f)$

2.1.1 Features of the Pinhole Camera Model

- Parallel lines in the world converge to a single point in the image;
- Parallel lines on the same plane lead to *collinear vanishing* points;
- The line is called the *horizon* for that plane;
- Vertical lines are perpendicular to the horizon.

2.2 Projection

The projection is the process of mapping 3D points (+time) to 2D points. The projection of a 3D point onto the image plane is defined by a line that passes through the point and the center of projection. The intersection of this line with the image plane gives the 2D projection of the 3D point.

$$f : \mathbb{R}^4 \rightarrow \mathbb{R}^3$$

$$f(X, Y, Z, t) = (x, y, t)$$

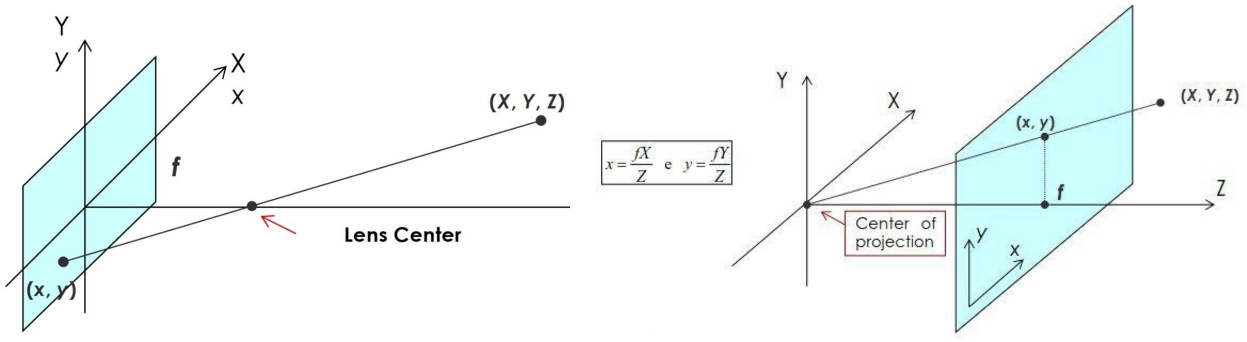
NB: contiuous variables.

Exist two types of projection:

- Perspective projection;
- Orthographic projection.

2.2.1 The Perspective Projection Model

The perspective projection model is an extension of the pinhole camera model. It is used to describe the imaging process in a more realistic way. The model is based on the principle that light travels in straight lines and that the image is formed by the intersection of these lines with the image plane. For simplicity we usually consider the image plane on the same side of the “real world”, to avoid the picture flip. The center



of projection corresponds to the origin of the 3D space. The plane (X,Y) is parallel to (x,y).

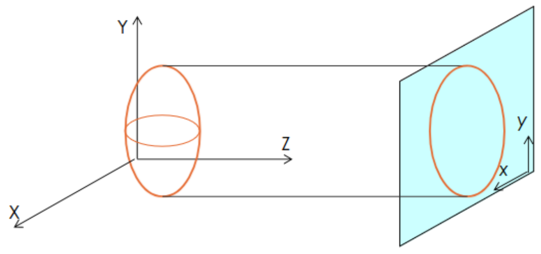
PS: This approximation is allowed when Z>>f.

From a computational point of view nothing changes, we simply shift the image plane from back to front in order to have the image straight. For what concerns the projection, we can use the same formulas as before but we remove the minus f from the denominator(too small).

$$\begin{cases} \frac{x}{f} = \frac{X}{Z-f} \\ \frac{y}{f} = \frac{Y}{Z-f} \end{cases} \Rightarrow \begin{cases} x = \frac{fX}{Z-f} \\ y = \frac{fY}{Z-f} \end{cases} \Rightarrow \begin{cases} x = \frac{fX}{Z} \\ y = \frac{fY}{Z} \end{cases}$$

2.2.2 The Orthographic Projection Model

The orthographic projection model is a simplified version of the perspective projection model. It is assumed that all rays originated from the 3D object, and from the scene in general, are parallel among each other. In the drawing, the image plane is parallel to (X,Y).



Assuming that the image plane is parallel to (X,Y), the orthographic projection can be simply described in Cartesian coordinates as:

$$\begin{cases} x = X \\ y = Y \end{cases}$$

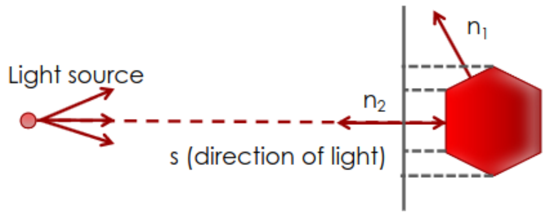
Or in form of a matrix:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

NB: The distance of the object from the camera does not affect the intensity of the image projected onto the 2D plane. It is a good approximation when the distance of the object is much bigger than the depth of the object itself.

2.3 Illumination Models

Illumination is a key element, it models the appearance/perception of objects. When a light source hits an object, light can be absorbed, reflected, or transmitted. It's very complex to model, we perceive objects because they reflect light in specific wavelengths. Reflection can be specular or diffuse. Specular means that more energy is concentrated in the light source direction. While in diffuse the energy is constant in all directions, and the position of the observer is irrelevant. Different surfaces vary in specularity, some of them (matte) reflect light uniformly in all directions, while glossy objects reflect light in specific directions. It also depends on the distance and the inclination of the light source. The problem is to determine how the surface is irradiated by the light source.We can make an assumption that the light source is far away, so we can assume that all rays can be represented by a single unit vector s (orthographic projection). For each surface element (btw dashed lines) the light is irradiated considering the cosine of the angle between the surface normal and the light direction.



NB: intensity we perceive is equal to dot product of the normal of the surface and the light direction. Dot product ⇒ cosine ⇒ more close to 90 degrees, less we see.

2.3.1 Lambertian Reflectance

Is a model for diffuse reflection, it is based on the assumption that the surface is rough enough w.r.t. the light wavelength. This means that each surface element reflects light evenly in all directions. The luminance of the surface is the same regardless of the viewing angle. The specular component is neglected.
Assumes that the surface we're dealing with is ideal, so looking at it from any angle we'll see the same intensity.

$$I = \rho n \cdot s$$

Where:

- I is the intensity of the light;
- ρ is the albedo, the ratio of the reflected illumination to the total illumination,

intrinsic property of the surface (not true hihih, some surfaces may reflect light differently depending on the view angle);

- n is the normal to the surface;
- s is the direction of the light.

An element is not visible if the angle between the normal and the light direction is greater than 90 degrees. The pixel in image

$$I(r,c)$$

depends on the light source direction and the normal of the element direction.

2.4 Remarks on cameras and lenses

Cameras are equipped with lenses (no pinhole in real life!). This means that the object is on focus if the distance from the center of the camera and the image plane obeys to the thin lens equation. If not, we have aberrations.We typically assume the object is on focus, but let's see what this means first.

2.4.1 Typical issues with lenses

Spherical aberration:
The lens does not focus light rays that strike the lens far from the center. This means that the image is not sharp (Causes blur). In simple terms the lens does not focus all the rays in the same point, so the light is reflected wrongly because the lens is not properly manufactured.
Chromatic aberration:
The lens does not focus all the colors in the same point, it's not able anymore to convey correctly the ray (at different locations). es tv a tubo catodico.
Vignetting:
The lens does not focus all the rays in the same point, so the image is darker in the

corners. The spreading of light is not uniform due to the impurity of glass.

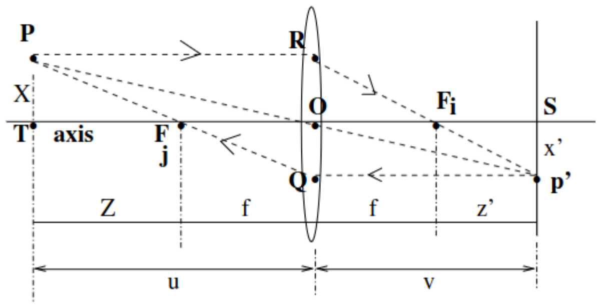
Barrel distortion:

The focal length is too short or the lens is too wide, so the image is distorted and the lines are not straight anymore. In simple terms, the lens is able to grasp a wide area of view. To compensate we can use a software to correct the distortion. (Linee storte possono dare problemi nell'analisi del percorso tipo... problema della distorsione-¿ecco perchè i cell moderni hanno più lenti)

2.4.2 Focus

Pinhole is an abstract model, in real life we have lenses. The lens is able to focus the light rays in a single point. The distance from the center of the camera and the image plane obeys to the thin lens equation.

$$\frac{1}{f} = \frac{1}{u} + \frac{1}{v}$$

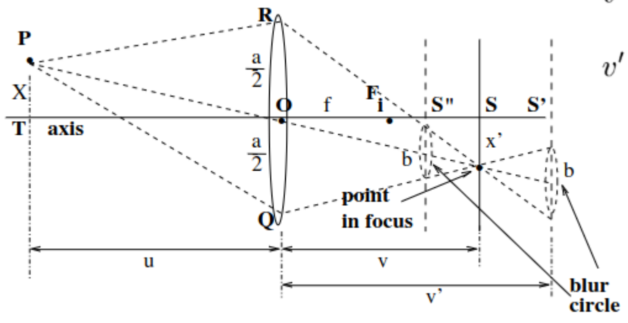


If we move the image plane, point p' is out of focus $\Rightarrow v$ changes $\Rightarrow v'$. If P is moved u changes $\Rightarrow u'$. The result is that the image is blurred on the image plane (instead of a point I see a circle, the rays of light don't converge).

NB: If the object is too far you can try to set the focal length but it won't work.

We want to configure the camera in order to have the quantity b as small as possible. Notice that:

- In general $u > f = u_n < u$;



$$v' = \frac{a+b}{a}v \quad : in \;\; case \;\; v' > v$$
$$v' = \frac{a-b}{a}v \quad : in \;\; case \;\; v' < v$$

$$u_n = \frac{u(a+b)}{a + \frac{bu}{f}}$$

$$u_r = \frac{u(a-b)}{a - \frac{bu}{f}}$$

- If f becomes smaller, u_n is closer to the camera;
- If f becomes smaller, u_r is farther to the camera;
- $u_r > u$;
- If $u- > \infty$ rays are parallel and converge to the camera center;
- The difference between the far and near planes limiting b is called *depth of field.*

Autofocus

Is the capability of focusing a specific portion of the image, it can be active, passive or a combination of both.

Active:

The camera emits a signal and measures the time it takes to return. The time is used to compute the distance of the object from the camera. It is mainly used for point-and-shoot cameras. The issues are related to obstacles and glossy and bright surfaces.

Passive:

The camera uses the contrast of the image to focus. It is mainly used for professional cameras. It is more expensive SLR (single-lens reflex) cameras, it uses the phase detection method. Distance computed using image analysis: take a strip of pixels and analyze the distribution, then if values are too similar the object is out of focus, if contrast is high the object is on focus. The problems are related to flat surfaces, to low contrast and low light. Good cameras compute the metric on the vertical and horizontal axes.

Chapter 3

Motion Detection

Motion is generally given by a transformation. In video analysis and processing motion is a fundamental feature. Motion detection is the process of detecting a change in the position of an object relative to its surroundings or a change in the surroundings relative to an object. We'd like to have an accurate description of objects motion in the scene and camera motion. So we need context in order to understand if is the camera that is moving or it's the object (using displacement vectors). As before, we're losing information when we're converting a 3D world into a 2D image. So we can't really know if an object is moving or if the camera is moving. We can only know that something is changing in the image.

$$3D : D(X; t_1; t_2) = X' - X = [Dx, Dy, Dz]$$

$$2D : d(x; t_1; t_2) = x' - x = [dx, dy]$$

NB: x is the projection of X on the image plane.

We're able to perceive motion only if exist a projection of the image on the image plane. Single camera involves a problem: you can't understand if the point in front of you is growing or coming closer to you.

Let's consider an example: The questions that came up are:

- How can I extrapolate the motion field?

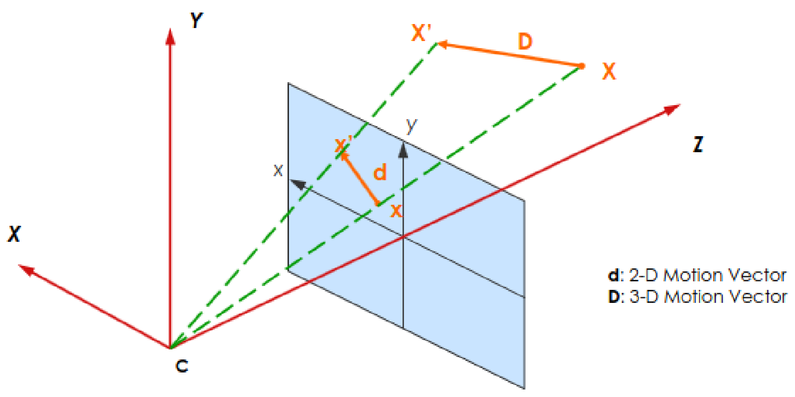
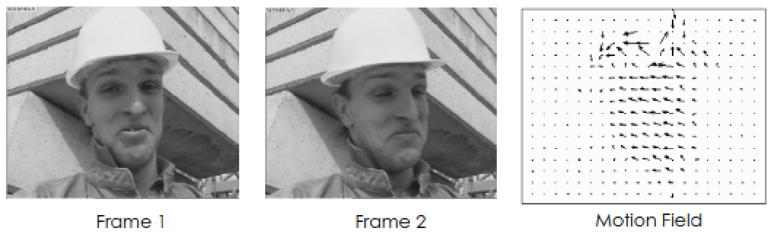


Figure 3.1: 2D Motion of a 3D object



- Does it really reflect the object displacement?
- Is the motion due to the camera or to the object?

We can notice an inconsistency in the area of the hat. This is because the hat is a flat surface and the color doesn't allows the window to understand the motion.
PS: esempio brutale, in una situazione con illuminazione ideale noi, potendo guardare solo una piccola porzione di un foglio bianco, non riusciremmo a capire s eil foglio si sta muovendo oppure no. Il movimento lo cogliamo solo guardando i bordi e la figura del foglio nella sua interezza e nel contesto! per quanto riguarda il cappello immagina di avere tre finestre, o meglio una che slida dentro la parte centrale del cappello, per tutte e tre le finestre ottieni lo stesso risultato quindi niente movimento, o comunque non si capisce un kaiser.

2D motion of a rigid object

The camera is supposed to be fixed, objects move in the 3D space, so motion is generated by a complex combination of arbitrary movements. The 2D projection we obtain is ambiguous because of combinations of different movements can generate the same image. Through an a posterior analysis it's very difficult to determine the different types of movements that affect the scene.

3.1 Motion detection

Some applications of motion detection are:

- Detection
- Segmentation
- Recognition
- Filtering (deblurring, noise suppression)
- Compression (transmission and storage)

Simplifying the problem, in general, we want to detect changes over time of the image intensity. This is a Ill-posed problem, the solution is not unique. (Other problem: lost of 1 dimension, zoom is viewed as a translation) So the problem is that it doesn't reflect the real motion, where the motion is defined as the velocity $v(x, y, t)$ of the pixels over two consecutive frames. The motion can be caused by object motion, camera motion and/or illumination changes.

3.1.1 Real vs Apparent movement

2D motion is perceived by changes in the scene over time, but it's not always identify correctly. With a constant illumination, for example, a perfect rotating sphere is perceived as not moving. With an illumination source that rotates around a static sphere, the sphere is perceived as moving.

3.1.2 Occlusion

Occlusion is a problem that arises when a surface is covered/uncovered by the movement of an object. Now take a look at this example of a circle sliding over a surface. There

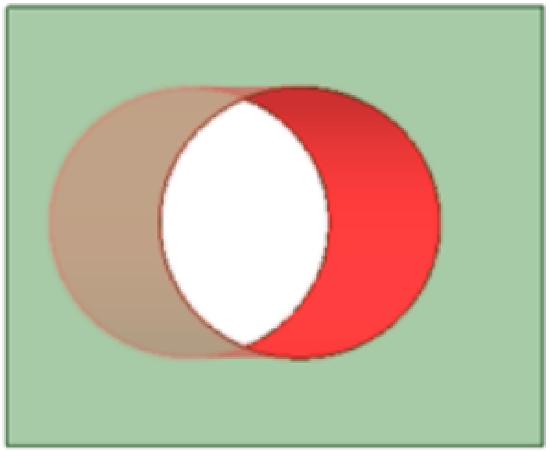


Figure 3.2: Occlusion

is a portion of the background (the vivid red one) that will be covered by the circle, and a region(the pale red) that will be uncovered, didn't exist in the previous frame. However, in the end we have notion of movement in this regions but we don't have any information about the white region where nothing seems to be happening.

3.1.3 Aperture

It's a problem that arises because movement can be detected only along the borders of the objects and only in the direction of the intensity gradient.
NB: only displacements of the borders are detectable.
To simplify, imagine to drag a paper along its diagonal. We can only see the 2 edges moving up and right but we can't see the actual corner moving.
TODO: controlla tutta sta parte e modifica l'immagine.

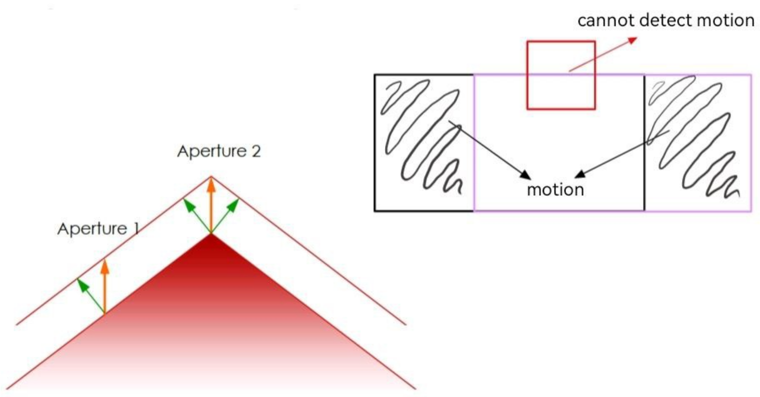


Figure 3.3: Aperture

3.1.4 Optical Flow

At this point things look complicated... but we can simplify the problem by making some assumptions:

- Object illumination does not change in $[t, t + dt]$;
- Distances do not change significantly;
- Each point $[x, y]$ is shifted in $[x + dx, y + dy]$.

NB: This is not true in real video but good enough for a first approximation.

Optical flow equation

Hypotesis: object points keep the same intensity over time

$$\psi(x, y, t) = \psi(x + dx, y + dy, t + dt)$$

We can use the Taylor expansion

$$f(x + \Delta x) = f(x) + f'(x)\Delta x$$

to approximate the function(for small dx, dy, dz):

$$\psi(x + dx, y + dy, t + dt) \approx \psi(x, y, t) + \frac{\partial \psi}{\partial x}dx + \frac{\partial \psi}{\partial y}dy + \frac{\partial \psi}{\partial t}dt$$

Comparing the two we obtain that

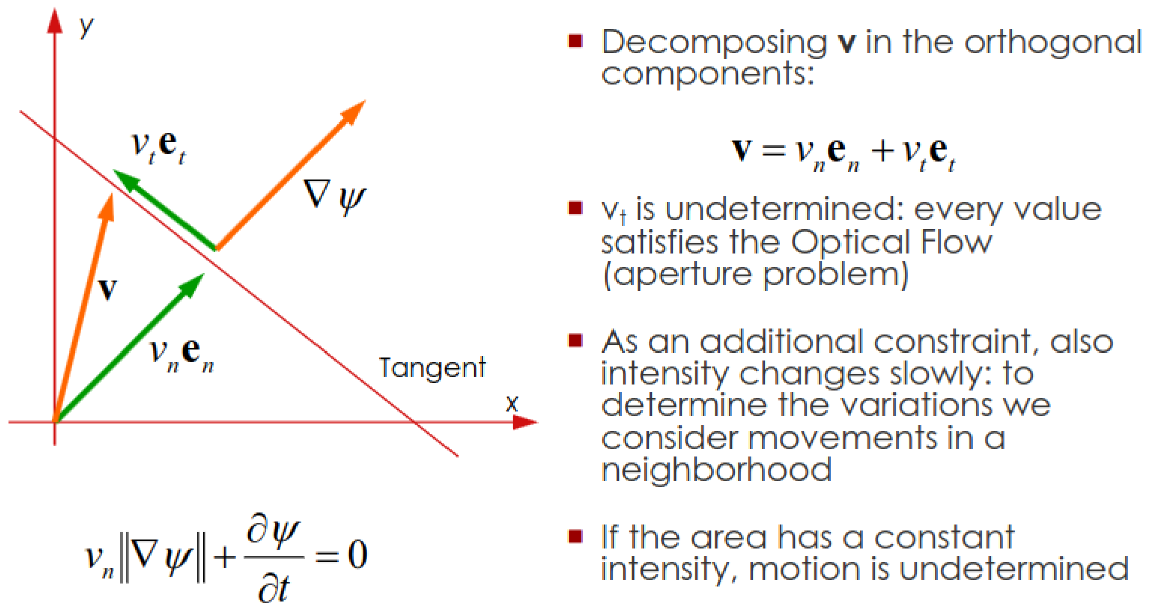
$$\frac{\partial\psi}{\partial x}dx + \frac{\partial\psi}{\partial y}dy + \frac{\partial\psi}{\partial t}dt = 0$$

Then, dividing by dt and rearranging the terms we obtain the optical flow equation:

$$\frac{\partial\psi}{\partial x}v_x + \frac{\partial\psi}{\partial y}v_y + \frac{\partial\psi}{\partial t} = 0 \quad \text{or} \quad \nabla\psi^T \cdot \mathbf{v} + \frac{\partial\psi}{\partial t} = 0$$

PS: if $v = 0 \Rightarrow$ we have no variation.

Comments:



$$v_n\|\nabla\psi\| + \frac{\partial\psi}{\partial t} = 0$$

- Decomposing **v** in the orthogonal components:

$$\mathbf{v} = v_n\mathbf{e}_n + v_t\mathbf{e}_t$$

- v_t is undetermined: every value satisfies the Optical Flow (aperture problem)
- As an additional constraint, also intensity changes slowly: to determine the variations we consider movements in a neighborhood
- If the area has a constant intensity, motion is undetermined

Figure 3.4: Optical Flow

3.2 Motion detection in practice

We have two possibilities to detect motion: Intensity based and Feature based. However, we have some problems, like the representation of the motion field and the choice of discriminant parameter.

3.2.1 Change Detection

To begin, just take a look at the following example:

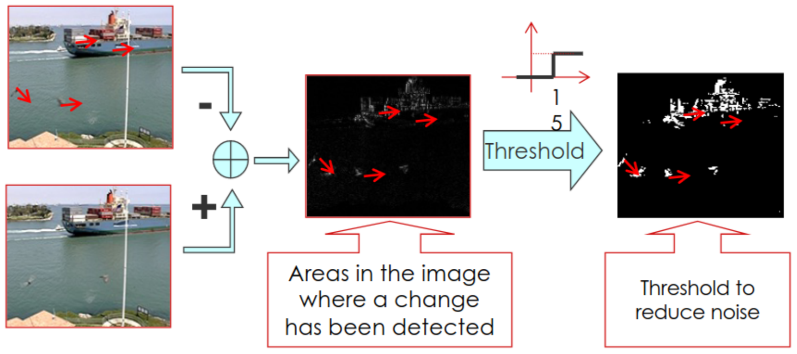

-
+
Threshold
1
5
Areas in the image where a change has been detected
Threshold to reduce noise

Figure 3.5: Change Detection

The two image on the left are the reference image and the current image. We compute the pixelwise subtraction between the two. The result is a binary image, where the white pixels are the pixels that have changed.
Looking at the first binary image we can notice a problem, the boat, that is moving, is not detected so well. Indeed, we are not able to understand very well the motion. Our goal is to obtain a motion map that is able to describe the motion field, so the displacement of the pixels. In this case we choose to change the threshold of the subtraction, so we can obtain a better result. In this way, we reduce noise removing the pixel with loud motion components. Another problem present in both images regards birds. There are two birds flying in the scene but we see four of them. This is due to the fact that the choice of Δt is wrong, more specifically is too large. So we need to reduce it increasing the number of samplings. Another problem that we don't see here but can occur is when we have only one camera and we can't compute correctly the velocity of an object moving in the scene. I'll explain myself better: if there is a car moving on the long diagonal of a rectangle and we have only one camera on the bottom border of the rectangle we probably think, watching the binary image, that the car is moving slower the more it goes away from the camera.
TODO: riscrivi sta parte

Formally speaking: Change detection is the process of identifying regions in an image that have changed between two or more images. The goal is to obtain a motion map that describes the motion field, i.e. the displacement of the pixels. Can work if the illumination is almost constant.
In addition, frame difference corresponds to: $I_k - I_{k-1}$

$$FD_{k,k-1}(x_1, x_2) = s(x_1, x_2, k) - s(x_1, x_2, k-1)$$

If FD is non-null, a change has occured. Change can be due to noise, so a threshold is needed to control it:

$$z_{k,k-1}(x_1, x_2) = \begin{cases} 1 & \text{if } |FD_{k,k-1}(x_1, x_2)| > \tau \\ 0 & \text{otherwise} \end{cases}$$

NB: a good strategy to avoid noise is to use the so-called cumulative difference.

3.2.2 Frame differencing

Is simply subtracting the current image from the previous one. The result is a binary image where the white pixels are the pixels that have changed. A key assumption for this method is that camera must be stationary. The core of this method is to replace background at each frame.
For example:

```
B(0) = I(0);
...
loop time t
I(t) = next frame;
diff = abs[B (t-1) I(t)];
Map(t) =
threshold(diff);
...
B(t) = I(t);
end
```

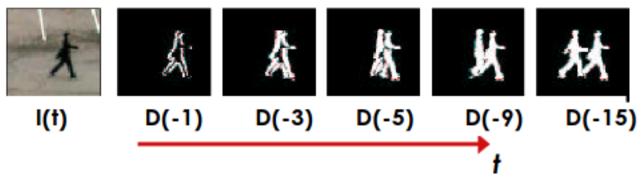
Main things to consider are:

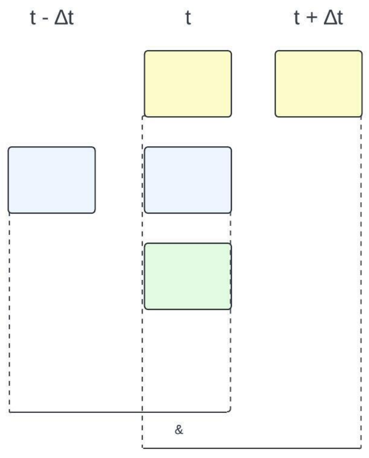
- Quickly adapts to changes in the background;
- Once an objects stops, it is not detected anymore;
- Can detect contours of objects;
- Only few pixel are usually detected;
- If motion is parallel to the edge, then it can lead to artifacts;
- By changing the time scale, results can be significantly different.

Regarding the last point, we can observe the following:

$$D(N) = ||I(t) - I(t+N)||$$

The contour of the object becomes clearer, until it doubles: departure and arrival





points.

NB: doubling is not good, you can't tell which one is the real one.

Solution ⇒ AND between 2 different frames(I understand the real movement).

3.2.3 Background subtraction

Background is modeled as a static image, in general acquired without moving object. The current frame is compared with the background to detect moving objects. The background is updated over time to adapt to changes in the scene. So, we calculate the difference between current frame and background and assign a label: 1 if it's part of foreground, 0 if it's part of background.
Example:

```
B = I(0); #initial background
...
loop time t
I(t) = next frame;
diff = abs[B - I(t)];
Map(t) =
threshold(diff);
...
end
```

Comments:

- Good results are obtained if the color of objects differs from the background;
- A problem is that if an object enters the scene, it will always be regarded as foreground (aka it won't be detected);
- In addition, if a background object moves, both real object and its "ghost" are detected.
- It is sensitive to illumination and background changes such as trees, reflections on glossy surfaces like cars and water, shadows, etc.
- In addition it is not robust to camera motion.

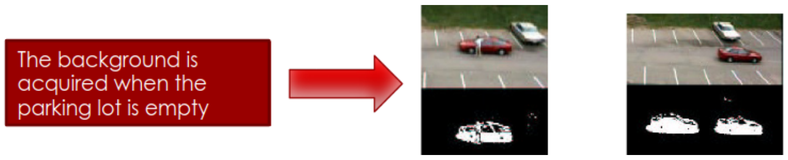


Figure 3.7: Possible problems of Background Subtraction

Adaptive background subtraction

–TODO: tutto, era un casino e devo ragionarci su.

3.2.4 Gaussian average

The idea is to model the background as a Gaussian distribution. The mean and the variance of the Gaussian are updated at each frame. The current frame is compared with the background model to detect moving objects. It may use a training sequence to initialize the background model, such as a possibly empty scene. It take the current pixel value and the previous average and standard deviation. It is very similar to adaptive background subtraction, but more stable.

$$\mu_t = \alpha I_t + (1 - \alpha)\mu_{t-1}$$

The advantages are its computational efficiency and the low memory requirements. Pixel goes to foreground if: $|I_t - \mu_t| > k\sigma_t$ Quality depends on a, a depends on the application, a too high value can lead to a slow adaptation to changes, a too low value can lead to a noisy background. Slow update = slow response to changes.

Improving the Gaussian average

To improve the Gaussian average we can use a binary operator to update values selectively, so only background values. M=1 if the pixel is foreground and 0 if it's background. This produces a more stable model and prevents pixels from being considered as background when they are not.

$$\mu_t = M\mu_{t-1} + (1 - M)(\alpha I_t + (1 - \alpha)\mu_{t-1})$$

NB: only if you know the situation, perturbations of BG are legit.

3.2.5 Mixture of Gaussians

With most techniques new objects are sooner or later absorbed in the background. This is because there are changes that appear and disappear faster than the update rate, such as tree leaves, rain, snow, water of a fountain, etc. It can be that the same pixel represents at different time instants tree leaves and sky/building behind. The idea is to model the background as a mixture of Gaussians, so using **K** Gaussian instead of a single one.

$$p(x_t) = \sum_{i=1}^{K} w_{i,t}\mathcal{N}(x_t, \mu_{i,t}, \Sigma_{i,t})$$

Each of them is used to model a foreground or background object. Theoretical model is complex, we have multi-variate gaussians to model RGB or YUV. In practice we have that components are independent (matrix is diagonal), and variance of each component is the same (*NB: this means that collapse to* σ^2).
Let's take a deeper look... $w_[i,t]$ = weight for the current Gaussian, $\mu_{i,t}$ = mean for the current Gaussian, $\Sigma_{i,t}$ = variance for the current Gaussian. Select K, rank the Gaussians on the basis of weight, Standard deviation and Peak amplitude. Hint: the higher the peak and the more compact the distribution (low variance), the more pixel is likely to belong to the background $\Rightarrow$ strong evidence. Among the K Gaussians, some belongs to the background and some to the foreground. The first B Gaussians are associated to the BG if $\sum_{i=1}^{B} w_i > T$. Each incoming pixel is checked against the available models. In case no match is found, a new Gaussian is created. If a match is found (Es a match is found in the pixel is within $(x_t - \mu_{i,t}) < 2.5\sigma$), the model is updated. If the pixel is not matched with any model, the least probable Gaussian is replaced by the new one centered in x_t with low weight and high variance. Also here, update of the weights is necessary: $w_{k,t} = \alpha M_(k,t) + (1-\alpha)w_{k,t-1}$, where $M_(k,t)$ is 1 for the matching model and 0 otherwise (is it's not the matching model, the weight is decreased) and α is still the learning rate.

Chapter 4

Motion Tracking

Just to introduce the topic, we can say that motion tracking is the process of determining the movement of an object in a video sequence. More in general, it regards the understanding of WHAT is moving in a scene and HOW it moves/interacts with the environment. This is a very important topic in computer vision, and it is used in many applications, such as video surveillance, traffic monitoring, human-computer interaction and biological applications (cell tracking). At high level, motion tracking can be used for the discovery of activities, behavior understanding, and detection of threats. In this chapter, we will discuss the basic concepts of motion tracking, and we will present some of the most common algorithms used in this field.

4.1 Object Tracking

Object tracking is the process of locating a moving object over time. On the basis of the applications requirements, we can track the 2D coordinates (centroid), track in 3D (more cameras are required), or determine the position of complex objects (human body articulations). The main goal of object tracking is to estimate the state of the object at each frame, and to predict its future position. The applications of object tracking are numerous, and they include monitoring and surveillance, such as motion classification, identification of anomalous/suspicious behaviors, following a trajectory.

Or again, human machine interfaces, such as interacting with a device removing physical barriers (mouse, keyboard), natural language understanding; or even virtual reality, such as immersive presence, animation of virtual characters. Even mining and retrieval, such as browsing databases containing specific motion patterns.
Some benefits of object tracking are:

- In HCI, control PC (or systems in general) à no need for additional tools;
- In surveillance, Automated / Semi-automated systems à reduce the stress of human operators;
- Virtual reality, computer animation à animate and drive the avatar;
- But also: Training of athletes, Gait disorders detection, Medical applications, etc.

4.2 2D Tracking

Sometimes it is enough to track the object in 2D, for example when the object is moving on a plane. Exists different approaches to 2D tracking:

- **Region-based** ⇒ set of pixels that share similar features (color);
- **Contour-based** ⇒ determine position and shape of an object over time. Useful to track deformable objects;
- **Feature-based** ⇒ select meaningful points (contours, corners);
- **Template-based** ⇒ use specific models (hands, faces, eyes).

4.2.1 Region-based tracking

When it comes to real-time applications, tracking regions with uniform appearance offers several benefits. It enables swift processing, often exceeding 30 frames per second, while maintaining a commendable balance between quality and speed. The idea revolves around identifying areas in an image that exhibit consistent color characteristics. These regions are akin to patches of similar color projected onto the image plane, often achieved through segmentation techniques like background suppression.

One fundamental requirement for successful region tracking is ensuring that these regions possess distinguishable colors. However, this method faces challenges, particularly in scenarios with variable illumination. Changes in lighting conditions can destabilize the uniformity of color, complicating the tracking process. To mitigate this issue, various compensation techniques can be employed, such as utilizing the hue and saturation components of the HSV color space or normalizing the RGB values. While these techniques work reasonably well indoors, outdoor settings may present more difficulties due to unpredictable lighting changes.

In terms of what we aim to track, the possibilities are diverse. It could involve monitoring any moving object, distinguishing between skin and non-skin regions (useful for applications like hand and face tracking), identifying specific colored areas, and more. The approach typically involves techniques such as color thresholding for uniform regions or utilizing color histograms.

However, tracking regions with uniform appearance isn't without its challenges. Color variations over time due to changes in illumination or object posture pose significant hurdles. Additionally, models of tracked objects need continual updates to adapt to evolving conditions.

One conceivable approach to tackle these challenges involves breaking down the object into smaller regions. Each region is then associated with a color vector or histogram, representing the average color values within that region. During tracking, the color of each region is computed, and the similarity to a reference model is assessed. If the ratio between the reference and current values is close to 1, it indicates a good match.

Histograms serve as a valuable tool in this process. They provide a quantified representation of the color distribution within a region, enabling comparisons between reference and current models. Different similarity measures can be employed for evaluation, such as bin-by-bin comparison:

$$\cup(O_i^t, O_i^r) \sum_{n=1}^{U} min\{O_{i,n}^r, O_{i,n}^t\}$$

or Sum of Squared Differences (SSD):

$$SSD(O_i^t, O_i^r) = \sum_{n=1}^{U} (O_{i,n}^r - O_{i,n}^t)^2$$

Careful consideration must be given to the number of bins used in the histograms, striking a balance between granularity and computational efficiency.

In summary, tracking regions with uniform appearance offers a promising avenue for real-time applications, but it requires robust techniques to address challenges such as variable illumination and color changes over time. Techniques involving region division and histogram analysis can help in achieving accurate and efficient tracking in such scenarios.

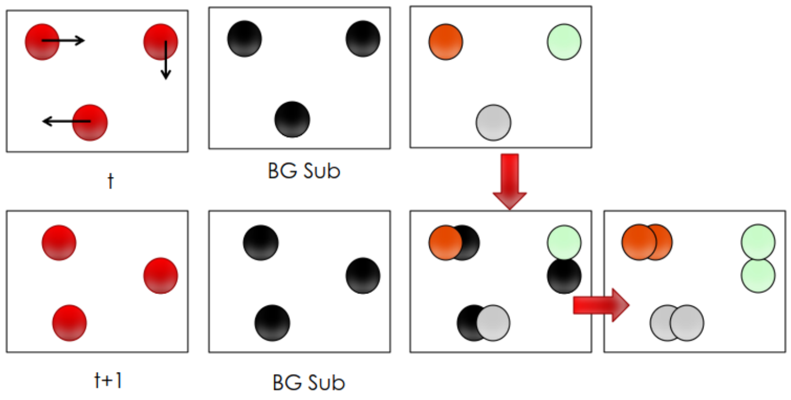
NB: Shadows can be a problem in region-based tracking because are source of noise $\Rightarrow$ false positive. A shadow does not correspond to the motion of a real object, but it is a change in the color of the object. More specifically it's a variation of the luminance while chrominance remains ideally unaltered. To solve this problem, and obtain a proper tracking, shadows should be removed before tracking using a suitable algorithm.

4.3 Blobs extraction

An object can be made up by several blobs (*NB:one for the torso, one for head...*), where blobs are aggregations of connected pixels that share common features. Talking about features, we can consider also position, so pixel with similar color but far from the object must be discarded. The typical application is in combination with a background suppression.

4.3.1 Target association

This is a procedure common to all trackers, not only region-based. In general it is worth noting that detection is not carried out on a frame-basis. This could lead to a lot of false positives, and it is computationally expensive. The idea is that once detected, targets are followed on a proximity basis. So, for example, background subtraction informs about the presence of motion, histograms characterize each moving objects, and then, unless occlusion occur, the target in the next frame should be the closest blob.



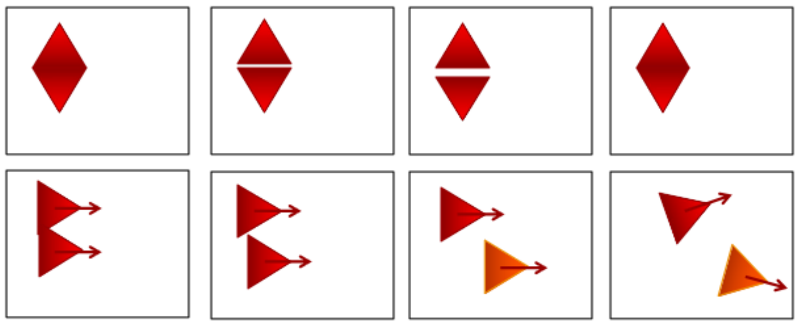
The steps of target association can be sum up as follows:

- Overlapping blob (issues with scale ⇒ depends on objects size);
- Centroid with minimum distance (as above);
- Overlapping bounding boxes (may fail due to perspective);
- Bounding box with minimum distance;
- Bounding box to centroid distance.

NB: When association has been completed, for each object or blob update the appearance model to account for small variations. In presence of occlusions, the last saved model can be used to disambiguate.

4.3.2 Splitting

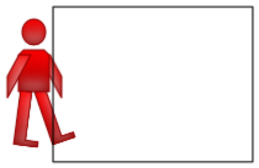
The splitting is a common problem in blob extraction, and it is due to the fact that the object is composed by several blobs. If objects are identified as single blobs, there are no problems. However, background subtraction may return ambiguous results. Indeed, objects can be split into several small blobs, so this means that there is not enough separation between BG and FG. Or again, two objects can enter the scene together and then separate.



It is important to accumulate evidence before saying if it is case 1 or 2. It's impossible to understand what's going on immediately, so it is necessary a temporal interval to tell if the object merged even though it's fragmented or if it's two objects that are close to each other.

4.4 Merging

Merging is the opposite of splitting, and it is due to the fact that two objects are identified as a single blob. This can happen when two objects are close to each other and move together consistently (then perhaps they're the same object). An example can be when a person's arm and foot enter the scene first and are detected as two well-separated FG blobs. Then also the rest of the body enters and a single blob is created.



4.4.1 Criteria for splitting and merging

Observation is the key. We need to monitor the regions of interest and evaluate consistency in terms of:

- Direction of motion;

- Distance between centroids/bounding boxes;
- Temporal range in which the phenomenon is observed;
- Velocity;
- Matching in features.

4.5 Occlusion

Occlusion is an anomalous, but very frequent, situation in tracking, and it occurs when moving objects overlap. Consequently, one or more objects disappear from the scene and bigger blobs appear as a result of the occlusion, with properties that don't belong to any of the models acquired previously, so the acquisition models are not reliable anymore.*NB: Model update should be avoided during occlusions.* In order to solve this problem, we need to re-associate "A to A" and "B to B" (where A and B are the objects that are occluded). *NB: Histograms are a good way out.*

4.6 Tracking: Feature-based

The objective is to retrieve the motion information of a set of features (such as points, corners, edges. . .) in the image.
Considering

$$A = \{A(0), A(1), A(2), \ldots, A(j-1)\}$$

as a set of images and

$$m_i(x_i, y_i) i = [0, j-1]$$

the position of the feature in the image plane in each frame. The objective is to determine the displacement vector

$$d_i = (dx_i, dy_i)$$

that best estimates the position of the feature in the next frame

$$m_{i+1}(x_{i+1}, y_{i+1}) = m_i + d_i$$

If needed, points can be grouped and objects can be represented using the bounding box or the convex hull.
TODO: inserire immaginina maybe
The question now is how to choose a good feature point that must has distinctive characteristic such as brightness, contrast, texture, edges, corners or point with high curvature.
The main idea is that for each candidate point, we compute:
TODO: inserisci matrice
Where J_x and J_y are the gradients evaluated on the point in x and y direction within W (nxn window). A good feature point is the one that maximizes the function, so the point where the smallest eigenvalue of Z is larger than a specific threshold. In practice, it highlights corner points and texture.
To sum up a little bit the important things:

- Eigenvalues should be above the image noise;
- Small eigenvalues imply strong similarity within the window;
- A large and a small eigenvalue means unidirectional patterns;
- If both eigenvalues are large, the point is of high interest (salt&pepper texture, corners).

For what concerns tracking them, we must ensure that the same points are tracked throughout the video sequence. Ideally we would expect that $A_i(Dm - d) = A_{i+1}(m)$, where A_i, A_{i+1} are the successive frames at time i and $i+1$, m is the 2D position of the feature point, D the deformation matrix (affine transformation model) and d is the displacement vector (translation models).
However, due to noise the equality does usually not hold. In addition, motion across

successive frames is assumed to be small $\Rightarrow$ a translational model is a good approximation. Feature dissimilarity measure is used to quantify the change of appearance between the first and current frame

$$\varepsilon = \iint_W (A_i(m-d) - A_{i+1}(m))^2 \, w(m) \, dm$$

where w is a weighting function such as Gaussian to emphasize the center of the window...
When the feature dissimilarity grows too large, the feature point should be abandoned and a new one should be selected. To minimize the residual we differentiate w.r.t unknowns (d)

$$e = 2 \iint_W (A_i(m) - A_{i+1}(m)) \, g(m) \, w(m) \, dm$$

and

$$g(m) = \begin{bmatrix} \frac{\partial(A_i(m) - A_{i+1}(m))}{\partial x} \\ \\ \frac{\partial(A_i(m) - A_{i+1}(m))}{\partial y} \end{bmatrix}$$

In this case the solution for the displacement vector can be expressed bu the 2x2 linear system of equations $Zd = e$.

4.6.1 The Lucas-Kanade optical flow

Substantially is a two-frame differential method for optical flow estimation. Consider $u = [u_x, u_y]$ in frame I and $v = [v_x, v_y]$ in frame J the goal is to find d that satisfies the equation $v = u + d$ such as I and J are similar(translation model). Because of the aperture problem, similarity must be defined in 2D. d is the vector that minimizes

$$\varepsilon(d) = \epsilon(d_x, d_y) = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} [I(x,y) - J(x+d_x, y+d_y)]^2$$

TODO:Da qui in poi è letteralmente copia incolla delle slide
We use an integration window to compute the displacement vector.

4.6.2 Pyramidal implementation

The two key components to any feature tracker are accuracy and robustness. Accuracy relates to the local sub-pixel accuracy attached to tracking ⇒ small integration window preferable to limit smoothness and preserve detail information (two image patches moving rapidly in different directions). Robustness relates to the sensitivity of tracking with respect to changes of light and big motions ⇒ a large window is preferable. ⇒ Pyramidal implementation *disegnino* Level 0 is the image at original resolution. Level 4, in the example, is the image at lowest resolution. The L-th level is defined as a linear combination of the elements in the previous level.
Conti da fare

At each level of the pyramid an initial guess g of the flow is computed at the lower level, which is then refined at the current level.

g is used to pre-translate the image patch in the second image J d should be small the information is then propagated at the upper level

overall the displacement becomes

4.6.3 Bayesian tracking

The basic idea is to estimate the state of a system over discrete time steps. At each step, we compute noisy measurements of the state and use them to update our estimate. The state of the system is represented by the x-y coordinates, velocity and acceleration along each dimension(6D vector). Important thing is that noise is typically smaller than the information about the state. The state can be defined as

$$x_k = f_k(x_{k-1} + w_{k-1})$$

Linking the measurement with the state vector we have

$$z_k = h_k x_k + v_k$$

where w_k is the process noise, v_k is the measurement noise, f_k and z_k are in general nonlinear functions defined in the state space. System and measurement should be available in probabilistic form. Every time the measurement is available, the estimation can be computed. It is an online method $\Rightarrow$ for every step k an estimate can be computed based on the previous observations z_k up to instant k. The initial probability distribution of the state vector is given $\Rightarrow$ $p(x_0|z_0)$. z_0 contains no measurement information, so it is the prior distribution. The goal is to compute the posterior distribution $p(x_k|z_k)$ at time k. The process consists of two steps:

- Prediction: $p(x_k|z_{k-1})$ is computed;
- Update: $p(x_k|z_k)$ is computed.

Let's consider the prediction step...
From the previous representation we have

$$p(x_k|z_{k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|z_{k-1})dx_{k-1}$$

This means that the posterior distribution at time $k-1$ is used to compute the prior distribution at time k, so it is propagated forward in time using the system model.
Using the Bayes theorem, it is possible to obtain the desired distribution

$$p(x_k|z_k) = \frac{p(z_k|x_k)p(x_k|z_{k-1})}{p(z_k|z_{k-1})}$$

where $p(z_k|x_k)$ is the likelihood function, $p(x_k|z_{k-1})$ is the prior distribution and $p(z_k|z_{k-1})$ is the evidence and is used for normalization and computed as

$$p(z_k|z_{k-1}) = \int p(z_k|x_k)p(x_k|z_{k-1})dx_k$$

4.6.4 Bayesian traking- toy example

TODO- tutto
sectionKalman filter The Kalman filter is a recursive algorithm that estimates the state of a linear dynamic system from a series of noisy measurements. In a nutshell we can say the following things:

- It is in line with the Bayesian tracking;

- It take a measurement;
- The measurement is subject to error;
- Derive the state of the system from the measurement.

We start from

Chapter 5

Geometry

Chapter 6

Local Feature Extraction

Chapter 7

Classification

Let's start this chapter saying what means implementing a classifier. This matches a variety of different pattern recognition problems that requires you to take decisions about what you are actually seeing in a picture. So basically, it is when you have to determine what the object present in the picture represents. And in order to do it you need to use the features of the appearance of the object and determine the category it belongs to. Also, in the previous chapter, we saw how pedestrian detector was working at that level. We just focus mostly on the feature extraction part and not really too much about the classifier itself, but what we did for the history of gradients was exactly to determine what were the silent elements that could characterize the shape of a human body. But there are tons of scenarios in which things like that are useful, some examples are:

- In a supermarket implement a system that recognizes vegetables;
- At the gate of your house, restrict the access to specific subjects or cars;
- At the entrance of a parking lot determine the category of the entering vehicle;
- On a robot, look for objects (doors, humans, obstacles).

The first element that we need to understand is actually what is a class. In a general representation a class is nothing but a set of properties, which can consist in colors, textures, patterns ...

NB: Objects or some parts of objects can be overall represented under the same umbrella over class.

A class is typically made up of a set of descriptions provided by observing a large number of examples and classes contain items that share similar properties Basically, goal of a classifier is to take an object as input and to output the class label it belongs to. So, we typically start from an input image and it can be taken as a whole or it can be segmented in advance. For example, if we want to detect the pedestrians in a scene, it could be an option to run a detector in a way that I can isolate the portion of the screen where things are moving, and I'm expecting that pedestrian are moving. So I can isolate some segment, so areas, which are potentially good for being a pedestrian (just a simplify example). Or again, I can do a segmentation of the picture filtering out the colors that do not mach the range of color of the skin. To sum up, segmentation is an option, a step that can be conducted and otherwise I'm proceeding with one of the most relevant steps in the classification, which is the extraction of features.

Extraction of features means that i'm looking to the history of the gradients which are being located in a certain portion of the segment of the picture itself. Basically we compare the picture vector with the feature vector that I used for the training phase, where the feature vector is the one that describes what I have extracted from the picture. So I feed this information into the machine that will start reasoning about what it sees as an input and in the end it would return an output. But here we are doing a process called recognition, which is an additional step forward to the detection.

7.1 The classification process

Just to wrap up a little bit, The classification process can be summarize in 4 steps:

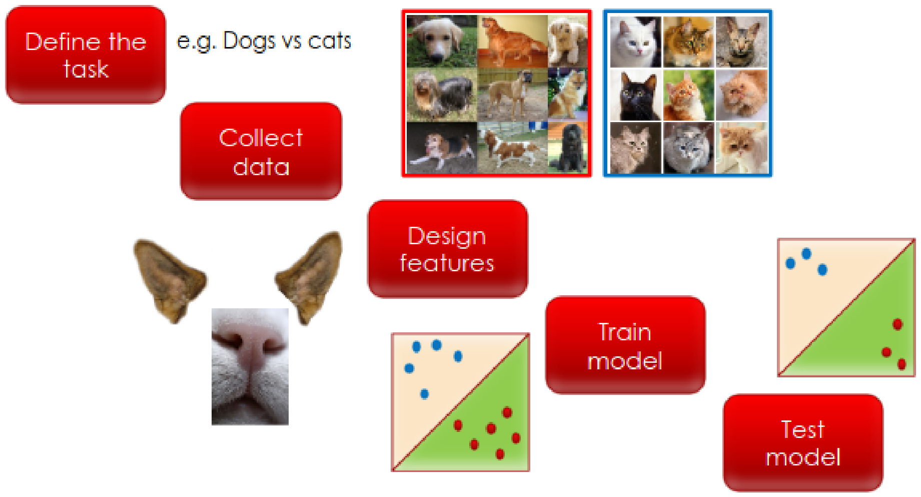
- Segment the image if necessary;
- Extract the features;
- Use the feature vector to feed a classifier;

- Output a label.

In order to make the concept simple let's take as example children. They learn new things by examples, so, to distinguish a dog from a cat they rely on observing samples of these two animals. We realize they have learned the concept by the time they see a new sample and they're able to recognize it. Basically, teaching a computer works the same way. Without entering in the realm of NN, let's take a generic recognition system. It requires a high number of examples in order to be able to distinguish across different classes. Indeed, the creation of a dataset is the most tough job, moreover the dataset need to be annotated (labelled).

7.1.1 Subject to failure

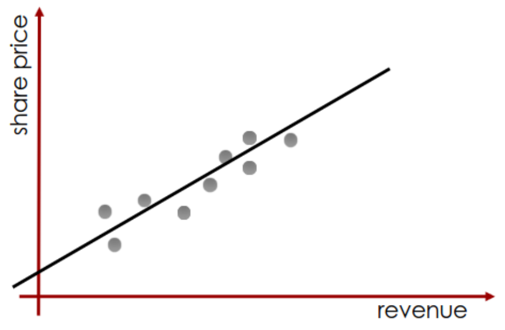
Machines may take days or weeks in order to learn concepts. However, they still tend to fail because of lack of data, wrong annotations, occlusions, perspective, similarity among classes ... Systems typically are not able to generalize, so for example, you have classifiers that are mounted on cars which are very robust in detecting cars, but only from a precise perspective. But what is the pipeline overall over classifier? So basically,



once the model is trained and has found the boundary that separates the two classes, if we put a cat image it falls on the right part, while if it is a dog one it falls on the left area. But if th points fall around the line it can be that we are making mistakes. In this case we would reduce the performance of our classifier. From this we obtain that one of the simplest problems in classification is regression.

7.2 Regression

To explain it we take an example as a use case. We are trying to predict the share price of a company that is about to go public. So, if we want to go public, what is the expected price? We need to have some examples in order to determine what is the shape, the inclination of this line in terms of slope and intercept. And we're doing it by looking at other companies similar to ours, at their revenue and the corresponding share price(features). In this way we create the training set that will be useful for us to define the parameters of the line, By looking at the line we have we could try to predict a new pair of coordinates and then compare with the real point. This process is the simplify version of the creation of a loss function. Meaning that we are trying through this loss function to understand how close we grt to our ground truth. More formally,



- Define loss function (our score to be minimized)
- Find the parameters of the line (slope, intercept)
- Use the line for testing

In our problem:

- We use the training to determine the line
- In testing, given the revenue we can predict the share price

- We have a set of training samples $D = \{Z_1, Z_2, \ldots, Z_n\}$;
- We have an unknown process $P(Z)$;
- And a loss function $L(f, Z)$ where f is the decision function;
- In supervised learning each example is a pair $Z(X, Y)$ where:
    - X contains the features for that sample;

- Y is the known/expected output;
- f takes X as an argument and outputs something in the range of Y

- Loss function defined as:

$$L(f,(X,Y)) = ||f(X) - Y||^2$$

Overall, a classifier can be schemed down in a very, very, very simplistic way to a function like that. So it tries to minimize the difference between the estimate and the corresponding ground truth. So far we were talking about binary classification but usually we have something which is a multi class detector and needs a more complex loss function.

It tends to not have very sharp boundaries and the function which is typically used is a lot like a conditional log-likelihood them. Very quickly, we are moving into the probabilistic domain, in which we say that $f_i(X)$ estimates $P(Y = i|X)$, where Y is a class (finite integer) and the loss function is the negative conditional log-likelihood. Of course we have the constraint that the entire sum of all these functions are mapped with one.

$$f_Y(X) >= 0$$

$$\sum_i f_i(X) = 1$$

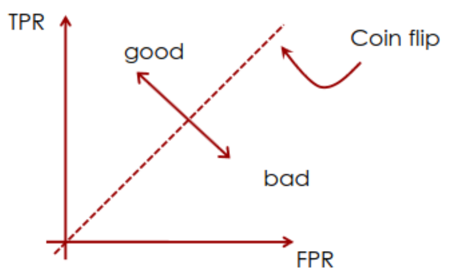
Another way for multi class classification is in the world of unsupervised learning, we are talking about clustering. In this case we do not provide any prior knowledge to the classifier about the input data. We feed samples into the algorithm and he progressively adjust the positioning of the different clusters and associates the different input in blind. Basically, through clustering the space is partitioned in regions centered around a prototype (centroid).

As an output you may have correct or incorrect detection, in a binary classification problem you can have True positive (hit), True negative, False positive or False negative (miss).

If we have a classifier for cat and dog and feed a bird into the system we obtain that the likelihood is extremely low, so we can implement a so-called reject option, in which the system prefers to just reject the examples and not give an answer.

7.2.1 The ROC curve, Precision and Recall

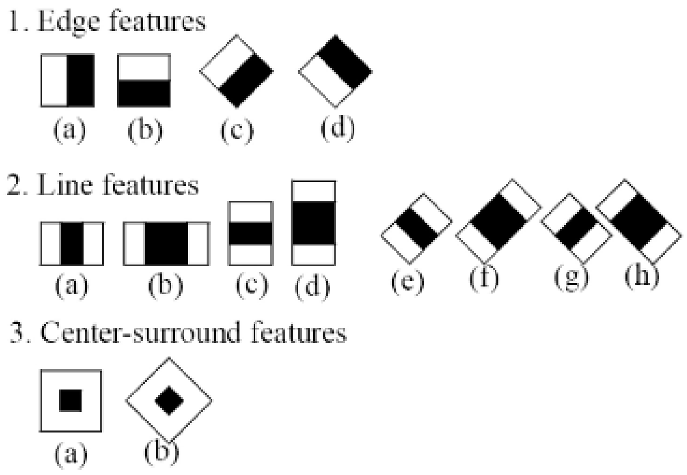
The ROC (Receiver Operating Characteristic) curve is a plot of the True Positive Rate vs the False Positive Rate. Each setup of the system is a point in the ROC space. We



basically would like to have something which stays above the coin flip. More precisely we want something that minimize as much as possible the contributions along the x axes and instead maximizes the contributions on the TPR axes.
(*NB: If the performance is on the line or under you better use a coin to classify.*) Given TRP and FPR we can define other two basic parameters which are the precision and the recall.
Precision: $TP/(TP+FP) = hit/(allretrieved) \Rightarrow$ probability that a randomly selected elements is relevant.
Recall: $TP/(TP+FN) = hit/(hit+miss) \Rightarrow$ probability that a randomly relevant element is retrieved in the search. If we extend the concept to more complex scenario where we have multiple classes, we can create a confusion matrix.

7.3 The face detection problem

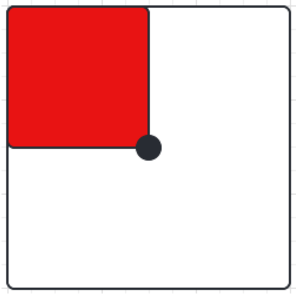
Once I have identified the significant features for the object that I'm looking for, I can take some samples and train the classifier. If I'm looking for faces the problem can be to find all faces (so binary detection), or find Bart-Lisa-Homer-Marge (recognition). The two scenarios are very different and involve the application of different algorithms. Let's talk about the Viola-Jones algorithm, that is probably the most widespread face detector also in terms of implementation because it is faster, quite robust and available in openCV. The goal was to implement a robust classifier using simple binary features. So the classifier is constructed using 14 Haar-like features: Basically, the sum of the



pixels within the white rectangles are subtracted from the sum of pixels in the black rectangles. Rectangle features can be computed easily using an intermediate representation for the image called the *integral image*.

$$ii(x, y) = \sum_{x'<=x, y'>=y} i(x', y')$$

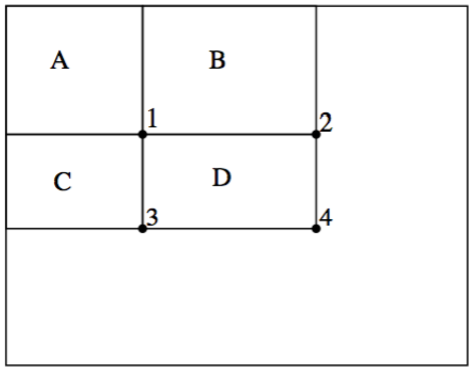
It is an intermediate representation of the image in which we are defining a single value $ii(x, y)$ that represents the contribution of the pixels before that location.



So, for example, if we want to compute the integral image at the black point, we just need to dum up the contribution of the pixels in the red area.

This makes the computation faster, if we start scanning the image from the top left

corner and we go to the bottom right corner, we can compute the integral image in a single pass. At any shift we can rely on the previous summation, add the last contribution and so on and so forth.
Remember that we are evaluating binary features, so we are looking at the difference between the sum of the pixels in the white area and the sum of the pixels in the black area.



So, looking at the image above, we can compute the sum of the values in D in the following way. Knowing that:

$$1 = A, \quad 2 = A + B, \quad 3 = A + C, \quad 4 = A + B + C + D$$

We obtain that the sum of the values in D is 0:

$$\begin{aligned} D &= 4 - (A + B + C) \\ D &= 4 - (2 + C) \\ D &= 4 - (2 + 3 - 1) \end{aligned} \tag{7.1}$$

7.3.1 Recursion

Using the following recursions, the integral image can be computed over the entire image in one single pass:

$$s(x, y) = s(x, y - 1) + i(x, y)$$

$$ii(x, y) = ii(x - 1, y) + s(x, y)$$

where s is the cumulative row sum.
I can basically rely on what has been computed up to the previous role and then add the last contribution.

Now we go into the detail of the classifier. In literature we have a lot of different classification algorithms. We have deep networks which are the most popular nowadays, but there is a time when there were other algorithms which are still fine. Usually the most popular algorithms are:

- SVM (Support Vector Machine);
- Artificial Neural Networks such as Multi Layer Perceptron or Radial Basis Function;
- Boosting algorithms such as AdaBoost;

The paper that professor is referring, Viola-Jones, is actually based on the AdaBoost algorithm (adaptive boosting classifier), so now let's see how it works.

7.3.2 AdaBoost

The AdaBoost algorithm is a boosting algorithm that combines multiple weak classifiers to create a strong classifier. They decided to create a final function f, that we recall it to be what we want to get in our classifier, as a linear combination of the weak classifiers h_i. The weak classifiers are trained sequentially, each new classifier is trained on the misclassified samples of the previous classifiers. So in terms of output, what we have is a sum of different contributions, where each contribution is weighted by a coefficient α_i. After defining the threshold, it will return either 1 or -1.

$$f(x) = \sum_{i=1}^{T} \alpha_i h_i(x)$$

where h_i is the weak classifier and $H(x) = sign(f(x))$ is the final strong classifier.

$$h_i(x) = \begin{cases} 1 & \text{if } f_i(x) threshold \\ -1 & \text{otherwise} \end{cases}$$

So, if it returns 1 it means that somehow the object is relevant, otherwise it means that I don't find this element as relevant.

$$f_i = Sum(r_{i,white} - Sum(r_{i,black}))$$

More formally, given a set of points $(x_i, y_i), i = 1 \dots m$, with $y = \pm 1$, and a set of weights all set to the same value $D_1(i) = 1/m$, what we do is to evaluate the features and try to find the configuration of the classifiers that minimizes the error. So, we try to minimize the sum of the weights of the misclassified samples.

$$h_t = argmin_{h_j \in H} \epsilon_j = \sum_{j=1}^{m} D_t(i) I(y_i \neq h_j(x_i))$$

where I is the indicator function(binary). If error is less than 0.5 we can use the classifier, otherwise we can discard it. Now we need to choose the weight of the classifier and from here we start to update progressively the system, we adjust the learning procedure for specific features inside a specific image.

$$\alpha_t = \frac{1}{2} log(\frac{1 - \epsilon_t}{\epsilon_t})$$

$$D_{t+1}(i) = \frac{D_t(i) e^{(-\alpha_t y_i h_t(x_i))}}{Z_t}$$

where Z_t is a normalization factor. *NB: Samples that are more difficult to classify will have higher weight in the next iteration.* As we say before, the final output is the summation of the different weak classifiers with the corresponding weight and the sign which basically tell us if the object is relevant or not from a graphical perspective.

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

Let's do a quick recap...

- In AdaBoost the combination of the weak classifiers improves the speed of the classification process and in the algorithm proposed by Viola and Jones, classifiers are used in cascade, further reducing the computational complexity;

- The goal is to quickly remove false negatives and focus on the positive samples.

Remember that in face detection, basic features can be used to exclude non-faces. The output of the first classifier is used to trigger the second one and at each stage the negatives are rejected. Let's make an example, if we have a face detector and we have a face in the image, the first classifier will check if he can find a symmetry in between the left and right part of the window, if it doesn't find it, it means that at least we have not a frontal face. Then it discards all the windows that he checked so that the second classifier will not waste time on them and has less tasks to perform.

- In the cascade, binary features are evaluated on training images of equal size (24x24pix);
- During the training stage, the boundaries of the classifiers are learned;
- During the testing phase, the learned classifiers are used to evaluate unknown samples;
- Images are not scaled at 24x24, so a multi-scale analysis must be conducted (at the feature level $\Rightarrow$ fast by considering the type of detector [binary + integral image]);
- In case of multiple detections at different scales, windows are averaged.

For what concerns the complexity we must say that it depends on the number of classifiers, the number of levels in the multi-scale analysis and the sampling distance between the windows.