Intensity Transformations (Module 3)

Image Histogram

An **histogram** represents the distribution of **graylevel intensities** in the image.

Contrast Enhancement

- **Linear Stretching**
  - Enhances the dynamic range by linearly stretching the original gray levels
- **Logarithmic Transformation**
  - $T(k) = c \cdot log(1 + k)$
  - useful to **enhance dark images**
- **Gamma Transformation**
  - $T(k) = c \cdot k^{\gamma}$
  - useful to:
    - $\gamma = 1$ **identity**
    - $\gamma < 1$ **brightening**
    - $\gamma > 1$ **darkening**

Histogram Equalization

Given a pdf p_x of an input image f, we want to apply an intensity transformation T such that the output image g has an uniform pdf p_y.

> $T = (L - 1)f_x$

but if the image has **bimodal**-like histogram, equalization may worsen the contrast.

In the discrete, the **histogram equalizing transformation** is:

$$T(k) = (L - 1) \sum^{k} n_j^{norm}$$

Achille Cannavale

Adaptive Histogram Equalization

It's an histogram equalization based on a histogram obtained from a portion of the image. One example is the **Contrast Limited AHE (CLAHE)** that limits contrast expansion in flat regions by clipping histogram values.

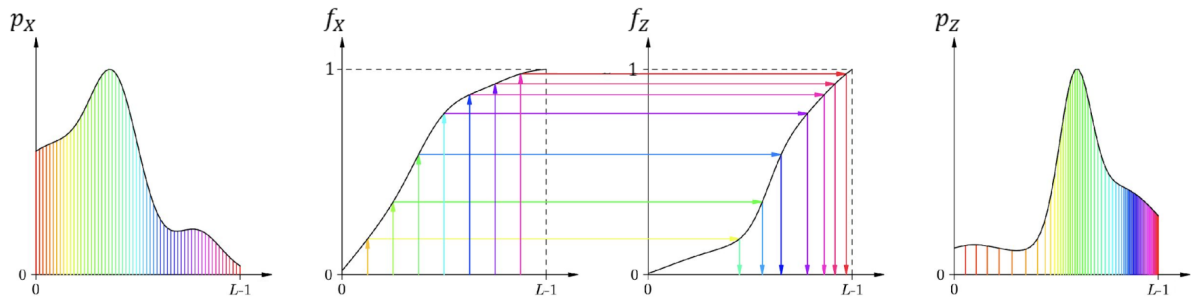
Histogram Specification

Starting from an image X we want to transform the histogram of this image to be like the histogram of a **target** image Z. This method is called **Histogram Matching**.

- Method: the zs values are calculated inverting the CDF of Z and applying such transform on top of the CDF-equalizing transform of X:

$$z(x) = f^{-1}[f_x(x)]$$

- Algorithm with discrete quantities:
  - calculate source and target CDFs $f_x(k)$ and $f_z(j)$ from their pdfs
  - for each k find the $f_z(j)$ closest to $f_x(k)$ and store the mapping in a look-up table $M(k)$
  - apply $M(k)$ to the source image



Spatial Filtering (Module 4)

A **spatial filtering** is a transformation described by a matrix called **filter**. If it can be defined as a **linear function of the neighborhood intensities**, it is called spatial linear filtering, otherwise it is a spatial non-linear filtering.

Linear Filtering as Convolution

$$I'(x,y) = I * F = \sum_i \sum_j I(x-i, y-j)F(i,j)$$

The **convolution** with F is equivalent to **correlation** with F flipped horizontally and vertically:

$$I * F = I \star \hat{F} = \sum_i \sum_j I(x+i, y+j)\hat{F}(i,j)$$

The convolution has some properties, like:

- **commutativity**
- **associativity**
- **linearity**
- **separability**
  - if $F = F1 * F2 * ... * Fn$
  - then $I * F = (...((I * F1) * F2) * ... * Fn)$
  - (split into simple kernel)

Filter Normalization

If the output of the linear filtering is an image, we want to keep the original gray level intensity range. In order to do that:

- all filter coefficients should **sum up to one**
- all filter coefficients should be **nonnegative**

Type of Linear Spatial Filtering

- **Average Filtering**
  - all filter elements are equal and the result is a blurred image
- **Gaussian Filtering**
  - like Average Filtering, but pixels are weighted according to a 2D Gaussian

Image Derivatives

First Derivative

We can calculate the first derivative in this way:

$$\Delta f(x) = \frac{1}{2}(f(x+1) - f(x-1))$$

In **convolution filters** way is like that:

$$H = 0.5 \cdot [-1\ 0\ 1]$$

Prewitt and Sobel filters

- **Prewitt** filters

$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \longrightarrow \qquad P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \downarrow$$

- **Sobel** filters

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \longrightarrow \qquad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \downarrow$$

Gradient

The **gradient** of an image is the** vector formed by its partial derivatives**. This vector **points toward the direction of maximum variation** and has **magnitude proportional to the rate of variation**.

Second Derivatives (Laplacian Operator)

The **second-order derivatives** are often combined into one single **isotropic operator**.

> But what means isotropic (rotation invariant)? => the second-order derivatives not depend on edge orientation

And the **Laplacian** is the simplest isotropic derivative operator (it has often negative values, so it have to be normalized).

which translated into linear convolutional kernel becomes:

$$L_{90^\circ} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Laplacian filter invariant to 90° rotations

$$L_{45^\circ} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Laplacian filter invariant to 45° rotations

Sharpening

To **sharp** an image we have to **subtract second derivative** from the image so that the slope of edge ramps are **exaggerated**.

$$sharp(I) = I - k \cdot \nabla^2 I$$

by applying the properties of convolution and $L_{45°}$, we get the following sharpening filter:

$$S_L = \begin{bmatrix} -k & -k & -k \\ -k & 1+8k & -k \\ -k & -k & -k \end{bmatrix}$$

Sharpening with Gaussian Filtering

- **Smooth** the original image with **Gaussian Filtering**
- Obtain the mask as the **difference** between the original image and its smoothed copy (we are isolating higher frequency)
- **Add** the mask to the original image

Nonlinear Filtering

Median Filtering

Replace each pixel by the **median** calculated within the filter region. This method **remove noise** but preserves the edges.

Noise Reduction

- **Salt and Pepper Noise**
  - Using an **Averaging** or a **Gaussian** filter is **not a good idea**
    - because new values will be created and **noise and information will be mixed**
  - We have to use a **Median Filter** instead
- **Gaussian Noise**
  - Gaussian Filter, but to achieve a better result we require more complex methods

Bilateral Filtering

Idea: Replaces the intensity of each pixel with a **Gaussian Weighted Average** of intensity values from nearby pixels, like the **Gaussian Filter**, but the weights depend not only on **spatial distances** between pixels, but also on the intensity difference between them.

There are two main parameters:

- σ_r that defines the **intensity extent**
  - the larger, the more different intensities will be mixed
- σ_s defines the **spatial extent**
  - the larger, the farther the pixels that will be mixed

Nonlocal Means Filtering

Idea: Replaces the intensity of each pixel with a **Gaussian Weighted Average** of intensity values from similar patches that are found in the image. So, the **weights depend on the similarity** between patches, not on their euclidean distance.

Template Matching

Idea: The correlation between the image I and a template T has its highest values in the region where I and T are equal or nearly equal.

This method works only if the two images have similar intensities.

Normalized Cross Correlation

To achieve the **invariance to intensity**, we have to subtract the **averages** from both template and image subregion and dividing by the corresponding **standard deviations**.

Binary Images (Module 5)

The process of generating a binary image from a grayscale image is called **binarization** and the **thresholding** is the most common technique.

Thresholding

- **Global Thresholding**: use a fixed threshold T
- **Region-based Thresholding:** T changes over the image
- **Local Thresholding**: T is calculated from a neighborhood of (x,y)

Global Thresholding

Simple

Calculate the **middle value** of the **histogram** to assign it to T.

OTSU

This algorithm checks every possible value of T and computes the variance between the intensities of the foreground's pixels and the background's pixels. The aim is to find a value of T that **maximizes** the **variance** between these **two classes** and **minimizes** the **inner variance** (it can be used also for N>2 classes).

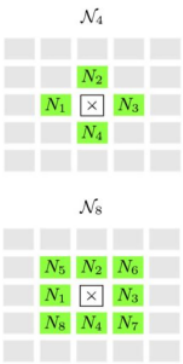
Triangle

The key concept of this method is to construct a **triangle** with a vertex on the peak of the histogram plot and the other two on the x-axis. The threshold value T is the **mean-point** where the distance from the triangle from the plot is **maximized**.

Adaptive Thresholding

- **Adaptive Mean Thresholding**: uses the **mean** to choose the right T
- **Adaptive Gaussian Thresholding:** uses the **local standard deviation** to choose the right T

Connected Components



Connected Component Labeling

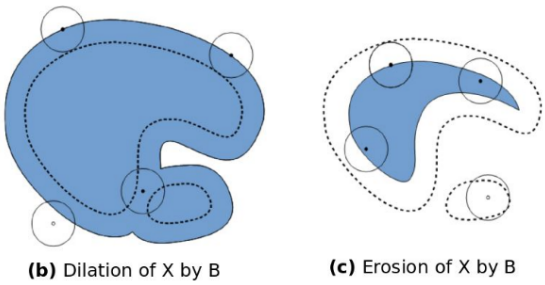
- For each foreground pixel, either assign the **smallest** label among its **top/top-left/left neighbors**, or assign a new label
- For each foreground pixel, assign the **smallest** label among its **bottom/top-right neighbors**

Mathematical Morphology

Morphological operations are defined with a set called **Structuring Element**.

Dilation and Erosion

- **Dilation**: for each point x of the plane, check if B centered on x intersect A
- **Erosion**: for each point x of the plane, check if B centered on x is contained in A

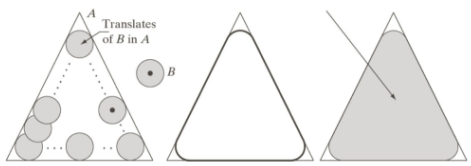


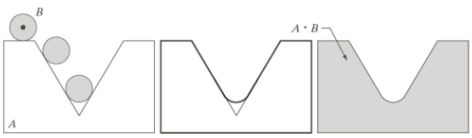
(b) Dilation of X by B **(c)** Erosion of X by B

Erosion and **dilation** are **dual operations**.

Opening and Closing

- **Opening**: eliminates **foreground** where the SE **can't fit**
  - **cuts protrusion**
  - erosion + dilation
- **Closing**: eliminates **background** where the SE **can't fit**
  - **fills holes**
  - dilation + erosion





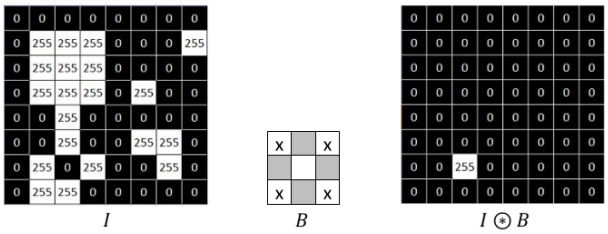
Hit or Miss

This method **detects all occurrences** of a given **(foreground, background)** pattern. A foreground-background SE can be made like this:

- $filled\ cell$: foreground to match
- $void\ cell$: background to match
- $cross$: don't care

Algorithm:

- **erode** the image A with B_1 (hit)
- **erode** A^c with B_2 (miss)
- **intersect** the two results



Thinning and Skeletonization

The **Thinning** of a set A through a SE B is defined as:

> Remove from A all the instances of B found by the hit-or-miss transform (the result depends from the orientation of B).

The **Skeletonization** can be achieved by iteratively thinning with a sequence of **oriented SEs**.

Hough Transformation

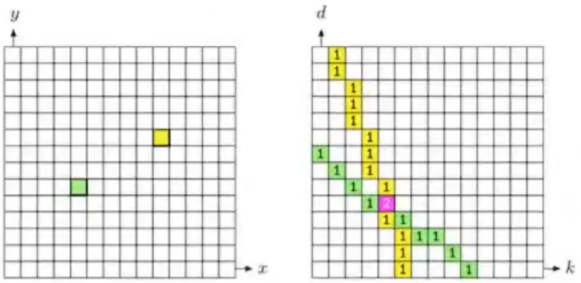
The **Hough Transform** is a tool used to detect **parametrizable curves** from **binary images**.

Starting from the simple case of a **line**, we can assume that if multiple points lie on the same line, they will share the parameters k and d. The idea of this method is to transition from the **space of points** in the binarized image to the **space of the parameters** k and d.

A point in the point space maps as a line in the parameter space.

The intersection of multiple lines in the parameter space corresponds to multiple points lying on the same line in the point space.

This relationship between the two spaces is stored in an **accumulation matrix**:



Where the lines can be ordered based on the number of intersections.

However, the problem with the classic line equation is that **vertical lines** have $k = \infty$. Therefore, it is necessary to switch to polar coordinates, which will generate in the parameter space.

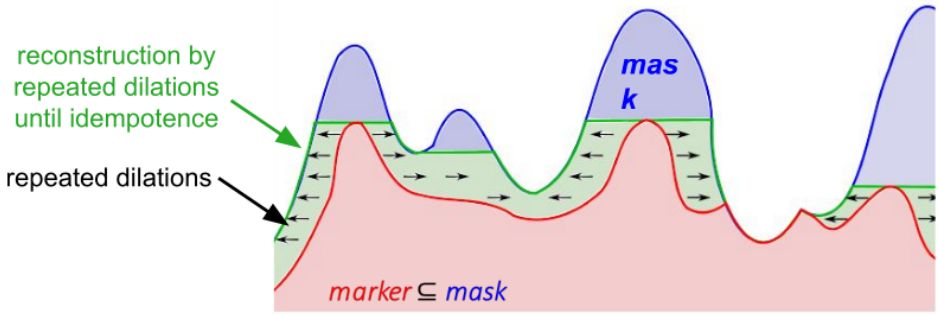
Grayscale Morphology (Module 6)

Top-hat and Bottom-hat transforms

- **Top-hat**
  - $f - (opening(f, b))$
    - the **opening removes structures brighter than surroundings and smaller than** b
    - the difference enhances these structures
- **Bottom-hat**
  - $closing(f, b) - f$
    - the **closing removes structures darker than surroundings and smaller than** b
    - the difference enhances these structures

Morphological Reconstruction

The key concept of the **Morphological Reconstruction** is to repeat dilations or erosion of an image, called **marker**, until the intensity profile of the marker fits under or above a second image, called **mask**.



> Maybe something more (VIDEO)

Segmentation (Module 7)

Edge-based Segmentation

In real images, **ramp edges** are the most common edges and we can handle them with:

- the **first derivative to detect an edge**
- the **sign of the second derivative**, that tells if the pixel is on the dark or on the light side of the edge
    - however, even a tiny amount of noise strongly affects the second derivative.
    - since that, three fundamental steps have to be performed in edge detection:
        - smoothing for noise reduction
        - detection of edge points
        - edge localization

Gradient-Based Edge Detection

The **gradient of an image** allows to find **edge strength and direction**. In particular the direction of the edge is orthogonal to the direction of the gradient.

The algorithm works in this way:

- Apply **gaussian smoothing** for noise reduction
- Calculate the **gradient** using (for ex) **Sobel**
- **Threshold** the gradient magnitude using a certain percentage of its max
- post-processing:
    - **edge thinning**
    - **edge linking**

Canny Edge Detection

The **Canny's** method has this model:

- **edge model**:
  - **ramp edge corrupted by additive white gaussian noise**
- **edge detector**:
  - **convolutional filter** with **antisymmetric response**, that can be well approximated by the **derivative of a Gaussian**
  - but we can **split** these things:
    - **gaussian smoothing**
    - **derivative with sobel kernel**

Algorithm

- **Filter** the image with an $n \times n$ **Gaussian Filter** (choose 6σ)
- Compute the **Gradient of the image**
- Apply **Non-Maxima Suppression** to get rid of **spurious response** to edge detection
- **Robust detection** and **linking of the edge** points based on **hysteresis thresholding**

Non-Maxima Suppression

The **gradient vector direction** is approximated to **one of the four principal directions**. If at least one of the neighboring pixels along the edge direction has larger gradient magnitude, the corresponding pixel in the gradient image is set to 0 (suppression), otherwise it is set to the gradient value.

Hysteresis Thresholding and Edge Linking

(this step is mean for delete false edges)

Let be g the **gradient image** after **non-maxima suppression**. The **hysteresis thresholding** consists in:

- let be T_h and T_L two **thresholds** so that:
  - $T_L < T_H$
  - $g_L = T_L \leq g \leq T_H$ (**candidate** edges)

    - $g_H = g \geq T_H$ (edges for **sure**)

- **Edge Linking** (to link the holes)
    - for every point in g_H, mark as valid all the points in g_l that are 8-connected to the considered points, until convergence.

Region-Based Segmentation

Region Growing

Based on **controlled growing** of initial pixels named as **seeds**.

Seeds are manually or automatically selected. Regions are grown from the seeds to connected points depending on a **region membership criterion**.

Algorithm

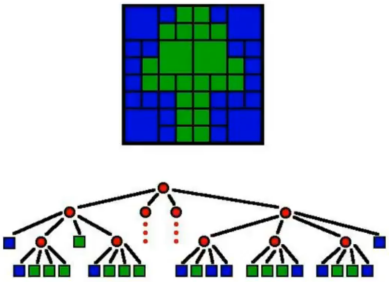
Given:

- f, the **image to be segmented**
- S, binary image with the **seeds**
- Q, **predicate** to be tested for each location
    - workflow:
        - do the **dilation** to get the **seeds neighbor** pixel
        - compute the **difference** with the **seeds image**, so we get an image only with **predicates**
        - with some **criteria** accept some **predicates**
        - continue until **convergence**

Region Splitting and Merging

This method consists into two main phases, **Splitting** and **Merging**. The splitting splits the image into **homogeneous quadrants**, recursively until all subregions respect the **predicate condition**.

At this point we have to **merge adjacent regions** such that their **union** respect the predicate condition again. Repeat until convergence.



Watershed

This algorithm is a **region growing approach** on a topological interpretation of the **gradient image**.

The **input** of this method is the **profile of the gradient of the image**.

We start with **seeds**, that are the **local minima** of the profile of the gradient image, that are manually or automatically determined.

Then we have to flood with **integer flood** increments from the **min value** to the **max value + 1**.

All the pixels **below the water level** are marked with "white". At each iteration we compare each **connected component at the current iteration** with the **connected component at the previous iteration**.

- if the connected component at the current iteration don't overlap any other connected components at the previous iteration, it's a **new region**
- if it overlap one of the connected components at the previous iteration, is a **growing region**.
- if the current intersect with more than one previous component, it means that two regions are touching, so -> **dam construction**

A possible way to construct the dams is to do a dilation before the touch, and do a subtraction to get the borders of the regions to set on a max value.

Marker-Controlled Watershed

To avoid **oversegmentation** we can use **Marker-Controlled Wateshed**. The idea is to **not select all local minima.**

So we can **choose some internal markers** (foreground) associated to the object, and some **externals markers** (background).

Statistical-based Segmentation

Mean Shift

The **mean shift filtering** replaces each pixel with the **mean** of the pixels in a neighborhood defined in a **joint-spatial-color domain**, to **cluster** all regions with

similar intensities.

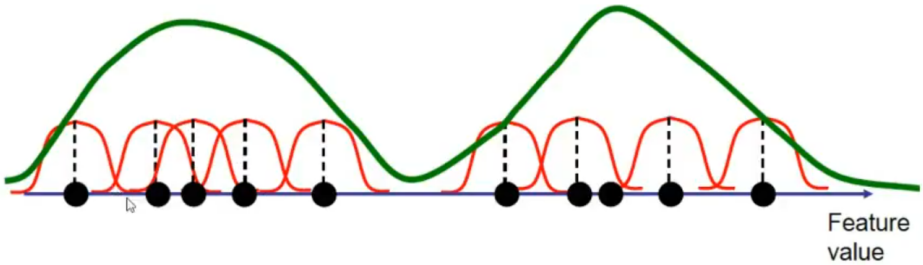
Explanation

Consider a set of points in 1-D feature space:

We want to generate two groups, corresponding to the two parts of the feature space in which we have a high density of points.

The "high density" notion is captured with the **Kernel Density Estimation**.

We can define a **kernel** at each data point x_i and sum up the result into a single function:



Attention: To avoid **overfitting**, a **smoothing parameter** h is used to scale the kernel, to make it **not too thick** and **not too fat**.

But, which kernel we have to use? One that satisfies the **radially symmetric rule**:

$$K(\mathbf{x}) = c \cdot k(||\mathbf{x}||^2)$$

where $k(t)$ is called **kernel profile**.

One of the most used kernel is the **Epanechinkov** kernel, with this kernel profile:

$$k(t) = 1 - t, \quad t \in [0, 1]$$

Now we can can "**climb the cliffs**" to find local peaks of the PDF, **following the gradien**t, in such a way that we are **shifting all samples into the direction of gradient vector.**

Algorithm

- let be y the **data point we want to move**
- $y_0 = y$ the **initial location** of y
- at each iteration j, **move** y_j by the corresponding **mean shift vector** (that points toward the direction of the gradient vector):

$$y_{j+1} = y_j + m(y_j)$$

- **until convergence**

Another way to view this method is with the **mean window shifting interpretation ** (moving all points toward their local mean, in a window of radius h).

But which feature domain we have to consider?

- **color**?
  - no, distant pixels having similar colors will converge to the same color
- **spatial**?
  - no, pixels close to each other will converge to the same color
- solution: **joint spatial-color domain**
  - but location and color have **different scales**!
    - we can **split the kernel** into two kernels with **two different bandwidth parameters** h_s and h_r

K-Means Segmentation

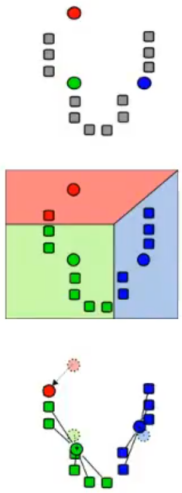
The **k-means method**, works knowing in advance the number of clusters.

The key concept is to **find a set of Clusters that minimizes the differences between samples and means**.

> Difference with the mean shift:
>
> The mean shift was to move the samples toward the local mean, iteratively. In the k-means, the samples aren't moving, instead they are assigned to the cluster with the nearest mean, and the means are iteratively refined until convergence.

Algorithm

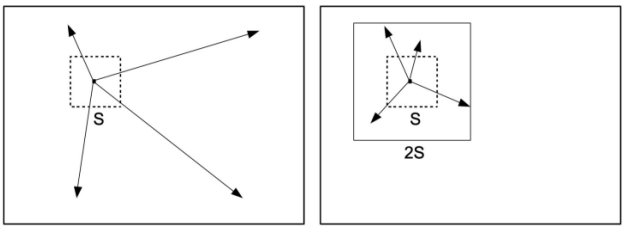
- Initialize the **means**
- **Assign** samples to clusters
  - whose mean is the closest
- **Update** the cluster centers (**means**)
- repeat until **convergence**



Superpixels

The idea of this method is to use **k-means** to perform **local clustering**.

We define a **region** size S and the algorithm will compute the distances from each cluster center (**means**) to pixels into a search space of $2S \times 2S$.



Now the **cluster centers** will be **updated recursively** until convergence.

But...which **distance measure rule** we have to adopt? We are in a **color+spatial space!!**

Euclidean distance?

- for large superpixels, spatial distance may outweigh color distance
- for small superpixels. color distances may outweigh spatial distances

Solution: **normalize** spatial and color distances and combine them!!!! (vabbè anche meno)

Graph Cuts

The image is represented with a **weighted graph** with:

- **vertices** as **pixels**
- **edges** formed between **adjacent pixels**

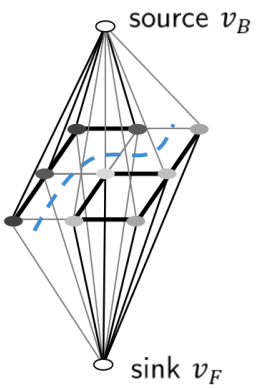
    - the **weights** on the edges are proportional to the **similarity between the two nodes**

The goal is to **partition the graph into 2 subsets,** one for **background** and one for **foreground**, such that:

- the similarity among the nodes within a subset is **high**
- the similarity across the nodes of different subsets is **low**

And this is **equivalent** to find a **minimum cut of the graph by removing the edges connecting the two subgraphs.**

In graph theory, **minimum cuts are equivalent to max-flow problems** where the goal is to **maximize the flow between a source and a sink node**. **Source** and **Sink**, in our case represent **background** and **foreground**.



> A cut C is a set of edges that, if removed, would disconnect the sink from the source.

But we are interested to find the **minimum cut.**

Shrinking Bias

The **minimum cut** will **cut small sets of isolated points **characterized by a very low similarity with neighboring regions.

Solution: **Interactive Graph Cuts** -> ask the user to provide **seeds** to force high weights in the corresponding terminal edges.

N-links

N-links weights are chosen proportional to the **intensity similarity** and to the **inverse distance** the two nodes.

T-links BG

T-links between a pixel node and ν_B are proportional to the **probability of that node to belong to the background**, taking into account **user-provided seeds**.

T-links FG

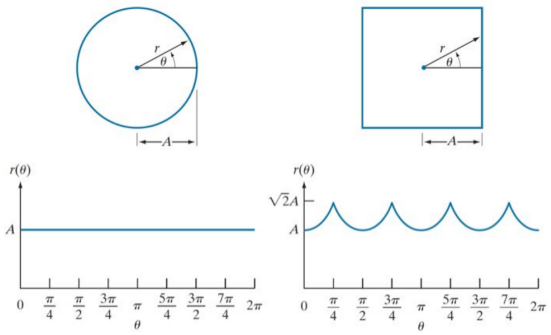
T-links between a pixel node and ν_f are proportional to the **probability of the node to belong to the foreground**, taking into account **user-provided seeds**.

> Maybe something more on energy minimization???

Descriptors (Module 8)

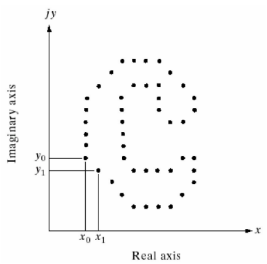
Signatures

A **signature** is a 1-D **representation of a boundary**, with a function of angle θ:



- **invariant to translation**
- **not invariant to rotation**
  - **select a distinct starting point**
- **not invariant to scale**
  - **rescale and normalize**

Fourier Descriptors



A shape can be represented with K **contour points**, or equivalently as a **discrete signal of 2 variables**:

$$s(k) = [x(k), y(k)]$$

We can treat each coordinate pair as a **complex number**:

$$s(k) = x(k) + iy(k)$$

This is a **discrete**, **periodic**, **1D complex signal**, so we can apply the **Discrete Fourier Transform**. This transformation can give us the **complex coefficients** (Fourier descriptors) $a(u)$.

We can also apply the **inverse Fourier transform** of these coefficients to restore $s(k)$.

So, the idea is to take the first $P < K$ coefficients and reconstruct $\hat{s}(k)$. In this way we can get **rid of high-frequency components** that capture the **finer details**, and keep the **low-frequency components**, that capture **general shape properties.**

Fourier descriptors are **not directly invariant to transformations**, but can be made invariant:

- **translation by** δ
  - just **ignore** $a(0)$
- **rotation** by θ
  - **estimate** θ and **divide** by $e^{i\theta}$
- **scaling** by α
  - can apply **min-max normalization**
- **starting point**
  - the **magnitude** part of $a(u)$ is **invariant to the starting point**

Shape Moments

We have to define the **Geometric Shape Moments**, that is the **projection of the contour of the image** $I(x, y)$ **to the basis** $x^p y^q$**:**

$$m_{pq} = \sum_x \sum_y x^p y^q f(x, y)$$

where $f(x, y) = 1$ if $(x, y) \in$ the shape.

This formula has nice characteristics, like:

- $m_{00} = \sum_x \sum_y f(x, y)$ that is the **area** of the shape
- $m_{10} = \sum_x \sum_y x f(x, y)$ **sum** of **x** coordinates
- $m_{01} = \sum_x \sum_y y f(x, y)$ **sum** of **y** coordinates

- Combination of 0^{th} and 1^{th} order moment give us the **centroid coordinates**
  - $x_c = \frac{m_{10}}{m_{00}}\ xy_c = \frac{m_{01}}{m_{00}}$

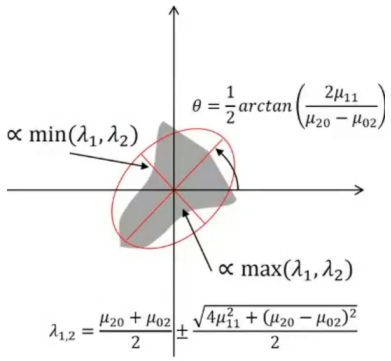
Translation Invariance

We can achieve the **Translation Invariance** by **moving the origin** of the reference system to the **centroid position** (so we get the **Central Moment** μ_{pq}):

$$\mu_{pq} = \sum_x \sum_y (x - x_c)^p (y - y_c)^q f(x, y)$$

Second Order Moments

The **second order moments** give the **Ellipsoid of Inertia**, that is a shape that has the same **area**, **orientation** and **eccentricity** of the object, and is centered on its **centroid**.



In particular given the **covariance** matrix:

$$\begin{bmatrix} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{bmatrix}$$

We can use its **eigenvector** that correspond to the **major** and **minor** axes.

Its **eigenvalues** are proportional to the **squared length** of the eigenvector axes.

And the orientation θ can be extracted from the **angle** of the eigenvector associated with the **largest eigenvalue**.

Scale Invariance

Let be $f'(x, y) = f(x/\lambda, y/\lambda)$ a new image scaled by λ. We can define a **scale-invariant central moment** by calculating the **central moment of the scaled image** (μ'_{pq}) and then set $\mu'_{00} = 1$ (unit area):

$$\mu'_{pq} = \lambda^{(p+q+2)} \mu_{pq}$$

$$\mu'_{00} = \lambda^2 \mu_{00} = 1 \longrightarrow (\mu_{00})^{-1/2}$$

So we can define the **Central Normalized Moment** η_{pq} as:

$$\eta_{pq} = \mu'_{pq}|_{\text{unit area}}$$

Rotation Invariance

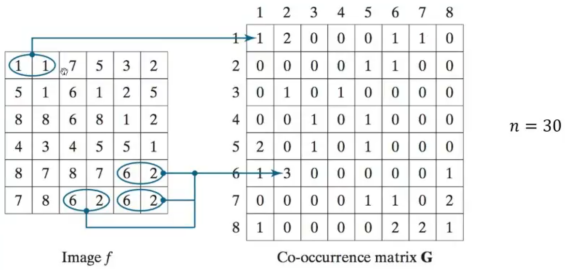
The basis $x^p y^q$ **doesn't have simple rotation properties**, so there are the **Hu moments** that are combination of **Central Normalized Moments**, that can be used to build up a basis with **Rotation invariance.**

GLCM (Gray Level Co-occurence Matrix)

The purpose of the **GLCM** embeds into a **matrix** all the information related to **intensities** and **spatial relationship**.

The first thing to do is to define a **position operator** Q (one pixel immediately to the right).

Now we define a matrix $G_{L\times L}$ whose element g_{ij} is the number of times that pixel pairs (z_i, z_j) occur in the image in the position specified by Q.



where n is the **sum** of the elements of G.

In particular:

$$p_{ij} = \frac{g_{ij}}{n}$$

is the probability that a pair of points satisfying Q have values (z_i, z_j).

In general $\frac{G}{n}$ is called **Normalized Gray Level co-occurrence matrix.**

An issue is that G is very **very big**!! So we can solve this problem by **grouping similar intensities together** in => $G_{K\times K}$ with $K \leq L$.

Correlation

$$\sum_{i}^{K}\sum_{j}^{K}\frac{(i-m_r)(j-m_c)p_{ij}}{\sigma_r\sigma_c}$$

This formula measures **how correlated a pixel is to its neighbor**:

- **random** textures have almost **0** correlation
- **regular** texture have almost perfect correlation (± 1)
- **textures** with a structure have **high absolute correlation**

-0.0005 0.9650 0.8798

Contrast

$$\sum_{i}^{K}\sum_{j}^{K}(i-j)^2p_{ij}$$

This formula measures the **contrast between a pixel and its neighbor.**

The less random an image is, the lower its contrast tends to be.

10838 00570 01356

Uniformity

$$\sum_{i}^{K}\sum_{j}^{K}p_{ij}^2$$

This formula measures **uniformity in the range [0,1]**

- **1** for **homogeneous** image
- **0** for **random** image

0.00002 0.01230 0.000480

Homogeneity

$$\sum_{i}^{K}\sum_{j}^{K}\frac{p_{ij}}{1+|i-j|}$$

This formula measures **the density of G in terms of its diagonal**

- **1** for diagonal G **(images with slow variations)**
- **0** for perfectly **sparse** G

0.0366 0.0824 0.2048

Entropy

$$-\sum_{i}^{K}\sum_{j}^{K}p_{ij}log_2(p_{ij})$$

This formula measures **the randomness of the element of** G

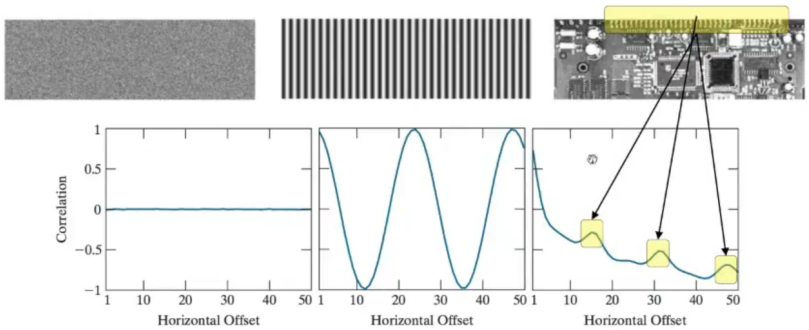
- **0** when the image is **flat** (constant intensity)
- **max** for **uniformly distributed** p_{ij}

(directly proportional to the information content)

15.75 6.43 13.58

GLCM-based signatures

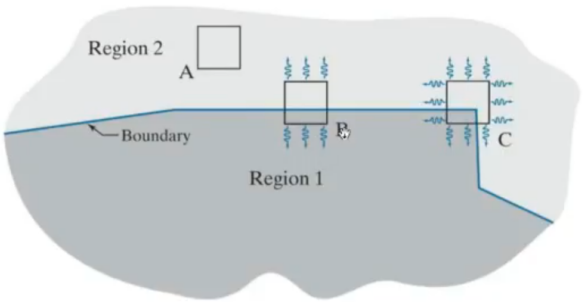
A signature with a **GLCM** method is a **plot of correlation changing the distance of adjacent pixels:**



Harris-Stephens Corner Detector

The basic idea to **detect a corne**r is to image a **sliding rectangular window** (patch).

- if the window is completely **outside** or completely **inside** our object, every movement of the window doesn't make changes
- if the window is on a **edge**, only movements toward one direction will make changes
- if the window is on a **corner**, changes will occur in all directions



Mathematically we can quantify the change of intensity for the shift (u, v) of a patch $\mathcal{P}$ by:

$$C_{\mathcal{P}}(u, v) = \sum_{(x,y)\in\mathcal{P}} w(x, y)[\text{shifted patch intensity} - \text{patch intensity}]^2$$

The **weighting function** can be a **box function** or a **Gaussian**.

Now, with very small shifts, we can use the **Taylor Expansion** to get:

$$C_{\mathcal{P}}(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} H \begin{bmatrix} u \\ v \end{bmatrix}$$

Where H is called **Harris Matrix**, that is the **weighted sum** of **products of gradient components calculated for all the points in the patch** $\mathcal{P}$:

$$H_{\mathcal{P}} = \sum_{(x,y)\in\mathcal{P}} w(x,y) \underbrace{\begin{bmatrix} f_x^2 & f_x f_y \\ f_x f_y & f_y^2 \end{bmatrix}}_{\text{Gradient Covariance Matrix}}$$

The matrix H is **symmetric** since the **weighted function** is **isotropic** and the **GCM** is **symmetric**.

> We know that, the eigenvectors of a real symmetric matrix point in the direction of maximum data spread, and the corresponding eigenvalues are proportional to the amount of data spread in the direction of the eigenvectors.

in our case:

- **eigenvectors** = **major axes** of an ellipse fitting the gradient data points (f_x, f_y)
- **eigenvalues** = **distances** from the center of the ellipse to the points where it intersects the major axes.

> A corner generate eigenvalues large and similar.

So now we can construct a **Corner Detector**:

Let introduce the measure R defined as:

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

- large positive values (**corner**)
- large negative values (**edge**)
- small values (**flat region**)

ISSUE: The eigenvalues are expensive to compute

SOLUTION: the **trace** of a square matrix is equal to the sum of its eigenvalues and the **determinant** is equal to the product of its eigenvalues:

$$R = det(H) - k \cdot trace^2(H)$$

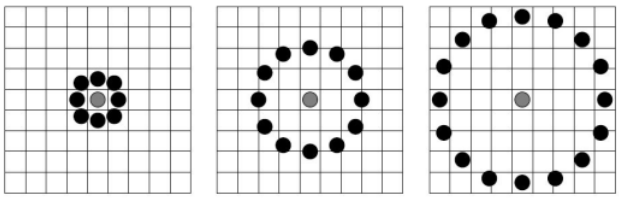
Algorithm

- Calculate one for all, **horizontal** and **vertical** components of the **gradient** and $f_x^2, f_y^2, f_x f_y$
- For each pixel (x, y) in the image, calculate the **corner response** R in a neighborhood $M \times M$ given by user
- Find the **local maxima** in R above a certain **threshold**

Local Binary Patterns

What is a local texture? A mathematical definition is $T_{P,R}$ centered on g_c, a pixel in a grayscale image, that is the joint pdf of the neighbors of g_c at distance F

$$T_{P,R}(g_c) \overset{\text{def}}{=} t(g_c, g_0, g_1, \dots, g_{P-1})$$



P=8, R=1.0 P=12, R=2.5 P=16, R=4.0

Let's do some approximations:

- **subtract** g_c from its neighbors
- we can assume that the **differences** $g_i - g_c$ are independent from g_c
- so we can **ignore** $t(g_c)$
- in order to remove dependency from the grayscale used, we consider only the **signs** of the differences

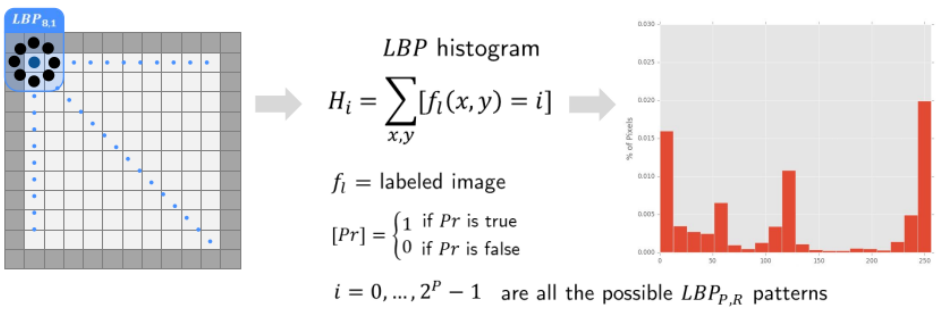
Meaningful interpretation:

Signs can be represented as **binary digits** (1 for positive and 0 for negative), that can be converted in a **decimal number** after choosing an appropriate convention for the starting point and direction used:

$$LBP_{P,R}(g_c) = \sum_{i=0}^{P-1} s(g_i - g_c)2^i$$

01100000 → 96

So the feature vector of **Local Binary Pattern** is the **normalized histogram** of the occurrences of a chosen **binary pattern operator** that is used to label each pixel with a **decimal number**.



Rotation Invariant

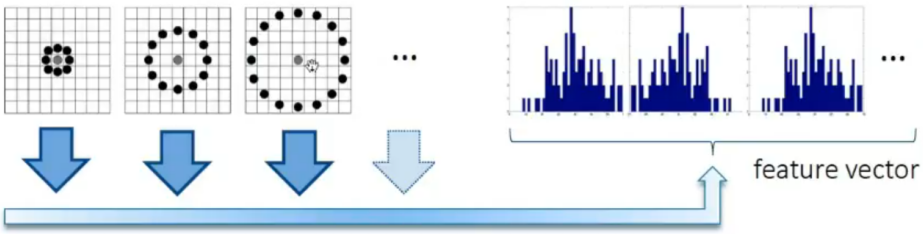
Achille Cannavale

LBPs are **not rotation invariant**. To achieve this property we can **group all the rotated LBP with the same pattern and assign to it the label of the minimum.**

Another simplification is to label all **LBP** that have at most two consecutive bit transition. Because, some **empirical studies**, show that this subset of LBP codes alone achieves 90% of the occurrences.

Multireosolution LBP

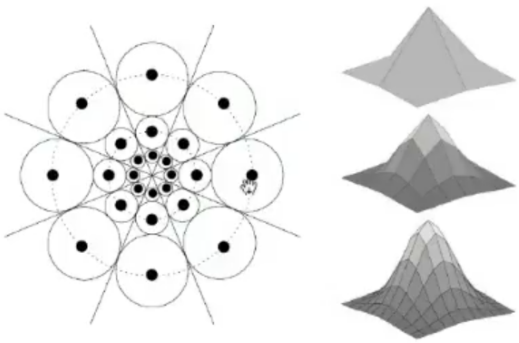
We can also use more than one **LBP**, assuming that all of them are **statistically independent** and **concatenating** all the histograms:



But there are some problems, like **noise** and **aliasing**!!!

Solution: **Use LBP Filtering**

Where pixels are replaced with the weighted sum of their neighborhood with a **Gaussian filter:**



HOG (Histogram of Oriented Gradients)

The idea under **HOG** method is to characterize objects with their **local distributions of edges.**

In particular, we have to use a **sliding window** over all image.

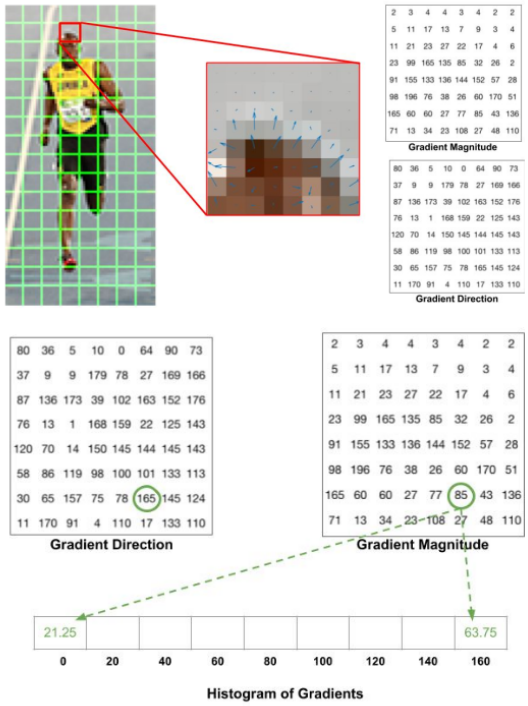
- **Step 1**

Compute the **horizontal** and **vertical derivatives** with **1D derivative kernels.**

- **Step 2**

The window is **partitioned into small cells** and for each of them, **gradient magnitudes** are accumulated to their **orientation bin**, that are usually 9 bins in [0, 180], so it's **invariant to noise**.

Voting at the **edges bins** are splitted proportionally:



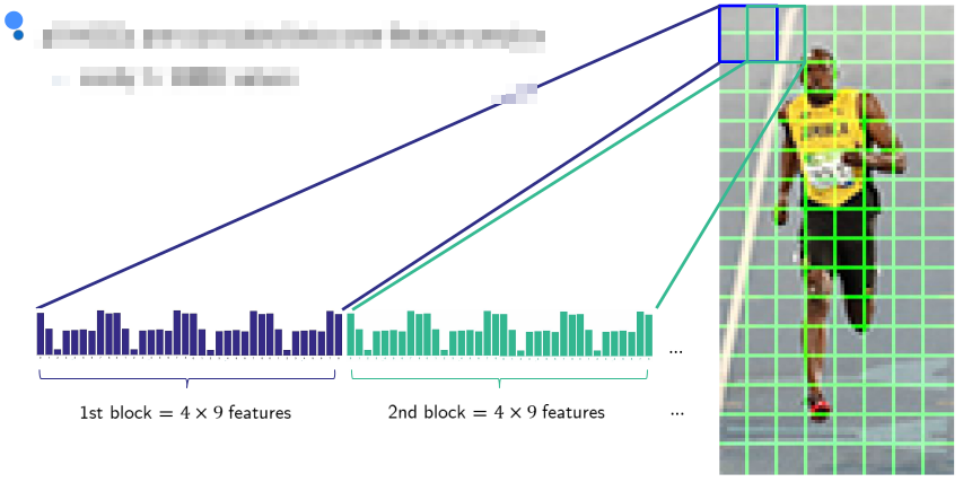
Now each cell is characterized by a **[1xnbins] HOG feature vectors**, but the problem is that **gradient magnitudes** that contributed to the **weighted voting** depend on **local illumination**, so we have to **normalize**!!

- **Step 3**

Cells are now grouped into **blocks** (2 or 8 cells for group) and each block will be **normalized** with a **L2-normalization** followed by a **clipping**, on the concatenated feature vector.

- **Step 4**

All **HOGs** are **concatenated** into **one feature vector**



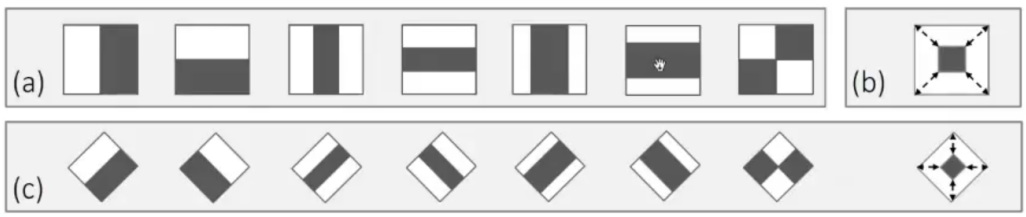
> Maybe something more

Haar-like Features

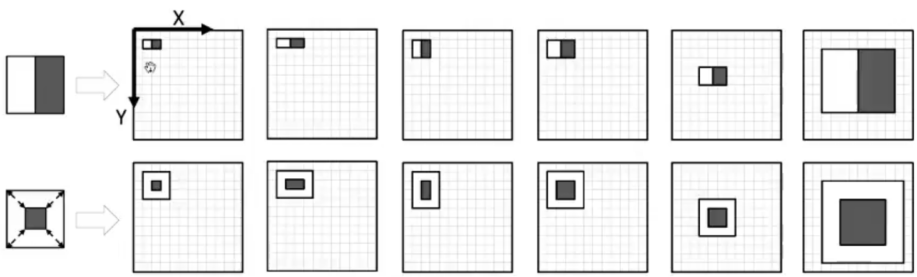
This feature are made by **adjacent rectangular regions** at a a specific location in a **detection window**. Then **sums up the pixel intensities** in each region and calculates the **difference** between these sums and will be used to categorize subsections of an image.

Haar prototypes

- three groups
  a) edge and line features
  b) center-surround features
  c) tilted features



After viewed what are **Haar prototypes**, we can define a **Haar Feature** set as each **prototype shifted and scaled across all possible combinations on a detected window:**



Integral Image

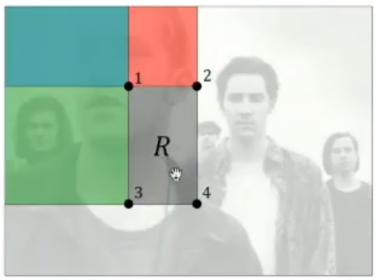
The **integral image** of $i(x, y)$ is:

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

In short, the **integral image** at (x, y) is the **sum of the intensities** within the **rectangle** that start at $(0, 0)$ and ends at (x, y)

And this can be done in one step time.

So let's implement this in **Haar features**!! (che belloo)



The **intensity sum** of R is:

$$S_R = ii(4) - ii(2) - ii(3) + ii(1)$$

in $o(1)$ time.

Invariance to Scale and Translation

Haar features are not **invariant to scale** and **translations**, but this can be obtained using a **multiresolution sliding detection window**.

The idea is to **downscale the image iteratively** with a fixed factor, so some other faces at different scales will be detected by the window:

Gabor Filters

The **Gabor Filters** have nice **localization properties** in the **spatial domain** and in the **frequency domain**. This method can check if in a subregion of the image there are some **frequencies** in some **specific direction**.

Mathematically, a **convolutional Gabor 2D filter** is a **sin wave** modulated by a **gaussian**.

It's a complex term where:

- The **real part** is the **even symmetrical** part of the signal
- The **imaginary part** is the **odd symmetrical** part of the signal

Optical Flow (Module 9)

An Optical Flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera. The usual result of this method is a 2D vector field, where each vector is a vector showing the movement of points from first frame to second.

Assumptions

- Brightness Constancy
    - Image intensities do not change significatively although their location may change
- Spatial Coherence
    - Neighboring points belong to the same surface and hence have similar motions

- Temporal Persistence
    - The image motion of a surface patch changes gradually over time (no hard changes)

With some mathematical manipulation and with only the Brightness Constancy constraint we can reach this form:

$$I_x v_x + I_y v_y + I_t = 0 \quad \text{Optical Flow Equation}$$

where:

- $v_x v_y$ are velocity vector components (unknown)
- $I_x I_y$ are spatial gradient component (Sobel)
- I_t is temporal gradient (frame difference)

But the Optical Flow Equation has two unknowns and cannot be solved. To solve this problem, we have to use some methods.

We can only estimate the projection of the velocity on the gradient of the intensity.

Lucas-Kanade

We have to apply another constraint, the spatial coherence constraint, so the pixel neighbors have simlar displacements and we can take a patch P of M pixels instead of a single pixel.

So we now have an overdetermined system, with more equations than unknown.

This type of systems are solved with the Least Squares Method, that minimizes the sum of squared residuals:

$$\min_v ||Av - b||^2$$

which has solution:

$$v = pseudo(A)b$$

IN OUR CASE:

$$A^TA = \begin{bmatrix} \sum_{(x,y)\in P} I_x^2 & \sum_{(x,y)\in P} I_x I_y \\ \sum_{(x,y)\in P} I_x I_y & \sum_{(x,y)\in P} I_y^2 \end{bmatrix} = H$$

HARRIS CORNER DETECTION MOD.

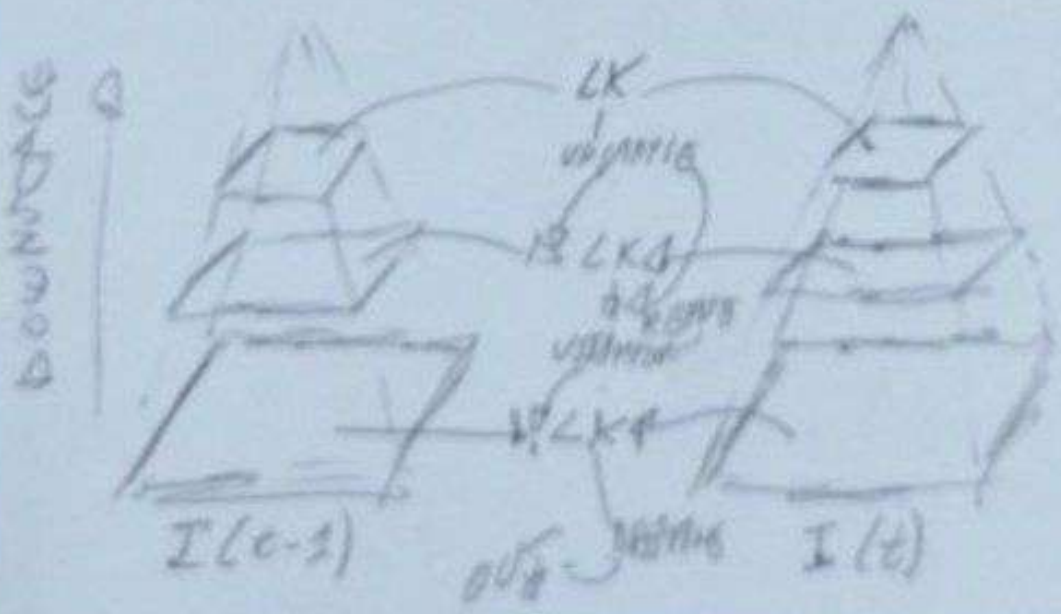
→ LUCAS-KANADE WORKS BEST AT CORNERS → BECAUSE CORNERS ARE NOT AFFECTED BY THE APERTURE PROBLEM.

→ A^TA:
- INVERTIBLE
- NOT NEARLY-SINGULAR
- WELL CONDITIONED

LARGE MOTION PROBLEM
- REDUCE Δt
- INCREASE M → LOSS ACCURATE

→ ITERATIVE LUCAS-KANADE → REWRITE APERTURE PROBLEM AS
$0 = I_x dx + I_y dy + I_t dt + \varepsilon$
← POLYNOMIAL ROOT FINDING PROBLEM
SOLVE WITH ITERATIVE NEWTON-RAPHSON METHOD → CALCULATE MOTION FIELD FOR EVERY PIXEL BY ABSOLUTE FORMAT.

→ ITERATIVE PYRAMIDAL LUCAS-KANADE



FARNEBACK (43)

LK → FIND OPTICAL FLOW ONLY ON CORNERS.

THE AIM OF THIS METHOD IS TO FIT AN IMAGE, REPRESENTED BY DISCRETE PIXELS INTENSITIES, IN A SURFACE:

$$f(\underline{x}) = \underline{x}^T \underline{\underline{A}}\, \underline{x} + \underline{b}^T \underline{x} + c$$ → WE CAN GET THESE COEFFICIENTS IN TERMS OF NORMALIZED CONVOLUTION WITH THE BASIS FUNCTIONS $\{1, x, y, x^2, y^2, xy\}$

SUPPOSE WE HAVE A GLOBAL DISPLACEMENT d:

$$f_1(\underline{x}) = \underline{x}^T \underline{\underline{A}}_1 \underline{x} + \underline{b}_1^T \underline{x} + c_1$$
$$f_2(\underline{x}) \approx \underline{x}^T \underline{\underline{A}}_2 \underline{x} + \underline{b}_2^T \underline{x} + c_2$$ → $f_2(x) = f_1(x - d)$

→ KEY EQUATION

$$\underline{b}_2 = \underline{b}_1 - 2\underline{\underline{A}}_1 \underline{d}$$ (d: UNKNOWN)

$\underline{\underline{A}}_1 = \underline{\underline{A}}_2$ → PROBLEM!!! IN REAL CASE IT CAN NOT HAPPEN

→ $\underline{\underline{A}}(\underline{x}) = \dfrac{\underline{\underline{A}}_1(\underline{x}) + \underline{\underline{A}}_2(\underline{x})}{2}$

OTHER PROBLEM → IT'S IMPOSSIBLE TO HAVE A GLOBAL TRANSLATION → $d(\underline{x})$

→ KEY EQUATION → $\underline{\underline{A}}(\underline{x})\, d(\underline{x}) = \Delta \underline{b}(\underline{x})$ → $\Delta \underline{b}(\underline{x}) = \dfrac{\underline{b}_1 - \underline{b}_2}{2}$

OTHER PROBLEM → IT'S TOO NOISY → AVERAGE INFORMATION OVER A PATCH P

→ THIS GIVE US AN OVERDETERMINED SYSTEM

→ SO WE CAN MINIMIZE THE LEAST-SQUARE PROBLEM