Computer Vision's Notes

Daniele Bertagnoli

2023/2024

Contents

1 Physics Main Concepts **3**
- 1.1 Determine 2D Coordinates . . . 3
- 1.2 Colors . . . 3
  - 1.2.1 Metamerism . . . 4
- 1.3 Digital Images . . . 4
  - 1.3.1 Image Types . . . 5
  - 1.3.2 Spatial Resolution . . . 5
  - 1.3.3 Basic Relationships between Pixels . . . 6
- 1.4 Histograms . . . 6
  - 1.4.1 Histogram Equalization . . . 7
  - 1.4.2 Histogram Projection . . . 7
  - 1.4.3 Plateau Equalization . . . 8
  - 1.4.4 Histogram Specification . . . 8
  - 1.4.5 Contrast Stretching . . . 8

2 Filters & Edge Detection **8**
- 2.1 Smoothing Filters . . . 8
  - 2.1.1 Simple Smoothing Filter . . . 9
  - 2.1.2 Weighted Smoothing Filter . . . 9
  - 2.1.3 Median Filter . . . 9
  - 2.1.4 Correlation and Convolution . . . 9
- 2.2 Sharpening Filters . . . 10
  - 2.2.1 Laplacian Filter . . . 11
  - 2.2.2 Unsharp Mask & Highboost Filter . . . 13
- 2.3 Sobel Filter . . . 13

2.3.1 Considerations . . . 14
2.4 Edge Detection . . . 14
2.4.1 Canny Detector . . . 15
2.5 Corners & Blobs . . . 15
2.5.1 Corners . . . 15
2.5.2 Blobs . . . 17
2.5.3 Affine-Adapted Blobs . . . 18

3 Machine Learning **18**
3.1 Fitting . . . 18
3.1.1 Hough Transform . . . 18
3.1.2 Polar Representation . . . 18
3.1.3 Least Squares Line Fitting . . . 19
3.1.4 Total Least Squares . . . 19
3.1.5 RANSAC . . . 19
3.1.6 Incremental Line Fitting . . . 20
3.1.7 K-Lines . . . 20
3.2 Image Alignment . . . 20
3.2.1 Transformations . . . 20
3.2.2 Feature Descriptors . . . 21
3.2.3 SIFT . . . 21
3.2.4 Feature Matching . . . 21

1 Physics Main Concepts

1.1 Determine 2D Coordinates

Each object's surface has 5 main properties related to their behaviour with the light:

- Absortion
- Reflection
- Refraction
- Diffusion
- Transparency

To capture an image, the light is been projected on a lens that impresses that image on a film. Mathematically, for real world images we need the incident light on a point $E(x, y, z, \lambda)$ and the reflectivity function $r(x, y, z, \lambda)$. By combining these two components, we obtain the reflected light that has been captured by the camera: $C(x, y, z, \lambda)$. In particular, the we have to project this C in a 2D space, this can be done by following one of these two projection type:

- Perspective Projection: the projection on which both cameras an human eyes rilies on (mathematically more complicated).
- Ortographic Projection: the object sizes do not depend on the camera distance (unrealistic but much easier).

Moreover, the 2D coordinates depends also by the camera sensitivity $V(\lambda)$, that expresses the how sensitive is the camera in capturing different wavelengths.

$$f(x', y') = \int c_p(x', y', \lambda) V(\lambda) d\lambda$$

1.2 Colors

In the camera sensors there are 3 main color sensors: Red, Green and Blue (RGB colors). By determining the sensitivity with respect to each of these three colors, we obtain a full-colored image.

Each color in the real world corresponds to a wavelength (we can see the light in the 400-700nm spectrum range). The observed color is the result of the interaction of the light source spectrum with the surface reflectance.

The coordinates of the color using RGB, are given by the weights of those 3 colors. Using RGB we can create a **linear color space**, RGB are all monochromatic lights, to express some wavelengths we need to subtract rather then add the others.

1.2.1 Metamerism

In colorimetry, metamerism is a perceived matching of colors with different spectral power distributions. Colors that match in this way are called matamers.

$$\begin{aligned} f_{\mathrm{R}}(x', y') &= \int c_p(x', y', \lambda) V_{\mathrm{R}}(\lambda) d\lambda \\ f_{\mathrm{G}}(x', y') &= \int c_p(x', y', \lambda) V_{\mathrm{G}}(\lambda) d\lambda \\ f_{\mathrm{B}}(x', y') &= \int c_p(x', y', \lambda) V_{\mathrm{B}}(\lambda) d\lambda \end{aligned}$$

1.3 Digital Images

Digital images are rapresented using pixels, each pixel can be either a single value (sensitivity images) or a 3D vector (color images) depending on the digital rapresentation. Each pixel has a coordinate, the origin (unlike classical Cartesian Space) is the top-left corner of the image.

The result obtained by the previous illustrated formulas are real numbers, hence they can have infinite numbers after the decimals. However, in digital images we must discretezie that numbers using one of these two techniques:

- Sampling: group set of coordinates into the same point which has an integer value.
- Quantization: reduce the possible values that the function can assume (so lower the number of colors that can be displayed).

Moreover, we must choose how many bits we have to use to rapresent each pixel's value. We call this paremeters as levels, so the number of colors that can be expressed (2 levels means 1 bit per pixel, 4 levels means 2 bits, ..., up to 256 levels). The number of levels determines the size in memory of the image. The total size of the image in the memory is:

Size = M*N*k, where:

- M: number of pixels per row
- N: number of pixels per column
- k: number of bits used to represent levels

1.3.1 Image Types

The images can be:

- Intensity Images: each pixel is an integer number that express the light intensity, hence the resulting image is a gray scale image.
- Color Images: each pixel contains a 3D vector with the three color components.
- Binary Images: each pixel can be either 1 or 0, so the image is composed only by black and white pixels.
- Index Images: each pixel contains an index that points to a color table.

1.3.2 Spatial Resolution

It is the smallest distinguishable detail in the image, expressed as:

- ppi (Pixel Per Inch)
- dpi (Dot Per Inch)
- M*N

1.3.3 Basic Relationships between Pixels

By considering a single pixel, we can the define 4-neighbors (either diagonal or vertical-horizontal neighbors) and the 8-neighbors. Based on this we can define the **connectivity property**, two pixels are connected if they are in the same class (same intensity or same color depending on the image type). The connectivity is adapted on the neighborhood relation:

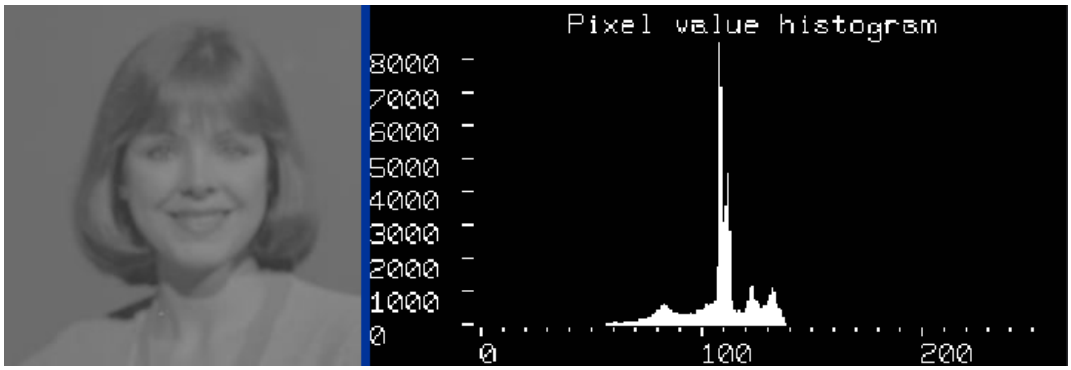
- 4-connectivity: 4-neighbors
- 8-connectivity: 8-neighbors
- Mixed-connectivity: two pixels p and q are m-connected if q is a 4-neighbor of p or q is a 4-diagonal-neighbor of p and they have at least one 4-neighbor in common.

The **adjacency property** is strictly related to the connectivity, two pixels (or two sub-images) are adjacent if they are connected. As for connectivity, we also have 4-adjacency, 8-adjacency and m-adjacency (based on the connectivity type).

A **path** is a sequence of adjacent pixels, we can have 4-path, 8-path and m-path (based on the adjacency type).

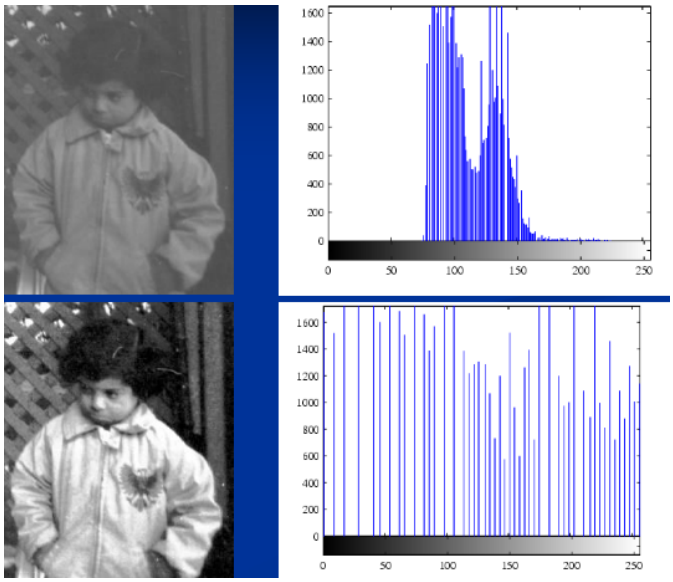
1.4 Histograms

We can use histograms to represent images, this allow us to process the image through mathematical formulas. A useful usage is to represent the image with gray scales and save the number of gray-levels into an histogram. We can also represent RGB images through histograms, in partiuclar, we will have an histogram for each color (3 in total).



1.4.1 Histogram Equalization

When the histogram's values are clusterized, so they are all grouped in a defined range, we can **obtain a better image trying to equalize the histogram**. This mean that we can distribute in a more flat way those values. However, when the image has a cluster in the histogram but other values are even present, we should not equalize the whole image. In this case, a better choise could be to equalize only the part of the images that creates that cluster in the histogram.



We can use several **techniques in order to model the histograms to obtain different versions of the same image in order to create a more complete dataset**. This allow us to use those versions to train the model in a more fine-graned way. We can also **use these techniques to normalize the images, to obtain a better dataset**.

Note: This is the formula used (also by OpenCV) to obtain gray scale levels of a pixel, starting from the RGB values of it:

$$Y = 0.299R + 0.587G + 0.114B$$

1.4.2 Histogram Projection

In this technique, we are **shifting the histogram in order to keep the curve of the original histogram**.

1.4.3 Plateau Equalization

This technique combines HP and HE to obtain a better result by mitiganting the defects of both techniques. We can decide how much we should apply HP and HE (in percentage). We can try to flat the histogram's values by keeping the curves.

1.4.4 Histogram Specification

We can **use an image as referenece to transform the histogram of another image**. In particular, by giving the histogram of the referenc we can try to model the other histogram in a way that it will look similar to the provided one.

1.4.5 Contrast Stretching

We **stretch the contrast of the image in order to obtain a full-range image in term of histogram values**.

2 Filters & Edge Detection

To apply filters we can work using the neighborhood concept. The neighborhood is choosen using the size of the kernel window. At the edges the kernel windows are reduced, in this case we have some solutions to deal with this problem:

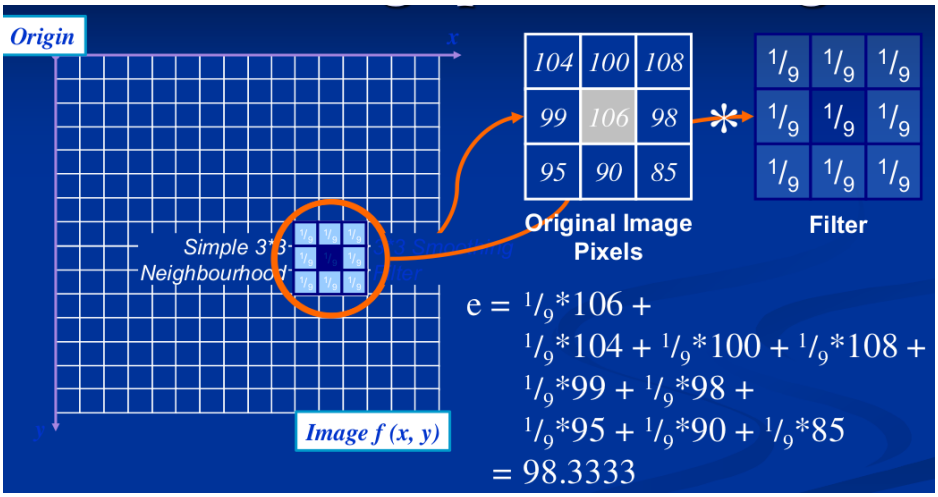
- Omit missing pixels: it works only with some filters and we have to manage manually the borders using more code.
- Replicate the border pixels
- Truncate the image: remove the pixels in order to make all the kernel windows with the same size.
- Add black or white pixels.

2.1 Smoothing Filters

This **filters are used to remove some fine-grained details or to remove noise** (that can be seen as very small details).

2.1.1 Simple Smoothing Filter

It is one of the most simple filters, all the values in the kernel is taken to compute the average. This is applied to the center pixel. It could be useful to remove noise and highlight gross details.



2.1.2 Weighted Smoothing Filter

It is like the simple smoothing filter, but the more closer the pixels are to the center, the more is the weight.

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

2.1.3 Median Filter

In this case we do not avarage the pixels' values but we use the median. This could be more useful to remove noise but it also produces less defined shapes.

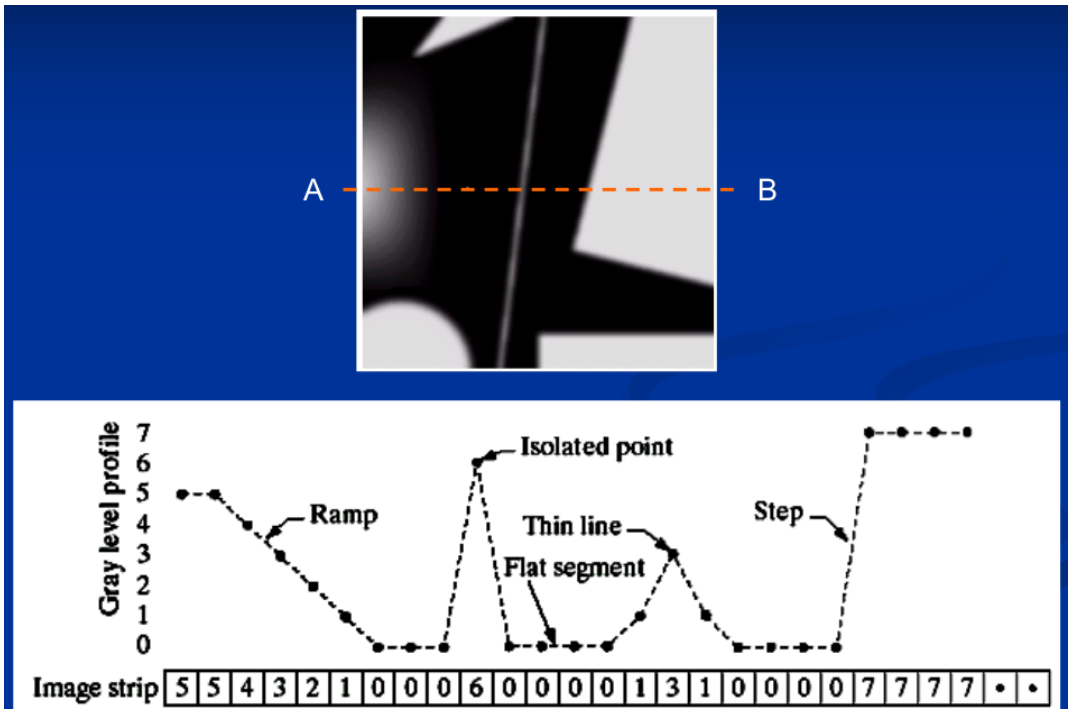
2.1.4 Correlation and Convolution

The just seen filters are called correlation filters, since pixels are correlated inside the same kernel. The convolution is a similar operation, but instead of associating the right corner of the mask

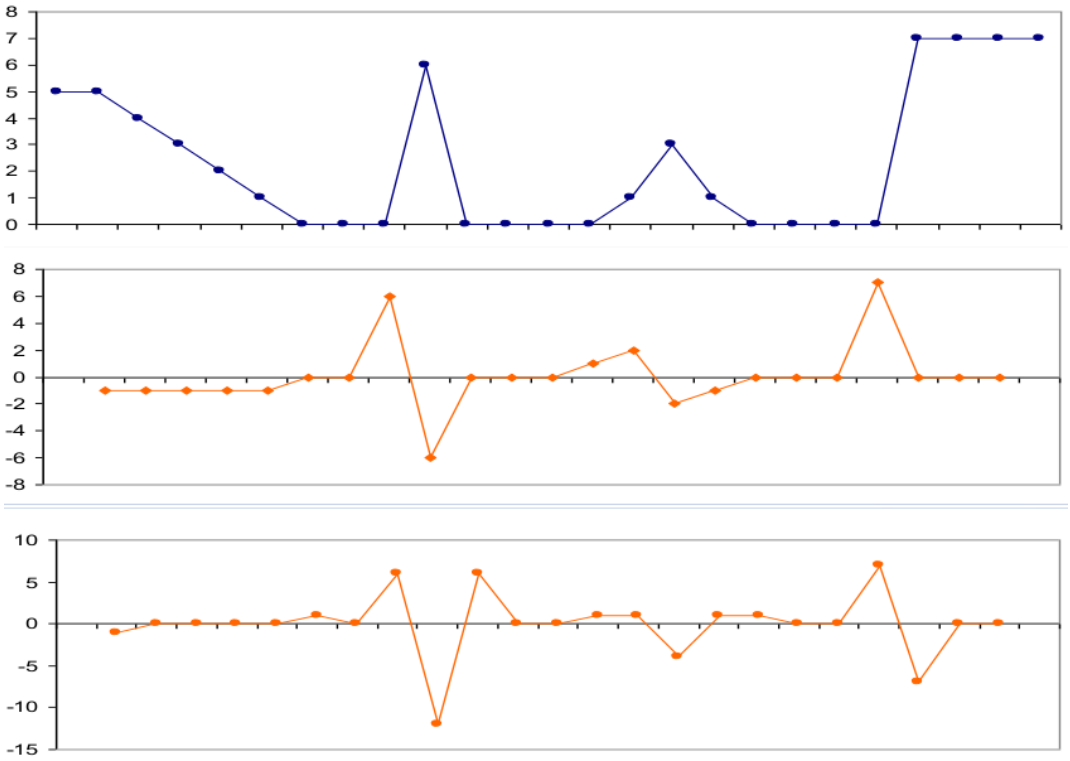
with the right corner of the kernel, we will use an inverse technique, so each pixel in the kernel is associated with the opposite cell of the mask.

2.2 Sharpening Filters

These are **filters are used to highlight details inside the image, so to remove the blur from an image and highlight edges of the objects**. They are based on **spatial differentiation**, that is the rate of change of a function. For simplicity, let's consider a single line of pixels to explain how it works. If we track the line that describes the changes in pixels, it will be like this:



We could use the first derivative of the Gaussian function in order to use the slope of the function to spot the edges, however we can see that using the first derivative we are not able to check if those



The Gaussian second derivative is more useful since is easier to implement and provides more stronger response to fine details.

2.2.1 Laplacian Filter

It **uses the Gaussian second derivative to detect intensity transitions, hence, can be used to detect edges inside an image**. The Laplacian filter is mathematically composed by the sum of the two partial derivative, respectively to x and y.

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

$$\frac{\partial^2 f}{\partial^2 x} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

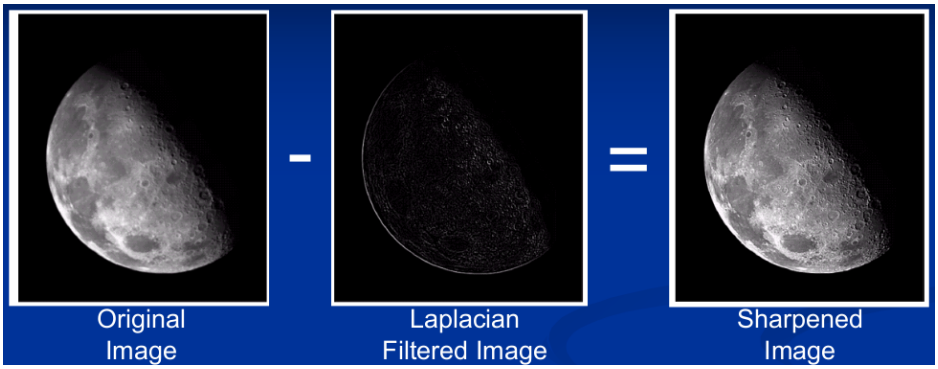
$$\frac{\partial^2 f}{\partial^2 y} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

$$\begin{aligned}\nabla^2 f = &[f(x+1,y)+f(x-1,y)\\ &+f(x,y+1)+f(x,y-1)]\\ &-4f(x,y)\end{aligned}$$

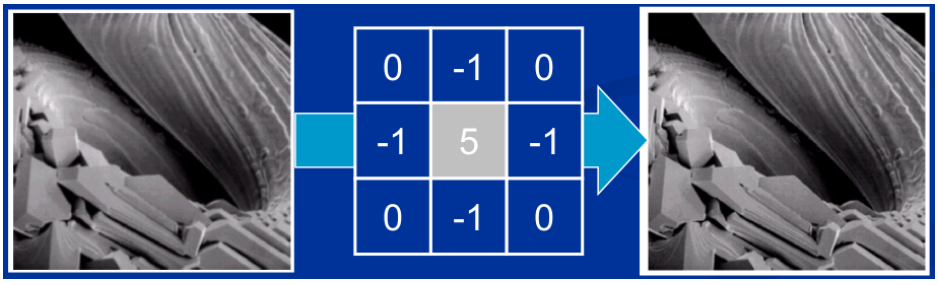
Using the final formulation of the laplancian filter, we can easily build a **3x3 mask for the kernels**:

0	1	0
1	-4	1
0	1	0

The result of the laplancian filter is not an enhanced image, in fact, we have to subtract this image to the original one to achieve a sharpened image.



We can apply all these steps using a single mask that is the following (obtained by subtrating the original image to the laplancian filter imaage):



2.2.2 Unsharp Mask & Highboost Filter

We can use a sequence of linear spatial filters in order to obtain a sharpening effect:

1. Blur the image.
2. Subtract if from the original image and multiply it for a factor K which controls the level of detail.
3. Add this mask to the original image.

2.3 Sobel Filter

It is a filter that is based on the first derivatives. Use the first derivative is quite complicated, for a function $f(x, y)$ the gradient f at (x, y) is given by the column vector:

$$\nabla \mathrm{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

In order to calculate the magnitude of this vector, we must apply some semplifcations, obtaining this:

$$\nabla f \approx |G_x| + |G_y|$$

We obtain from this, two masks for both vertical and horizontal filtering, depend from our needs:

-1	-2	-1
0	0	0
1	2	1

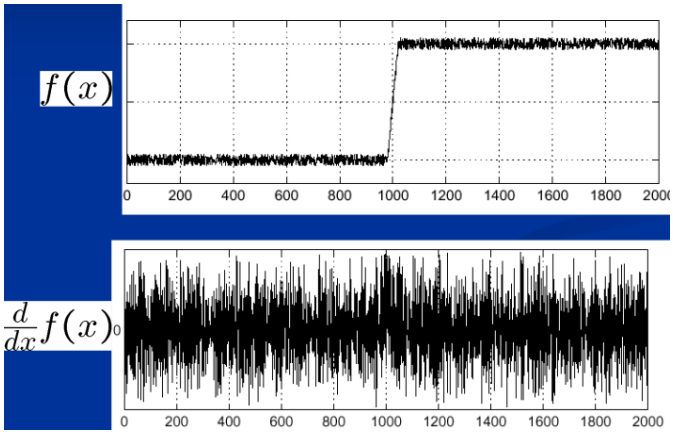
-1	0	1
-2	0	2
-1	0	1

2.3.1 Considerations

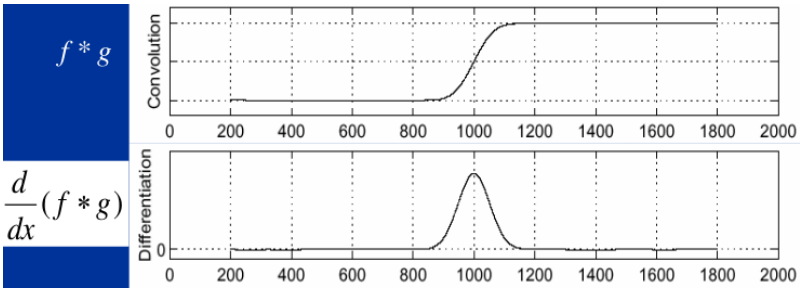
The **first derivative methods can generally produce thicker edges with a stronger response to gray gradients**. On the other hands, **second derivative methods have a more precise response to details** (thinner edges) **but can produce double responses at step changes in gray levels** (so edges are drawn twice, one for every color change).

2.4 Edge Detection

An **edge can be seen**, in term of first derivative function, **as a rapid change in the intensity**. The most simpler edge detector that could use is applying the first derivative to spot where the function changes. However, this is not a good idea since if the signal has some noise, the function changes rapidly very frequently.



The solution is to smooth the image using the seen filters, in this way the function should be able to spot a single peak rather than detect the noise peaks.



This obviously implies that edges are blurred, **the more is larger the kernel used to smooth the noise, the more will be the blur**. We have to manage this problem, indeed, a

possible solution is to find in that edge a local maximum and set it as part of the edge. To locate a maximum we have to **understand the gradient direction and then find the maximum**. Even in this case there are some problems and difficulties that we have to face, first of all we should minimize the number of local maxima around the edge. We must be able to draw a line that is continous and be as much robust as possible against noise.

2.4.1 Canny Detector

It is the mostly spread and used edge detector, it involves in these steps:

- Filter the image using a derivative of Gaussian.
- Find magnitude and orientation of gradient.
- Reduce the edge width using local maxima.
- Define upper and lower thresholds (hysteresis): the size of the threshold window define the "strenght" of the edges that we want to consider. The smaller is the value for the bounds, the smaller will be the size of the details that we want to take account (and viceversa).

2.5 Corners & Blobs

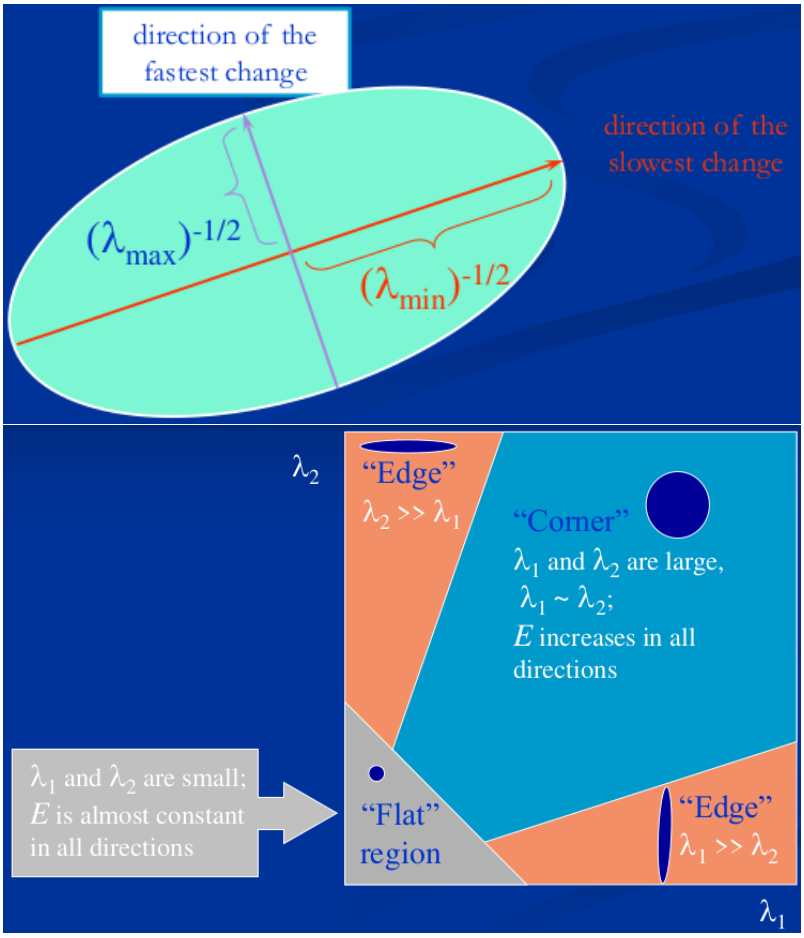
In order to extract some useful features from the images, we could use both corners and blobs. Features should be found multiple times in all the images, occupying only a small area.

2.5.1 Corners

A corner is the junction between two edges. We can use corners since they represent implicitly the change of the color in all the directions. **The Harris detector** is used to detect corners, the idea behind this process is to take a window (set of pixels) around every pixel and measure the gradient change by shifting that window in all the directions. This function can be approximated into:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

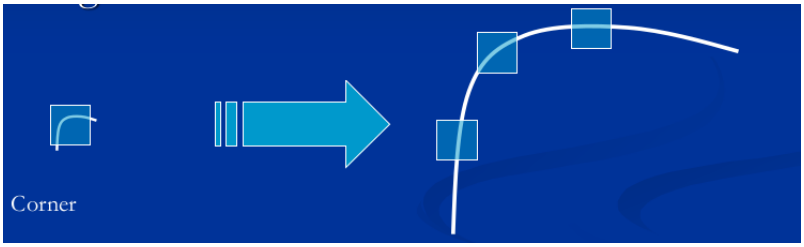
With M called **second moment matrix**. The R is a score calculated for each window. In particular the two lambdas are eigenvalues that can be seen as the axis length of an ellipse. This ellipse is the visualization of the gradient change in the windows.



So the harris detector works by following these steps:

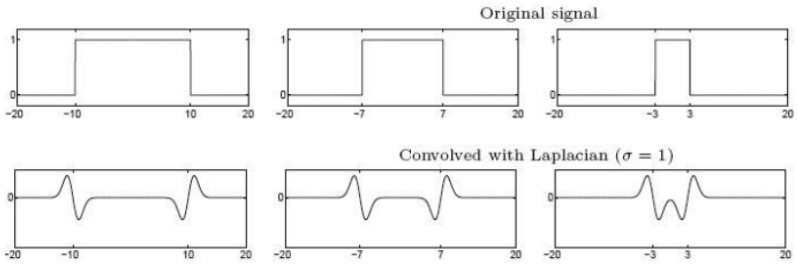
- Compute Gaussian derivative of the pixels.
- Compute the second moment matrix in a Gaussian window around each pixel.
- Compute the corner response.
- Apply the threshold over lambdas.
- Find local maxima, in this case is the centroid of the circles.

However, the **harris detector has the invariance to the rotation** (so it works even if the image is rotated), **but is partially invariant to the intensity changes** (depending on the threshold, some corner could be not detected) **and is not invariant at all to the image scaling** (when a corner is zoomed, for small windows it will appear as an edge).



2.5.2 Blobs

We want to achieve something that can used even when the scale of the image changes. A blob, mathematically speaking, is what lies between the two peaks in the Laplacian (Gaussian second derivative) function.



However, our aim is to have a single point between the two ripples. We could increase the σ of the laplancian, but this will end up having a flattened function. So **we can multiply the Laplacian by σ^2, after using a trial and error we can find the right value for sigma in order to have a single point rather than having an area**.

Scale-space Blob Detector: This algorithm is used to find the blobs in an image:

1. **The original image is processed at multiple scales using Gaussian smoothing**. In other words, the image is blurred with Gaussian filters of different standard deviations (sigma values). This creates a series of increasingly smoothed or blurred images, with each image corresponding to a different scale level.

2. **On each of these smoothed images, a blob detection method is applied**. These methods help identify regions where the intensity changes in a way that resembles a blob at the particular scale.

The blob detection is done based on the Laplacian function or the difference of Gaussians that approximates the Laplacian (better implementation).

2.5.3 Affine-Adapted Blobs

This variation of the classical blobs that guarantees that invariance to some affine transformations such as rotation, stretching and scaling. The steps are:

1. Similar to other blob detectors, the first step is to **find potential blob locations**.
2. To **make the blob descriptor invariant to affine transformations**, you need to describe the local image region inside the affine region in a way that is not affected by changes in scale, rotation, and shearing. One common approach is to compute the gradient information within the affine region and then normalize it to make it affine-invariant.
3. **Affine-adapted blobs include information about the orientation of the keypoint**. This orientation can be computed based on the gradient direction within the region. It helps to achieve rotation invariance. These information are used to build a descriptor that encodes information about the gradient magnitudes and orientations in a way that is robust to affine transformations.

3 Machine Learning

3.1 Fitting

Fitting is about choosing a paremetric model to represent a set of features. We can use a **voting scheme** to recognize a mode, so **let every feature vote for most compatible model**. This allow as to deal with both noise and missing information since is not probable that we have both problems on every feature.

3.1.1 Hough Transform

For **every edge point in an image we compute a set of parameters that represent a possible line** ($y = mx + b$) in the Hough space. The **intersection between all the possibile lines is the point in the Hough space that represents the line that passes from all the points in the original image**.

The main problem is that in the Hough space we cannot draw vertical lines due to the fact that the slope of that lines is equal to infinite.

3.1.2 Polar Representation

It is a different representation that solves the problem of the Hough transform. Each point will add a sinusoid in the (θ, ρ) parameter space. In particular, **for every edge point a sinusoid is**

drawn from any angle between 0 and 180 degrees. **The most voted point is the edge that passes from every point** (edge).

In order to vote, we must **create a grid where the votes will be accumulated**. The size of the grid **shouldn't be too large** in order to avoid that points in differnt edges contributes to the same bucket, **neither too small** in order to have edge points relative to the same edge that votes for a different bucket.

This approach is **rebust against occlusion but not too much against noise**, moreover the **grid size must be tuned a lot and the complexity increases exponentially with respect to the number of model parameters**.

3.1.3 Least Squares Line Fitting

This is a technique that tries to **draw the line in the space for which the sum of the squared distances are minimum**. In this case, **only the vertical distance is considered** so if all the points are aligned on the x axis, the distance from the line is infinite.

3.1.4 Total Least Squares

Same as for Least Squares but in this case also the horizontal distance is considered. Thanks to the second moment matrix that takes account of both x and y, this method is more noise-robust.

3.1.5 RANSAC

This technique is more **robust even if we have a lot of outliers**. **We choose a small subset of our points, then we fit the model with respect to that subset and discard all the points that are too far from the obtained line**. Then we repeat this process by considering the another subset of points. We have to set **3 parameters**:

- N: number of samples.
- t: distance threshold, if the distance of a point is greater than t is considered as outlier.
- s number of points that composes the subset.

The main advantage is that is a **simple technique and often works well in practice, but it requires a lot of parameter tuning and the result highly depends on the random initialization**.

3.1.6 Incremental Line Fitting

The idea is to **fit a line on the first s points. For every point the line is refitted in order to reduce the distance between all the points and itself**. **If the distance of a point from the line is too high, a new line is generated** by keeping the previous line as before, so the **final result will be not a single line but a segmentated line**. This method can't cope with occlusion and is too sensitive to noise.

3.1.7 K-Lines

We can either partition the points into k sets and then fit a line for every set or randomly initialize k sets of parameters. We will assign every point to the closest line and iteratively refit the line on the new points. This method is robust against occlusion but we have to set k and it is very sensitive to the initialization.

3.2 Image Alignment

Image alignment means "recognize" that two image are similar. There are two main approaches:

- **Direct alignment**: search an alignment for which most of the pixels agree.
- **Feature-based alignment**: search an algorithm for which the extracted features agree.

The alignment can be also seen as fitting, in particular, **we can fit a model to a transformation between pairs of features that match in two images**. As first we have to **compute the putative matches**, so try to associate the features in the first image with the features in the second one, **then we have to apply some transformation T such that a small group of putative matches is related to that transformation**.

3.2.1 Transformations

We have 3 main types of transformations:

- **Similarity**: translation, scale, rotation and shear.
- **Affine**: any combination of similarities.
- **Projective (Homography)**: plane projective transformation.

3.2.2 Feature Descriptors

If we don't know the correspondeces between the features, we need **feature descriptors**. So assuming that the local effect of geometric transformations is factored out, to compute the similarity we need these feature descriptors that are invariant to intensity changes, noise, perceptually insignificant changes of the pixel pattern.

The simplest descriptor is the **vector of raw intensity values** that **can be compared using Sum of Squared Distances** (SSD), however this method is not invariant to intensity change. Hence, we can use **normalized correlation** that is invariant to affine intensity changes and can be computed using Zero-Mean Normalized Sum of Squared Differences (ZNSSD) or Zero-Mean Normalized Sum of Absolute Differences (ZNSAD).

Using **patches** (pixels considered for the descriptors), even a **small shift could lead to high variations in matching score. The solution is to use histograms rather than raw pixels as descriptors**, in partiuclar the **directional histograms that indicates the gradient direction for every pixel in the patch**.

3.2.3 SIFT

SIFT is a feature detector. It splits each patch into 4x4 sub-patches and for each of those we compute the histogram of gradient orientation (8 possibile directions). The final descriptors have 4x4x8 = 128 dimensions. This guarantees a more robust solution against intensity changes and to small shifts.

3.2.4 Feature Matching

Generating putative matches means find for every patch in one image a short list of patches in the other image that could match with it based on the appearence:

- Exhaustive Search: for each feature compute the distance to all features and choose the closest one.
- Fast Approximate Nearest Neighbor Search: use hashing or other structures.

In order to **find the most reliable putative matches**, we can use the following heuristic: **compare distance of nearest neighbor to the distance with the second nearest neighbor. If these two distances are very close, than the match can be marked as less distinctive or ambiguous**, hence discarded. A **good ratio used as threshold could be 0.8**. However, the chances that our putative matches are still filled of outliers is high, so we can use different strategies to fit a geometrical transformation to a small subset of those:

- **RANSAC**: randomly select a seed group (subset of matches) and compute the transformation on those matches. Then we find the inliers and if they are enough we can loop on those to refit the transformation. The transformation with the highest number of inliers is kept. This strategy does not exclude the outliers very well.
- **Incremental Alignment**: use RANSAC but taking advantage of strong locality constraints, only close-by matches are considered when the process is started and then the matches in the neighborhood are progressively added.
- **Hough Transform**: The idea is to let every match vote for its transformation hypothesis using a Hough space with very coarse bins.
- **Hashing**: Make each invariant image feature into a low-dimensional "key" that indexes into a table of hypotheses. Then, given a new test image, compute the hash keys for all features found in that image, access the table, and look for consistent hypotheses. This can even work when we don't have any feature descriptors.