Computer Vision

Bravi N., Contreras R., Guelfi F., Ierardi A., La Rocca M., Rachiglia F., Trovatello A., Viola S.

December 18, 2024

Abstract

Notes and materials of Computer Vision course from master's degree in Robotics Engineering.

1 Image processing fundamentals

1.1 Convolution operator

Convolution operator is one the of the most operator used in computer vision, is can be thought as a general moving average. There is two different interpretation, in the **continuous domain** and in the **discrete domain**.

- Continuous domain:
$$(f * g)(t) = \int f(\tau) \cdot g(t - \tau)\, d\tau \tag{1}$$
- Discrete domain:
$$(f * g)(m) = \sum f(n) \cdot g(m - n) \tag{2}$$

We are interested in the digital version of the equation (discrete domain), since when you implement the convolution, you have signal in the digital domain, they are numbers. This is how we perform convolution in one dimensional signal:

- Flip (reverse) one of the digital functions
- Shift it along the time axis by one sample
- Multiply the corresponding values of the two digital functions
- Summate the products from step 3 to get one point of the digital convolution.
- Repeat steps 1-4 to obtain the digital convolution at all times that the functions overlap

1.2 Fourier transform

In the frequency domain we use the FFT; Note that convolution in the time domain is equivalent to multiplication in the frequency domain (and vice versa):

$$\mathcal{F}\{f * g\}(t) = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\} = F(\omega) \cdot G(\omega) \tag{3}$$

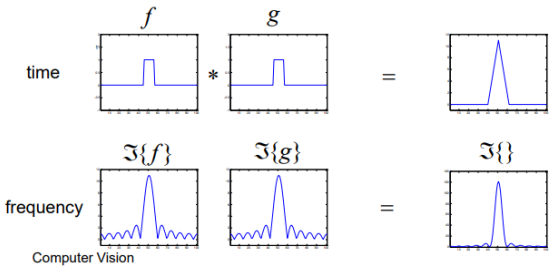


Figure 1: Fourier transform

The same information are two ways to see the world, in time or in frequency, but they have the same information, depends on application.

1.3 Delta Dirac function

Is the impulse function, it is important to describe the system, the response of the system.

$$\delta(x) = \begin{cases} \infty, & x = 0 \\ 0, & x \neq 0 \end{cases} \quad \int_{-\infty}^{\infty} \delta(x)\, dx = 1. \quad \int_{-\infty}^{\infty} f(x)\, \delta(x)\, dx = f(0)$$

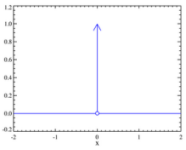
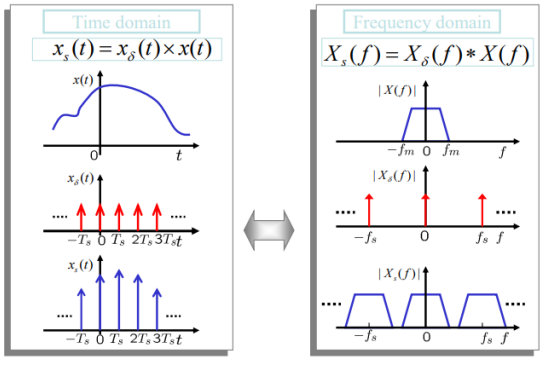


Figure 2: Impulse function.

In the world, normally there are two types of domain: Time domain and Frequency domain.



Normally the world is continuous, in time or space, but when you store the information in the computer, you need the digital version, so we need to perform **sampling**. The sampling consist in the product between the continuous signal and a train of delta of Dirac with some delays, so the result will be a signal point by point reconstructed where you store all the information of the signal.
In the frequency domain there are some difference as, **when you perform the product in time, you perform convolution in frequency**, the train of delta doesn't change but in the frequency the

delta of Dirac is the inverse of the sampling time.
The **sampling** is the replica of the band base of the signal to different frequency.

1.4 Nyquist theorem

One of the important sampling theorem is the Nyquist theorem, that tell us to avoid error wen need to sample 2 times of maximum frequency:

$$F_s = \frac{1}{T_s} = 2f_m \tag{4}$$

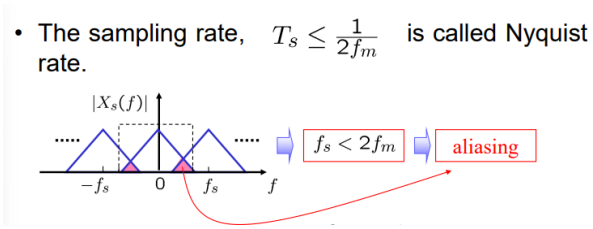
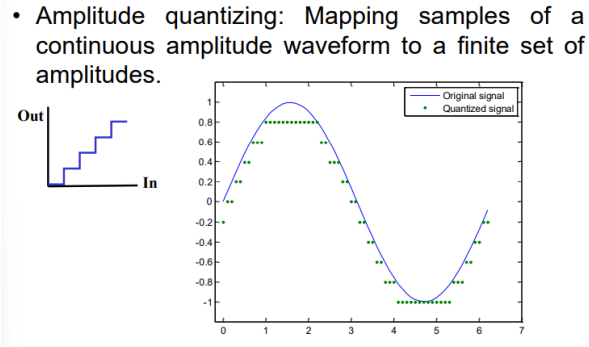


Figure 3: Sampling rate.

this allow us to reconstruct continuous signal, otherwise if you do not comply these constraints, the replica of the spectrum overlaps and the information is lost (Aliasing).
Another step of processing is **quantization process**. We need to choose a number of level for your amplitude.



There is a little quantization error, but if you have enough level you can handle it.

1.5 Image formation

When you store an information on your computer, you store an information of the object, then you have to create the matrix of intensity; each element of the matrix is a pixel and this is a sample of the image, this is called **spatial sampling** of the image.

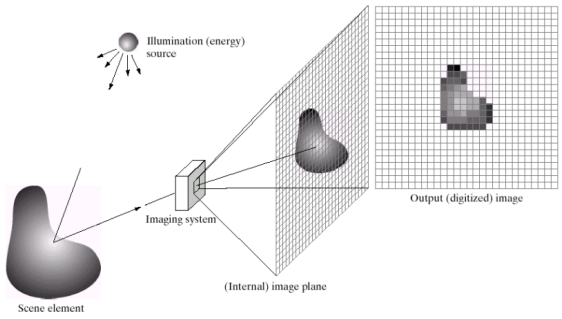


Figure 4: Spatial sampling

If you use a few samples, using Nyquist theorem, you have an error. As in the image, there are 20/25 pixels (levels) so the object contour is not smooth, you see the pixels.
To represent the transition of the gray color of the object, you need enough intensity level to make the transition more smooth.
POSSIBLE EXAM QUESTION:
How is composed a camera?
The camera is composed of three different system.

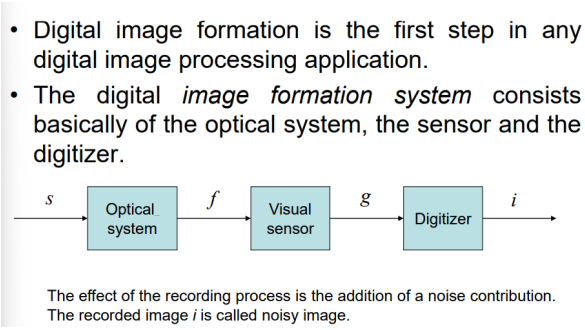


Figure 5: Camera components

So, there is the incoming light (S) that carries the information about the world, then there is my camera (all three blocks of the image). The camera is composed of Optical System that allow us to recover the light, the lenses. The output of the Optical system is a little transform of the original signal (F), then there is the Visual sensor that provide us the digital version of the original signal (I), the digital image.

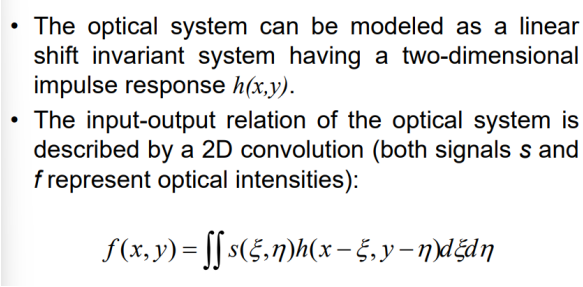


Figure 6: Optical system

The Optical System are the lenses of the camera. The lenses con be described in a mathematical terms as a filter, and in this case the filter is a two dimensional filter. The original signal S is modified

by the properties of the lenses H.
When we talk of Optical System is can be called also PSF (point spread function) and it's a Fourier transform of the H function that is the impulse response of the system describes the lenses. You can model this part of the camera to be able to handle errors, as for example, if you move too fast your camera or if the object is moving to fast, that produce a Blurred image, blurred by motion. You can implement an algorithm that allow you to reduce the error.
To not have a blur, the impulse response of the optical system must be egual to the impulse function (Dirac).

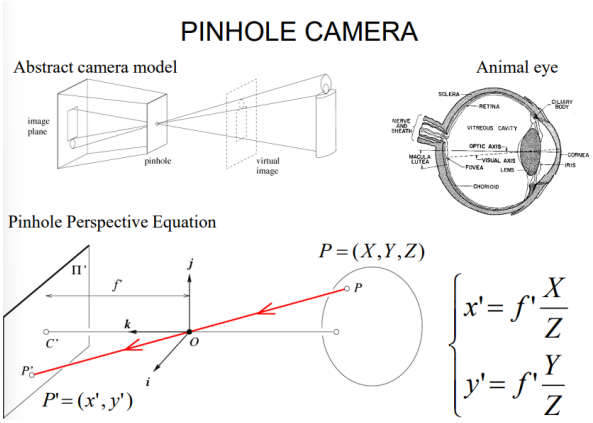


Figure 7: Enter Caption

The geometric model of the camera is the **pinhole camera**. The pinhole camera is the equation that allow you to take an object in the three dimensional world and put it in the 2D dimensional image.
P prime, is the projection of the three dimensional point.
The perspective equation is:
The projection on the image is the three dimensional X coordinate divided by the distance of the object Z, same for the Y. What we know is x prime and y prime, what we would know is X, Y and Z, so two equations and three unknowns variable, impossible to solve the system! This is the problem of computer vision.

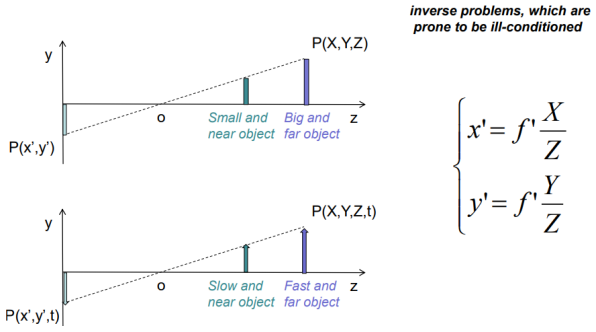


Figure 8: Enter Caption

As we can see, the problem is the projection of the object, it is impossible to say how the object is far! same for speed. To compute the equation we need to put some constraints.

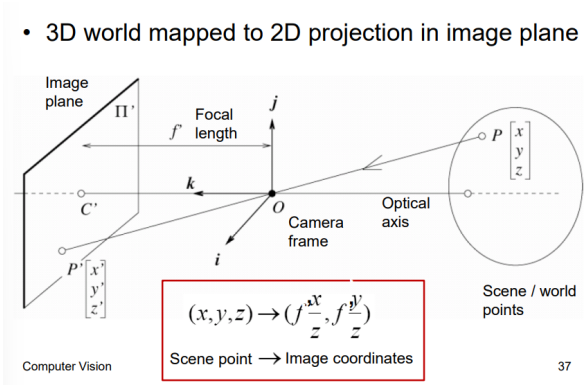


Figure 9: Enter Caption

The distance between the 2D image and the origin of the Camera frame is the **Focal length**.

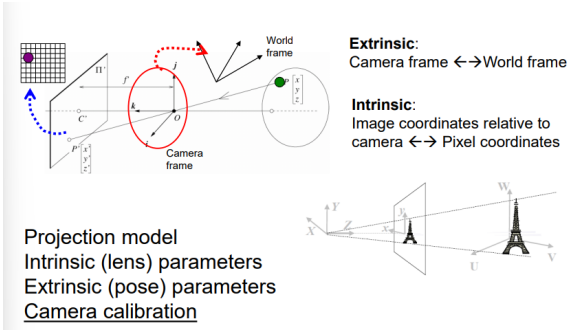


Figure 10: Enter Caption

There are two type of parameter: extrinsic and intrinsic.
Extrinsic: are the parameters that relate the camera frame with the world frame.
Intrinsic: are the parameters that relate the image coordinates and the pixel coordinates.

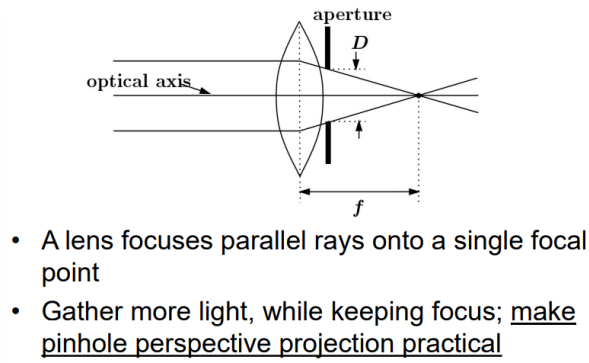


Figure 11: Enter Caption

In the image acquisition there is a lenses to acquire more light than pinhole, but this not change the geometry, there is also an aperture, that is the size of your camera.

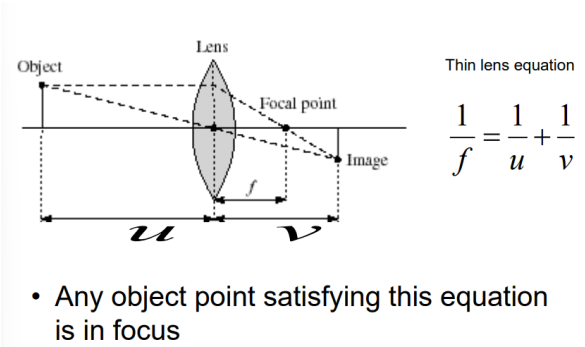


Figure 12: Enter Caption

There is a thin lens equation that tell you: only to a given distance of the lenses (Object), the projection of the point is a point, that means is focused without blur. Otherwise:

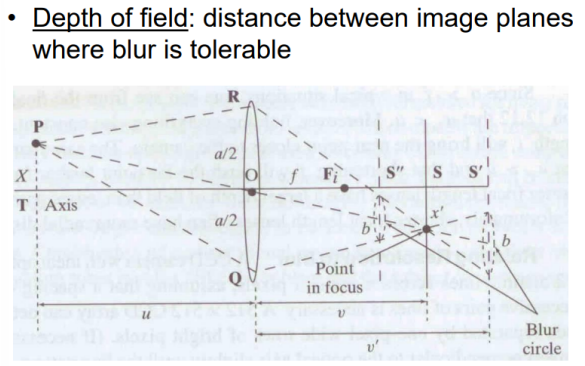


Figure 13: Enter Caption

When you have your sensor close to the lenses, what we see is the intersection, a circle (b), so you have the blur. Regarding the depth of field, the image is in focus if the intersection is inside a little range of depth of field, so the result is the flower in focus and the background blurred. Otherwise, id the depth of field is larger, the background is in focus and the flower more blurred.

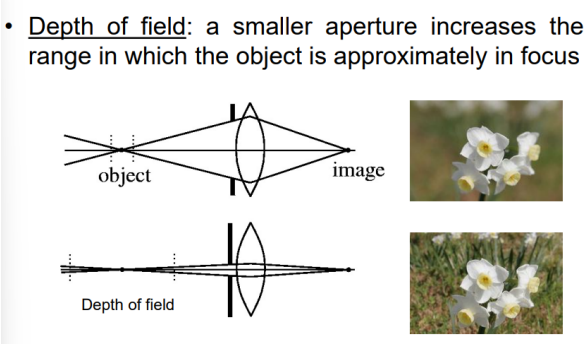


Figure 14: Enter Caption

Another aspect of the camera is the **Field Of View (FOV)**.

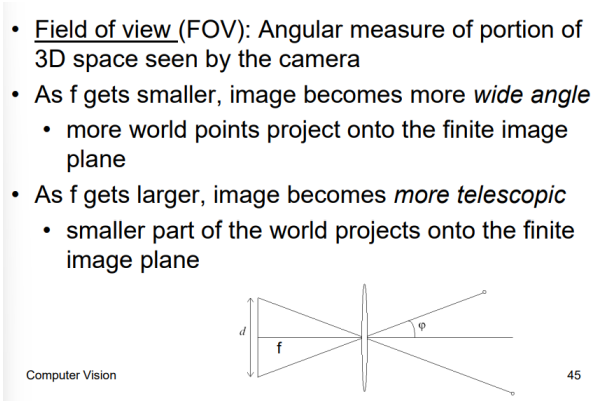


Figure 15: Enter Caption

FOV is how much see in front of you, and this is related to the size of the sensor (d). If you have a sensor closer to the lenses the FOV is larger (wide angle camera) and the focal leght will be smaller. If the device that acquire the light is not close to the lenses, the FOV is smaller and you obtain a zoom, and for the zoom the focal lenght is larger.

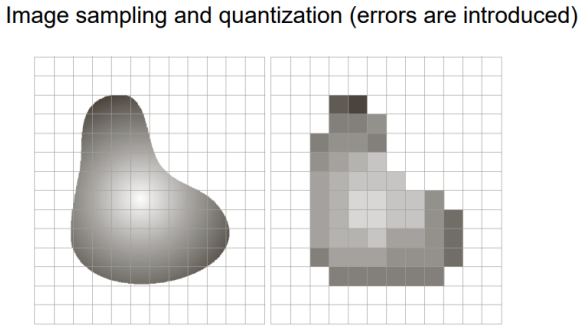


Figure 16: Enter Caption

The sampling produce the pixelization, that means the contour is not smooth.
The quantization produce sharp transition among pixels (error).
To solve you need more level and more pixels.

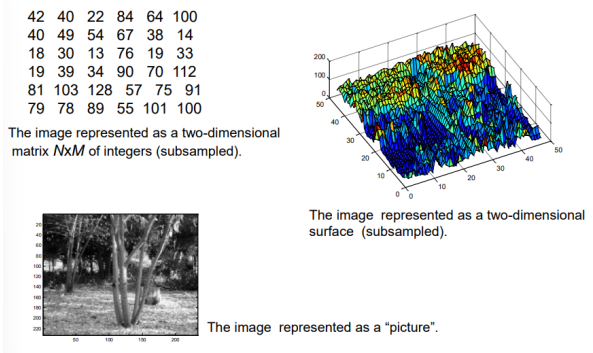
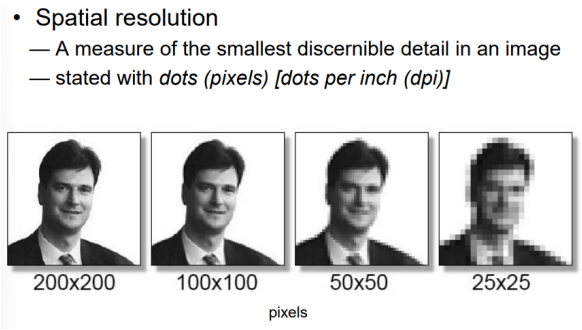
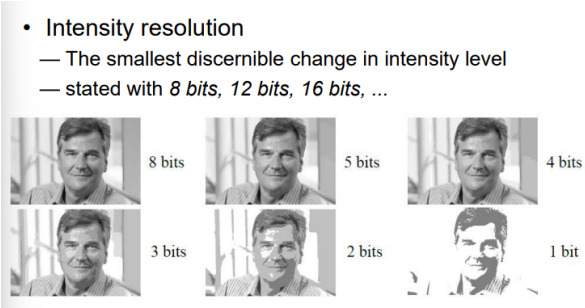


Figure 17: Enter Caption

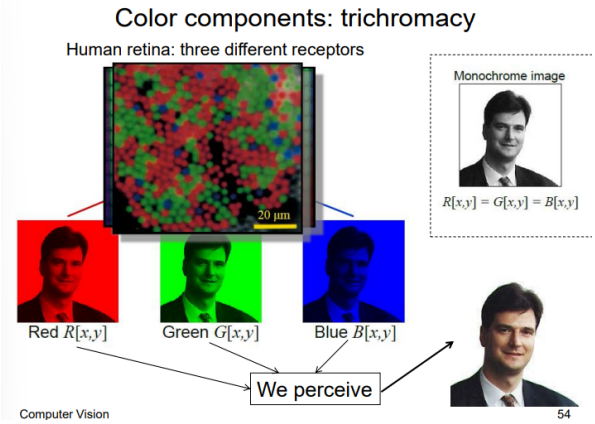
The numbers in the top left image, is a representation of an image, where the low number is a dark part and the higher a bright part of the image, since black is 0 and white 255.



If you have a good resolution you can see the contour very nice, but the meaning of the contour is that you can see (e.g.) the eyes, the details of the image.
If you decrease the number of pixel, you decrease the number of the resolution and the image will be blurred and you lose information. Sometimes we use a small image to speed up the computation, and in the 25x25 you detect the blobs, where there is an important information inside the image, and compute fast since the image is small. To resume, you use a small image to detect a possible face, a blob, then you compute the algorithm only in the blob area inside the larger image, this is called multi-resolution.



In the 8-bits image, the pixel can store 255 value (every pixel can store 8 bits) to store the gray level, and in this way you see the real "shape" of the image. If you want an binary image, you need to use 1 bit to represent the image.



In color images it's different. In our retina we have three kind of neurons that are sensitive to three different wavelenght. We are sensitive to red, green and blue, and our retina sample the world by using red, green and blue; our brain blend this three channel and then we perceive the world to the color. There is a mathematical way to describe that, we need to store 3 different matrices to store red channel, green channel and blue channel, this is called **RGB** model of the color, and in this way we can see the color image.
To obtain monochrome image, the red, green and blue must be the same.

TRUECOLOR IMAGES

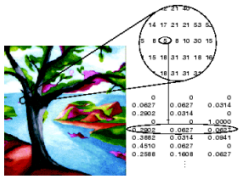
- A truecolor image is an image in which each pixel is specified by three values (RGB), one each for the red, blue, and green components of the pixel's color.

Computer Vision 55

In a truecolor image, each pixel store three values, each for color components. So, every image occupied 3x... memory.

INDEXED IMAGES

- An indexed image consists of an array and a colormap matrix. The pixel values in the array are direct indices into a colormap. The colormap matrix is an m-by-3 array of values in the range [0,1]. Each row of map specifies the red, green, and blue components of a single color.

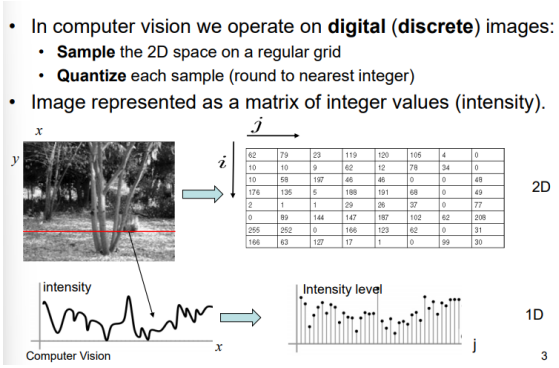


Otherwise, you can use Indexed images, this use the same memory of grey-scaling images. The number inside the image, is not the intensity of the image but the index of the look-up-table (LUT), where are stored the red, green and blue component of that pixel. Normally the LUT have a limited number of colors, so, you have a small error related to the color. That means, the color in the image is an approximation of the true color.

- In the case of a digital acquisition device, the two-dimensional signal is a digital image.
- We consider two types of cameras:
  - CCD (charge-coupled devices).
  - CMOS (complementary metal oxide semiconductor).
- There are separate photo-sensors at regular positions and no scanning.

In the current technologies, as CCD or CMOS, in each camera, the sensor is a matrix of device and each element of the matrix is a sensor that is able to acquire the incoming light and it is also a pixel of the corresponding image. The maximum resolution of the image is embedded in the device.

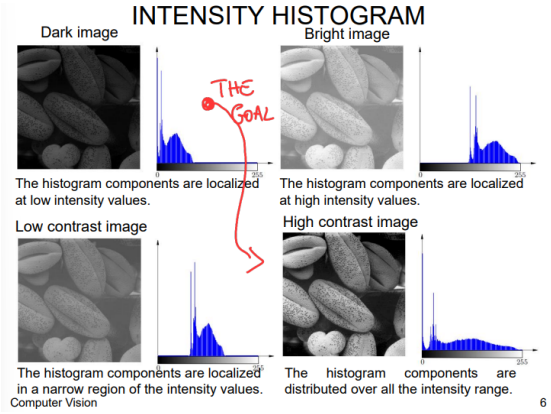
1.6 Image Filtering



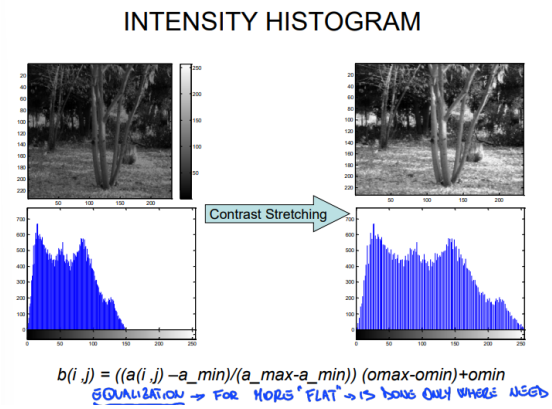
In a digital image each pixel describes the intensity of the pixel. A way to see the intensity within an image is Histogram.

1.6.1 Histogram

Histogram is a graph showing the number of pixels in an image at each different intensity value found in that image. Histograms can show different properties of the image, as: enhancement, compression and segmentation.

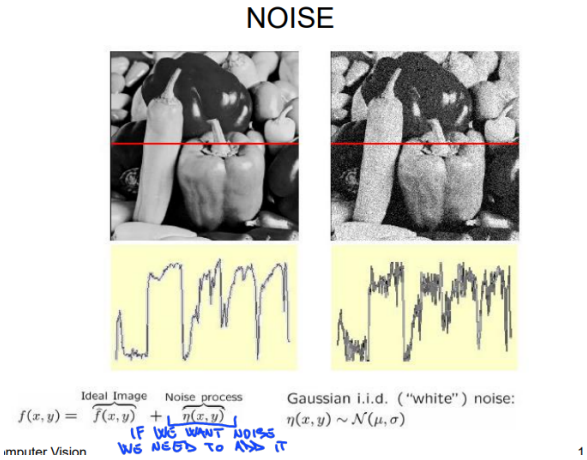


When we have a dark image, we can apply the enhancement to improve the image, as in figure. If the distribution is uniform, the image tends to have a high dynamic range and the details are more easily perceived. Another operation can be used is the Contrast Stretching, normalizing the intensity range between 0 and 255.

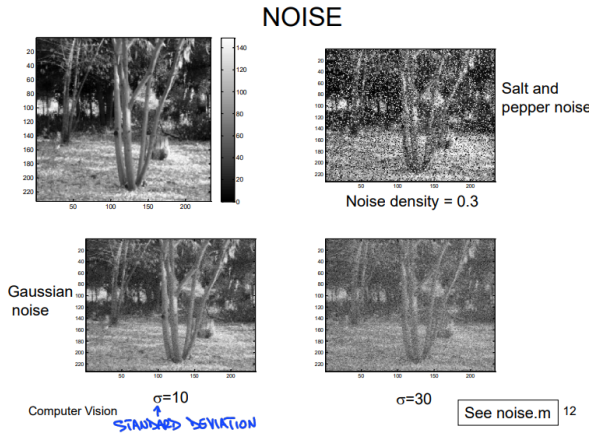


1.6.2 Noise

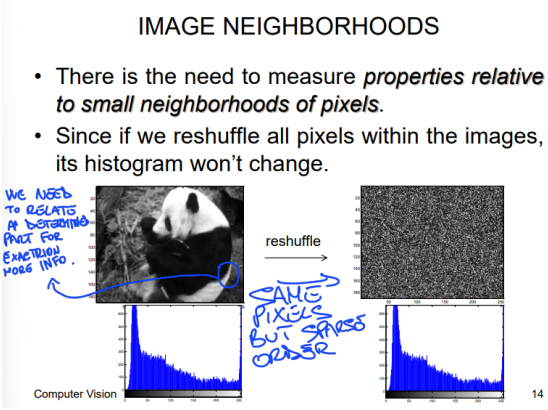
Digital images are corrupted by noise during image formation process.



In the figure is applied the Gaussian white noise. There are different types of noises:
Gaussian Noise: variations in intensity drawn from a Gaussian normal distribution.
Impulse ("Shot") Noise: there are random occurrences of white pixels.
Salt and pepper Noise: there are random occurrences of black and white pixels (0 or 255).



In the figure, we can see the Salt and Pepper noise controls by the noisy density (0.3 in the figure) and the Gaussian Noise controls by the standard deviation.



As in figure, Histogram is not enough to recognize a part in the image.

1.6.3 Linear filter

In the linear filter the output is a linear function of the input and the output is a shift-invariant function of the input, it means that if the input is shifted by n values, also the output will be shifted by n values.

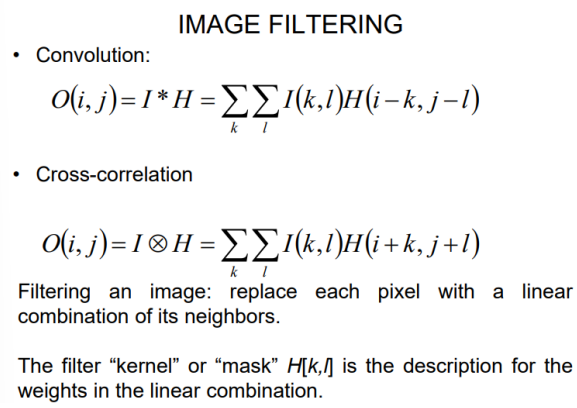
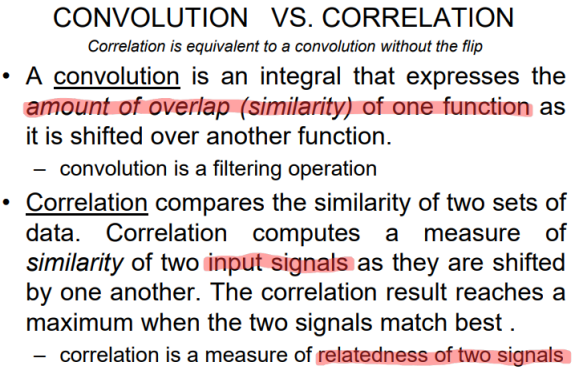
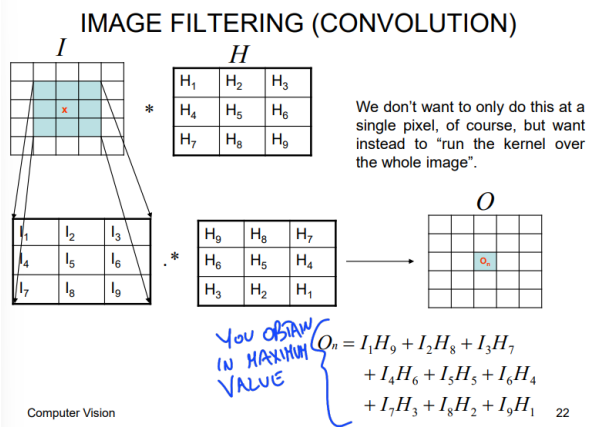


Figure 18: Enter Caption

The main difference between Convolution and Correlation is that the Convolution is a filtering operation while the Correlation is a measure of relatedness of two signals. An important aspect to consider is that the Convolution take the H image and flip from bottom to top and swap from right to left. In the Correlation no.

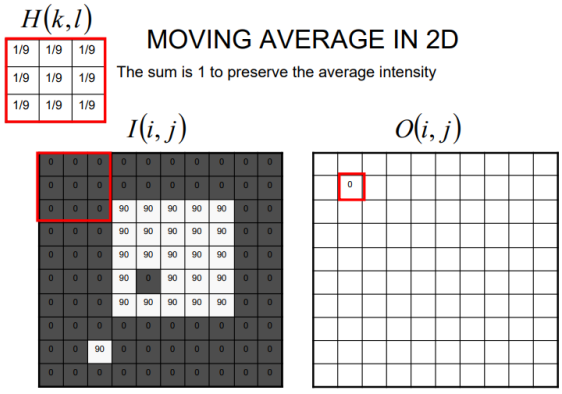


In the following images is show how the Convolution works.

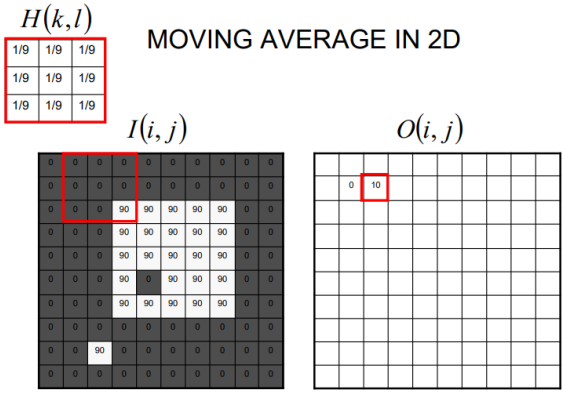


1.6.4 Moving average

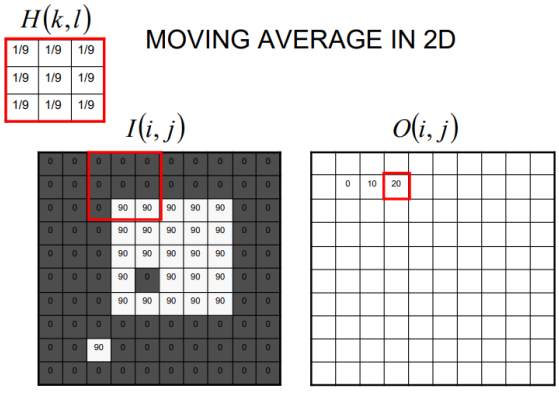
The Moving Average filter it works with average of the pixels.
In this case, the sum of the product between filter and I(i,j) is 0, so we put 0 in the O(i,j) image image.



In this case, the sum is 10, that's due to $\frac{1}{9} * 90$.



And so on...



The result will be:

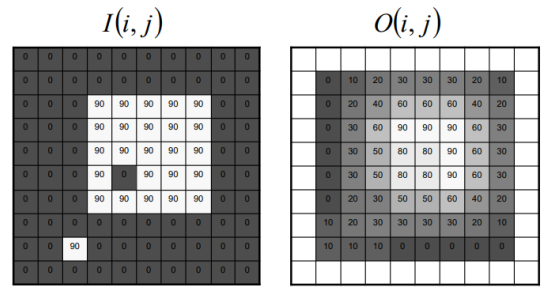


Figure 19: Enter Caption

Now there is a problem, what do we do for border pixels where the kernel does not completely overlap the image? One of the simplest methods is **Zero-Padding**.
So, we have the image and the kernel and what we do is to increase the size of the image by padding. After that we implement the convolution to get the result.

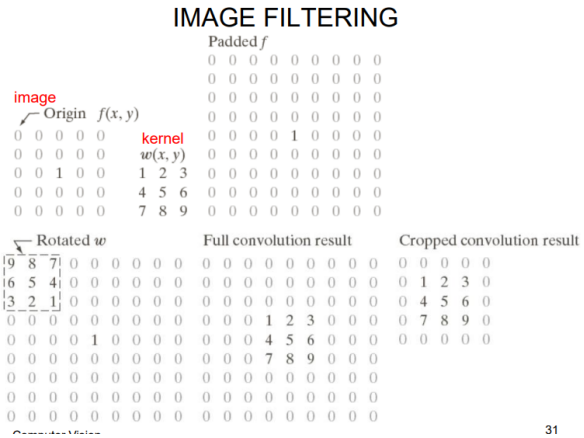


Figure 20: Enter Caption

The moving average is necessary to perform Smoothing, it means, the moving average produce Blurring for noise reduction and **bridging**.

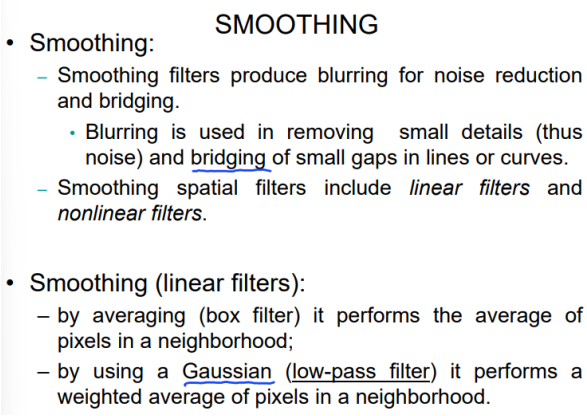


Figure 21: Enter Caption

The contour of a face my be not continuing, so you perform Smoothing to bridge the small gaps (for instance)in the contour line. The Smoothing can be performed by using the averaging (box filter) or by using a Gaussian (also called **low-pass filter**) that not have the box filter with weights but the weights are the amplitude of a Gaussian.

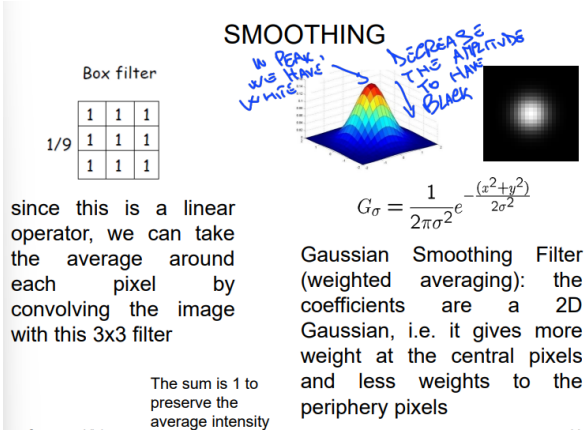


Figure 22: Enter Caption

The sum of the box filter MUST BE 1 to preserve the average intensity. The peak of the Gaussian means white part (high energy).
So, Smoothing is a technique to reduce the noise in the images.

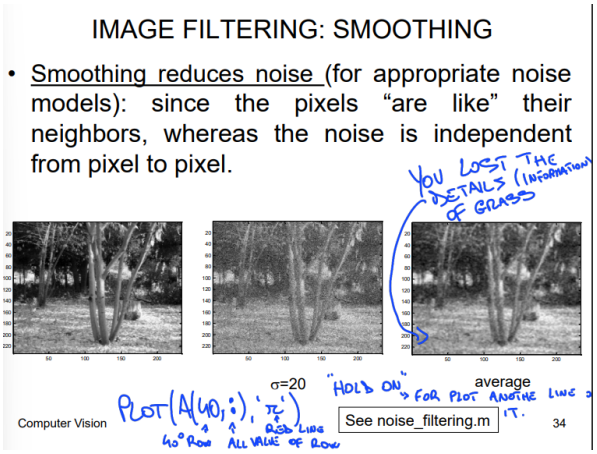
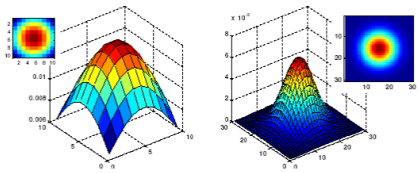


Figure 23: Enter Caption

As in the figure above, the original image have a lot of details, but when we perform the smoothing, we reduce the possible noise but we will lose also information about the image, for instance, details of grass.
The important parameters in the Gaussian filter are essentially two: the **size** of kernel and the **standard deviation**.

GAUSSIAN FILTERS

- What parameters matter here?
- Size of kernel or mask
  - Note, Gaussian function has infinite support, but discrete filters use finite kernels



σ = 5
with 10 x 10 kernel
(no)

σ = 5
with 30 x 30 kernel
(yes)

Computer Vision

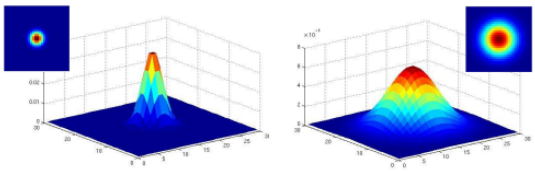
Set filter half-width to about 3 σ

35

Where the size of kernel is the size of the mask (box), while the standard deviation determines the extent of smoothing as below:

GAUSSIAN FILTERS

- What parameters matter here?
- Standard deviation of Gaussian: it determines the extent of smoothing



σ = 2
with 30 x 30 kernel

σ = 5
with 30 x 30 kernel

Computer Vision

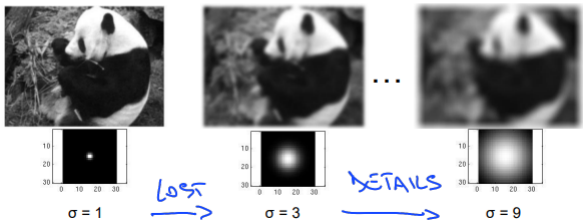
Set filter half-width to about 3 σ

36

The rule is to set the half-width of the kernel to about 3 times standard deviation. As above, standard deviation = 5 and the kernel is 30x30, where the half-width is 15. For instance, if we increase the standard deviation, we will lose details:

SMOOTHING WITH A GAUSSIAN

Parameter σ is the "scale" / "width" / "spread" of the Gaussian kernel, and controls the amount of smoothing.



To subsampling an image, you need to apply the low-pass filter and the sub-sampling (using Nyquist rate). Why do not sub-sampling directly? Because if we decrease the resolution without blurring, the image will have a lot of visible pixels.

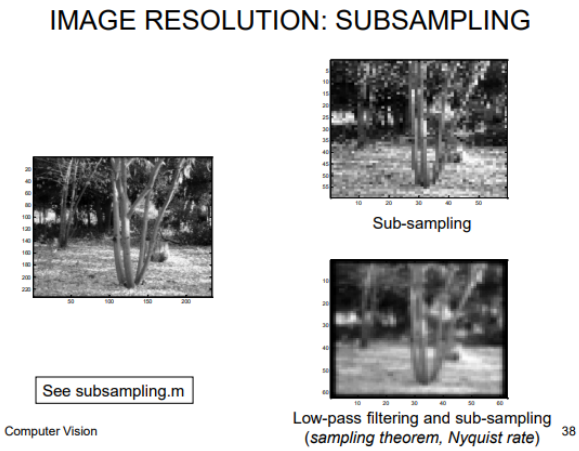


Figure 24: Enter Caption

1.6.5 Median Filter

Median filter is a non linear filter.

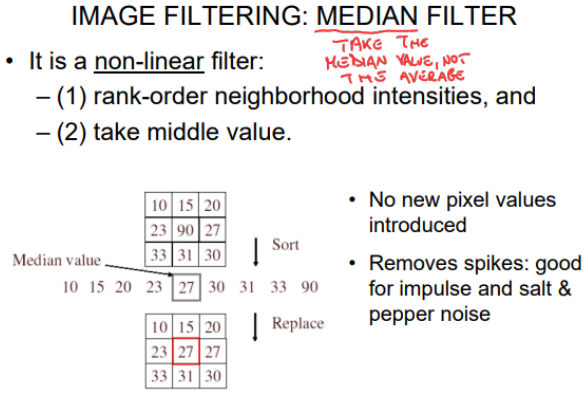


Figure 25: Enter Caption

It works with the middle value inside a box. So, if a value is too big with respect to neighborhood, it will be substitute with the median value of neighborhood (symmetric area around the big value). This filter is edge preserve and discard the spikes (value peaks).

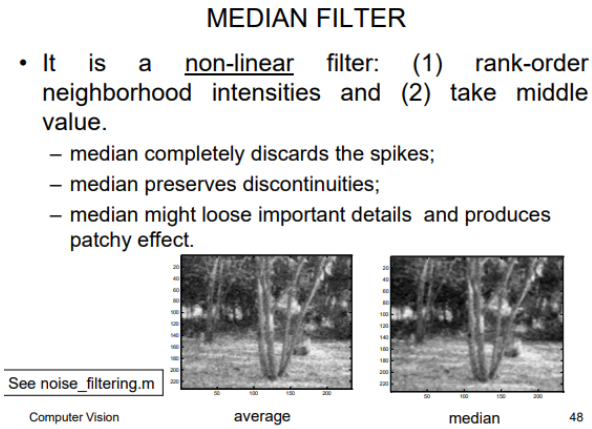
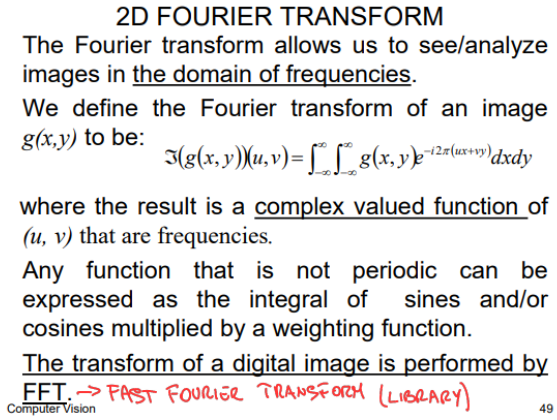


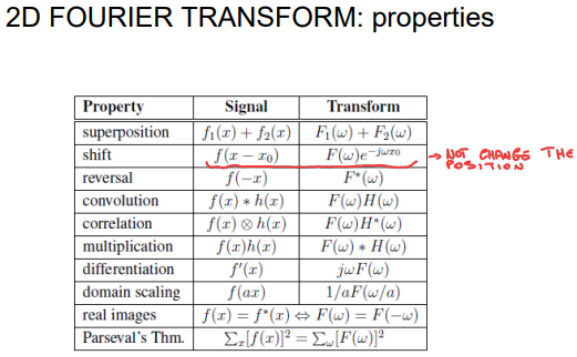
Figure 26: Enter Caption

1.6.6 Fourier transform

We use the Fourier transform when we want to see the same information seen in time but in the frequency domain.



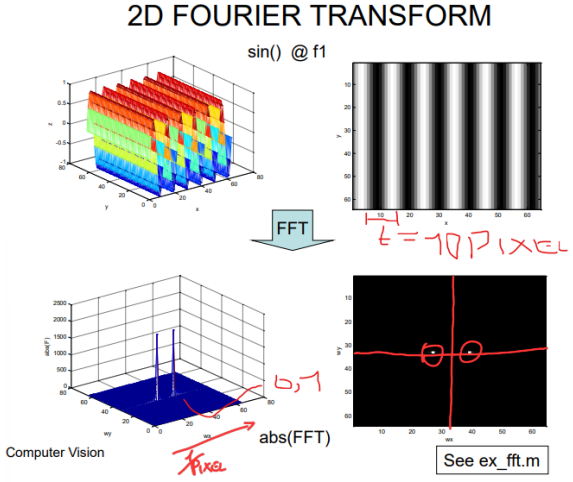
Some useful properties of Fourier transform:



Property	Signal	Transform
superposition	$f_1(x) + f_2(x)$	$F_1(\omega) + F_2(\omega)$
shift	$f(x - x_0)$	$F(\omega)e^{-j\omega x_0}$
reversal	$f(-x)$	$F^*(\omega)$
convolution	$f(x) * h(x)$	$F(\omega)H(\omega)$
correlation	$f(x) \otimes h(x)$	$F(\omega)H^*(\omega)$
multiplication	$f(x)h(x)$	$F(\omega) * H(\omega)$
differentiation	$f'(x)$	$j\omega F(\omega)$
domain scaling	$f(ax)$	$1/aF(\omega/a)$
real images	$f(x) = f^*(x) \Leftrightarrow F(\omega) = F(-\omega)$	
Parseval's Thm.	$\sum_x [f(x)]^2 = \sum_\omega [F(\omega)]^2$	

Table 3.1: *Some useful properties of Fourier transforms. The original transform pair is* $F(\omega) = \mathcal{F}\{f(x)\}$.

For instance, when i perform SHIFT in space, what we have is a change in phase only. Or, convolution is the product element by element in Fourier. Perform the convolution in space is faster than to perform product in the fourier transform.



In the figure above, we can see the fourier transform of the sinusoidal signal. In the top right figure, we can see the signal from the top and the bright part are the period, i.e. 10 pixels, so, the fourier transform will have 2 peaks of energy with value 0,1 [inverse of pixels].
To create low-pass and band-pass filters, we can modify the standard deviation of the Gaussian as below:

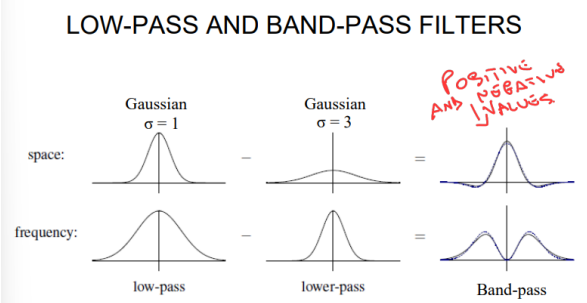


Figure 3.37: *The difference of two low-pass filters results in a band-pass filter. The dashed blue lines show the close fit to a half-octave Laplacian of Gaussian.*

1.6.7 Magnitude and Phase

In computer vision, magnitude and phase are two important concepts that are used to represent the frequency content of an image. The magnitude of an image is a measure of the strength of the frequency components, while the phase is a measure of the relative position of the frequency components 1.

For instance, in edge detection, the magnitude of the image gradient is used to identify the edges, while the phase is used to determine the orientation of the edges 1.

In summary, both magnitude and phase play important roles in computer vision, and they are used in various applications such as edge detection, feature extraction, and image processing.

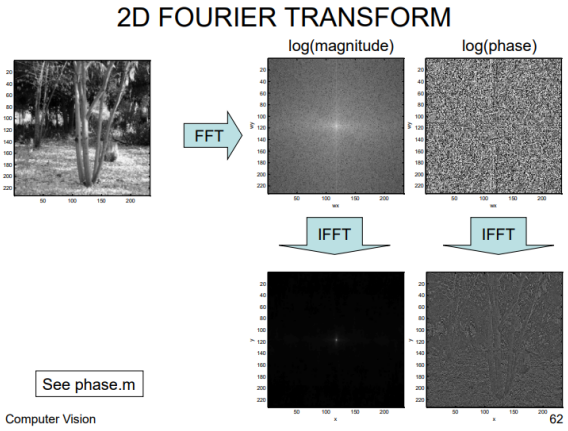


Figure 27: Enter Caption

2 Edge detection and Hough transform

We begin by employing filtering techniques to extract essential information from an image. In this chapter we will focus on demonstrating how, when presented with an object (such as a pedestrian), it is possible to detect the object's contour through edge detection.

This initial step involves capturing the most basic information (the contour of the object). Techniques based on derivatives or similar principles are utilized to detect gradients. In edge detection, the result is an image where a value of 0 indicates no edge, and a value of 1 signifies the presence of an edge. This breakdown occurs pixel by pixel.

When viewing such an image, clarity might arise. For example, a corridor or street may appear as two straight lines. Our minds reconstruct information beyond the individual pixel level, creating associations between pixels.

However, edge detection operates at the pixel level, providing only basic information. We are also interested in extracting more symbolic information from the image. For instance, recognizing a straight line as describing the floor of a room or a corridor and expressing it as a symbolic equation. This involves moving beyond pixel-level data and into a more abstract, geometric representation of the scene.

One of the method is known as Hough Trasform, this serves as an abstraction of boundaries, allowing us to detect not only straight lines but also geometrical shapes described in a parametric form. This marks a shift from mere pixel data to a mathematical or geometrical representation of the environment—an initial step towards interpretation.

2.1 Edges are Special

Edges play a crucial role in describing scenes as they efficiently convey vital information about the environment. They offer a powerful and compact representation by focusing on transitions or boundaries between regions, capturing essential details while minimizing data. Their simplicity has been historically utilized to describe objects like faces, horses, or planes. In essence, edges distill complex scenes into concise and informative snapshots, making them fundamental in image processing and computer vision. In the case of a plain white wall, the edges become essential as they define the wall's area. Since the interior lacks distinctive features, emphasizing the edges captures crucial information about the environment.

What are Edges? Edges in the context of image processing refer to discontinuities in the intensity of light. This definition characterizes edges as abrupt changes in image intensity, serving as boundaries that delineate different material properties within the image. Essentially, edges highlight the areas where there is a notable shift in visual information, making them valuable for identifying transitions and defining distinct regions in an image.

In image processing, extracting edges and performing segmentation are crucial steps. Segmentation involves identifying the area or volume of an object, which is vital for real-world interpretation.

Once edges and segmentation are obtained, blending this information allows for object identification and subsequent recognition. Simple algorithms are often effective in achieving these tasks.

When describing these concepts mathematically, there are four main types of edges:

1. Step Edge: Characterized by a sharp transition between black and white.
2. Ramp Edge: The transition is less steep, featuring a slope instead of a sharp change.
3. Roof Edge: Describes a gradual transition resembling the shape of a roof.
4. Spike Edge: Seen in features like street stripes, where there are sharp peaks in intensity.

Understanding these edge types helps in formulating mathematical descriptions, facilitating the interpretation of objects or events in an image.

2.2 Detecting Discontinuities

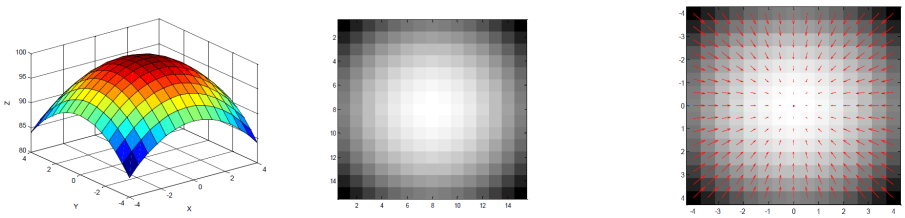
Edges, as intensity changes, can be identified using derivatives. Derivatives enhance these changes, providing an effective means to pinpoint and highlight edges by focusing on intensity variations.

Example: Consider an image where we analyze a row and represent it as an intensity function. By applying the first derivative to identify changes in intensity, the magnitude of the derivative highlights edges at maximum and minimum values. This approach allows for effective edge detection.

In the context of images, which are functions of two independent variables, the derivative concept extends to gradients. Deriving an image results in a vector composed of partial derivatives with respect to the independent variables, forming the image gradient. This can be mathematically expressed as demonstrated in the script (SEE MATH_2D_GRAD.m).

$$\nabla I(x,y) = [\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}]^T = [I(x,y)_x, I(x,y)_y]^T] = [I_x, I_y]^T$$

Consider a simple paraboloid with a known mathematical description. Computing partial derivatives yields a gradient vector, a function of image position. Visualizing this gradient on the image shows vectors for each pixel, indicating the direction of intensity change. Strong derivatives are evident, and the vectors point uphill towards maxima, revealing the steepest ascent direction. This visual insight is particularly useful when the mathematical description is unknown.



The gradient always points to the maximum change in intensity. The vector perpendicular to the gradient represents the tangent to the image's contour line, revealing the locus of points with equal intensity. This simple operation, calculating the gradient (∇) of an image, offers valuable information about its structure and contours.

Another method to visualize the derivative of an image is by considering changes in intensity along different axes. In an image transitioning from black to white along the X-axis, the derivative would show a non-zero value along the X-axis, and zero along the Y-axis (and vice versa for an image with a gradient in the Y-axis). For changes in intensity occurring in both X and Y directions, the derivative will reflect this bivariate change.
The gradient direction (orientation of edge normal) is given by:

$$\theta = tan^{-1}(\frac{\partial f}{\partial y}/\frac{\partial f}{\partial x})$$

The strength of an edge is determine by the magniude of the the gradient:

$$||\nabla f|| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$$

The gradient is how strong it is the edge and normally this is used to identify an edge.
But an image is not described as a formula, they are number, so you have to apply the numerical derivative, and there are different techniques to implement the derivative.

Looking into the formula take the function in a position move a little, subtract and then divide for that little, but when you move in a image you move form a pixel to another so the formula change to this:

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y) - f(x,y)}{1}$$

.

The numerical derivative in image processing is achieved through convolution with a kernel, typically $[1, -1]$. When applied horizontally, it captures differences between adjacent pixels along rows (X-axis), and when applied vertically, it detects changes along columns (Y-axis). Convolution involves sliding the kernel over the image, computing differences at each step. This method efficiently highlights edges and transitions in the image. If we can describe the derivative as a convolution. We can also use the properties of the convolution when you handle algorithm.

2.3 Simple Edge Detection with Gradient

Now we can write a simple edge detection algorithm by using gradient:

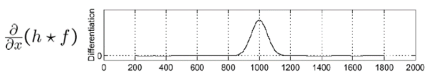
1. Compute Gradient: Use the convolution with kernels like $[1, -1]$ along the x and y axes to find intensity changes.
2. Compute the gradient Magnitude: Combine the x and y, usually using the square root of the sum of squared gradients.
3. Apply Threshold: Choose a threshold to classify pixels as edges or non-edges based on gradient magnitude.

What is above a given threshold that means the strength of the gradient is above the threshold and for us is an edge. Choosing the threshold is subjective and depends on image characteristics.

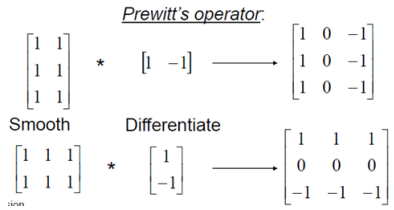
2.3.1 Issues to Address: NOISE

In any image measurement, the presence of noise, characterized by small, high-frequency variations, can significantly impact the accuracy of edge detection. The process of taking derivatives to identify edges tends to amplify this noise, resulting in a convoluted output that makes it challenging to distinguish true edges from noise. To mitigate this issue, it is better to incorporate a low-pass filter, such as a Gaussian filter, prior to performing the derivative operation. By convolving the image with a Gaussian filter, high-frequency noise are effectively smoothed. While this introduces a subtle blurring effect on sharp transitions, the overall benefit lies in the removal of noise and the facilitation of more reliable edge detection. The incorporation of a low-pass filter ensures a more robust and accurate outcome in the presence of noisy data.

You can do filtering for smoothing and filtering for convolution for a derivative, but now that we have described it in a mathematical term, we can exploit the properties of the derivative and convolution (they are linear) and combine it.

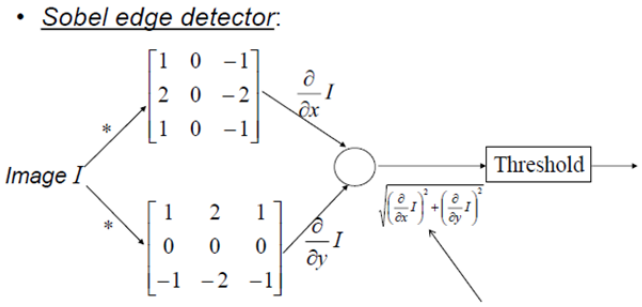


Mask to perform smoothing and derivative (PREWITT'S OPERATOR):



The approach involves integrating smoothing and edge detection by taking the derivative of the simplest smoothing filter, such as the box filter or average filter. Instead of applying two separate filters sequentially, the method combines both operations in a single step. This is achieved by first calculating the derivative of the blurring filter, where the convolution operation is employed. Subsequently, these two fundamental operations are convolved with each other, resulting in a 3x3 mask. This mask simultaneously performs smoothing and edge detection. The convolution operation constructs this metric, which combines the aspects of smoothing through the initial filter and edge detection through the derivative (specifically along the X-axis). The process is then repeated for the Y-axis by applying the same principle to the second row of the image. This straightforward convolution technique efficiently incorporates both smoothing and edge detection within a single operation.

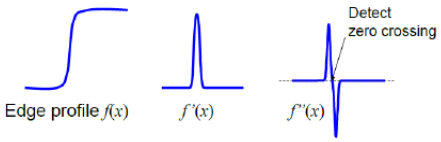
So, the Prewitt operator is no other than the convolution between them and average on the left and the derivative on the right. We will obtain the gradient smoothed. → Combining the two we will obtain the first edge detector SOBER EDGE DECTECTOR.



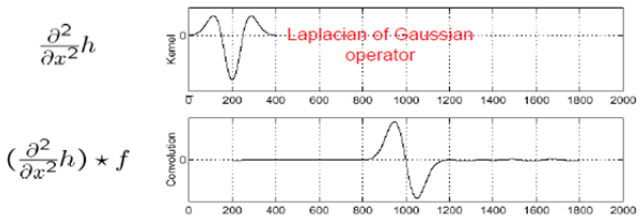
Convolving an image with a kernel for x-axis derivative gives one image; another kernel for y-axis derivative gives a second image. Applying a pixel-wise formula yields a sample. Introducing a threshold allows edge detection – low for all edges, high for selective edges. The Sobel matrix, a weighted average variant, follows a similar principle to Prewitt for edge detection.

2.3.2 Issues to Address: Choose the right THRESHOLD

Determining the appropriate threshold is tied to edge thinning and linking considerations. Striking a balance between smoothing and localization is crucial to avoid stripes and maintain pixel precision. Excessive blurring may lead to edge loss. Eldritch proposed using the second derivative. In the second derivative, changes in the first derivative result in maxima, forming an "area." Contrastingly, with the second derivative, our focus is on zero crossing rather than threshold values.



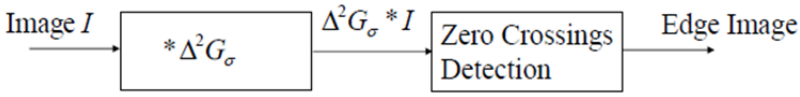
When we apply the second derivative to a Gaussian filter, you get the Laplacian of Gaussian (LoG). This filter exhibits both negative and positive shapes. Notably, at edges, there are neither maxima nor minima but zeroes, signifying the detection of zero transitions.



In two dimensions, applying the Laplacian of Gaussian involves convolving a Gaussian with the Laplacian, and the formula for this operation is as follows (SEE LAPLATIAN.m):

$$\Delta^2 G_\sigma = \frac{1}{2\pi\sigma^2}\left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right)e^{\frac{x^2+y^2}{2\sigma^2}}$$

Marr and Hildreth edge detector:



When examining zero transitions in the Laplacian of Gaussian, we are essentially identifying points where intensity changes from positive to negative or vice versa, indicating the presence of an edge. However, since noise is inherently present in any image, determining an appropriate threshold becomes crucial. Setting a threshold on this slope allows us to distinguish between zero crossings caused by genuine intensity changes and those induced by noise. This introduces a challenge within the problem, as the choice of threshold needs to strike a balance between capturing true edges and mitigating the influence of noise in the image.

2.4 The Canny Edge Detector

Criteria for an "optimal" edge detector include:

- Lower Error Rate: Aim for minimal errors in edge detection.
- Well-Localized Edge Points: Edge points should be accurately localized, pinpointing the exact position of the transition.
- Single Edge Point (Not Stripe): Ensure that a single edge point is identified, avoiding the creation of undesired stripes.

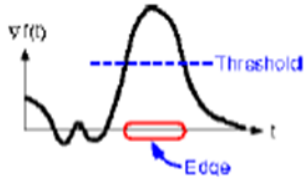
Achieving an optimal filter for these criteria lacks a closed-form solution, relying on approximations. However, it has been demonstrated that the first derivative of the Gaussian closely approximates the optimal solution.

When utilizing the first derivative of the Gaussian, the result is an optimal signal-to-noise ratio and precise localization of edges through linear filters. The typical edge detection algorithm involves the following steps:

1. Filtering with the Derivative of Gaussian: Apply the derivative of Gaussian along both the x-axis and the y-axis to the image.
2. Compute Image Gradient: Calculate the image gradient based on the filtered results.
3. Gradient Magnitude and Orientation: Determine the magnitude and orientation of the gradient at each pixel.
4. Thresholding the Magnitude: Apply a threshold to the gradient magnitude.
5. Non-Maximum Suppression: Thin multi-pixel-wide "ridges" down to a single pixel width. This involves starting with a stripe and obtaining an edge by retaining one pixel.
6. Linking Missing Connections: Utilize hysteresis technique and thresholding to link missing connections between two edges.

This comprehensive approach ensures effective edge detection by leveraging the benefits of the first derivative of the Gaussian, optimizing signal-to-noise ratio and edge localization.

When we have an edge, we have to put the threshold and indeed an edge is not one pixel but a stripe.



In the resulting image, the norm of the gradient magnitude is evident. Upon applying thresholding, certain lines become thinner due to the non-maxima suppression process. This suppression involves examining the edges, determining their directions, and conducting a check to retain only the maximum values. Non-maximum suppression is performed on the entire image, typically considering eight directions (pixels in the surrounding eight positions). This process effectively thins the detected edges down to a single pixel width.

$$NMS(x,y) = \begin{cases} |\nabla I(x,y)| \;\; if \; |\nabla I(x,y)| > |\nabla I(x',y')| \;\&\; |\nabla I(x,y)| > |\nabla I(x'',y'')| \\ 0 \; otherwise \end{cases}$$

To determine the direction of the edges, calculate the ratio of the magnitude of the partial derivative along the y-axis to the partial derivative along the x-axis. Then, use the arctangent to obtain a numerical value representing the direction.

What and how Hystheresis Thresholding works? Hysteresis operates by considering two strong segments of an edge, delineated by a high threshold. In the gap between these segments, where the intensity falls below the threshold, points that reside between the low and high thresholds are evaluated. The decision to create a bridge and establish an edge hinges on whether these intermediate points collectively surpass the low threshold. If all points within this range are above the low threshold, the edge is preserved. Essentially, hysteresis ensures the continuity of edges by selectively connecting intermediate points when they contribute to a coherent edge structure.

The filter size, often represented by the standard deviation or scale, is a crucial consideration. A smaller standard deviation implies a smaller filter size, enabling the detection of smaller objects. Conversely, a larger scale is effective for identifying larger objects.

The significance of standard deviation. A smaller standard deviation captures a multitude of details, including objects that may not be of interest. On the other hand, increasing the standard deviation provides information about the overall structure of the scene, such as buildings. Edges, depicted as pixels, can be perceived as straight lines. While mathematically described as such, the challenge lies in developing techniques that can accurately identify and connect these pixels to form a cohesive straight line, crucial for navigation and interpretation.

In scenarios with numerous potential matches, the **voting technique** becomes valuable. Each feature (pixel) casts a vote if it aligns with a straight line, and the challenge lies in determining a mathematical description that accurately represents these votes, providing a means to describe the detected straight lines effectively.

Example: To describe a straight line, parametric space with slope and constant terms is used. Points in Cartesian space correspond to lines in parametric form. For multiple points on the same line, their intersections in parametric space identify the common line. Voting is then employed, with each point contributing to determine the most significant intersection point, representing the straight line. This is often implemented through for loops on the detected edges.

2.5 Hough Transform

The Hough Transform, a voting technique, effectively addresses challenges in line detection.
Main Idea:

1. Record all potential lines that each edge point may lie on.
2. Identify lines that accumulate the highest number of votes, indicating their significance in the overall image structure.

Example: To describe a straight line, parametric space with slope and constant terms is used. Points in Cartesian space correspond to lines in parametric form. For multiple points on the same line, their intersections in parametric space identify the common line. Voting is then employed, with each point contributing to determine the most significant intersection point, representing the straight line. This is often implemented through for loops on the detected edges.
The Hough transform algorithm:

1. Initialize the $H(d, \theta) = 0$
2. For each edge point (x,y) in the image
3. Fot $\theta = 0 \rightarrow 180$ (sampling) $\rightarrow d = x \cos\theta - y \sin\theta$ (quantization), $H(d, \theta) = H(d, \theta) + 1$
4. Find the value of (d^*, θ^*) where $H(d, \theta)$ is maximal
5. The detected line in the image is given by $d^* = x^* \cos\theta^* + y^* \sin\theta^*$

However, the described approach faces challenges when dealing with vertical lines, as their slope is infinite. To address this, an alternative is to use the polar form of the straight line. In the polar form, the representation accommodates vertical lines.

Moreover, if the edges are not perfect due to noise, resulting in small blobs instead of single points, the Hough Transform remains effective. Despite these imperfections, the approach looks for the maximum votes, ensuring the identification of the corresponding straight line.

3 Scale-space and color

The first concept is scale space analysis, a study aimed at comprehending scale space theory. The primary objective is to automatically determine the size of objects within a scene.

In this context, "**scale**" refers to the size of an object, and the choice of filter size for detection is pivotal. Matching the filter size to the object size is a fundamental concept. For smaller objects, a filter of similar size is necessary, while larger objects require correspondingly larger filters.

The Laplacian of Gaussian can not only detect edges but also objects, we can automate this process. This approach is known as BLOB detection, where a BLOB signifies a distinct entity in contrast to its surroundings.

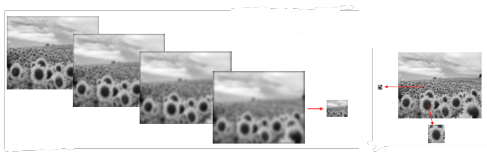
The introduction of color enhances the analysis, especially for area calculations, with a segmentation algorithm based on color detection being an effective and straightforward approach.

3.1 Multi-Resolution Representation

When considering various aspects of an object, such as different sides or resolutions, it becomes necessary to handle images of different sizes. In scenarios where the original image is either smaller or larger than the required size, resizing the image to match the target becomes essential.

In certain scenarios, utilizing a high-resolution image for computation may result in slow and non-real-time implementation of an algorithm. In such cases, downsizing the image becomes necessary for practical reasons. However, if the algorithm is inherently slow due to computational complexity, optimizing the code might be challenging.

To effectively describe various aspects of an image, a multiresolution representation is essential. The fundamental concept is that different scales are suitable for depicting different objects within the image. A full-resolution image enables the detection of smaller objects, given their small size and high-frequency characteristics.



Blurring the image strongly removes small objects, making it suitable for detecting larger objects. Using a smaller filter with a standard deviation of 1, helps in eliminating small noise while retaining fine details in the output image. Changing the filter size, or increasing the scale, removes small objects but preserves larger ones.

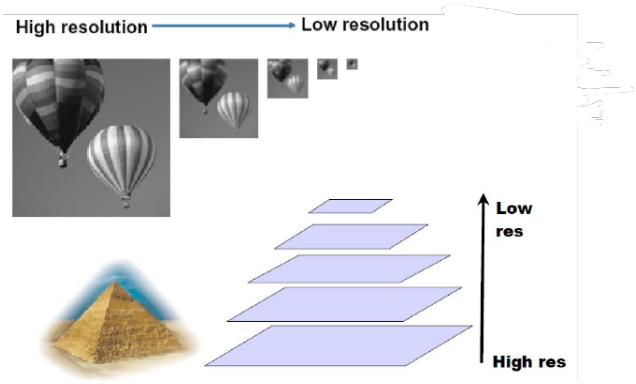
Smoothing, achieved through blurring, removes high-frequency content, resulting in many pixels having the same value. Downsizing the image accelerates algorithms, particularly in areas with uniform color. Once a significant object is found, determining its center of mass allows for higher-resolution analysis to better identify and understand the object.

3.1.1 Gaussian Pyramid

To create a multi-resolution representation, known as Pyramid Representation, we follow these steps:

1. Smooth the image using Gaussian blurring.
2. Downsample the image.
3. Upsample (Interpolation: to fill the empty spaces)

(Subsampling is a bad idea unless we have previously smoothed the image because of aliasing) The resulting set of images, stacked on top of each other, resembles a pyramid. When identifying objects in a scene, focusing on this pyramid allows for a clearer view. If interested in analyzing specific objects, it becomes crucial to locate their positions. Once found, the corresponding values in the high-resolution image can be extracted and analyzed in detail.



When performing down-sampling, caution is required to avoid artifacts like aliasing, caused by insufficient pixels to represent the image (we have the lost of information but is better than aliasing), in adherence to the Nyquist sampling theorem. To mitigate this, it's crucial to remove high-frequency contents by blurring the image before down-sampling. Proper down-sampling ensures that only larger objects survive in the reduced-size image, eliminating high-frequency details in the process.

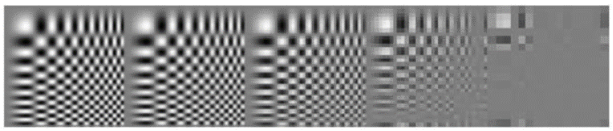
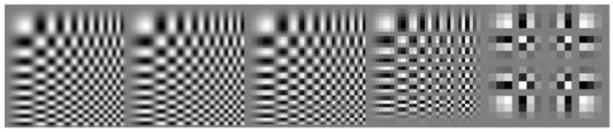


Figure 31: Sampling with smoothing

Performing down-sampling without prior blurring results in a new image containing artifacts. This discrepancy arises because parts of the image become artifacts due to sub-sampling. The original image, being high-frequency, can lead to surviving portions that should be removed, creating aliasing issues. Therefore, it is imperative to consistently apply smoothing before sampling to ensure accurate and artifact-free representation.

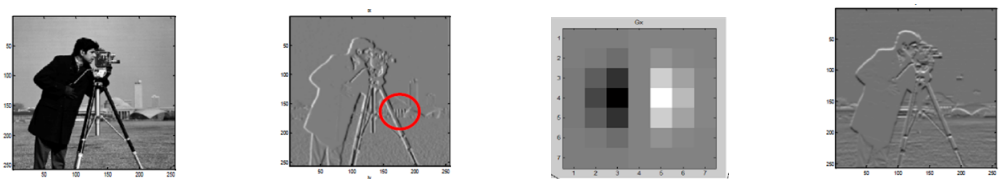


When dealing with a large image and a moving object between frames, detecting the object's speed using a large filter can be slow. To address this, downsizing the image, effectively reducing the distance, allows for the use of a smaller filter to compute speed. While the speed obtained at this level is an approximation (half of the real speed due to the size reduction), it serves as a useful estimate. To enhance speed computation accuracy, the obtained speed is used to apply warping, bringing the object closer to its position in the next frame. This process involves up-sampling, enabling the computation of the actual velocity of the object since these steps are interconnected.

The concept of up-sampling involves computing features like speed or disparity at a lower level, then increasing the resolution to match that of a higher level. This information is utilized in warping the two frames, bringing objects closer for further analysis. By using a small filter, residual speed that wasn't accurately computed at the lower level can be detected.

This illustrates another aspect of scale. To detect small edges, like those related to columns, a small filter is necessary, as its size aligns with the shape of the object. Using a larger filter, which increases the scale, renders it incapable of detecting these small edges and is effective only for identifying larger edges.

3.1.2 Template Matching



In filtering, a filter acts as a template, comparing a small representation of the desired feature with local regions. For instance, the derivative of a Gaussian represents bright and dark edges, defining patterns in image patches resembling the filter. When applied, like in the case of columns being black and white, template matching involves multiplying negative by negative and positive by positive, resulting in a sum that produces the maximum. As the filter size increases, oscillations become confined to one part, causing positive and negative values to cancel out, yielding an output of 0. This clarifies that filtering aims to match the shape of the filter, illustrating why the derivative of a Gaussian identifies edges with a similar shape.

It's crucial to remember that convolution involves mirroring the filter, encompassing both positive and negative parts. This process elucidates why the Laplacian of Gaussian is adept at detecting not only edges but also objects.

3.1.3 Blob Detection

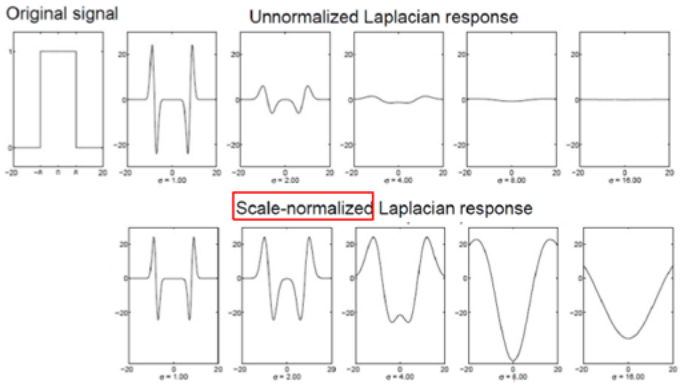
When applying the Laplacian of Gaussian to filter a specific part of an image, maximum or minimum values are obtained. These patches in the image resemble the filter, making it effective for identifying objects. For instance, when focusing on eyes, which are dark surrounded by a bright area, the Laplacian of Gaussian identifies maxima, signifying a significant difference from the surroundings. This ability to discern regions with distinct centers makes it valuable for object identification, often employed through techniques like Blob Analysis.

Using a larger Laplacian of Gaussian aids in identifying larger objects. A simple algorithm can be devised to determine the size of an object in a scene using different-sized Laplacian of Gaussian filters. By filtering and taking the maximum, a correspondence between the filter size and the object size is established.

For example, to find an object of size 2 pixels, where the object is defined as having a value of 1 compared to the surrounding 0, convolving the image with the Laplacian of a Gaussian we will have:

Where there is an absolute maximum, we locate the object. As the size of the object increases, the maximum shifts from the object's center to emphasize more on the edges, given the Laplacian of Gaussian's small size.

When employing a series of Laplacian of Gaussian filters with increasing standard deviation, we might not achieve a maximum response. This is due to the formula, where dividing the Laplacian of Gaussian by the square of the standard deviation decreases the amplitude as the standard deviation increases.



To address this, normalization is essential, accomplished by multiplying it by the square of the standard deviation. By applying the scale-normalized Laplacian, the response becomes more meaningful. The resulting Laplacian of Gaussian with various standard deviations provides an indication of the object's size, offering a way to determine the object's size in the scene.

We define the **characteristic scale** as the scale that produces the peak of the Laplacian response.

For example, consider finding the radius R of a circle. Using the relationship where the standard deviation is divided by $\sqrt{2}$, we can identify the size of an object in the scene by examining the responses of a Bank of Laplacian of Gaussian filters. This allows us to determine the characteristic scale of the object. (Lindeberg)

Indeed, this approach provides an automatic means to identify not just objects but the number of distinct regions in the scene that differ from their surroundings. With knowledge of their sizes obtained through the scale-normalized Laplacian, further, more specific processing can be applied to recognize and characterize each individual object within those regions.
The algorithm for **scale-invariant blob detection** involves:

1. Convolve the image with the scale-normalized Laplacian at multiple scales (FILTER THE IMAGE).
2. Identify the maxima of squared Laplacian responses in scale-space (apply a threshold and consider areas instead of points).
3. Implement non-maxima suppression in scale space to determine the position of the centers.

Smaller objects generate high responses for smaller filters, while larger objects respond to bigger filters. The output provides information for each pixel, helping to recognize the characteristic scale by finding the maximum in the scale dimension. (SEE BLOBS_DETECTION.m) (If the goal is to detect people, using Gaussian filters may be more suitable than circles.)

To avoid directly convolving with the Laplacian of Gaussian (LoG), an alternative is to approximate the LoG by the difference of two Gaussians (DoG). This approach takes advantage of the computations already performed during the pyramid (multi-scale level) generation, where blurring of the image, down-sampling, and up-sampling have already been carried out. The Laplacian Pyramid is constructed by taking the original image, down-sampling it, up-sampling the result, and then computing the difference, effectively approximating the Laplacian of Gaussian.

3.2 Color Fundamentals

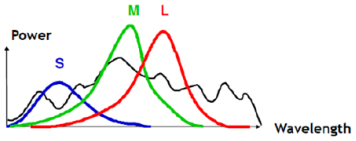
Color is a rich and intricate experience, arising from the human visual system's distinct responses to various wavelengths of light. A specific segment of the electromagnetic spectrum is perceived as color.

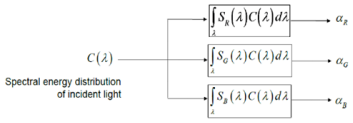
Monochromatic color, such as that produced by a laser, consists of a single wavelength. In contrast, normal daylight we perceive is a mixture of red, blue, and green wavelengths. In the context of image formation, color is intricately linked to the characteristics of light and the surface properties of objects.

Describing color might seem peculiar, but it relies on understanding how individuals respond to different hues. The primary method for gauging this is by asking people whether the colors they perceive are the same.

In our retina there are two types of neurons, RODS handle intensity for low-light conditions, and CONES, sensitive to different wavelengths, allow us to see colors during daylight. The three main types of cones respond to red, blue, and green wavelengths, collectively enabling our perception of color.

To describe the color perception system in our eyes, we examine the response as a function of wavelength, which is the inverse of frequency. This response is akin to the Fourier transform of the eye's filter. By characterizing this system, we can understand how it processes different wavelengths and contributes to our perception of color.



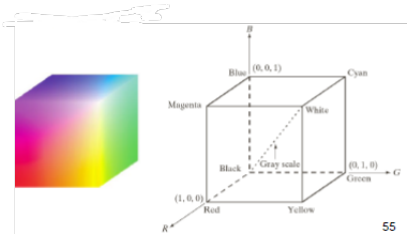


In the figure above we have three-receptor model of color perception, spectral energy enters our eyes, and neurons in our eyes filter the light, providing a numerical representation of the color intensity.

The trichromacy color theory asserts that color matching is a linear process. This theory allows us to establish a color model, with the RGB model designed for color monitors. However, in the case of printers, we utilize the CMY model since color reproduction involves subtraction rather than addition.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The RGB color space can be envisioned as a cube with red, green, and blue as its three axes (24 bits, corresponding to 3 bytes for these three colors). Printers, however, operate differently with cyan, magenta, and yellow as primary colors since they work on subtractive color mixing, unlike the additive approach of RGB.



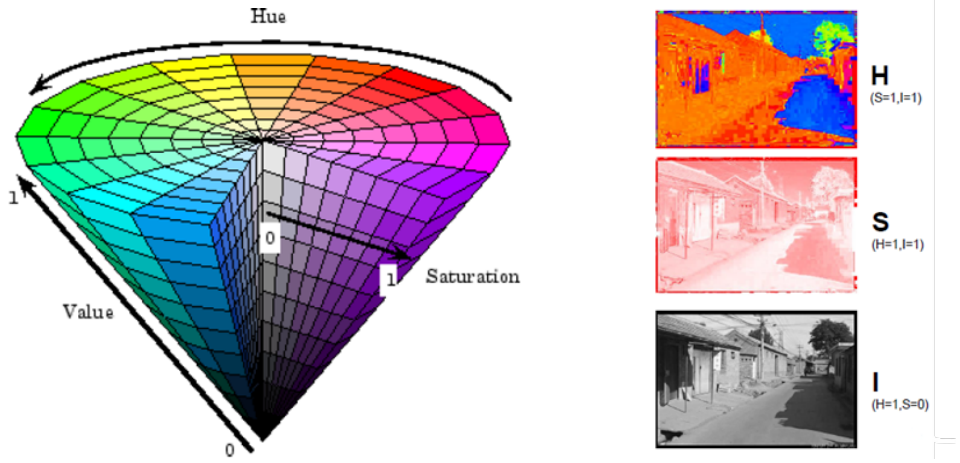
In terms of visual perception, luminance plays a crucial role compared to chrominance. Emphasizing intensity allows us to perceive the structural details of a scene. When we only observe color without variations in intensity, the scene's structure becomes less apparent.

3.2.1 Colour Space: HSI

If we aim to characterize human perception of color, we utilize the HSI color space, which considers not just a single RGB value but combines all three: red, green, and blue. However, for a more perceptually meaningful representation, we prefer a color space where color, saturation, and intensity are distinct channels. This approach is encapsulated in the HSI color space: Hue, Saturation, and Intensity.

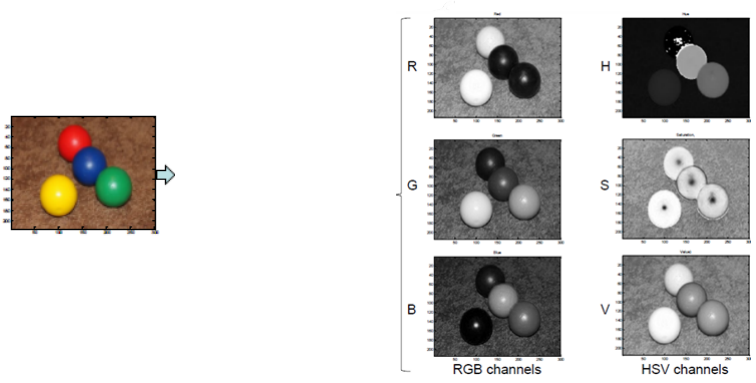
- Hue: The characteristic of color that changes while transitioning from red to green.
- Saturation: The attribute of color that varies between red and pink.
- Intensity: The property that changes between black and pink.

These transformations are not linear to accommodate our nonlinear perception.



It's crucial to analyze the outcomes. In the RGB color space (linear cube), examining its three channels (red, green, and blue) reveals that all channels essentially provide information about the scene's structure rather than what we perceive. On the other hand, in the HSI color space, the three channels distinctly separate intensity and color.

For example, consider a simple image. Viewing its red, green, and blue channels reveals the intensity of red. However, the red and yellow appear similar in this channel, making it impractical for describing color. In contrast, HSI assigns specific gray values to each color, providing a more effective color representation.



If we look at the red ball we have black and white this is because the red is the frontier of the to edges of the circular map. So, we have the red around the zero and the red also around the 255.

Object tracking can involve not just edges but also the area, and image segmentation is a technique used for this purpose. It divides the image into homogeneous regions based on certain properties. For

example, one can segment based on grayscale values, like regions where the grayscale value is between 100 and 120. However, this may not be robust under varying lighting conditions. Using color, such as focusing on red objects, can be more reliable since color remains consistent even with changes in illumination, saturation, or intensity.

The technique involves selecting a small area of the object, computing the mean and standard deviation of the color within that area. This provides a color range, and objects falling within this range are assigned a value of 1, while others get 0. To avoid errors, a region (BLOB) is identified based on geometrical properties like area, considering only larger regions and discarding smaller ones. The centroid or center of mass can then be computed to locate the center of interest. This segmentation method can also be applied by choosing the color of interest.

Geometrical properties of regions: The centroid, or center of mass, serves as a representation of the location of an arbitrarily shaped region. Additionally, the perimeter of a region is calculated as the sum of its border pixels. A border pixel is one that has at least one neighboring pixel from the background.

4 Corner detection and image matching

4.1 Matching

If we want to locate an object or a pattern within an image or video, there are two main techniques:

- **Template matching**: the process involves using a patch that represents the object of interest, which is then compared to different regions within a larger image. This comparison is typically conducted through intensity matching techniques, with normalized cross-correlation being a common method.
- **Feature point or local point**: We identify some critical points within an image are determined. By leveraging a select set of these pivotal points, it becomes possible to identify analogous structures within a larger image. This technique often involves the detection of numerous local key points or feature points.

This significantly transforms our search approach, as the use of a sub-image contrasts sharply with the alternative method of relying solely on points that maintain a specific spatial relationship.

The second technique comprises three integral components:

1. **Key Point Detection** (Corner Detection): This involves identifying key points, particularly corners, which possess distinct and special properties.
2. **Variance or Covariance Analysis**: In the subsequent stage, the algorithm examines the variance or covariance of these detected key points.
3. **Scale-Invariant Region Selection** (Harris-Laplace Detector): The final step involves scale-invariant region selection, exemplified by the Harris-Laplace detector. This optimization enhances both the speed and robustness of the code, ensuring stability even in the presence of varying lighting conditions.

4.2 Blob Detection and Template Matching

When identifying blobs, the filter recognizes regions in an image that resemble the filter. Expanding on this concept, the optimal linear operator for detecting a feature within an image is, remarkably, the sub-image itself. Therefore, the **most effective linear operator for finding an image patch essentially becomes the patch itself**. Subsequently, utilizing this sub-image as a filter enables the search for the specific pattern it represents in another image. This approach aligns with the principles of template matching.

The Laplacian of Gaussian is straightforward in its approach, targeting elements that deviate from their surroundings. When employed with patches as filters, it precisely seeks variations in intensity, facilitating the identification of similar patches in another image. This process is the TEMPLATE MATCHING.

The concept behind template matching revolves around the **pursuit of similarity, not exact matches**. When we have a sub-image representing an object of interest, the objective is to identify similar, though not identical, sub-images in other images. This technique proves effective when the images share sufficient resemblance. However, if variations are substantial due to changes in lighting or perspective, the use of a corner detector becomes necessary.

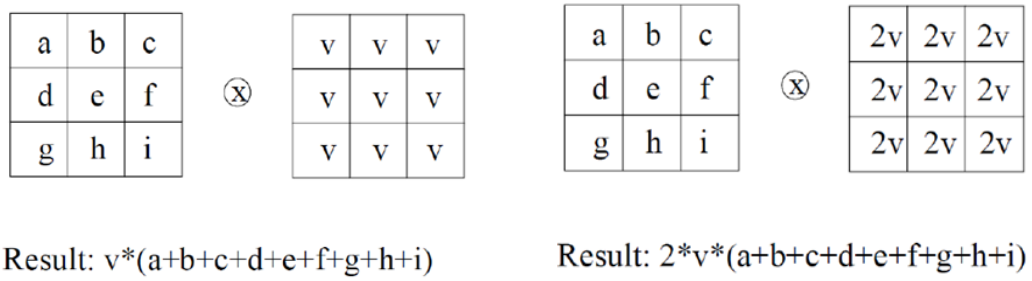
The idea is to apply this formula (not a convolution $\rightarrow$ no mirroring of the filter) but it is a correlation.

$$C_{fg}(i,j) = \Sigma f(u+i, v+j)\; g(u,v)$$

$$R_m(i,j) \;=\; \{u,v \mid x - \frac{m}{2} \leq u \leq x + \frac{m}{2}, y - \frac{m}{2} \leq v \leq y + \frac{m}{2}\}$$

We're basically comparing the patch we're interested in with the image using a method called cross-correlation, which measures how similar they are. In simpler terms, it's like checking how much the patch matches different parts of the image. This can be done with a loop or using functions available in a library. When we're comparing patches in the same image, it's called AUTO-CORRELATION.

For instance, if we use a picture of a face as a template and try to find that face in the same image, it's the simplest case because there are no changes in brightness (all values are the same). However, when we do the correlation, the result might show the maximum blur not on the face but somewhere else in the image. This happens because we can apply correlation without worrying too much about the details of the data. If we calculate the correlation between the face template and a plain-colored image, the results would look like this.



If you double the brightness of an image, the row cross-correlation result will be twice as large, regardless of the image's content. This makes row cross-correlation sensitive to signal amplitude, which is not ideal. To address this, we subtract the mean value of the template from each pixel. Now, we have both positive and negative values, capturing darkness as negative and brightness as positive.

With a zero-mean template, high scores will highlight the face, allowing us to identify the template. However, if we apply this to another template, like searching for a small building, the result may not represent the building itself. This is because we're detecting the maximum brightness, not the template. The issue arises because the contrast in the template is low, and patches with higher contrast will dominate over the template of the small building.

The solution to this problem is to use normalized cross-correlation, which handles changes in light by adjusting intensity. Subtracting the mean addresses the first issue, removing intensity problems with contrast. The second issue is tackled by dividing by the standard deviation, mitigating the effect of contrast changes within the patch. This approach is implemented in TEMPLATE_MATCHING.m.

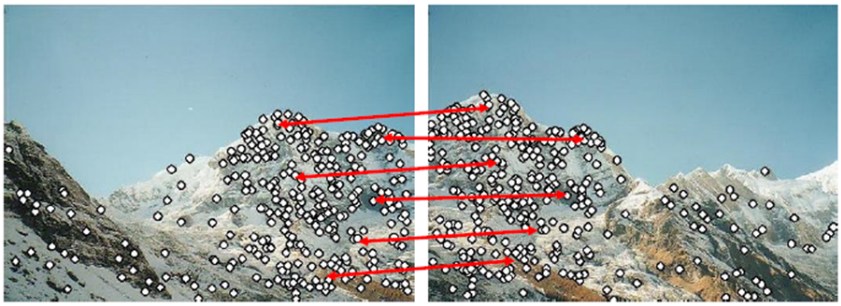
Now, we can detect more complex objects than blobs. Using only the Laplacian of Gaussian (or color), we can identify objects without specific textures. The Laplacian of Gaussian identifies ellipsoids or circles without precise shapes, and using color helps find shapes based on color rather than texture. However, in our case, we seek a specific texture, indicating particular intensity variations in the image. It's important to note that this technique is sensitive to light changes, and in a different frame where the object's position changes, the exact shape may not be the same, especially in videos.

4.3 Feature Point or Local Point

One effective technique is finding stable elements across various views, and these are called feature points. When we observe an object, our eyes focus on specific points, typically at the borders rather than the interior. To incorporate this in a program and detect it through coding, we need a mathematical description. These distinctive characteristics in an image are termed localized features, key points, or interest points.

To identify a point in an image, we need some surrounding context or area. This allows us to recognize the same point in another image (MATCHING.m).

1. **Extracting Keypoints (Detection):** To identify points, we extract keypoints. However, a single point is just one pixel. To make the computer understand, we need information from the surrounding area for point identification.
2. **Matching Keypoint Features (Description):** Identifying the same points in two images involves matching keypoints using surrounding information. Template matching, particularly using the histogram of the gradient (SIFT), is effective for this purpose.
3. **Aligning Images (Matching):** Now that we have the points and their surrounding areas, we can determine the transformation and apply warping. This process, discussed in the first lecture, enables zooming and merging.



Having only a few stable points is crucial. Images may differ in intensity, color, size, etc. Using stable points allows us to focus on their surroundings for characterization.

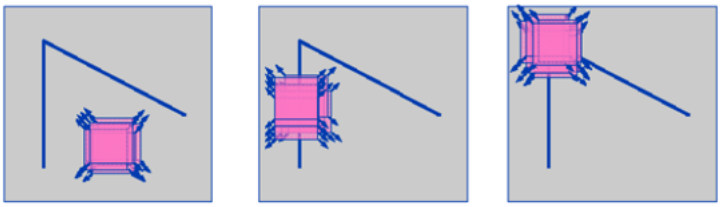
Characteristic of good features:

1. Repeatability and Accuracy: Invariant to translation, rotation, and scale changes, robust or covariant to out-of-plane (affine) transformations, robust to variations in lighting, noise, blur, and quantization.
2. Locality: Features are local, ensuring robustness to occlusion and clutter.
3. Quantity: A sufficient number of regions are needed to cover the object, but fewer than the total image pixels.
4. Distinctiveness: Regions should contain "interesting" and unique structures.
5. Efficiency: Close to real-time performance is essential.

Goal: We want to detect the same points in both images.

Features must be local to maintain stability; otherwise, they become unstable. The number of points is crucial—fewer than the image pixels, specific, and limited for real-time processing.

Identifying the same points in two images is key, done concurrently. A mathematical solution is needed to describe the uniqueness of a patch concerning changes when the window is moved.



This idea is essential to describe what makes a point unique. Points stand out due to differences in their surroundings. In a uniform area, moving a window doesn't change much. Along edges, the image stays the same, but moving perpendicular to an edge changes the image content. Corners, where these changes are most noticeable, are effectively detected by the **Harris Corner detector**.

4.4 Harris Corner Detector

We can describe it mathematically. We look for differences between a point and its neighbors when shifted a few pixels. By highlighting and squaring these differences, using a local window function like a box or a Gaussian, we characterize all windows and sum them. This process averages changes, producing zero energy in uniform areas and vice versa.

$$E(u,v) = \Sigma_{x,y}\, w(x,y)\, [I(x+u, y+v) - I(x,y)]^2$$

If we move in an image with the same color everywhere, the corner detector won't generate energy. The energy surface, represented by $E(u,v)$, will be flat. In contrast, moving in an image full of corners will result in different sub-patches, producing energy.

To simplify the description, we use a mathematical formulation. By considering a similar paraboloid and applying the Taylor series, we approximate the shift using the 1st order Taylor approximation. This involves the image itself plus the partial derivatives multiplied by the shift. Plugging this formula into our energy removes the image, giving us a weighted sum of the image gradients:

$$I(x+u, y+v) \approx I(x,y) + I_x u + I_y v$$

In a uniform area, where the gradient is 0, there's no description. If there's a gradient difference, it's captured in the energy, providing a metric for detection. Manipulating this equation further gives us a description of a metric.

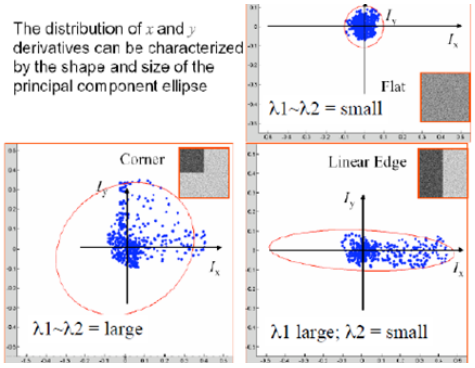
$$M = \begin{bmatrix} \Sigma I_x I_x & \Sigma I_x I_y \\ \Sigma I_x I_y & \Sigma I_y I_y \end{bmatrix} = \Sigma \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} = \Sigma \nabla I (\nabla I)^T$$

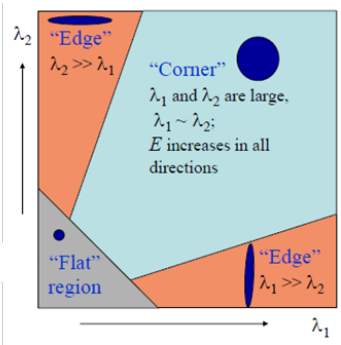
The two-by-two metric is formed by summing the partial derivatives inside the patches. Known as the second moment metrics or structure tensor, it's a tensor describing the image structure. This tensor mathematically encapsulates the idea of finding something that changes when we move around a specific point, particularly in comparison to flatter regions and edges.

$$M = \Sigma_{x,y}\, w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

This method describes corners using a matrix composed of partial derivatives. For an axis-aligned corner, the matrix shape resembles a paraboloid, representing the gradient of the image. The intensity of the derivatives informs us about the content in the patches.

For example, in three synthetic images with an edge, flat area, and corner, performing derivatives and plotting their intensity reveals that in a flat area, where the partial derivatives are small and similar, the points form a circular pattern around the origin.





In the presence of an edge, the horizontal derivative is more significant (in the case of a vertical linear edge). There won't be any points on the Y-axis, and we'll observe an elongated ellipse. In this scenario, a and b are different, with one being larger.

In the case of a corner, both partial derivatives have high values, resulting in a circular distribution. The eigenvalues a and b of the matrix are also large in this case.

The **relationship between eigenvalues and the type of region** is as follows:

- A flatter region is described by small and similar eigenvalues.
- An edge region is described by different eigenvalues, where one is small and one is large.
- A corner region is described by both eigenvalues being large.

This relationship holds even when the energies are not aligned with the axes. To find the orientation of the ellipsis axis and the eigenvalues, diagonalizing the matrix is required.

In the M matrix (second image), we can identify the eigenvector that represents the direction of the edge and the eigenvalues that signify the strength.

We can define a measure that indicates the presence of a corner in the image called "**cornerness**." Mathematically, this corner measure is the product of the two eigenvalues minus the square of their sum, multiplied by a certain parameter. This parameter is determined through fitting, typically falling between 0.04 and 0.06. This explanation can be simplified by this computation:

$$R = det(M) – \alpha * trace(M)^2$$

Summary: We compute the M metrics for all the pixel and the get the corresponding R score, the output is an image where is a store the score then put the threshold on the score and the local maxima are the key points. (See HARIS_CORNER.mat)

We can utilize cornerness to distinguish different regions in the image. A positive value of R indicates a corner, a negative value suggests an edge, and a small R implies a flatter region. However, what constitutes small or large depends on the image, and a threshold needs to be set.

For example, consider the same image that is rotated and has varying intensity. Identifying the face using template matching with patches is challenging due to the differences in intensity and rotation. However, by identifying specific stable points in both images, we can effectively recognize the object.

In the corner detector process (generating the cornerness map R), we need to set a threshold to distinguish corners. For points where R is larger than the threshold, those are considered corners. To identify the same point, we locate the maximum values using local maxima suppression.
The algorithm for the detector involves:

1. Computing the M matrix (using derivatives).
2. Defining the cornerness function (using eigenvalues).
3. Applying a new maximum suppression to identify only the maximum values.

Using this algorithm allows us to identify identical points in two vastly different images. To assess the robustness of these points, we must establish the concepts of invariance to photometric transformations and covariance to geometric transformations.

- **Invariance:** The corner locations remain unchanged even when the image undergoes transformations, such as changes in lighting.
- **Covariance:** In cases where two transformed versions of the same image exist, features should be detected in corresponding locations.

4.4.1 Properties of the Harris Corner Detector

We can alter the intensity of an image by adjusting its brightness or changing the contrast. If we modify the intensity, the algorithm remains invariant; however, if we alter the contrast, it loses invariance (thus becoming sensitive to contrast changes, i.e., scaling).

In the case of translation, a corner moves by the same amount as the image translation, maintaining covariance with the translation.

Rotation is handled by rotating the corner. Diagonalizing the matrix and reapplying edge detection ensures the corner's covariance with rotation.

The challenge arises with zooming, as the corner ceases to be a corner, breaking covariance. To address this, we employ scale space analysis of the Laplacian of the Gaussian. This allows us to automatically compute the characteristic size of an object and use it to identify matching corners. By utilizing a multi-scale approach, we analyze the image at different scales and identify the scale with the maximum response. Once we know the characteristic scale, we can compare corners by warping and zooming them to the same size. The Laplacian of a Gaussian provides the solution by indicating where the maximum occurs, corresponding to the object's dimensions.

5 Single View Geometry

In the realm of computer vision, the pursuit of comprehending the three-dimensional world from a two-dimensional image has been an enduring challenge. At the heart of this quest lies the intricate domain of Single View Geometry, which we will threat in this chapter: we are going to derive the relation between a point in the 3D world and a point in a image plane (2D).

Giving a look at the image below we define $\mathbf{X}(X, Y, Z)$ as the point in the 3D world and $\mathbf{x} = (x, y)$ as the 2D point.

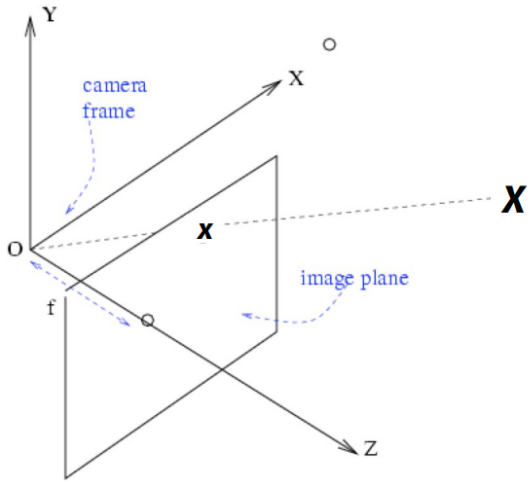


Figure 39: Enter Caption

We call f, which is the distance between image plane and origin of the camera's ref. frame, the **focal length**; it controls the type of acquisition that you can make. The origin of the camera's reference frame is called **camera center**. Looking the scene from another perspective (plane y-z) we denote a similarity between the 2 triangles OZY and Ofy.

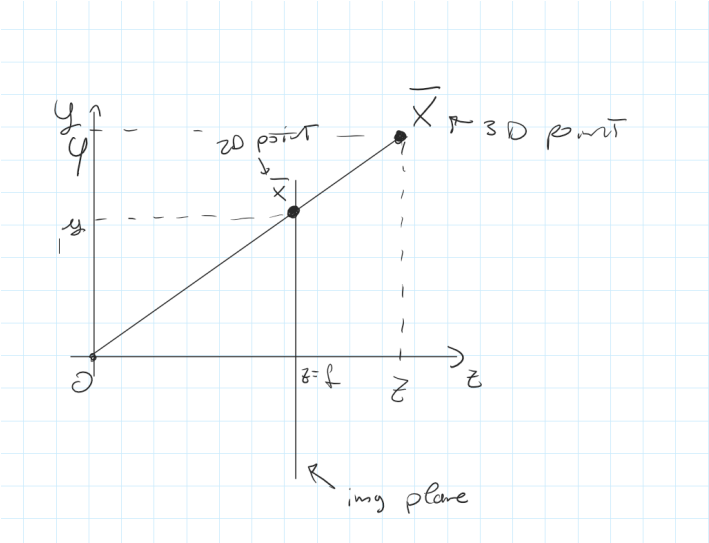


Figure 40: Enter Caption

Because of this we can write the following proportion: $\frac{Y}{Z} = \frac{y}{f}$. Knowing that the same thing can be done in the pane x-z, we obtain:

$$\begin{cases} x = f\frac{X}{Z} \\ y = f\frac{Y}{Z} \\ z = f \end{cases}$$

We now want to formulate the mapping between the 3D point (X, Y, Z) and (x, y):

$$(X, Y, Z) \rightarrow (x, y)$$

We can express them in homogeneous coordinates by adding a 1 dimension to these two points with value 1:

$$(X, Y, Z, 1) \rightarrow (x, y, 1) = (f\frac{X}{Z}, f\frac{Y}{Z}, 1) = (fX, fY, z)$$

We can formulate it with a matrix:

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

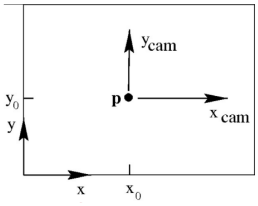
More compactly $\mathbf{x} = P\mathbf{X}$, where P is called **camera projection matrix** and can be decomposed as $diag(f, f, 1)[I_{3x3}|0_{3x1}]$.

A general model for a camera, to represent precisely the 3D world, should take into account 2 main factors:

- In general 3D points are not represented w.r. to camera' reference frame, this implicates that we have to do a rigid transformation between cam and 3D world's ref frame (EXTRINSIC PARAMETERS)
- The transformation from metric coordinates to pixel coordinates (INTRINSIC PARAMETERS)

We can say that calibrating a camera means estimate this two type of parameters.

Intrinsic parameters : in general, differently from how we did before, the origin of the image is not the principal point, so we need to add a translation in the P matrix:



$\mathbf{x} = \mathrm{P}\ \mathbf{X} \qquad \mathrm{P} = \mathrm{K}\ [\mathrm{I}\ ;\ \mathbf{0}]$

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

These quantities (distance of the focal length and position of point **p** must be expressed in pixel in stead of the unit of measure of length, so:

$$\begin{cases} \alpha_x = fm_x \\ \alpha_y = fm_y \\ x_0 = m_x p_x \\ y_0 = m_y p_y \end{cases}$$

Extrinsic parameters As said before, we did a big assumption: we considered the point from the point with respect to the camera's ref. frame, but generally it's not this. In fact usually the point is represented in a world reference frame, so we need to transform it using a transformation matrix in order to represent it with coordinates in camera's frame: $\mathbf{X_{cam}} = R(\mathbf{X_w} - \mathbf{c})$, where $\mathbf{X_{cam}}$ is $\mathbf{X}$ in camera's frame, $\mathbf{X_w}$ is $\mathbf{X}$ in world's frame, R is the rotation matrix from world's frame and camera's and $\mathbf{c}$ is the translation between the two frames.

Because of this we can write that:

$$P = K[R; \mathbf{t}]$$

5.1 Calibration pattern

How we may compute camera calibration? We need to associate a set of 3D points with the correspondent point in 3D world.

Taken the ref. frame, we can express the image points w. r. to the origin of the reference frame. It consists in making association between $\mathbf{x_i}$ and $\mathbf{X_i}$ for i = 1,2,...,N: we will obtain a set of the type $\{\mathbf{x_i}, \mathbf{X_i}\}_i = 1^n$ (both the points are expressed in homogeneous coordinates). The procedure is aimed to estimate the projection matrix P and then exploiting the decomposition of the matrix, i.e. K, R and the translation vector $\mathbf{t}$. P has 11 degrees of freedom, and for each correspondence we will have 2 equations; because of this we need to have $N \geq 6$.

5.1.1 Estimation of the projection matrix

We can certainly say that for i=1,2,..,N, $\mathbf{x_i} = P\mathbf{X_i}$. The fact that they're expressed in homogeneous coordinates means that the equation is true up to a scaling factor: so they're parallel vectors and $\mathbf{x_i} \times P\mathbf{X_i} = 0$. We can rewrite this last formula as:

$$\begin{bmatrix} \mathbf{0}^\top & -w_i\mathbf{X}_i^\top & y_i\mathbf{X}_i^\top \\ w_i\mathbf{X}_i^\top & \mathbf{0}^\top & -x_i\mathbf{X}_i^\top \\ -y_i\mathbf{X}_i^\top & x_i\mathbf{X}_i^\top & \mathbf{0}^\top \end{bmatrix} \begin{pmatrix} \mathbf{P^1} \\ \mathbf{P^2} \\ \mathbf{P^3} \end{pmatrix} = \mathbf{0}$$

where $\mathbf{P^1}, \mathbf{P^2}, \mathbf{P^3}$ are the rows of P but as column vector. We end up with a homogeneous system, where, for example, if N=6, the size of A is 12x12 and of $\mathbf{p}$ is 12x1.

$$A = \begin{bmatrix} \mathbf{0}^\top & -w_1\mathbf{X}_1^\top & y_1\mathbf{X}_1^\top \\ w_1\mathbf{X}_1^\top & \mathbf{0}^\top & -x_1\mathbf{X}_1^\top \\ \cdots & \cdots & \cdots \\ \mathbf{0}^\top & -w_N\mathbf{X}_N^\top & y_N\mathbf{X}_N^\top \\ w_N\mathbf{X}_N^\top & \mathbf{0}^\top & -x_N\mathbf{X}_N^\top \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} P_{11} \\ P_{12} \\ \vdots \\ P_{33} \\ P_{34} \end{bmatrix} = \begin{bmatrix} \mathbf{P}^1 \\ \mathbf{P}^2 \\ \mathbf{P}^3 \end{bmatrix}$$

DLT (Direct Linear Transformation) algorithm We want to minimize the difference between $\mathbf{x_i}$ and the projection of the 3D point $P\mathbf{X_i}$, so minimize $||A\mathbf{p}||$, with $||\mathbf{p}|| = 1$. Then we transform $A = UDV^T$, with U and V that are orthonormal matrices, and in D you can find the singular values of A in descending order. So, we have to minimize $||DV^T\mathbf{x}||$ with $||V^T\mathbf{x} = 1 \rightarrow$ (substitution) $||D\mathbf{y}||, ||\mathbf{y}|| = 1$. Because of the composition of D, it's obviously that to minimize this quantity y should be $(0, 0, ..., 1)$ (so that the products is the smaller singular value).

5.1.2 Matrix decomposition

If we want to extract the intrinsic and extrinsic parameters from $P = [M| - \mathbf{p}]$, where M is a 3x3 matrix:

- M gives the intrinsic parameters, from which we obtain R because of $P = K[R| - \mathbf{t}]$;
- the center can be computed as the null space of P.

5.2 Wrapping up

- Camera calibration is the process of estimating camera parameters from "ad hoc" images
- It allows us to (geometrically) relate the images acquired from the camera with the real 3D world
- Many available algorithms... today we discussed a classical procedure (estimate P and then extract the parameters)
- A user-friendly camera calibration tool is available in Matlab, try and play with it

6 Planar worlds - Homographies

6.1 Introduction

Synchronized images of the same scene can provide depth information or can help create a 3D reconstruction of the scene. This is an important part of 3D geometrical computer vision. There are some special cases where we cannot extract depth information from multiple views. One of these special cases we can use to compare two images in a sequence is the case when the corresponding points in different images are related by a projective transformation or homography. This is the case for a planar scene or when the images are acquired by a purely rotating camera.

6.2 Types of transformations

The four types of transformations are as follows:

Type	Block Matrix	DOF	Invariants
Isometries	$\begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	3	Length, angles, area
Similarities	$\begin{bmatrix} sR & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	4	Angles, ratios of lengths, ratios of areas
Affinities	$\begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	6	parallel lines, ratios of length of parallel line segments, ratios of area
Projectivities	$\begin{bmatrix} A & \mathbf{t} \\ \mathbf{v}^T & v \end{bmatrix}$	8	cross-ratio

6.2.1 Isometries

Isometries are transformations of the euclidean plane preserving the euclidean distance. They consist of a rotation and a translation.

Representation:

$$\begin{pmatrix} x_1' \\ x_2' \\ 1 \end{pmatrix} = \begin{bmatrix} \epsilon\cos\theta & -\sin\theta & t_x \\ \epsilon\sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} \qquad \epsilon = \pm 1$$

Block representation of H

$$H_E = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

Degrees of Freedom: 3 (θ, t_x, t_y), can be computed with two point correspondences
Invariants: length, angles, areas

6.2.2 Similarities

Similarities are isometries with isotropic scaling by scale factor s.

Representation:

$$\begin{pmatrix} x_1' \\ x_2' \\ 1 \end{pmatrix} = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} \qquad \epsilon = \pm 1$$

Block representation of H

$$H_S = \begin{bmatrix} sR & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

Degrees of Freedom: 4 (θ, s, t_x, t_y), can be computed with two point correspondences
Invariants: angles, ratios of lengths, ratios of areas

6.2.3 Affinities

Non singular linear transformation followed by a translation (also called an affine transformation).
Representation:

$$\begin{pmatrix} x_1' \\ x_2' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} \qquad \epsilon = \pm 1$$

Block representation of H

$$H_A = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

Degrees of Freedom: 6 (a_{11}, a_{12}, a_{21}, a_{22}, t_x, t_y), can be computed with three point correspondences
Invariants: parallel lines remain parallel, ratios of lengths of parallel line segments, ratios of areas
Deciphering the Affine Transformation: If we perform an SVD composition on A, it can be used to gain better intuition on the transform being performed. We can add a V^TV to the left of D because V^TV is the identity.

$$A = UDV^T \tag{5}$$
$$= UV^TVDV^T \tag{6}$$

We define the following:

$$UV^T = R(\theta) \tag{7}$$
$$V^T = R(\phi) \tag{8}$$

Then we can redefine A as:

$$A = R(\theta)R(-\phi)DR(\phi) \tag{9}$$

This shows that the affine transformation involves stretching along axes rotated by ϕ followed by a rotation by θ. Then the translation t_x, t_y is applied.

6.2.4 Projectivities

A general non-singular transformation of homogeneous coordinates.
Representation:

$$\begin{pmatrix} x_1' \\ x_2' \\ x'3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{31} \\ h_{21} & h_{22} & h_{32} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} \qquad \epsilon = \pm 1$$

Block representation of H

$$H_P = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{v}^T & v \end{bmatrix}$$

Degrees of Freedom: Though there are 9 unknowns in the H matrix, the H matrix produces the same transformation if it is linearly scaled, therefore there are only 8 degrees of freedom required to define the H matrix. 4 point correspondences are required.
Invariants: cross-ratio (ratio of ratio) of lengths
Note on ideal points: For the other types of transforms and ideal point (point at infinity, where the third component of the homogenous coordinate is 0) will be mapped to another ideal point. For homographies this is not always the case.

$$H_P = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{v}^T & v \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 0 \end{pmatrix} = \begin{pmatrix} A \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ v_1x_1 + v_2x_2 \end{pmatrix}$$

6.3 Homography

Homography also called the projective transformation is an invertible mapping from points P^2 in homogenous 3d vectors to points in P^2 homogenous 3d vectors. Maps a line as a line so if three points (x_1, x_2, x_3) lie on a line in the original image, they also lie on a line in transformed image (x'_1, x'_2, x'_3).

6.3.1 Properties of the Homography

- H is invertible
- The inverse of a homography is a homography
- The composition of two homographies is a homography
- Homography is a linear transformation of $\mathbf{x}$ in P^2 in homogenous coordinates
- Any invertible linear transformation is a projectivity.

Theorem: A mapping h: $P^2 \rightarrow P^2$ is a homography iff there exists a non-singular 3x3 matrix H such that for each $\mathbf{x}$, $h(\mathbf{x}) = H\mathbf{x}$

This can be proved in two steps:

1. Verify that any invertible linear transformation of homogenous coordinates is a projectivity (proved below)
2. Each projective transformation arises in this way (proof skipped in the lecture)

Proof:

- Assume $\mathbf{x_1}$, $\mathbf{x_2}$, $\mathbf{x_3}$ lie on the line $\mathbf{l}$.
- Thus $\mathbf{l}^T\mathbf{x_i} = 0$ for $i = 1, 2, 3$.
- Add $H^{-1}H$: $\mathbf{l}^T H^{-1} H\mathbf{x_i} = 0$
- This means the point $H\mathbf{x_i}$ lies on the line $\mathbf{l}^T H^{-1}$ for $i = 1, 2, 3$.
- Therefore the co linearity of the points after the transformation is preserved.

6.3.2 The Homography Transformation Matrix

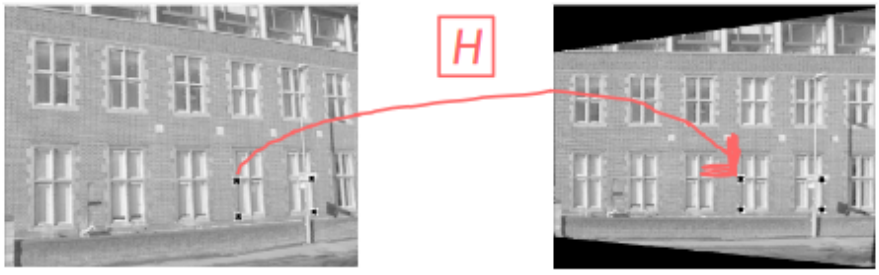
$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{31} \\ h_{21} & h_{22} & h_{32} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{10}$$

$$\mathbf{x}' = H\mathbf{x} \tag{11}$$

Multiplying H by a non-zero scalar does not alter the transformation. Therefore even though there are 9 matrix entries, there are only 8 DOF. We can describe H as a homogenous transformation.

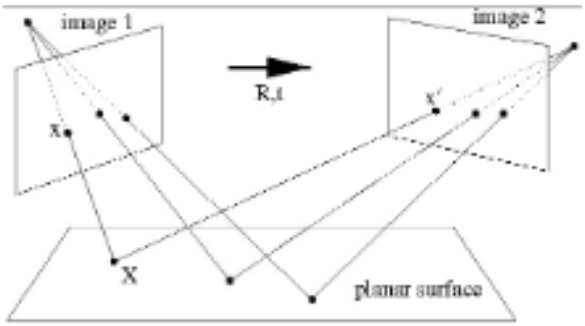
6.3.3 Examples and Use Cases

1. Correcting perspective distortion of a single image
Projection of plane of the building face in the original image onto the image plane is a projective transformation.



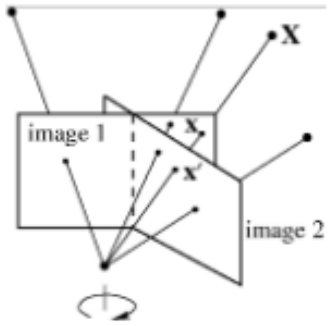
Identify four points on original image and then manually select on the other image a rectangle for the original image points to match to. Then the inverse of the homography derived from those correspondences is applied to each point of the synthetic image to get the value for each of the pixel.

2. Transformation between two images of a planar scene



This exploits the fact that the composition of homographies is a homography. There is a homography from image 1 to the plane and a homography from image 2 to the plane so there is a homography from image 1 to image 2.

3. Two images obtained by a camera rotating about its optical center (no translational component)



The rays of projection all go to the point around which the camera is rotating. This allows the homography to still function even though the points are not all in the same plane.

6.4 Estimating the Homography

6.4.1 Problem Statement

Given a set of point correspondences $\mathbf{x}$, $\mathbf{x}'$, estimate H 3x3 matrix such that $\mathbf{x}' = H\mathbf{x}$. We need at least 4 point correspondences because the homography has 8 degrees of freedom. If we have exactly 4 correspondences then we can calculate the exact solution (solving the minimal problem). If we have more, then we can find the best transformation by minimizing a cost function.

6.4.2 Direct Linear Transform (DLT) Algorithm

We know that $\mathbf{x}'_\mathbf{i} = sH\mathbf{x}_\mathbf{i}$ where s is a scaling factor because they are homogeneous coordinates. We need to pick a quantity to solve for that is invariant to this scaling difference. Therefore we chose this condition: $\mathbf{x}'_\mathbf{i} \times H\mathbf{x}_\mathbf{i} = 0$ because $\mathbf{x}'_\mathbf{i} \times s\mathbf{x}'_\mathbf{i} = 0$

We define:

$$H = \begin{bmatrix} \mathbf{h_1} \\ \mathbf{h_2} \\ \mathbf{h_3} \end{bmatrix} \qquad \mathbf{x}' = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} \qquad \mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{x}' \times H\mathbf{x} = \begin{bmatrix} x'_2\mathbf{h_3}\mathbf{x}^T - x'_3\mathbf{h_2}\mathbf{x}^T \\ -x'_1\mathbf{h_3}\mathbf{x}^T + x'_3\mathbf{h_1}\mathbf{x}^T \\ x'_1\mathbf{h_2}\mathbf{x}^T - x'_2\mathbf{h_1}\mathbf{x}^T \end{bmatrix}$$

Isolating $\mathbf{h}$ the equations for a single point correspondence:

$$\begin{bmatrix} \mathbf{0}^T & -x'_3\mathbf{x}^T & x'_2\mathbf{x}^T \\ x'_3\mathbf{x}^T & \mathbf{0}^T & -x'_1\mathbf{x}^T \\ -x'_2\mathbf{x}^T & x'_1\mathbf{x}^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} \mathbf{h_1}^T \\ \mathbf{h_2}^T \\ \mathbf{h_3}^T \end{bmatrix} = \mathbf{0}$$

This is overdefined so for each point correspondence we only take the first two rows for each point correspondence. Concatenating the two rows for each correspondence generates the A matrix.

- For each point we get two homogenous equations
- Using all the points we get a system of equations $A\mathbf{h} = 0$
- We find $\mathbf{h}$ such that $A\mathbf{h} = 0$ and then transform h back into H
- In the ideal case $rank(A) = 8$ the 1-dim null space contains the non-trivial solution to the system.
- In an overdetermined system, look for a that minimizes $||A\mathbf{h}||$ such that $||h\mathbf{h}|| = 1$

6.4.3 Solving the system for h

Minimize $||Ah||$ subject to $||h|| = 1$

1. Using the SVD decomposition, let $A = UDV^T$
2. Switch to solving the equivalent problem of minimizing $||DV^T\mathbf{h}||$ subject to $||V^T\mathbf{h}|| = 1$
3. Change variable to minimizing $||D\mathbf{y}||$ subject to $||\mathbf{y}|| = 1$
4. Because D is diagonal with elements in descending order, the solution which minimizes the norm is $\mathbf{y} = (0, 0, 0, 0, ..1)^T$
5. Thus the solution $\mathbf{h}$ is the last column of the V matrix (not the V^T) matrix

6.4.4 DLT with normalization

To optimize the performance, it is best to normalize the points by centering them about (0,0) such that the average distance from the center is $\sqrt{2}$. For this it is better to use only finite points

- Compute T and T'
- Normalize the points to get (${}_bfx_{norm}$, $\mathbf{x'}_{norm}$)
- Calculate H_{norm} using the DLT algorithm from ($\mathbf{x}_{norm}$, $\mathbf{x'}_{norm}$)
- Calculate $H = T'^{-1}H_{norm}T$

6.4.5 Robust Estimation

Some of the correspondences can be wrong because of errors in matching points or because some points are outside the plane. If we have more than four correspondences we want to be able to find the best correspondences. We can do this using the RANSAC method (RANdom SAmpling Consensus).

1. Given a set of N points pairs ($\mathbf{x_i}$, $\mathbf{x'_i}$)
2. Select a subset of M pairs (M is the minimum size allowed by the algorithm, for DLT M=4)
3. Estimate H on the set of M pairs
4. Compute the error obtained by applying H to all pairs by computing the geometric reprojection error
   (a) This is the euclidean distance between $\mathbf{x'_i}$ and $H\mathbf{x_i}$
   (b) count the number of points that have a distance less than the threshold (inliers)
5. Repeat for K iterations and keep the H producing the fewest outliers

6.5 Notes from the Lab

1. The first part of the lab was correcting perspective distortion in images. First we selected four point that were the corners of a rectangle on the original image and then we selected four points that were the corners of a the rectangle in the image plane. The homography transformation then corrects the perspective distortion. This also shows why inverse mapping is better than direct mapping because the direct mapping image had a lot of gaps.
2. The second exercise had us use the homography to find the transformation between two images in a sequence. From this we learned that it is important that the surface that the corespondent points are on needs to be a planar surface. In dataset 5 with the notebook and the figurine, the notebook matches nicely but the figurine does not because it is three dimensional.
3. The third exercise shows how homography can be used for mosaicing images taken by a purely rotating camera. In this case, the scene does not need to be planar for the homography to work well because the camera position does not change.

7 Image matching

7.1 Introduction

In general terms image matching is a way to estimate the similarity between image pairs. Similarity is usually estimated with image matching process where we try to match one image with another.

Two main approach to estimate similarity:

- Global matching.
- Local matching.

7.1.1 Global Matching

Compute global descriptor, such as color histograms, color mean and variance or brightness, for each image and estimate the similarity between descriptors (e.g. Histogram intersection)

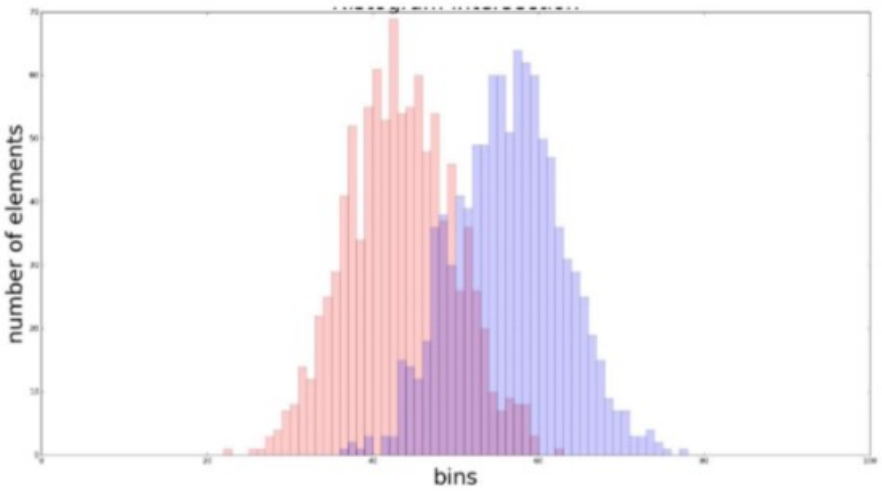


Figure 41: Caption

For normalized histograms, histogram intersection is in the range [0,1]

7.1.2 Local Matching

Extract local feature correspondence problem.

It involves in two decisions:

1. which element to match:
   - All pixels from an image (dense correspondences), look for correspondences for all points in the image. Dense correspondences are in general applied to stereo pairs.
   - A subset of pixels meeting some requirements (sparse correspondences)
2. which feature description + similarity measure to adopt:
   - If image pairs are similar enough, image patches are an appropriate feature description
   - Other feature descriptors, such as SIFT, are more suitable to deal with scale, rotation and view point changes.

7.2 features descriptor

The simplest feature descriptor is the **image patch** surrounding the key point.

Invariant descriptors:

7.2.1 SIFT (scale invariant feature transform)

The idea is to detect interesting points on a multi scale image representation, and then associate with them a vectorial description which is invariant to translation, scale and rotation changes. It is very robust, fast and efficient. SIFT is also tolerant to illumination changes.

Each local feature is represented collecting the information of the gradient around its location, considering a patch of size corresponding to its scale and rotated according to its orientation.

- Scale-space extrema detection: This step involves finding the maximum and minimum "extrema" points in scale space, a method used in computer vision which looks at an image at different scales (sizes) to find features. The term "DoG keypoints" refers to the Difference of Gaussians method used to find these points.
- Accurate key point location: This step refines the location of the keypoints found in the previous step. It involves sub-pixel analysis, which is a method to find a more accurate location of the keypoint within the pixel. It also involves removing keypoints that have low contrast or are located along edges, as these are less reliable.
- Keypoint orientation assignment: This step assigns an orientation to each keypoint based on the local image gradient directions. This ensures that the keypoints are rotation-invariant, meaning the same features can be recognized even if the image is rotated.
- Keypoint descriptor: This step involves creating a descriptor for each keypoint, which is a numerical representation of the keypoint's local image patch. This descriptor can then be used to compare keypoints between different images to find matches.

7.3 features matching

We start from a set of features extracted from the two images.

$$F_{I_1} = \{f_i\}_{i=1}^{N} = \{(x_i^1, y_i^1, Q_i^1)\}_{i=1}^{N} \tag{12}$$

$$G_{I_2} = \{g_j\}_{j=1}^{M} = \{(x_j^2, y_j^2, Q_j^2)\}_{j=1}^{M} \tag{13}$$

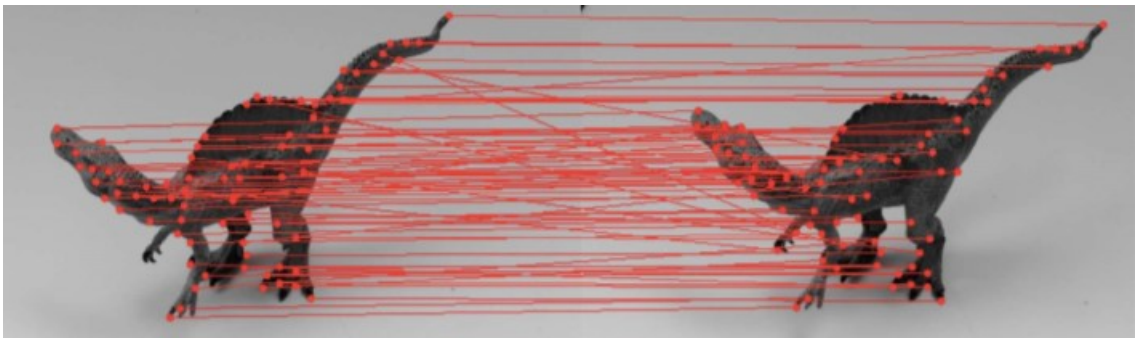


Figure 42: Caption

It is possible to look for match through an affinity matrix, let S be a similarity measure between feature descriptors, we can build an N$\times$ M affinity matrix A so that:

$$A_{ij} = S(f_i, g_j) \tag{14}$$

where f_i are the row of the first image and g_j are the column of the second image.
We may look for one-to-one matches on A:

- fix a certain row;
- look for the maximum of each row of A

- check that the identified value is the maximum of its column too. If so, and if the matrix value is above a threshold, a match is detected.

We may compare features considering their position and/or their descriptors. To improve analysis S should return values in a predefined compact range, e.g. [0,1].

7.4 Examples of similarity

7.4.1 Comparing feature by positions

$$E(i,j) = e^{-\frac{||p_i^1 - p_j^2||^2}{2\sigma^2}} \tag{15}$$

The matrix E has value in [0,1]. Each entry measures how spatially close two points from the two images are. Sigma controls the maximum distance we are willing to consider, allow us to choose the size of the window that we use to look for a certain point.

7.4.2 Comparing features by patches

Let us assume the feature descriptor is a neighbouring patch. Let N1 and N2 be two square image patches of size W × W, we can measure the similarity using:

- Sum of squared differences
- Normalized cross correlation

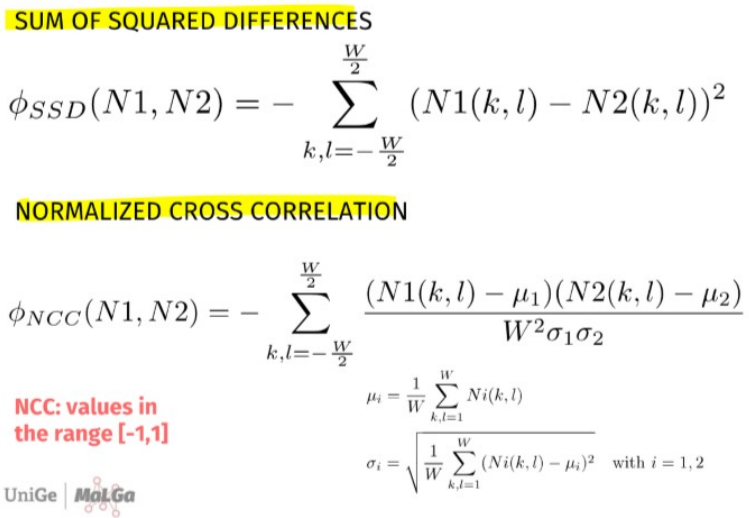


Figure 43: Caption

It is also possible to combine the previous formulas in the following way:

$$A(i,j) = E(i,j) \times \frac{1}{2}(\Phi_{NCC}(Q_i^1, Q_j^2) + 1)$$

Figure 44: Caption

If you want to reduce the dependence from the euclidean distance you can use the following formula with a low value of α

$$A(i,j) = \alpha E(i,j) + (1-\alpha)\frac{1}{2}(\phi_{NCC}) \tag{16}$$

8 Stereopsis

The Stereo Vision is the problem of extracting 3D information (structure and distances) from two or more images captured from different viewpoints (example: how humans use both eyes). So, a Stereo System is a system in which you have at disposal two cameras rather than just one.

1 scenario) a couple of cameras in different position;
2 scenario) a camera which is moving (scene).

Problems in Stereo Vision:

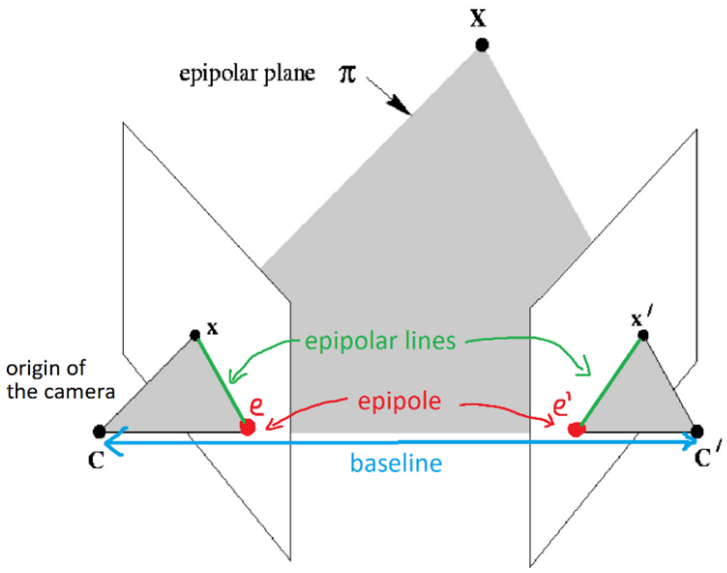
1. Finding Correspondences: Identifying the same physical point in the scene across different images.
2. 3D Reconstruction: calculating the real-world coordinates of a point based on its projections in images. This involves triangulation methods that use the geometry of the camera setup.

8.1 Epipolar Geometry

With Epipolar Geometry, we refer to the projective geometry describing the relationships between the two view. So we assume we have 2 cameras and the origin of the reference frames of the cameras are represented by the points **C** and **C'**. While **X** is the point in the 3D world, **x** and **x'** represent the point **X** in two different projection, which are the two image planes. The **baseline** is the line between the origin of the cameras, **C** and **C'**. The intersection between the baseline and the two image planes are called **epipoles**, which are **e** and **e'.**

IMPORTANT: if you have, **x** and both the epipolar lines, the point **x'** must be on the Epipolar Line!

- **Epipole:** Where baseline intersects the image planes. If the cameras are parallel, the epipoles are at infinity.
- **Epipolar Plane:** Contains the baseline and a point in 3D space. Every point in one image has its epipolar plane.
- **Epipolar Line:** The intersection of the epipolar plane with the image plane. It simplifies the correspondence problem by reducing the search space.



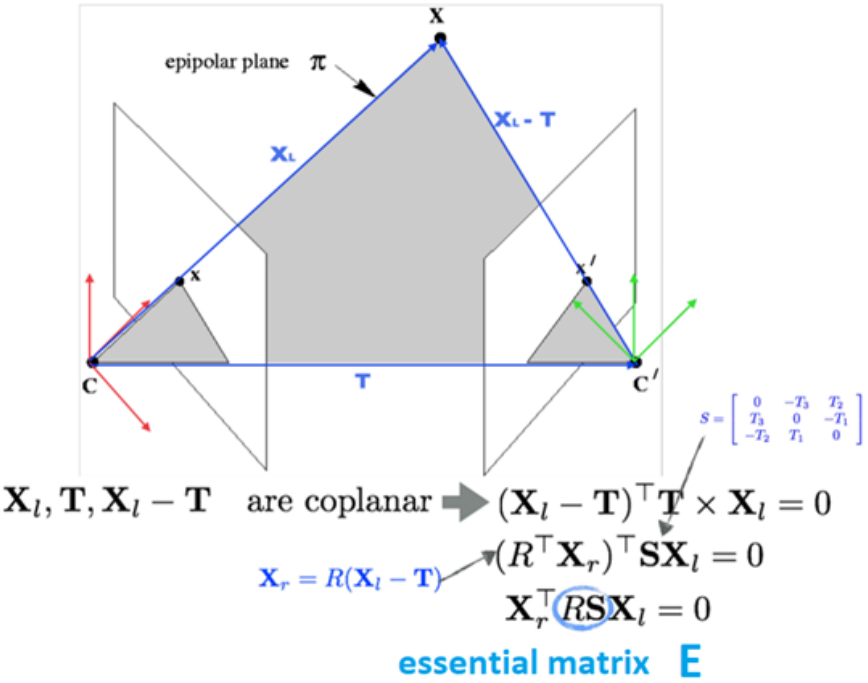
We need to know how to relate the two cameras, because we want to express the information coming from one camera with respect to the other. The geometrical relationship between the camera C' with respect to the camera C can be expressed in terms of a rototranslation.

We assume that the center of the left camera C is the same of the origin of the world reference system so that we can basically express everything with respect to this origin that is C.

- **Intrinsic Parameters:** Characterize the mapping of an image point from camera to pixel coordinates in each camera. They are unique to each camera and describe its internal optical properties.
- **Extrinsic Parameters:** Describe the relative position and orientation of the two cameras **(R, T).** Essential for aligning and merging images from different cameras.

$$X_r = R(X_l - T) \tag{17}$$

The Epipolar Constraint is a simplification of search, it restricts the search for corresponding points from 2D to 1D, because the point MUST BE ALONG THE EPIPOLAR LINES, this simplifies the matching process.



8.2 The Essential and Fundamental Matrices

- The **Essential Matrix** is a matrix $E = SR$ such that $X_r^T E X_l = 0$. It is satisfied by each (x_m, x'_m) pair of corresponding points in mm coordinates. E is 3x3, but it has rank 2 (R has full matrix, but S has rank 2). E has 5 DOF: 3 from R, 3 from T, but with a scale ambiguity 3+3-1=5.

  A 3x3 matrix is an essential matrix iff 2 of its singular values are equal, and the third is 0.
- The **Fundamental Matrix (F)** generalizes the Essential Matrix, the purpose is to derive an equivalent equation relating points in pixel coordinates. Then the fundamental matrix F is the unique 3x3 rank 2 matrix so that, for each corresponding pair (x,x'). F has 7 DOF since it is a 3x3 homogeneous matrix but has the further constraint of $det(F) = 0$.

WHILE **E** REPRESENT CORRESPONDING POINTS IN **mm COORDINATES**, THE **F** MATRIX REPRESENT CORRESPONDING POINTS IN **PIXEL COORDINATES**.

$$\mathbf{x}_m = K_l^{-1}\mathbf{x} \qquad \mathbf{x}'_m = K_r^{-1}\mathbf{x}'$$

$$\mathbf{x}'^\top K_r^{-T} E K_l^{-1}\mathbf{x} = 0$$

$$\boxed{\mathbf{x}'^\top F\mathbf{x} = 0}$$

8.3 Estimating the Fundamental Matrix

To estimate the Fundamental Matrix (F) we know the 8 Point Algorithm and the RANSAC.

8.3.1 8-Point Algorithm

The **8-point algorithm** is used to estimate the Fundamental Matrix F from at least 8 pairs of corresponding points between two images. Consider $x = (x, y, 1)^T$ and $x' = (x', y', 1)^T$, the matrix A is like: (found with equation of F). $Af = 0$:

$$\underbrace{\begin{bmatrix} x_1'x_1 & x_1'y_1 & x_1' & y_1'x_1 & y_1'y_1 & y_1' & x_1 & y_1 & 1 \\ x_2'x_2 & x_2'y_2 & x_2' & y_2'x_2 & y_2'y_2 & y_2' & x_2 & y_2 & 1 \\ & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\ x_n'x_n & x_n'y_n & x_n' & y_n'x_n & y_n'y_n & y_n' & x_n & y_n & 1 \end{bmatrix}}_{\mathbf{A}} \begin{pmatrix} f_{11} \\ f_{12} \\ \cdot \\ \cdot \\ \cdot \\ f_{33} \end{pmatrix} = \mathbf{0}$$

- We use **SVD** decomposition $A = UDV^T$
- We pick the last column of V as solution
- We reshape the 9x1 vector in a 3x3 matrix, obtaining a rank 2 matrix
- Rank enforcement: we need F to have rank 2! We look for the matrix F close to estimated one but with rank 2. So we use again **SVD** decomposition: we set the last element in the diagonal matrix **D** to zero.

$$F = UDV^\top$$

$$D = \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \sigma_3 \end{bmatrix} \quad D' = \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & 0 \end{bmatrix} \quad F' = UD'V^\top$$

In many other situations, a good practice is to actually **normalize** the points before doing the estimation of the fundamental matrix. Normalizing the points in a specific way, includes two different steps:

- **translate** points so that the image coordinate center corresponds to the points centroid.
- **scale** points so that their average distance from the origin is $\sqrt{2}$

How do the normalization:

- Consider points pairs (x,x')
- Normalize points obtaining (x_n, x_n') and matrices (T, T') such that $x_n = T \cdot x$
- Compute F_n with 8 point algorithm
- Denormalize the matrix as $F = T' \cdot F_n \cdot T$

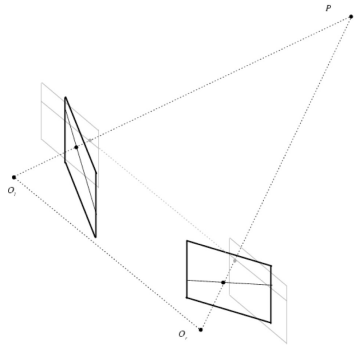
8.3.2 RANSAC (RANdom SAmpling Consensus)

With algorithms like *image matching algorithm* it is possible to find a lot of correspondences pairs, which could be correct or not. The bad points pairs can involve to errors in finding the Fundamental Matrix. We want to find the inliers (correct matches) and compute the F matrix on these points only. With the RANSAC (RANdom SAmpling Consensus) method, it's possible to handle this kind of problem doing a robust estimation in the presence of potentially incorrect correspondences in homography estimation.
Here's a basic outline of the RANSAC algorithm:

1. **Find correspondences points:** For instance through *image matching algorithm* it is possible to find a lot of correspondences pairs, which could be correct or not.
2. **Randomly Select a Subset of the Data Points**
3. **Fit a Model to the Subset:** Estimate the parameters of the model using this subset.
4. **Identify Inliers:** Determine how many points from the entire dataset fit this model with an error smaller than a threshold T. These points are considered inliers.
5. **Repeat:** Repeat steps 2-4 a fixed number of times N, each time producing a new set of inliers.
6. **Select the Best Model:** Choose the model with the highest number of inliers.

8.4 Rectified image pair

A rectified image pair it's basically a pair of stereo images where pairs of *conjugate epipolar lines* become *collinear and parallel* to one of the image axes. It can be obtained by a stereo system with parallel optical axes and coplanar image planes. This property can be obtained from a given image pair through an appropriate **rectification algorithm**.



8.4.1 Correspondence problems

The **correspondence problem** involves two decisions:

1) which image elements to match; 2) which feature description and similarity measure to adopt.

The (1) can be solved in two ways:

- use all pixels in the image (obtaining *dense correspondences*)
- use subsets of pixels meeting some requirements (obtaining *sparse correspondences*)

For (2) we will adopt correlation methods.

We consider the so-called *correlation methods* applied to image patches (neighbourhoods of pixels). We assume we have two rectified images, where conjugate points lie on corresponding scanlines of the image ("rows"). *The goal is to obtain a* ***Disparity Map*** *giving the relative displacement for each pixel.* In a standard color coding bright areas correspond to high disparities (closer objects).

8.4.2 Algorithm for Disparity Calculation

The algorithm inputs a stereo pair of rectified images I_l and I_r, which are images from the left and right cameras that have been transformed to align on the same plane. As **input** we have:

- A stereo pair of rectified images I_l and I_r.
- Size of a correlation window W.
- A search range $[d_{\min}, d_{\max}]$.

Algorithm Steps for Each Pixel:

1. For each pixel p_i of (i, j) coordinates in I_l, the algorithm proceeds to evaluate the disparity.
2. For each disparity d in the search range, the algorithm computes the similarity measure:

$$c(d) = \phi(N1(i, j), N2(i, j + d)) \tag{18}$$

   where $c(d)$ is the cost associated with the disparity d, and ϕ is a function that measures the similarity between the neighborhood of the pixel in the left image $N1(i, j)$ and the neighborhood of the corresponding pixel in the right image $N2(i, j + d)$.

3. The disparity of the pixel $\hat{d}$ is determined by:

$$\hat{d} = argmax_{d \in [d_{\min}, d_{\max}]} c(d) \tag{19}$$

   The value $\hat{d}$ is the disparity that maximizes the similarity measure, suggesting the best match between corresponding image patches.

4. A disparity map is constructed by assigning the estimated disparity $\hat{d}$ to each pixel, providing a representation of the scene's depth.

Thanks to the epipolar constraint the search is limited to one line. Thanks to the fact the images are rectified the line is exactly the same row. With a further prior on minimum and maximum disparity the search interval can be shortened (and is only positive).

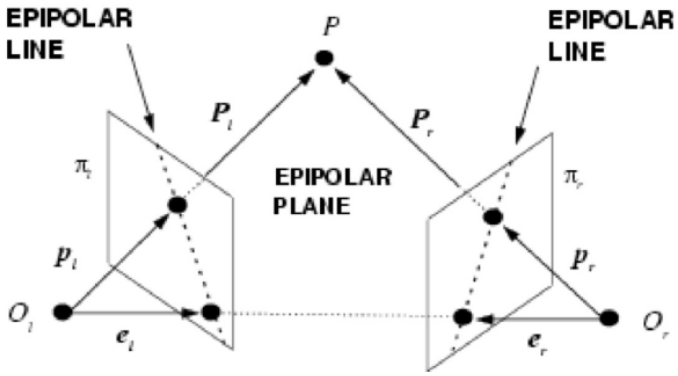
Disparity map from left to right is the opposite of the Disparity map from right to left:

$D(i, j) = d$ iff $D_{lr}(i, j) = -D_{rl}(i, j + d) = d$

8.5 Stereo vision pipeline (3D reconstruction)

The following steps outline the pipeline for reconstructing a 3D model from a stereo pair of images:

1. **Extract Interesting Points from an Image Pair:** Identify distinctive features within both images, such as corners or other points of high contrast.
2. **Determine a Set of Matching Points:** Identify matches
3. **Estimate the Epipolar Geometry:** With the matching points, to estimate the epipolar geometry. The *Fundamental Matrix* could be enough.
4. **Rectify the Two Images:** Transform the images so that corresponding epipolar lines are aligned horizontally. It simplifies the matching problem from 2D to 1D, searching along the horizontal lines.
5. **Compute Dense Stereo Matching on the Rectified Pair:** Generate a *Dense Disparity Map* by determining matching points across the rectified images, which provides a pixel-wise representation of depth.
6. **Triangulate 3D Points:** for each correspondence (x, x') compute the 3D point that projects to these two image points by triangulating the 2 corresponding optical rays. This step is supported by the **stereo camera calibration** parameters.


EPIPOLAR
LINE
P
EPIPOLAR
LINE
P_l
P_r
π_l
EPIPOLAR
PLANE
π_r
p_l
p_r
O_l
O_r
e_l
e_r

9 Change Detection and Tracking

9.1 Motion Analysis

In the context of motion estimation, analyzing a sequence of images over time provides valuable insights into the dynamic aspects of the scene. This allows for a more comprehensive understanding of the scene's evolution and motion patterns compared to a single static image.

In the scenario of observing a scene with a single camera, acquiring a video translates to capturing a sequence of images at equidistant discrete time instances, denoted as t_k. This sequential image data forms the basis for analyzing temporal changes and motion patterns in the scene.

Image sequence: series of N images, or frames acquired at discrete time instans

$$t_i \; = \; t_o + k * dt$$

The small time delta between consecutive images (captured at T_k and T_{k+1}) results in a high degree of similarity in the content of these images. This similarity facilitates the task of solving the correspondence problem, as discussed in previous sessions regarding image matching.

Motion information is very useful from a perceptual standpoint. There are changes that we can immediately perceive when there is some kind of motion occurring in the scene.

There is a fundamental assumption in all the methods: they presume that **illumination remains constant during the observation interval**. In other words, changes in the image from time t_1 to t_2 are attributed to the relative motion between the scene and the camera, not variations in lighting conditions.

The assumption of constant illumination is essential for identifying moving regions, ensuring that changes observed between two images are solely attributed to the relative motion between the camera and the 3D scene. This allows us to perceive apparent motion in the representation of the scene.

In scenarios where the camera is stationary, as in typical video surveillance, the **APPARENT MOTION is induced by the actual motion in the scene**, such as a person moving. In this case, the derived apparent motion is specifically related to the moving person, and other elements in the scene should not contribute to motion since the camera itself is static.

In scenarios where the scene remains static, but the camera is in motion, the apparent motion include the entire image. **This motion induced by the camera's movement is termed EGO MOTION**, signifying that it's the camera itself that is in motion. Analyzing the image sequence in this context provides information about the motion characteristics of the camera.

In a complex scenario, both the camera and dynamic events in the scene contribute to the apparent motion. For instance, when the camera is in motion, and there is also an object, like a walking person, the resulting apparent motion is a combination of ego motion (caused by the camera) and the actual motion of objects (caused by the moving person).

When dealing with motion estimation and analysis, we encounter two primary problems:

- **The Correspondence Problem:** The objective is to establish a mapping or matching between elements in two images, typically consecutive frames in a video. The challenge lies in finding corresponding elements in images captured at times t and t+1. Due to the small time interval between consecutive images, the content similarity often simplifies the correspondence problem compared to general matching scenarios.
- **The Reconstruction Problem:** This problem involves deriving information about the 3D motion and, potentially, the structure of the 3D scene. Once corresponding elements are identified (solving the correspondence problem), the goal is to infer details about the motion of the

3D structures responsible for the apparent motion in the images. The reconstruction problem is inherently more complex due to the limited information available from the small time delta between images.

Then there are other problems:

- **Motion segmentation** (2D problem): Identify regions in an image where motion occurs.In scenarios involving a stationary camera, such as video surveillance, this task is often referred to as **CHANGE DETECTION**. The fundamental concept involves analyzing an image acquired at time t and identifying regions that correspond to moving entities
- **Tracking** (2D or 3D): the fundamental objective is to link information associated with different regions segmented at various time instances to reconstruct the trajectory of a moving object or person. The goal is to associate the region, often described by its centroid, with corresponding regions in subsequent images.

9.2 Change Detection

The problem of change detection, also referred to as motion segmentation or **Motion-Based Image Segmentation**, involves segmenting an image based on motion information, particularly when the camera is stationary.

In the simplest case change detection is often achieved by computing the difference between images. **This method requires a reference image**, representing the background or the scene when no significant motion is happening. This reference image serves as a background model, encompassing information unrelated to noteworthy motion events in the scene. This method is call **BACKGROUND MODEL**.

Therefore, the basic approach involves comparing the reference frame with the frame at time t. The objective is to detect significant changes at each point in the image concerning the reference image or background. If a substantial change is detected, it suggests the presence of a FOREGROUND EVENT, indicating some form of motion in the scene. This approach relies on the assumption that the illumination remains constant between consecutive time instants.

Mathematically, this involves computing the absolute difference between the pixel values in the image at time t and their corresponding values in the reference image. If this difference exceeds a certain threshold, we assign a value of one, corresponding to white in the binary map M_t, indicating a detected change at that position. Otherwise, we assign zero.

$$M_t(x,y) = \begin{cases} 1 \; if \; |I_t(x,y) - I_{ref}(x,y)| < \tau \\ 0 \; otherwise \end{cases}$$

In the end, the result is a binary map where connected white components can be associated with moving regions in the scene. To create the reference image, a straightforward approach involves deriving it as an average of a certain number N of empty frames.
The are disadvantages for this strategy:

1. Limited availability of a sufficient number of empty frames in most cases.
2. The uniqueness and stability of the background, which may not reflect natural variations in the scene over time.

For instance, changes in natural illumination or other variations in the scene may necessitate an adaptive background model that can capture these dynamics. In such cases, the **RUNNING AVERAGE PROCEDURE** is employed, allowing the background image (or reference image) to evolve over time, denoted by the index "t" indicating the temporal adaptation of the background.

$$M_t(x,y) = \begin{cases} 1 \; if \; |I_t(x,y) - B_{ref}(x,y)| < \tau \\ 0 \; otherwise \end{cases}$$

And the idea is that in a certain position (x, y), the background at time t can be in two different ways, depending on what happened on that specific point.

How can decide if the pixel is moving or not ? We built another binary map, by comparing images at time t and t-1

$$D_t(x, y) \ = \ |I_t(x, y) - I_T t - 1|$$

If the difference between the background at the previous time instant and the current information at that position is too high, it indicates that something is happening at that specific position in the current instant (a foreground object). In this case, the information corresponding to the current position is not considered reliable for updating the background, and we choose to retain the old information.

In the other case, when the difference between the information found at a certain position in the current instant and the information in the background model is low, it indicates that the current image at that position is similar to the background. In other words, the information at that position in the current instant is still related to the background information.

$$B_t(x, y) = \begin{cases} B_{t-1}(x, y) \ \ if \ \ D_t(x, y) > \tau' \\ (1 - \alpha)B_{t-1}(x, y) + \alpha I_t(x, y) \ \ otherwise \end{cases}$$

Then the idea is to use this information to update your background with a weighted sum that combines the old information and the new information, weighted using two weights guided by a parameter α, where α is a value between zero and one. This allows us to control the importance given to the old information (the information in the old background) and the information in the current image (α depends on the specific environment).

For example, when people are moving very slowly, we must be careful not to give a high value to α because it would incorporate the current information into the background too quickly. This could risk incorporating information related to the appearance of the foreground object into the background. This is influenced by both the choice of α and the choice of τ.

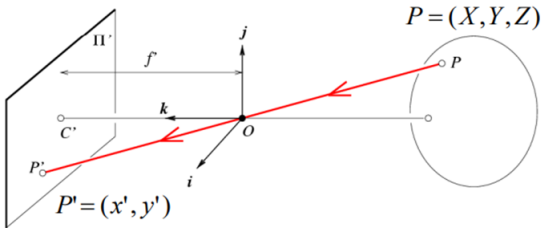
The cons of this method is its uni-modality, indicating that it only uses one value to represent the appearance of a point when it is considered a background point. This approach is not suitable for outdoor scenarios where multi-modal models, representing the background with more than one possible value at each point, would be more appropriate.
This is extremely important when you might have elements belonging to the background but moving.

Under general conditions, the correspondence problem can be framed as the task of **estimating the apparent motion of the brightness pattern**, often referred to as **OPTICAL FLOW**. The goal is to determine the correspondence between elements in one image and their counterparts in another image by estimating the amount of movement for each element from one time to the next.

To achieve this, we require a review of certain concepts related to image formation, including the pinhole camera model. What's crucial is understanding the relationship between the coordinates of a 3D point in the world (denoted as P) and the corresponding coordinates of its projection (P') onto the image plane. This relation involves factors such as the focal length of the camera, the ratio between the first two coordinates, and the depth coordinate.

10 Optical Flow

10.1 The Motion Field of Rigid Objects



We have a point in the 3D world denoted as P, and it is undergoing motion with a certain velocity V in the 3D world. The motion field is essentially the projection of this 3D velocity vector onto the image plane. The objective of optical flow is to estimate this 2D projection.

Properties of the motion field include the fact that rigid objects exhibit a constant 3D velocity field. Now, let's consider a 3D point $\mathbf{X} = (x, y, z)^T$. The relative motion between $\mathbf{X}$ and the camera will have both translational and angular components.

$$\mathbf{V} = -\mathbf{W} - \boldsymbol{\Omega} \times \mathbf{X}$$

translational angular

In general terms, we can express the relative motion between $\mathbf{X}$ and the camera as a combination of translational and angular components. Thus, we can represent $\mathbf{V}$ as the sum of a translational component $\mathbf{W}$ and an angular component $\Omega \times \mathbf{X}$ (the minus sign indicates the velocity related to the 3D velocity of the point with respect to the camera reference system). There are two reference systems involved: the camera reference system and the image reference system. In this case, we are referring to the camera reference system.

We can describe the velocity of the projected point, i.e., the motion field, as the derivative of its position.

$$\mathbf{v} = \dot{\mathbf{x}} = \frac{\partial \mathbf{x}}{\partial x} = f \frac{\partial}{\partial t} \left(\frac{\mathbf{X}}{Z} \right)$$

$$\mathbf{v} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ 0 \end{pmatrix} = f \begin{pmatrix} \frac{Z\dot{X} - \dot{Z}X}{Z^2} \\ \frac{Z\dot{Y} - \dot{Z}Y}{Z^2} \\ 0 \end{pmatrix} = f \frac{Z\dot{\mathbf{X}} - \dot{Z}\mathbf{X}}{Z^2} = f \frac{Z\mathbf{V} - V_Z\mathbf{X}}{Z^2}$$

If we put together the two observations, so basically few substitutes $\mathbf{V}$, we end with this.

$$\mathbf{V} = -\mathbf{W} - \Omega\, \mathbf{X}$$

$$\mathbf{v} = f \frac{Z\mathbf{V} - V_z\mathbf{X}}{Z^2}$$

This separation allows for the final formalution of the motion field $\mathbf{v}$.

$$\mathbf{v} = \underbrace{-f\frac{\mathbf{W}}{Z} + \frac{W_Z\mathbf{x}}{Z}}_{\text{translational component}} \underbrace{-\boldsymbol{\Omega}\times\mathbf{x} + (\boldsymbol{\Omega}\times\frac{\mathbf{x}}{f})^\top \hat{k}\mathbf{x}}_{\text{rotational component}}$$

We can identify a translational component and of rotational components. We can observe that:

- · The angular velocity Ω and the information on the depth of a scene never appear together. In the case of pure rotation (where the traslational component is zero), the motion field does not depend on the scene structure → This implies that with pure rotation the motion field is independent of the scene structure.
- · The translational component $\mathbf{W}$ is always divided by the depth, leading to an ambiguity known as **DEPTH VELOCITY AMBIGUITY**. This ambiguity arises because the same motion field (or projection) can be associated with very different scenarios. For instance, objects that are closer to the camera but moving slowly and objects that are far away from the camera but moving faster can produce the same visible effect on the image plane. Therefore, distinguishing between these two conditions is challenging due to the similar projections.

10.2 How to Estimate the Motion Field

The computation of optical flow involves making a strong assumption on image brightness constancy, which assumes that the apparent brightness of a moving object remains constant. This assumption is fundamental for estimating the motion field accurately.

$$\frac{dI}{dT} = 0$$

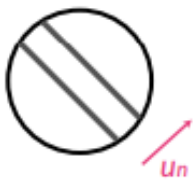
$$\frac{d(I(x,y),t)}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

To compute optical flow, we begin by calculating the derivatives of the image with respect to x, y, and time. This involves treating the image as a function of spatial coordinates (x, y) and time. The fundamental principle here is to set the formula equal to zero, which is an implication of the image brightness constancy assumption. By writing down the partial derivatives of the image with respect to time, we use the chain rule of derivation for each component. Next, we define the components $u = dx/dt$ and $v = dy/dt$, and we combine them into a vector $\mathbf{U} = [u, v]$. This vector represents the optical flow associated with time.

$$(\nabla I)^T \quad \mathbf{u} + I_t = 0$$

The optical flow u is a vector with two components, subject to the image brightness constancy constraint. However, one constraint alone is insufficient to compute the optical flow, as there are two unknowns. The APERTURE PROBLEM is another challenge arising from the assumption that led to the optical flow formulation. This assumption, based on the image brightness constancy equation, only enables the determination of a portion of the information related to the optical flow component parallel to the spatial image gradient.

In the presence of a change in brightness in the image, the gradient is perpendicular to the change. Consequently, the optical flow estimation is limited to the component parallel to the gradient of the spatial gradient of the image, which is perpendicular to the change in the image.



So basically the image brightness constancy equation allows us to determine the optical flow component parallel to the spatial image gradient.

Analytically we can describe it as:

$$u_n = \frac{(\nabla I)^T \mathbf{u}}{||\nabla I||} = \frac{-I_t}{||\nabla I||}$$

Indeed, the "aperture problem" is named so because it involves looking at the problem in a localized or partial manner, as if through a small aperture. This approach involves examining the information in the neighbourhood of the observed point, considering only a limited portion of the overall scene.

When visually perceiving the motion of an object between time t and t+1, the challenge is that even if the observed motion appears to be along a specific direction, there could be additional components contributing to the motion that were not discernible in the earlier observation. This limitation arises from the fact that, in the actual motion field, we can only analytically identify one of the main directions, leaving the other components indeterminate. Many algorithm algorithms start with the idea of adding other constraints to derive the optical flow estimation.

10.3 Lucas Kanade Algorithm

The Lucas-Kanade algorithm operates under the assumption of local motion constancy, implying that the motion is constant in the immediate neighbourhood of a given point.

In essence, when examining the x and y coordinates of a specific point, the algorithm presupposes that the motion estimation regarding to that particular point extends consistently to nearby points in the neighbourhood.

So, we must build the system using the image brightness constancy equation and putting together the contribution of the point x (to focus the attention on the point and also of the points in the neighbourhood of x).

Given an image, our approach involves estimating the optical flow vector u corresponding to the point x. This estimation utilizes not only the information associated with the position x but also takes into account the information from points in a close proximity or small neighborhood around x

$$(\nabla I(\mathbf{x}_i, t))^T \mathbf{u} + I_t(\mathbf{x}_i, t) = 0 \;\; x_i \in N$$

If we develop the system, we can write down a linear system like this Au = b (where b is the unknown term):

$$A = \begin{bmatrix} \nabla I(\mathbf{x}_1, t)^T \\ \nabla I(\mathbf{x}_2, t)^T \\ \\ \nabla I(\mathbf{x}_m, t)^T \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -I_t(\mathbf{x}_1, t) \\ -I_t(\mathbf{x}_2, t) \\ \\ -I_t(\mathbf{x}_m, t) \end{bmatrix}$$

To solve this system, we can do it with the pseudo inverse.

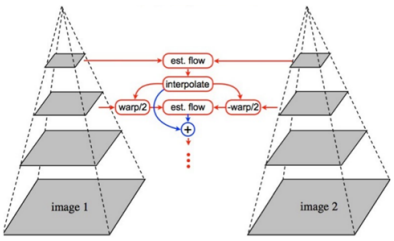
$$\mathbf{u} = A^{\dagger}\mathbf{b} \quad \text{with} \quad A^{\dagger} = (A^{\top}A)^{-1}A^{\top}$$

In conclusion, for every point in your image, there exists an associated vector representing the optical flow estimation specific to that point.

$$\begin{bmatrix} \Sigma I_x I_y & \Sigma I_x I_y \\ \Sigma I_x I_y & \Sigma I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \Sigma I_x I_t \\ \Sigma I_y I_t, \end{bmatrix}$$

Because we employ a local strategy, our approach involves utilizing points within the neighbourhood of a selected point to determine the optical flow estimation. It's crucial to emphasize that the region surrounding the central point is deliberately kept small. This decision is based on the assumption that motion remains consistent within a limited area around the point, and we refrain from enlarging this region excessively.

The basic framework for computing optical flow might encounter challenges in the presence of significant displacements or large motions. To address this issue, a potential solution is to adopt a coarse-to-fine(Hierarchical refinement strategy) approach for estimating optical flow. This involves creating a hierarchical representation with different layers, where the estimation is progressively refined. The iterative procedure entails moving from a resized version to a rescaled version of the images, gradually reaching a smaller size, and then reversing this process to return to the full original dimensions of the image.



To achieve this, we need to take the float value estimated at the previous level and interpolate the estimation to match the dimensions of the current level. At a given level i, we use the estimation from level $i-1$, where the images were smaller, and then upsample the estimated flow to align with the size requirements of the image at level i. This initial estimation provides an optical flow estimate with the current dimensions.

Subsequently, we warp the image at time t based on this new estimation and apply the estimation using the Lucas-Kanade method on the image at the subsequent time.

By following this process, we introduce a correction that can be applied to the estimate identified in the previous level, adjusting it to the current level. This iterative refinement is carried out until all the images are processed.

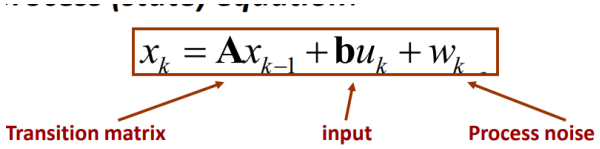
This method helps reduce errors in regions with significant motion. Initially relying on the assumption of a locally constant translational optical field, we can extend to more complex models. As optical flow estimation methods become more intricate, visualizing the optical flow is essential.

11 Kalman filter

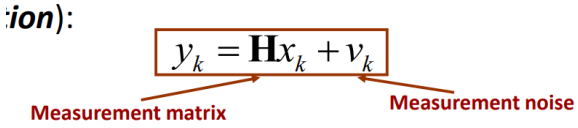
The Kalman filter is an optimal estimator, it works of probabilities. It estimates quantities of interest from noisy observations. It is a statistically optimum way to combine and exploit sensor data.
Used to :

- Spatio-temporal estimation
- Tracking in computer vision applications
- Tracking targets
- Autonomous navigation
- Robotic applications

It is an **estimator**: it estimates the system state from **noisy measurements**. It is **model based**: based on the state equation of the system. To use it, it's necessary to have the system equations **linear** and the noise to be Gaussain with zero mean, otherwise we have to use the *extended KF*. It is **least square**: it minimizes the mean square error of the estimated parameters, this is why it's considered to be optimal. It is **recursive**: KF can process new measurements as they arrive. It is **stochastic**: KF quantities are found statistically.
We can use KF if we have the probability distribution of the system and the measurement of it at time t.



where x_k is the new state, x_{k-1} is the previous one, u_k is the current input, which drives the change of the state, and the w_k which is the process noise, it is the **uncertainty** of the system. The matrix A is the matrix telling how state changes.
N.B. when we write code, we don't have to add noise, since it's already there in the sensor!! (same thing in the measurement equation)
The noisy measure at instant k is:



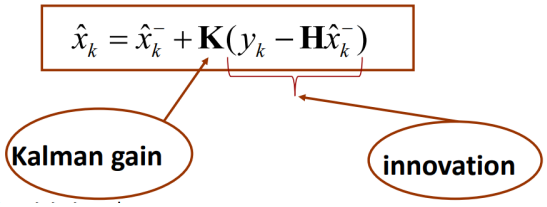
and x_k is the state at time k, v_k is the noise of the sensor.
Noise can change as a function of time, but we can assume both measurement and process matrices to be constant in the filtering stage.
Error on the a priori estimate:

$$e_k^- = x_k - \hat{x}_k^-$$

The error on the a posteriori estimate:

$$e_k = x_k - \hat{x}_k$$

We can define the **a posteriori estimate** at time k, given the measurement at time k, as:



being x_k^- the **a priori estimate**, y_k the real measurement, and $H * x_k^-$ the **estimated measure**, based on the estimated state.The **Kalman gain** minimizes the a posteriori error covariance; the innovation is the difference between the current and the predicted measurement.
Then there are two parts:

- **temporal update**: compute the a priori estimate based only on the a posteriori estimated model of the state; no noise here: it is our uncertainty, which is in the covariance:

$$\hat{x}_k^- = \mathbf{A}\hat{x}_{k-1} + \mathbf{B}u_k$$

  This was the prediction of the error covariance.

$$P_k^- = \mathbf{A}P_{k-1}\mathbf{A}^T + \mathbf{Q}$$

- **measurement update**: The new kalman gain is:

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}^T(\mathbf{H}\mathbf{P}_k^-\mathbf{H}^T + \mathbf{R})^{-1}$$

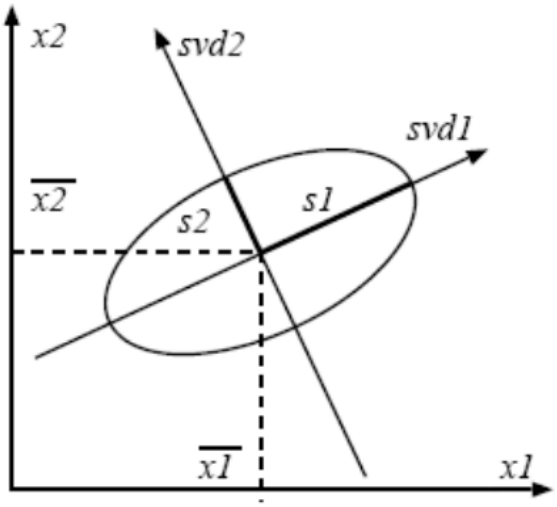
  At this point we can update the estimate, an a posteriori estimate as a weighted sum of the a priori and the product of the kalman gain and the innovation=real measurement y - the measurement expected from the a priori estimated state:

$$\hat{x}_k = \hat{x}_k^- + \mathbf{K}_k(y_k - \mathbf{H}\hat{x}_k^-)$$

  Then we can update the error covariance:

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^-$$

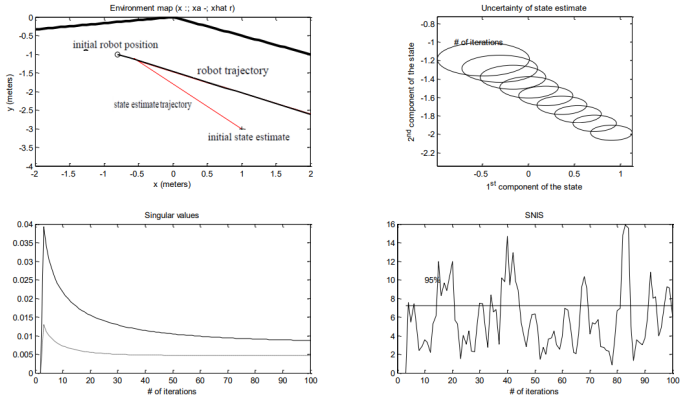
then we start again the process, by the temporal update.
First check: **convergence**, which is related to the reduction of the uncertainty of the state estimate, we want the matrix P to decrease over recursion. A convergence test for the KF is to use the singular value decomposition (SVD) of the error covariance matrix: we monitor if the singular values decrease over time, or we look at them in the state space:



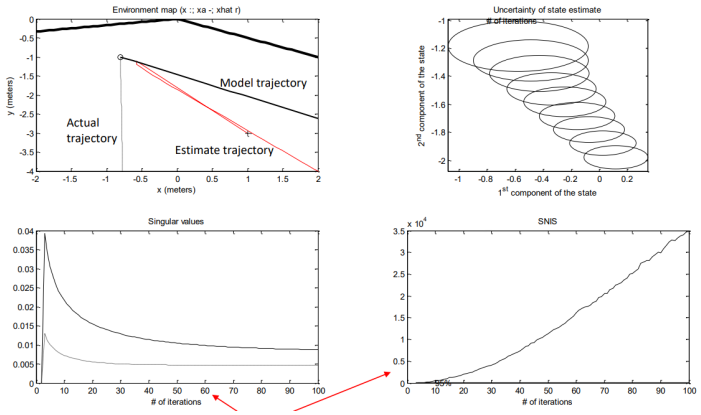
singular directions: x_1 and x_2; singular values: svd_1 and svd_2. At the next time we want a shrinked ellipsis -¿ convergence.
Second check: **consistency**, if the model is wrong, even if the system converges, it will be wrong. We need a technique to understand if the estimated model is wrong or right. To test it: check the consistency between the innovation and the model (i.e. between observed and predicted values) in statistical terms. A measure of the reliability of the KF output is the Normalized Innovation Squared (NIS), which is a function of time (k), that has a chi-square distribution with m degrees of freedom, related to the size of the transition matrix:

$$NIS_k = a_k^{\ T} P_k^{-1} \alpha_\kappa$$

where m is the number of statistically independent measurements and is the innovation. This is related to the error covariance of the a posteriori estimate (P_k) and the innovation. If values grow up at each time, something is wrong. The α, which is the innovation, must be inside a range. It is a quadratic form. N.B. a is actually α, the slide is wrong!!
This is the right model, and we can see that singular value are decreasing, ellipsis is shrinking, and the state estimated trajectory is close to the robot trajectory. 95 is the threshold for the chi-distribution, oscillations is due to the noise:



This is the wrong model:



Real model:

State equations:

$$x_1[k] = x_1[k-1] + v\cos(\theta)\Delta t$$
$$x_2[k] = x_2[k-1] + v\sin(\theta)\Delta t$$

These are the equations of the model, the wrong one:

$$x_a[k] = x_a[k-1] + v\cos(\theta - \pi/3)\Delta t$$
$$y_a[k] = y_a[k-1] + v\sin(\theta)\Delta t$$

The convergence is good, but the consistency is very different!! This is where we understand that consistency is not good, otherwise we can't detect an error in the model.
This can be used also to track objects: the corner detector can be wrong, due to noise. The kalman filter gives you the estimated position of the object. We get less error. Do it recursively.

EXTENDED KALMAN FILTER: it is used to handle non-linear filters. Use a linearization of the system, using taylor, of the system and of the noise.
This is the measurement expressed in an exended way:

$$y = \phi(x)$$

If it is not too non linear, use taylor.

Σ_x is small, ϕ is not too nonlinear,

$$y \approx \tilde{y} = \phi(\bar{x}) + D\phi(\bar{x})(x - \bar{x})$$

These are the approximations for mean and covariance.

$$\bar{y} \approx \phi(\bar{x}), \qquad \Sigma_y \approx D\phi(\bar{x})\Sigma_x D\phi(\bar{x})^T$$

If the covariance is not too small, we need other solutions, which is the **MONTE CARLO** simulation:

$$\bar{y} \approx \bar{y}^{\mathrm{mc}} = \frac{1}{N}\sum_{i=1}^{N}\phi(x^{(i)})$$

$$\Sigma_y \approx \frac{1}{N}\sum_{i=1}^{N}\left(\phi(x^{(i)}) - \bar{y}^{\mathrm{mc}}\right)\left(\phi(x^{(i)}) - \bar{y}^{\mathrm{mc}}\right)^T$$

where $x^{(1)}, \ldots, x^{(N)}$ are samples from the distribution of x, and N is large

estimate average of the average measurement performing the average of some sample of the state equation, and describe covariance with the formula of the covariance. Try to approximate mean and covariance using a large number of sample -¿ large number theorem -¿ probability again gaussian.

another method: use Monte Carlo formulas, with a small number of nonrandom samples chosen as 'typical', *e.g.*, the 90% confidence ellipsoid semi-axis endpoints

$$x^{(i)} = \bar{x} \pm \beta v_i, \qquad \Sigma_x = V\Lambda V^T$$ Sigma points

They have an high value of confidence. This is the **UNSCENTED KALMAN FILTER**. THIS IS THE IDEA, I DON'T ASK YOU THE EQUATIONS!!!!!!!!!

12 Recognizing and classifying, Visual Recognition

In this chapter the mention some higher level tasks on images

Image Understanding task

Refers to the process of extracting semantic information from images or sequences of images. The problem, in general, is unsolved, but but there are quite effective solutions for more specific tasks and questions.
From an image can be extracted a lot of different informations, such as: structural elements of the environment, provide an idea of the type of activity, the type of interactions that a group can have etc...
Describing an image the focus is to derive some kind of semantic informations. Having the sequence of the images is also possible to notice the mutual positions of the people.
For now everything was low-level analysis, now the aim is to look to the image in a high level, and to do that, are needed some tools, such as the **machine learning** for the image representation (reasoning on what to extract from the image depending on the specific tasks to address) and the image understanding (try to solve the understanding problem). So starting from basic and classical computer vision elements, given a task and a set of data, design and validate an image representation strategy (meaning also to design a similarity measure to compare different representations).
This representations usually are based on machine learning (they are provided to a higher level), but the idea is to derive learning the knowledge from the data to have the representation.
In the last 8 years **Deep learning** from the theory of neuronal network (very fast and easily: the image representation is learned with the function to solve the requested task).

From the image or the image sequence to goal is to have a knowledge about the content.
There are three main categories of tasks to accomplish it:

- image retrieval:
- image classification and its nuances (categorizations, detection, recognition)
- image segmentation: in which the classification is applied to each pixel mainly independently

All the methods are built on top of the design of an appropriate representation based on the content. An idea can be to extract knowledge from sparse feature detection and then describe the appearance of the image region of the points.

1 Image retrieval

The problem is to have a Disposal (a gallery) of images and a query image (i don't know anything about the query). So use the notion acquired from the similarity and estimation from all the images of the gallery to order all the images of the gallery for similarity with the query ones.

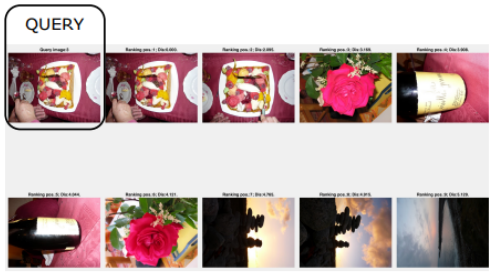


Figure 45: Image reteival

The first image can be discard because is the very same image. The bottle of wine is similar for the color
When the gallery is very big (a lot of images) I may want to only save the firsts images for similarity and not order all the others.
The retrieval is the skeleton of a Google search starting from an image that returns images similar to the query one. (remember our search is very much easier because is a pure one-to-one image matching)

2 Classification

The problem is having a sets of images associated to a some kind of labels (have people faces and you can label every face with the name, so the label of the class of the individuals).
The goal is to learn how to separate and distinguish different sets of images (e.g faces and not faces), so having a new unknown image with the function *training set* of machine learning terminology (developed with the data-set) I can distinguish to which label associate the image
To choose the classification between two categories, starts from the disposal (= n images) and n pairs to find: (x_i, y_i) where:

- x_i represents the image that $x_i p_R$, so lives in a space X that is subset to general space p_R
- y_i represents the label

It is a binary classification problem, so two sets of possible images (classes). A common practice, in this case, is to consider a set of y, so $y = 0, 1$ or $y = -1, 1$ Each sample x_i is a vector of elements and is the representation designed and derived image.
The idea is to derive a function f_n (n because n examples in the training set and the f depends on the training set) that applying to one of the samples I'm able to derive the right label.
$f_n(x_i) = y_i$
I want to have this function generalized to all the possible new images that doesn't belong to the data-set to estimate the right label

$$(x_1, y_1), \dots, (x_n, y_n)$$
$$x_i \in \mathbb{R}^p \quad \text{and} \quad y_i \in Y, \quad i = 1, \dots, n$$

$$X_n = \begin{pmatrix} x_1^1 & \dots & \dots & \dots & x_1^p \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^1 & \dots & \dots & \dots & x_n^p \end{pmatrix} \qquad Y_n = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Figure 46: ML function

Object Detection

It is a image classification problem
Given a category of interest (e.g cats, vehicle) and a image, I want to identify the portion of the image, if any, in which the object is present. Can be represented by the bounding box.

Considering CV before the deep learning, the idea was to start from the entire image and with a region box to look for the entire image to find the requested object also from different sides, not knowing the size, the shape, the type and sometime the color of the object.
It belongs to the classification problem because for every region I can consider with a binary classification if that belongs or not to an object. The response will be quite always "yes", because you can have a positive response also around the object itself (taking only a portion of that), so it is important to apply maximal suppression to save only the best window.

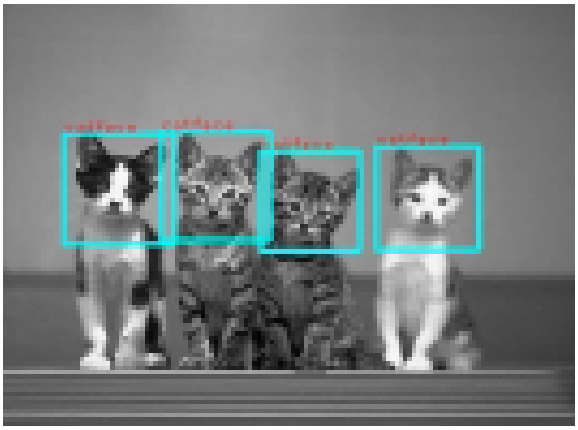


Figure 47: Best window detection

Image Categorisation

An other instance of the image classification when I have n classes, so I'm in multi-class classification problem.

Object Recognition

Problem of the recognition

- general: recognize different classes of objects (e.g recognize mugs from bicycle so positive answer for all the mugs)
- instance: recognize a specific object of a class from another one of the same class (e.g only positive answer for the chosen mug)

More complex situations

The situation becomes more complex in the **unsupervised** state when the objects doesn't belong to the the training-set available for input and output, so the case in which you have lots of images where you can derive a lot of representations, but not the label. In this case you can try and derive some kind of common patterns or group among the images, so the representation, but the problem has to be formulate in a different way.
Also is possible to have a **multi-classification problem** in which the different classes are no characterized by the similar amount of information (one class may be difficult to gather data and annotate them for example when the label is about a pixel).
Another problem that makes the situation more complex is that the data-set can have very different size

3 Image segmentation

The idea is to identify regions (not rectangular), so portion of the image of whatever shape, that corresponds semantically to the very same object. It can be done in:

- unsupervised state (without the labels) reasoning on the coherency of the color or the texture, the presence of boundaries or edges inside the image
- supervised state done using deep learning where the annotation becomes really complex because is needed to annotate each single pixel with a certain label

Methods for Object Recognition

A simple way to associate a label (meaning) to a image starting from local features release to two assumption:

- the use of a template in which the object of interest appears with no background noise (is clean)
- few object in the scene of the image (to look to a small number of them)

There are two possible attempts:

- **feature matching** so object recognition from an example or template.
  Starting from the templates (little on the left) we want to find the local features of them using **SIFT** and the corresponding descriptors and doing the same on the images on the right and perform the matching between the descriptors of the templates and the images using the localization of the matching point to localize the object

Figure 48: Enter Caption

  You don't have any prior on the object on the right image on the position or the side or the size. Having a lot of objects these procedure is going to fail because of the difficult computation also to try all the points combination.
  A possible solution is to resort the **retrieval task** (next item)

- **content based image retrieval**, derived from local feature, with a more refined description. It is a way to organize the information coming from local feature Design a retrieval procedure in which the information is based on the content of the image.
  A common practice is to use representations that are derived from the notion of Inverted indexes or describing the distribution of the frequency of a particular Visual Words.
  From the gallery we can look for the most similar images in base of the content.

Figure 49: ciuf ciuf

The **context** is often important:

Figure 50: tredici

It could be "13" or "B", and we can't know without knowing the context information (having other numbers or characters we can understand)
The idea is to enforce with the next topic the meaning of a single element putting together and seeing the contribution of every element.

Bag of words or Bag of Features or Bag of Keypoints it is a classical representation coupled with this problem. It is a representation relying the use of image representation relying on the use of visual information, but it's inherited from the text retrieval in which the representation is used to compactly represent the content of a text.
The idea is to define the dictionary (a set of words) or code book and to represent the text by associating to each word in the dictionary the frequency with which the word is appearing in the text. You don't have usual words, but visual words.
So two steps in the image representation:

- identification of the dictionary for visual words, so Build a visual dictionary from data
- the actual representation of the images with respect to the dictionary (image to representation with the dictionary) based on key points to have the basic context of the image, so represent all images by quantizing its key points with respect to the dictionary.

Figure 51: dictionary

The one in the top is the dictionary
In the middle the local patches (patches representations), so the visual words
The kind of histograms are the frequency of appearance of every key word in the image
The codebook is derivable from patches of similar appearance or the most frequents and so we have a dictionary of visual words
From the codebook and the frequency of appearance we can restore the images on the top
Starting from a gallery of images G, to describe the images (for instance) extracting the key points of the image and compute the descriptor on the portion of the image around the key points using the SIFT descriptor. So we move from the gallery to set V in which there is the collection of all the SIFT descriptors.
The idea is to build the dictionary with a method of clustering method (unsupervised method), to generate group of coherent descriptors.

Once you have this groups of visual words it is possible to represent each group with a single element which may be the centroid (the average). The collection of centroids compose the dictionary, in which I can decide the size.

Dense way = it is also possible to cover the image with a grid of image regions, so changes the identification of points to embed in the representation, but the rest is pretty much the same.
So now it is possible to represent every image with the dictionary. Seen by the mechanical standpoint it can be formalized as:

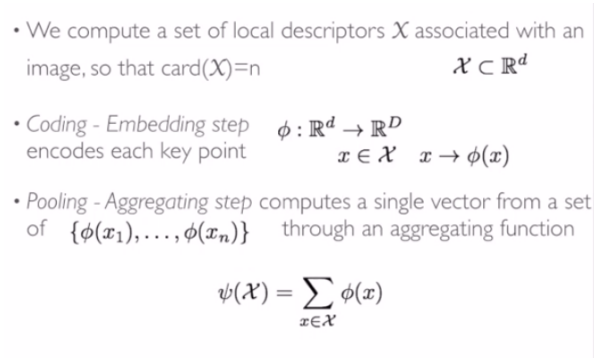
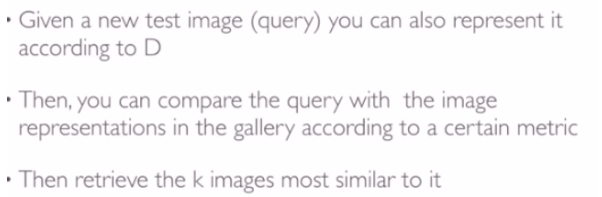


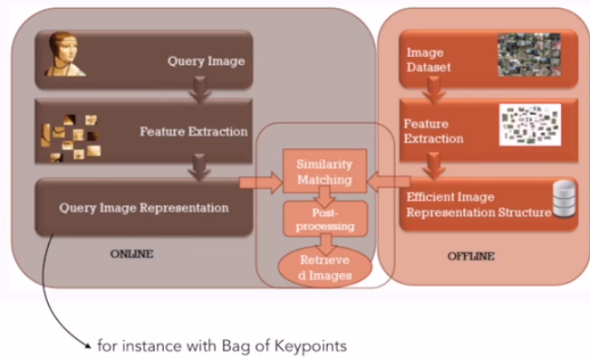
Figure 52: Enter Caption

X is a set of local descriptor living in a certain space
t can be formalized in two steps:

- **Coding - Embedded step** in which you encode each key point. So it's a feature map from the d-dimensional space of the original key points to the capital D. You can take each file descriptor and transform in a new representation of the very same information.
- **Pooling - Aggregating step** compute a single vector from a set of representations using a certain aggregation function (this case sum). Having an image, each image associated with a set of key points, (transform it with Coding), then with pooling I put together all the different contributions to derive a singe final representation

Simple strategy:
Defining the feature map $\phi_B OW(x) = [0, ..., 0, 1, 0...0]$ that goes from d (that can be SIFT) to D. So associate to each descriptor to D descriptor (D is size of dictionary) in which I have all 0 and 1 where I find the visual world (derives as centroid of all the elements of a common group) the most similar to the SIFT here (to original descriptor).
The final image is the sum of all these different vectors that can be normalized just to control numerically the stability.
The retrieval with back of word is the very same of what discuss till here.





Can SKIP (not in our slides) Many times the representations with 0 and 1 is not enough because a single visual word is not sufficient to represent the image path, so it is needed a more general definition of coding

- GOAL: given a dictionary of atoms
$$\mathbf{D} = [\mathbf{d}_1|\mathbf{d}_2|\cdots|\mathbf{d}_K] \in \mathbb{R}^{d\times K}$$
decompose a datum **x** in a linear combination of the atoms such that:

or better $\mathbf{x} = \mathbf{D}\mathbf{u}$

$$\mathbf{x} \sim \mathbf{D}\mathbf{u}$$

- the vector **u** encodes the information of **x** with respect to the dictionary **D**

(we can get rid of **x** and use **Du** which is less noisy/redundant)

Decompose a vector x (in our case SIFT) by identifying u such that can encode the info of x with respect of the dictionary approximating that.
For a number of visual words I can have numbers greater than 0 (the big difference).

- we may compute **u** by minimizing the reconstruction error
$$\min_{\mathbf{u}} ||\mathbf{x} - \mathbf{D}\mathbf{u}||_2$$
- sparsity is good! this means that we would like to approximate a given **x** with a few atoms
- Vector quantization (it can't be sparser than this)
use only one atom
u will have only 1 non-zero component (equal to 1)
$||\mathbf{u}||_0 = 1$

I want the vector u with few entrance different to 0.
.... Skipped (In Podcast to 31 min)

The dictionary may be fixed or learnt from the datas.

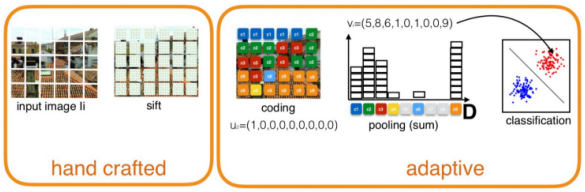


Figure 53: Enter Caption

The idea is to use this representation as training set to have the identification function having the part adaptive to make ML learn.

Figure 54: Corri cavallo corri

The pooling can be done globally on the image, as we did in this example, let's say, but also you might want to organize a little bit your representation in a more structured and hierarchical way by applying the very same pipeline coding plus pooling, not to the entire image, but to different images. Images so it may be two different portions of the image so it may be in the portions one two three and four or even with higher granularity.
Let's say of your partition in the image and then the idea is to is that the the final representation associated with the image is derived as the concatenation of all the single representations derived from all these different regions here. What you gain here is the fact that if the object of interest is not, exactly in the center of the image, you can, in a way, focus on it, by analysing the information in a smaller portion of the image, and most importantly, you can gather some hints on the different parts composing the object and how they are located, specially located in the image. The ones with respect to the other. So you are kind of including some type of information about the topology, of the object of interest.

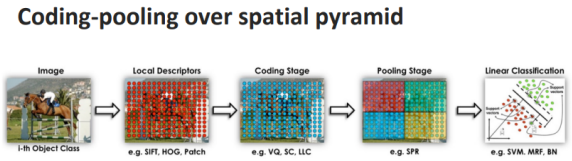


Figure 55: Enter Caption

12.0.1 What is next

There is the Convolutional Neuronal Networks that are a subset of the generalised deep learning extension to neural networks.

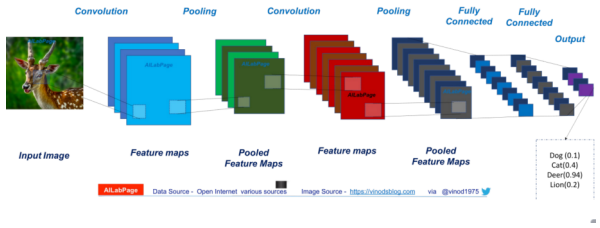
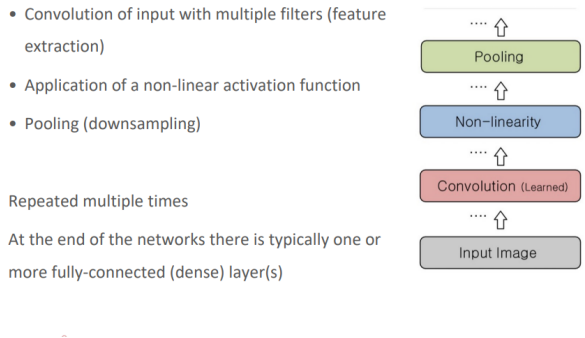


Figure 56: CNN

- cnn in the first layers immediately reduces the complexity of the input in a structured and meaningful way (based on learned weights)



pro and cons
yes: CNN and deep architectures in computer vision brought remarkable results
yes: "ready-to-use" implementations
No: not that easy to start from scratch in the design of a new architecture
No: there is also the need for very significant amounts of labeled data

Not written because not so much important, but look to all the actual challenge and success of computer vision nowadays (from 48 to 61)