Module 1

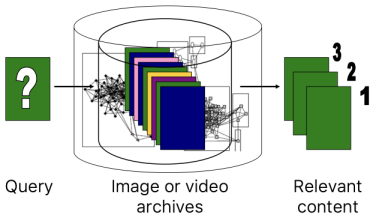
Computer Vision Intro

@February 20, 2023

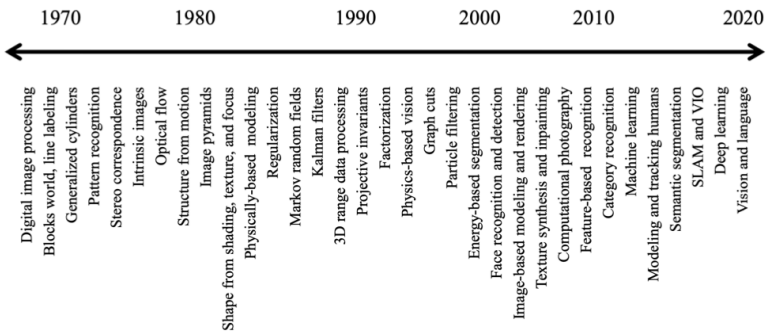
Started in the 50s with the objective to replicate the human vision system. Hubel and Wiesel's 1959 Nobel Prize winning study on the visual processing mechanisms in mammals, starting from the brain of a cat (they sticked some electrodes in the back of the cat's brain, where the primary visual cortex area is, and detected which stimuli made the neurons respond (discovery of the simple cells). The result was that visual processing starts with simple structure of the visual world (oriented edges), complexity is built up until it can recognize the complex world. Marr theorized in the 60s the steps for 3D object recognition.

Computer Vision deals with extraction of information from images. Image Processing is about improving the quality of images (to help computer vision tasks).

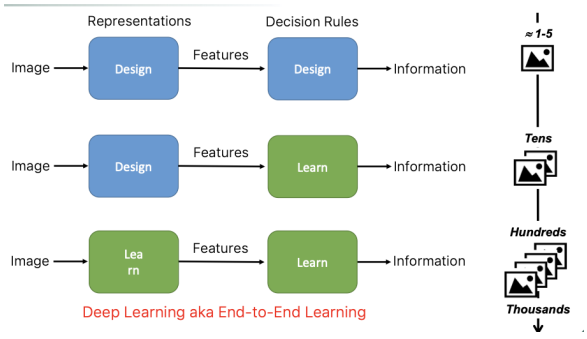
Object Detection, Classification. Measurements (extract 3D world properties - Depth Estimation, Structure from Motion (COLMAP)-Tracking), Mining Visual Data.

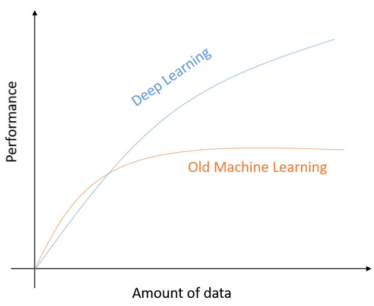


Computer sees a matrix of numbers basically (should be the 3D RGB system if I'm not wrong): basically depth, scaling, instance variation, illumination, shape and occlusions are challenges.



The first paradigm shift comes in the 00s as the Decision Rules are learned and not anymore implemented (Machine Learning approach - still used in controlled environment). In 2012, with AlexNet (first CNN to participate - and "win" - the ImageNet Large Scale Visual Recognition Challenge), the Deep Learning paradigm shift happens - we learn also the Features. This requires a huge load of data w.r.t. Machine Learning approach, but makes for the possibility of use in "in-the-wild" settings (e.g. my thesis). AlexNet was not actually the first CNN, there was some foundational work in the 90s - LeNet for digits&letter recognition - though it was not really acknowledged. AlexNet, then SuperVision, draws a lot of inspiration from it.





The graph shows that for small amounts of data, it is still actually better to use ML algorithms - ML algorithms are still actually used in Industry in controlled environments

Images

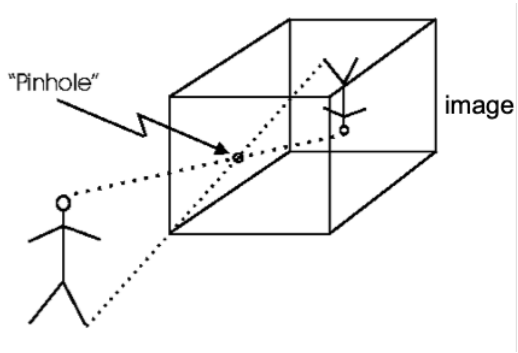
@February 23, 2023

Somehow, the objects in the world are hit by light, light is reflected by the objects and can be acquired by some sort of mechanism. Assumption: the object has a reflectant property that is not specular. The imaging device acquires the light reflected by the object in a scene (3D) to create a bi-dimensional representation of the scene. In Computer Vision we want to infer knowledge about the object from the digital images (so we want to basically invert the process). The elements involved are:

1. The geometric relationship between the scene points (points from we're observing the scene) and the image points
2. The digitization process: how we transform the light we capture in a matrix
3. How do we transform lights in those intensities: the radiometric relationship between the brightness of image points and the light emitted by scene points.

Pinhole Camera Model

Simplest model for cameras/imaging devices: it is a box with a very very small hole, called pinhole. Somehow, in the back of the box you have a photometric sensitive machine (an analog field - CMOS - etc), whatever acquisition which is sensitive to light.

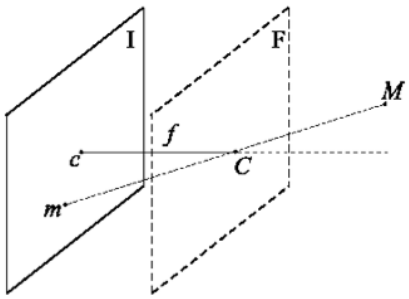


Geometrically, the image is achieved by drawing straight rays from the scene point through the hole, up to the image plane

The resulting image is reversed on both axes. The assumption made on the surface is that it is non-specular, else would make it a different result. It is a good approximation, but not optimal, because it doesn't have any lenses. Plus, you'd need eternal exposure times. Can't obtain useful images, but can obtain good mathematical models.

Geometric model is known as **Perspective Projection.**

M : scene point
m : corresponding image point
I : image plane
C : optical centre (pin hole)
Optical axis: line through C and orthogonal to I
c : intersection between optical axis and image plane (image centre or piercing point)
f : focal length
F : focal plane

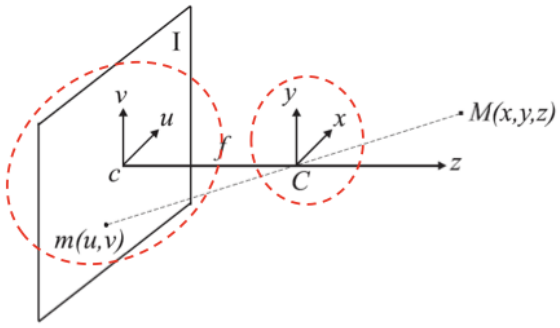


A point from the real world is hit by light and traverses the Optical Centre to the image plane. As a convention, we use capital letters for the 3D world. Focal length defines the distance between the Optical center and the Image Plane. For each point in the real world which are projected from the real plane we define an Optical Ray.

We now want to find a mathematical relationship between 3D and 2D points.

We need two coordinate systems, which now we see as parallel (though we can disprove this assumption later):

- Camera Reference System → it is the 3D reference system, because it is attached to my camera (its source is the pinhole) $M(x, y, z)$
- Image Reference System: $m(u, v)$

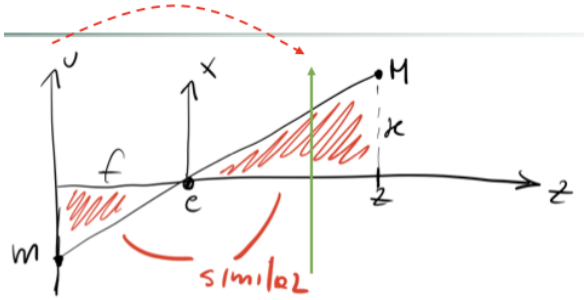


Distance between the two coordinate systems is f (focal length). For the perspective model the axes must be parallel.

Equations:

$$\frac{u}{x} = -\frac{f}{z} \implies u = -x\frac{f}{z}$$
$$\frac{v}{y} = -\frac{f}{z} \implies v = -y\frac{f}{z}$$

Point (0,0,0) is the pinhole (usually when working with images, the 0 is in the upper left corner).



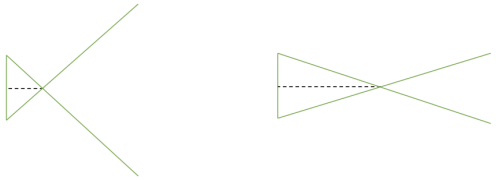
The relationships is based on geometrical similarity rules. In real world we don't get reflected images, but a solution to remove the minus, that comes from the geometrical relationship (image plane is behind the scene), is bringing the image plane in front of the pinhole. It is all mathematical.

Image coordinates are a scaled version of scene coordinates. It is a function of depth. z defines the depth (distance from the camera - which we lose in the 2D representation).

$$u = x\frac{f}{z} \qquad\qquad v = y\frac{f}{z}$$

Assume that the focal length (which is related to the distance between the light sensitive material and the pinhole) is fixed. The scale is proportional to depth. The object size stays the same, but the scale changes:

1. Depth (main reason) → distance of the object. Scales (inversely) proportionally to depth
2. Focal Length → the larger the focal length, the bigger the object in the image (rays traverse a greater distance). Short focal lengths have a larger field of view, while the longer the focal length, the more the field of view is restricted (it is compressed).

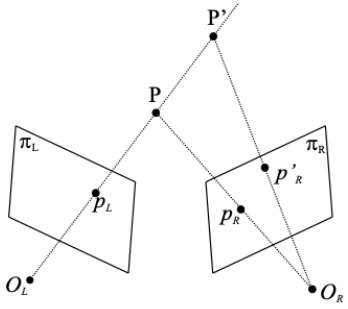


We scale the world inversely with respect to the depth. Scale invariance is one of the features I should wish in an algorithm. Fundamental concept.

Overall, we map a 3D space onto a 2D space, which means that information is lost. The mapping is not a bijection. The inverse problem is an ill-posed problem (a given image point is mapped onto a 3D line). I cannot univocally decide which plane I was observing (I can't say anything about the distance to the camera), every point maps to a ray.

Stereo-Vision Systems

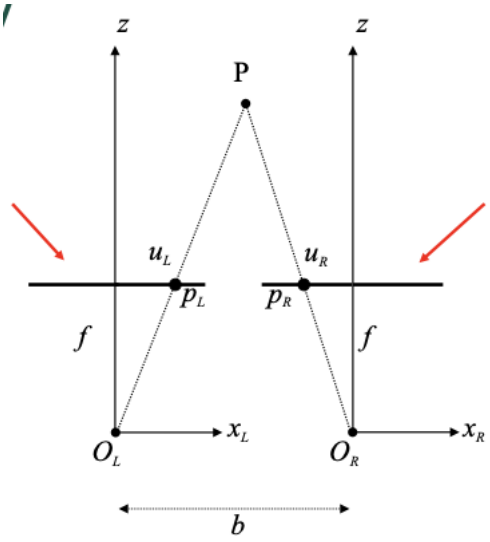
Use multiple images to infer the distance (triangulation): this is what happens in the human vision system, which is a stereo vision system. Before being able to do triangulation, I need correspondences (determine that the two points in the two images are the same): algorithm that observes matching. We assume that we are given the correspondences.



I need to make assumptions: I have two reference systems; in the coordinate system, the two cameras are "perfectly" aligned. (x, y, z) axes are parallel and the images are coplanar (same focal length).

Transformation is a translation (in Calibration we will see that there's also a rotational component).

$$P_L - P_R = \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{aligned} x_L - x_R &= b \\ y_L = y_R &= y \\ z_L = z_R &= z \end{aligned}$$



b is called the baseline (displacement of the two cameras - difference between the horizontal coordinates). I need "scene overlap" between the two cameras to make this work - logically, they should represent the same scene, otherwise why?

The baseline is in the Camera Reference System. So, we know that v_L and v_R are parallel and so the points coincide, but the us are different because of the baseline $\rightarrow u_L - u_R = b\frac{f}{z}$. This difference is also called "disparity". We can use it to derive depth:

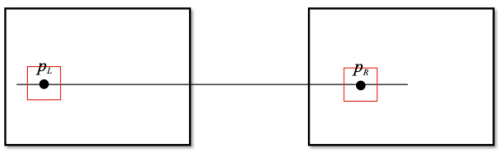
$$z = b\frac{f}{d}$$

Fundamental Relationship in Stereo Vision

Depth is inversely proportional to the disparity. Usually we know b and f as we can calibrate the imaging device to obtain them/are given as pre-defined specifics by the producers of the system. So it is possible to compute the disparity map for every point of the image.

@February 27, 2023

So, in the simplest case we've seen, I have two images, vertically aligned, with a baseline. This assumption that we're not moving in the vertical axis allows me to reduce the search space of correspondences to a parallel line.



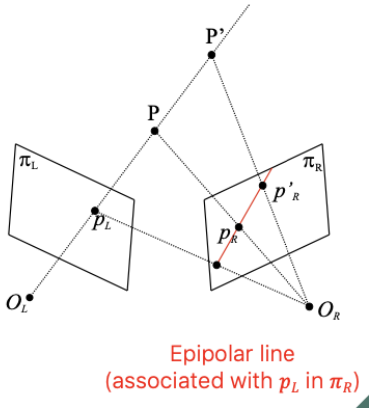
The simplest solution would be to check for the same color, though it could cause ambiguity. An evolution would be to use neighbors search and look for discriminative patterns. In uniform surfaces, the Projector, which is the third component in stereo rigs, can be useful for correspondence matching.

Epipolar Geometry

We break an assumption: the cameras are no longer aligned; we have a rotation-translation that moves the right camera. In epipolar geometry we can still search over a line; in detail, I can project the 3D line that relates to point P_L along the right plane and search there (this is called epipolar line).

Issue: I can compute this projection only if I know the transformation between the two cameras (relative mapping between the two cameras)

Plus, looking through oblique epipolar lines is awkward and less efficient. Plus, building a perfectly aligned stereo rig is highly difficult



Usually, the image is rectified (conversion from epipolar geometry to standard geometry - which uses horizontal and collinear conjugate epipolar lines) and then we look for correspondences. The transformation is an homography (see Module 2).

Stereo Correspondence

Given a point in one image, corresponding points are the point in the second image which is the projection of the same 3D point.

KITTI Dataset → Benchmark Suite for Autonomous Driving. Acquired using a Stereo Rig camera with disparity provided as label. The objective is depth estimation. When we look for matches at larger distances, we find smaller disparities. Points that are closer, have larger disparities.

Properties of Perspective Projection

The farther objects are from the camera, the smaller they appear in the image. The image of a 3D line segment of length L lying in a plane parallel to the image plane at distance z from the optical centre will exhibit a length given by:

$$l = L\frac{f}{z}$$

Need to know the 3D Reference frame in case of arbitrarily oriented 3D segments. But anyway, for given position and orientation, length always shrinks alongside distance.

Perspective projection maps 3D lines into image lines. Ratios of length are not preserved, unless the scene is planar and parallel to the image plane. Proportions are not maintained. We observe perspective distortion (e.g. the street seen by a driver).

The images of parallel 3D lines intersect (this is actually a consequence of the definition of vanishing point) at a point called **Vanishing Point**. If the lines are parallel to the image plane, they meet at infinity.

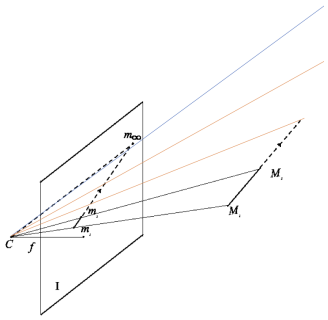
The vanishing point of a 3D line is the image of the point at infinity of the line (every line has its own point at infinity).

I can find it using the line parallel to the image plane that passes through the optical centre (and this is just one line) and see where it intersects the Image Plane.

All parallel 3D lines share the same vanishing point, in the sense that they meet at their vanishing point in the image.

If the 3D lines are parallel in the image plane (weak perspective) they will not meet at infinity. This means they stay parallel.

Weak Perspective → all lines in the image plane are parallel. The projection is called (scaled) orthographic projection. In this case the image of an object does not have the same size of the object

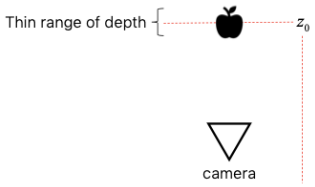


The operation to find the vanishing point is: I take the parallel line M1-M2, I translate it to be parallel to the image plane and passing through the optical center. The point in which this line intersects the image plane is the Vanishing Point. I am still in a pinhole camera model

$$\begin{cases} u = x \\ v = y \end{cases} \rightarrow \begin{cases} u = sx \\ v = sy \end{cases}$$

Scaled Orthographic Projection

Range of distances of the framed subject $[z_0 - \Delta z, z_0 + \Delta z]$. I have weak perspective if Δz (variation of depth) is small compared to z_0, like for example aerial images like Google Maps. This means the perspective scaling factor is constant (approximately).



$$\frac{f}{z_0 + \Delta z} \approx \frac{f}{z_0 - \Delta z} \approx \frac{f}{z_0}$$

$$u \approx \frac{f}{z_0} x \quad v \approx \frac{f}{z_0} y$$

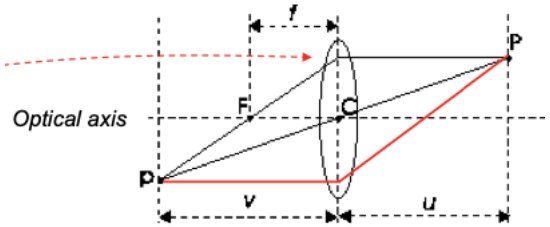
Depth of Field (DOF) and Lenses

In the pinhole device we have a huge issue: we have a tiny hole and only one ray can pass. But this makes the image a lot dark, which in reality is not sufficient.

- If I enlarge the hole, the image is bright but not anymore on focus (I project on a circular region → I don't have rays anymore, but cones of light).
- If I integrate over time, which means that I can obtain bright scenes from the Pinhole Model, I can only work with static scenes, otherwise I incur in motion blur.

Pinhole Model has Infinite DOF: we always have on focus points. This condition decays if we enlarge the pinhole

We can use Lenses to gather more light from a scene point and focus it on a single image point. This enables smaller exposure times and avoids motion blur in dynamic scenes.



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

P : scene point
p : corresponding focused image point
u : distance from P to the lens
v : distance from p to the lens
f : focal length (parameter of the lens)
C : centre of the lens
F : focal point (or focus) of the lens

Attention to this!!

The DOF is no longer infinite: so I don't have all points simultaneously on focus. I see on focus only points up to a certain f.

Modern camera systems (complex optical systems) use multiple lenses

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

Thin Lens Equation

All optical rays are deflected so that they pass through F except for the ones that pass through the optical center.

The thin lens equation links the planes (f stays fixed). If I vary u, v varies accordingly.

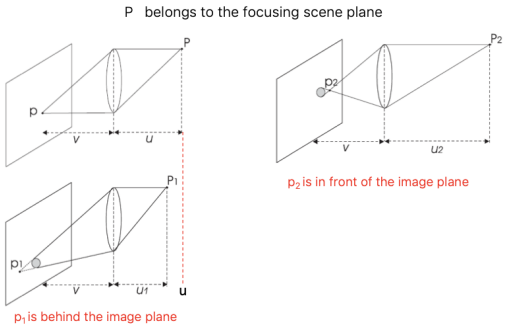
- Choosing the distance of the image plane determines the distance at which scene points appear on focus in the image

$$\frac{1}{u} + \frac{1}{v} = \frac{1}{f} \rightarrow u = \frac{vf}{v - f}$$

- To acquire scene points at a certain distance we must set the position of the image plane accordingly

$$\frac{1}{u} + \frac{1}{v} = \frac{1}{f} \rightarrow v = \frac{uf}{u - f}$$

Given the chosen position of the image plane, scene points both in front and behind the focusing plane will result out-of-focus, thereby appearing as circles, known as Blur Circles/Circles of Confusion, that are not points.



Blur Circles

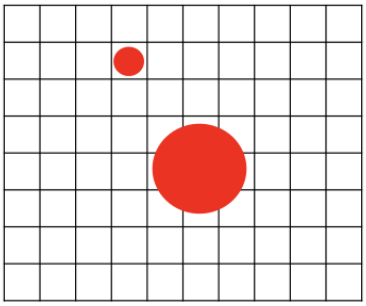
We essentially trade a small exposure time for less DOF.

Diaphragm

@March 2, 2023

We can control somehow the Circle of Confusion, using a Diaphragm. Imagine a camera sensor (photo-matrix); if they hit on a specific pixel, we don't know the real dimension of the photo-sensitive cell.

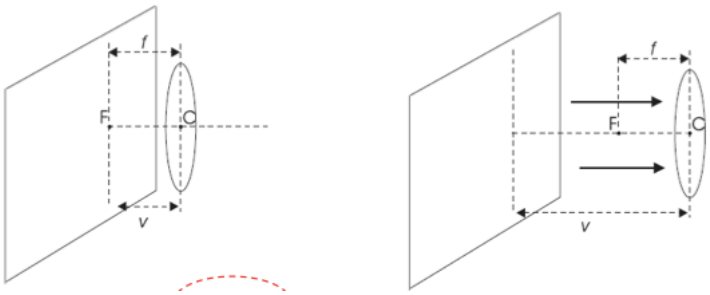
If the blur circle is contained in a single cell, the point will still look on-focus. Else, it will be blurred. The dimension of the photo-sensors gives us a tolerance on the sharpness of the image: this means, it influences the DOF



The diaphragm controls the amount of light I'm letting through (the larger the openings the larger the blur circle). The point is, by closing the diaphragm we increase the DOF, but need more exposure, which can lead to motion blur. It's a trade off.

Focusing Mechanism

Helps us with Depth Of Field. It allows the lens to translate along the optical axis w.r.t. the fixed position of the image plane. By moving the mechanism, I increase/decrease the sharpness of the image.



First end: minimum I can move my lens from: u is $\infty \rightarrow$ the focus is the farthest from the camera
Second end: we increase v and obtain focus for closer point up to $u \rightarrow$ the focus is closer to the camera

Image Digitization

Images are a discretized version of the world. We are working with a planar sensor composed of photo-sensitive elements that register the irradiance into an electric quantity (voltage) that will be then converted.

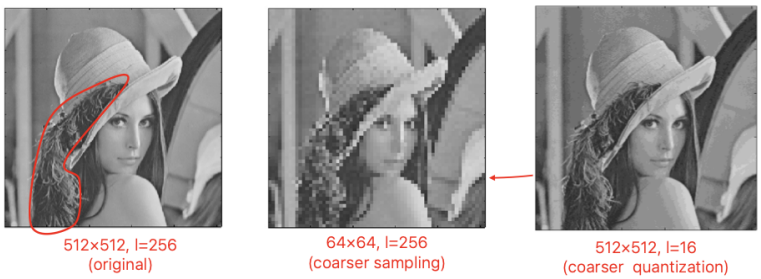
First step is sampling: the photosensors are finite. The number of photosensors determines the resolution. The sampling is according to a two dimensional grid. Then there's quantization: we need to quantize the continuous values to the discrete ones (fixed number of levels). Apart from the level we get from the grid, the quantization process will give only a finite number of labels.

Sampling consists in creating a matrix of pixels by sampling the planar continuous image along both the horizontal and vertical directions. Pixels: both intensity and position.

$$I(x,y) \implies \begin{bmatrix} I(0,0) & I(0,1) & ... & I(0,M-1) \\ \vdots & \vdots & \vdots & \vdots \\ I(N-1,0) & I(N-1,1) & ... & I(N-1,M-1) \end{bmatrix}$$

Quantization → takes the continuous electrical range values and transform them into discrete values (we work with computers so: $l = 2^m$). m depends on how many pixels we want to use to represent the intensity (how much a $N \times M$ image occupies in memory: $N \cdot M \cdot m$). To represent colors, you essentially occupy more memory (typically, it is one byte for each of the RGB channels).

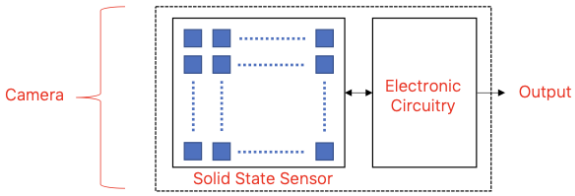
In Classic Computer Vision, most of the algorithms don't need color, so it is often discarded. CNN can exploit all the channels.



Loss of dimension → we lose details. Lots of Aliasing
Loss of quantization → loss of smoothness, we have less possibility of changing the colors.

Camera Sensors

2D arrays of photodetectors (photo-gates, photo-diodes). Camera sensors come with electronic circuitry. We have in fact also the Analog-Digital Converter circuitry on digital cameras.



During Exposure time, each detector converts the incident light into a proportional electric charge (i.e. photons to electrons). The companion circuitry reads-out the charge to generate the output signal, which can be either digital (thus the ADC circuitry) or analog. Nowadays, there's never a continuous image in practice. The image is sensed directly as a sampled signal.

We have two main sensor technologies:

- CCD (Charge Coupled Devices) → better technology. Higher SNR, DR and better uniformity. Medical Applications.
- CMOS → more common. The electronic circuitry can be integrated within the same chip as the sensor ("one chip camera") - smartphones, webcams… they can be miniaturized more and are also cheaper. Also, they can focus on arbitrary regions instead of accessing the whole image

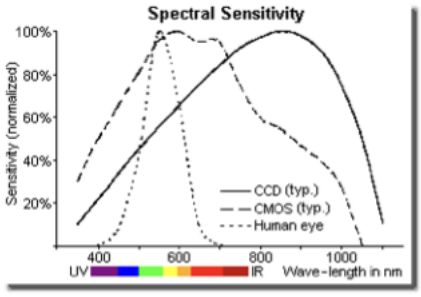
Color Sensors

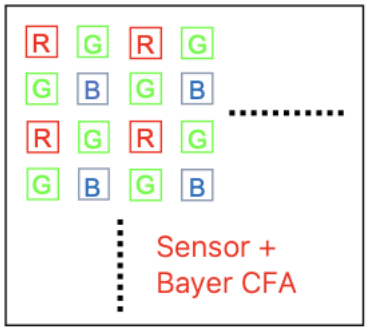
Spectral Sensitivity of humans is mostly concentrated around the green. CCD and CMOS have much wider range of spectral sensitivity (from near-ultraviolet to near-infrared). The point is, CCD and CMOS cannot sense color.

Actually, the only thing we can do to see colors, we can only put optical filters (Color Filter Array) in front of the photodetector. This way, you render each pixel sensitive to a specific range of wavelength. The issue here now is that we're losing resolution, because we're subsampling (to obtain one pixel I am subsampling

2 times for green and 4 times for red and blue). We combine the subsamples to obtain the full channels: more in detail, we interpolate samples from neighboring pixels (demosaicking).

We can use the more expensive full-resolution: three sensors and a prism (three times the photodetectors).





Signal-to-Noise Ratio

It tells us how much signal we're able to measure w.r.t. the random noise present. The irradiance already contains some noise. The pixel value is not deterministic, but rather a random variable
Main noise sources:

- Photon Shot Noise → the photons take some time to arrive and is governed by a Poisson statistic. Therefore, the number of photons collected during exposure time is not constant
- Electronic Circuitry Noise → generated by the electronics which reads-out the charge and amplifies the resulting voltage signal
- Quantization Noise → we're approximating (the ADC conversion in digital cameras)
- Dark Current Noise → related to thermal excitement due to the electronics.

Quantifying the strength of the true signal with respect to the unwanted fluctuations induced by noise →The higher the better.
Can be expressed both in dB and bits.

Dynamic Range

When you observe a scene, you have both areas of high illumination and areas of shadows. Ratio between:

- E_{min} which is the minimum detectable irradiation, that depends on how much noise is already there
- E_{max} which is the saturation irradiation (that depends on the hardware) → the amount of light that would fill up the capacity of a photodetector

The DR is defined as $\frac{E_{max}}{E_{min}}$ → good if it is higher. Also specified in dB or bits.

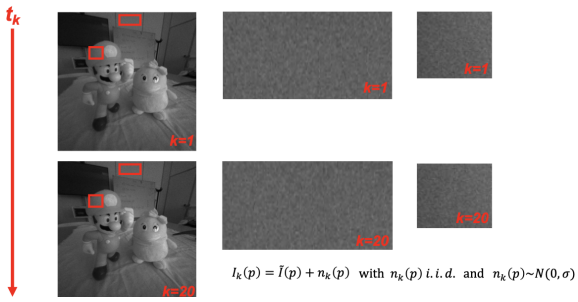
High Dynamic Range

Spatial Filtering

@March 6, 2023

We need to deal with noise. In images, noise appears as random fluctuations in values, and somehow, this noise varies across

We are observing an image at time t_0. In the cut-out sections we can observe spurious variations. The chosen sections, where the noise is more noticeable, are in regions that present a uniform color (wall, Mario's hat…).



At time t_{20} we see that the cut-outs are different: noise is a random process that depends on all four sources seen before.

If we're given a point p on the image, we would want to know $I_k(p)$, the real intensity that pixel should have at that time. This is computed as:

$$I_k(p) = \tilde{I}(p) + n_k(p)$$

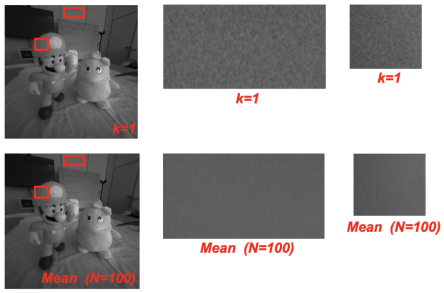
where $\tilde{I}(p)$ is the ideal value and $n_k(p)$ is the noise of p at time k. I need assumptions: $n_k(p)$ is independent (noise on a point is not dependent on the other points) and identically distributed (come from the same distribution), and $N_k(p) \backsim N(0, \sigma)$

I could solve this issue, if the scene and camera are fixed, by averaging across time.

$$O(p) = \frac{1}{N}\sum_{k=1}^{N} I_k(p) = \frac{1}{N}\sum_{k=1}^{N}(\tilde{I}(p) + n_k(p)) = \frac{1}{N}\sum_{k=1}^{N}\tilde{I}(p) + \frac{1}{N}\sum_{k=1}^{N} n_k(p) \cong \tilde{I}(p)$$

Remember the noise has a 0-mean.

You can see in the image in slide 6 that by using the mean across time you can reduce noise.



But what if I don't have time or I can use only a single image? If we can't exploit time, you will exploit space.

$$O(p) = \frac{1}{|K|}\sum_{q \in K} I(q) = \frac{1}{|K|}\sum_{q \in K}(\tilde{I}(q) + n(q)) = \frac{1}{|K|}\sum_{q \in K}\tilde{I}(q) + \frac{1}{|K|}\sum_{q \in K} n(q) \cong \tilde{I}(q)$$

K is called "Supporting Window" and it defines a neighboring region close to my pixel. There's an issue still. We can have defects along the edges if we use too large windows (edges define the contours of objects). It's a matter of trade-off. This is some sort of De-noising (Spatial) Filters

Image Filters

Image filters (or spatial operators) are image processing operators that compute the new intensity of a pixel p based on the intensities of those belonging to a neighborhood (support) of p. Filters can be used to "de-noise" an image and sharpen it. Why: better looks and improving quality for elaboration (edge enhancement). Important family of filters: Linear and Translation-Equivariant (LTE) operators. Main definition comes from signal theory: their application in image processing consists in a 2D convolution between the input image and the impulse response function (point spread function or kernel) of the operator. LTE operators are used to extract features in CNNs.

LTE Operators and Convolution

A local operator is linear if I take a 2D signal, an operator T applied to the signal → Linear combination is legit.

Given an input 2D signal $i\{x, y\}$, a 2D operator, $T\{.\}$: $o(x, y) = T\{i\{x, y\}\}$ is said to be linear iff:

$$T\{ai_1(x,y) + bi_2(x,y)\} = ao_1(x,y) + bo_2(x,y)$$

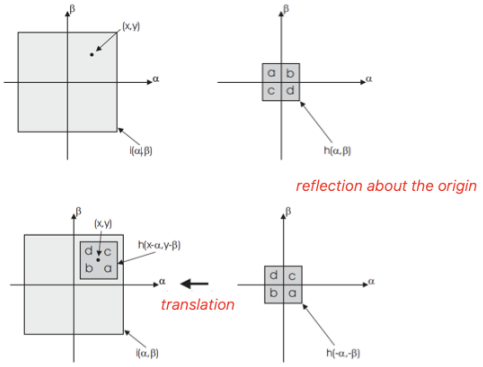
with $o_1 = T\{i_1\}, o_2 = T\{i_2\}$, a, b two constants.

The operator is said to be Translation-Equivariant iff $T\{i(x - x_0, y - y_0)\} = o(x - x_0, y - y_0)$

If the operator is LTE, the output signal is given by the convolution between the input signal and the impulse response $h(x,y) = T\{\delta(x,y)\}$ of the operator

In signal theory each signal can be expressed as a combination of impulse signals. The idea is to multiply and sum (integral). $h(x,y)$ is the response of the input to my operator. δ is the Dirac delta function

$$o(x,y) = T\{i(x,y)\} = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(\alpha,\beta)h(x-\alpha, y-\beta)\,d\alpha\,d\beta$$



What happens? I have $h(\alpha,\beta)$, in my integral h is applied to $-\alpha$ and $-\beta$. This means that I'm reflecting about the origin on both directions. Then I translate by x and y. Convolution is about taking multiplication and sum of input and response reflected and then translated.

$$o(x,y) = i(x,y) * h(x,y)$$

Properties:

- Associativity → gives us the possibility of performing more efficient operations through chained convolutions.

  $f * (g * h) = (f * g) * h$
- Commutative

  $f * g = g * f$
- Distributive w.r.t. Sum

  $f * (g + h) = f * g + f * h$
- Commutes with Differentiation → useful in edge detection. You can compute the derivative you want if you need to compute the derivative of the convolution.

  $(f * g)' = f' * g = f * g'$

Correlation

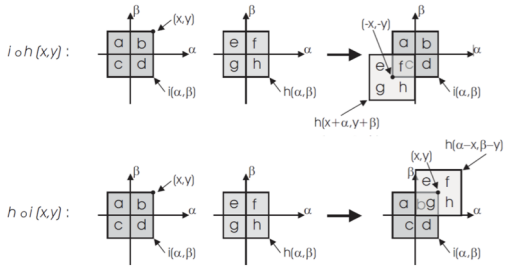
Gives me as output how much a signal is correlated to another signal. The formula is similar to the convolution.

$i(x,y) \circ h(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} i(\alpha,\beta)h(x+\alpha, y+\beta)d\alpha d\beta$

$h(x,y) \circ i(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(\alpha,\beta)i(x+\alpha, y+\beta)d\alpha d\beta$

Correlation is not commutative:

$$h(x,y) \circ i(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(\alpha,\beta)i(x+\alpha, y+\beta)d\alpha d\beta = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} i(\xi,\eta)h(\xi-x, \eta-y)d\xi d\eta = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} i(\alpha$$



I have my input signal i and h (response). If I correlate, h is not reflected and is just translated. The same operation by changing the order is more similar to convolution: I still don't reflect, but I translate over X and Y.

If I have a symmetric kernel (h is an even function), I have that convolution is the same as correlation of h w.r.t. i.

CNNs actually compute correlations (we don't reflect).

Discrete Convolution

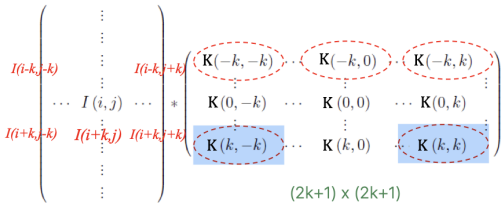
We're in the discrete world:

$$O(i,j) = T\{I(i,j)\} = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} I(m,n)H(i-m, j-n)$$

In this case, my discrete 2D input and output signals are respectively $I(i,j)$ and $O(i,j)$, and $H(i,j) = T\{\delta(i,j)\}$ is the kernel of the discrete LTE operator, i.e. the response to the 2D discrete unit impulse (Kronecker delta function), $\delta(i,j)$.

Properties and definitions stay the same.

In Image Processing, both the input image and the kernel are stored into matrices of given finite sizes, with the image being much larger than the kernel. One would cycle through the kernel. Conceptually, to obtain the output image we need to slide the kernel across the whole input image and compute the convolution at each pixel.



Remember! You flip the kernel before you work.

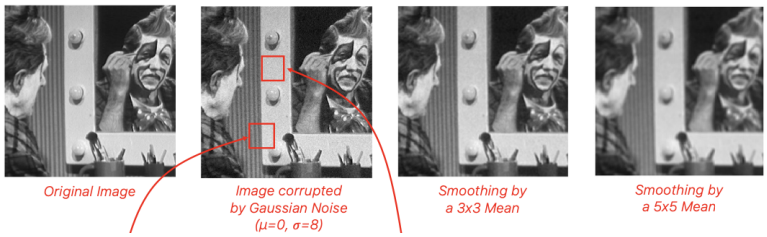
As for the borders:

- I can crop → lose $2k$ rows and $2k$ columns. Approach for classic CV
- I can use forms of padding (e.g. zero-padding) → CNN usual approach.

Mean Filter

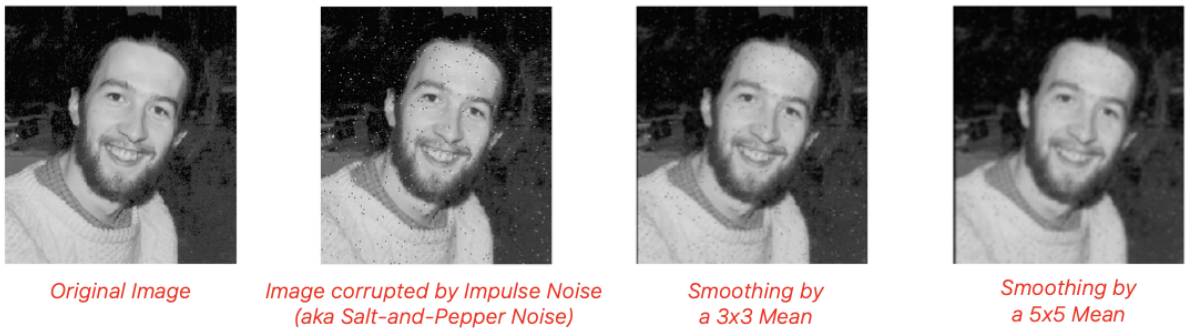
It is the simplest way to de-noise an image. It is also the fastest. Still an LTE operator. What is really happening in this case is that I'm applying a low-pass filter (we cut the high frequencies)→ image smoothing. As the higher frequencies in images are where there's abundance of details (e.g. hair), the mean filter loses details.

Related to scale of objects. Smoothing image = removing details = influencing the scale of details in a scene. Used to create scale space = smooth image at different levels to obtain different levels of details.



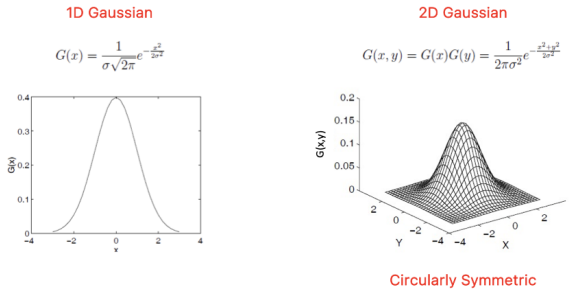
@March 9, 2023

Impulse noise (aka Salt&Pepper Noise) introduce a huge amount of outliers inside an image. Can be introduced by an electrical defect in the sensor/faulty pixels, transmission errors (one bit gets swapped), algorithmic error (e.g. computation of disparity maps). Noise can even worsen: we spread the outliers all over the other pixels. So Mean Filter cannot really deal with impulse noise. It is the faster, but not the most effective

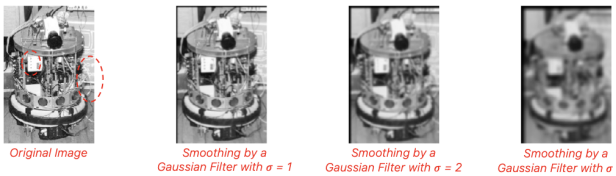


Gaussian Filter

Weights follow a 2D gaussian distribution. I can center the pixel and assume that pixels closer to the center are more related to the object than the farther ones.



Take into account an image with a high number of details. When we apply a Gaussian Filter, we see that somehow, we're smoothing the details, we lose some of them. With small σs, we are de-noising a little. By increasing σ, the image is getting smaller because of cropping, we don't see the details anymore, but we can still have a bigger picture.



We use in CV GF to build scaled spaces to extract invariant features to achieve scale invariance. So σ is somehow related to the "scale of interest"

How can we discretize a Gaussian Kernel? By sampling the corresponding continuous function, for a finite number of values, chosen according to the following observations:

- The larger the size, the more accurate the discrete approximation

- The larger the size, the more computation is necessary
- Higher σ = higher smoothing

We need a rule of thumb to decide the dimension of the kernel given the σ value. We know that the support of the Gaussian is theoretically infinite, but we can say that $[-3\sigma, 3\sigma]$ captures 99% of the energy (area) of the function. Therefore $(2k+1) \times (2k+1)$ with $k = 3\sigma$ is a valid choice.

As the Gaussian is circularly symmetric, I can exploit separability: the product of gaussians is always a gaussian. We can split the original convolution in two 1D convolution. Now I will apply two 1D kernels, via the associativity property (on the respected assumption that it is a LTE kernel). We can see that the speedup factor will be around k (now we reach a complexity of $O(k)$).

$$\begin{cases} \text{2D Filters} : N_{OPS} = 2(2k+1)^2 \\ \text{1D Filters} : N_{OPS} = 2 \cdot 2(2k+1) \end{cases}$$

Dividing the two values you obtain the speedup factor.

As for impulse noise, it's still bad because it will still spread it, it is still a weighted sum. Also, when you apply gaussian filters, you are not creating any kind of artifact, you don't introduce new structures.

Median Filter

Non-Linear filter that works well with impulse noise. Median is the "middle" element in a sorted collection. Computing the median of the sum is not the same as computing the sum of the medians, and here lies the non-linearity. Essentially we apply the filter to a window, sort the values and assign to the pixel the median. In this case, if there is impulse noise, in a sorted array it will be at the extremities and not in the middle, so it won't be taken.

Also, they preserve sharpness, because we're not averaging values, but choosing a value.

Still, Median Filter cannot deal with Gaussian-Like Noise. The solution is to mix-it-up and apply Median then Gaussian.

Bilateral Filter

Non-Linear edge preserving smoothing filter.

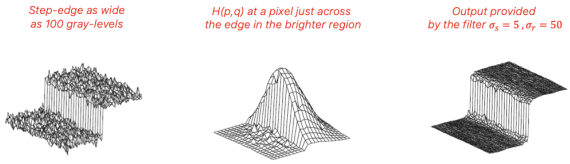
I have an edge when I have a sharp difference on pixel intensity.

I can use the values of the intensity to identify where to smooth and where don't. So I also exploit the intensity pixel value. $O(p) = \sum_{q \in S} H(p,q) \cdot I_q$.

We define a kernel $H(p,q) = \frac{1}{W(p)} G_{\sigma_s}(d_s(p,q)) G_{\sigma_r}(d_r(p,q))$ which is a multiplication of two Gaussian functions, one that depends on the spatial distance and one that depends on the intensity values.These gaussians depend on distances, one spatial (classic euclidean) and one a distance on intensities (that can be an L1 Norm). The products of the two gaussians influence each others. One thing (that also features in the Gaussian) is the Normalization factor: I weight the product using the sum of all the values, so that we don't change the overall intensities → the final sum would have a different "energy".

$$\begin{aligned} d_s(p,q) &= \|p-q\|_2 \rightarrow \text{ Spatial Distance} \\ d_r(I_p, I_q) &= |I_p - I_q| \rightarrow \text{ Intensity Distance} \\ W(p) &= \sum_{q \in S} G_{\sigma_s}(d_s(p,q)) G_{\sigma_r}(d_r(p,q)) \rightarrow \text{ Normalization Factor} \end{aligned}$$

The sigmas are hyper-parameters. The issue is that it will be less efficient than a Gaussian/Mean… etc. At each pixel we compute run-time a specific kernel.



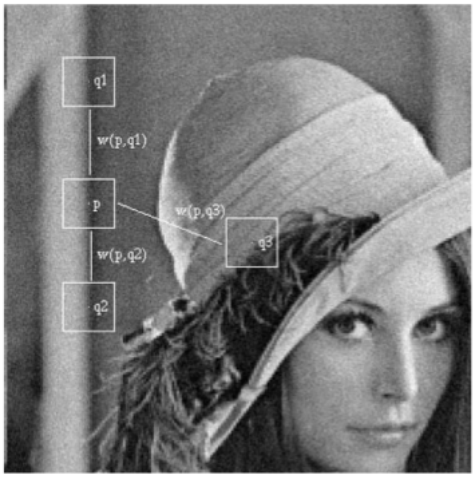
Given the supporting neighborhood, neighboring pixels take a larger weight as they are both closer and more similar to the central pixel. At a pixel nearby an edge, the neighbors falling on the other side of the edge look quite different and thus cannot contribute significantly to the output value due to their weights being small.

Non-Local Means Filter

Another non-linear edge preserving smoothing filter.

The idea is that I can use multiple windows to decide how my output should look like. These windows should represent patches in the image that have a similar look and hopefully different noise, to be inspired to determine a filtering that preserves sharpness. I weight them based on distance between intensities.

$$O(p) = \sum_{q \in S} w(p,q) I(q)$$
$$w(p,q) = \frac{1}{Z(p)} e^{\frac{-\|N_p - N_q\|_2^2}{h^2}}$$
$$Z(p) = \sum_{q \in I} e^{\frac{-\|N_p - N_q\|_2^2}{h^2}}$$



h is an hyperparameter called bandwidth and must be found with trial-error.

Image Corrupted by Gaussion Noise (σ =20)

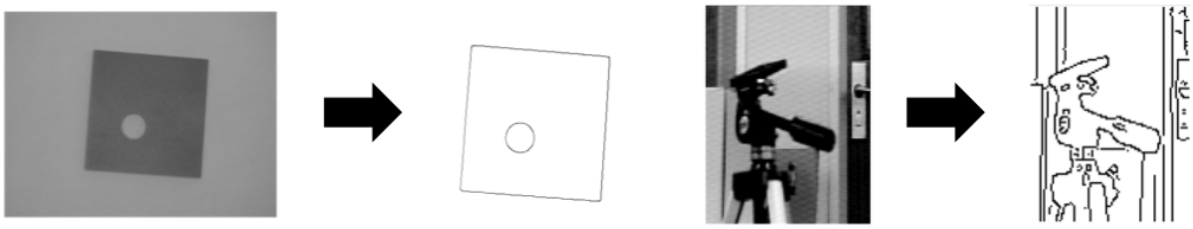
Gaussian Filter

Non-local Means Filter
$N = 7\times7$,
$S = 21\times21$,
$h = 10 \cdot \sigma$

It will work spectacularly, but has a high computational cost. The original paper's solution is using instead of the whole image, just a small window where to search patches.

Edge Detection

Why are we so interested in edges? Edges are local features of the image that capture lots of information related to its semantic content. Edges are the contours of something and represent the border between objects or can be related to the object itself. Can also measure (if the camera is calibrated).



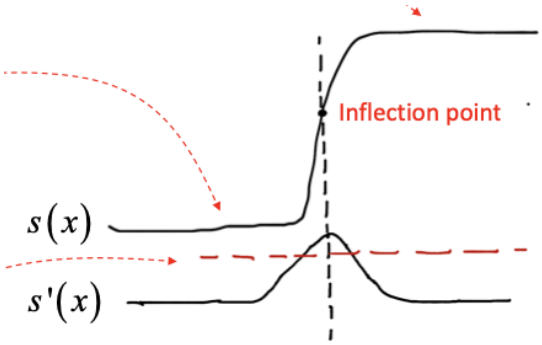
Edges can be defined as pixels that can be thought of as lying exactly in between image regions of different intensity, or, in other words, to separate different image regions.

@March 13, 2023

An edge can be seen as a sharp change of a 1D signal.

We can use derivatives, which give a function that's 0 in the constant zones and in the transition region increases up to the inflection point and then descends until 0.

The simplest Edge Detector relies on thresholding the absolute value of the derivative of the signal.

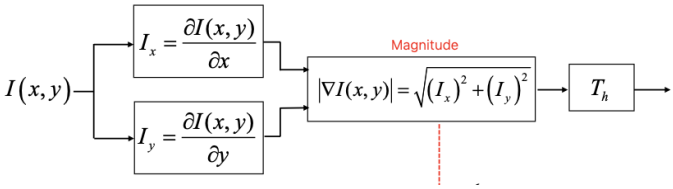


In the transition region between the two constant levels, the absolute value of the first derivative gets high (and reaches its peak at the inflection point).

Why the absolute value? We have a positive edge. If we have a negative edge, the derivative has a reversed graph, which would yield negative values in the transition region.

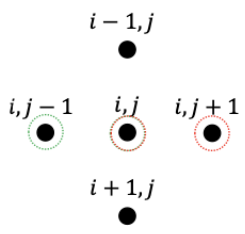
2D Step-Edge

I need to take into account strength and direction. I therefore will use the gradient, which is a vector composed of the partial derivatives along horizontal and vertical axis. We no longer have a scalar. We need to look at the place where is the maximum variation, which is given by the gradient's direction.



Structure of the new Edge Detection

We still miss something, which is the orientation, the direction. This computation depends on whether you care about polarity (the sign) (partial or full reverse tangent).



$\frac{\partial I(x,y)}{\partial x} \cong I_x(i,j) = I(i,j) - I(i,j-1)$, $\frac{\partial I(x,y)}{\partial y} \cong I_y(i,j) = I(i,j) - I(i-1,j)$ backward

$\frac{\partial I(x,y)}{\partial x} \cong I_x(i,j) = I(i,j+1) - I(i,j)$, $\frac{\partial I(x,y)}{\partial y} \cong I_y(i,j) = I(i+1,j) - I(i,j)$ forward

As a discrete approximation, I can work using differences between pixels (either backward or forward). Both methods yield a vector. We can also use central differences. In this case, I can compute it using correlation:

$$I_x(i,j) = I(i,j+1) - I(i,j-1) \quad I_y(i,j) = I(i+1,j) - I(i-1,j)$$

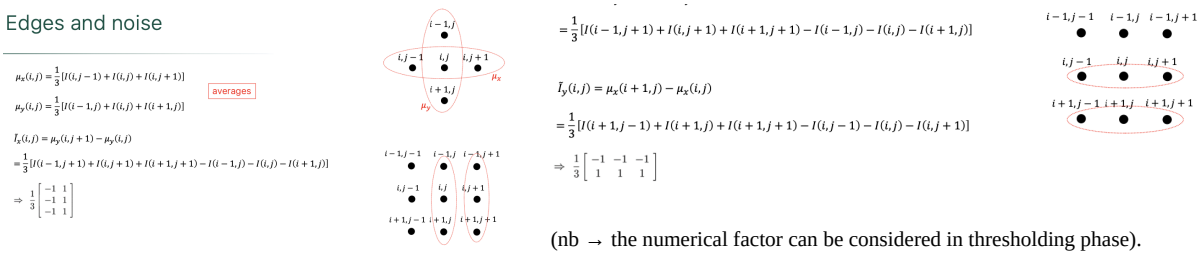
with correlation kernels:

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

To estimate the magnitude we can use different approximations: L2 norm (euclidean norm), L1 norm (Manhattan Norm), $L_{\max}$ Norm (which is the fastest and the best one, because it is invariant w.r.t. edge direction). L1 and L2 will yield not-isotropic responses.

But, still, we have noise, so we will detect noise-induced spurious edges. Derivatives amplify noise (taking derivatives of noisy signals is an ill-posed problem: the solution is not robust to input variations). We can use filters to de-noise the image, but we would blur also real edges, therefore we would anyway lose sharpness.

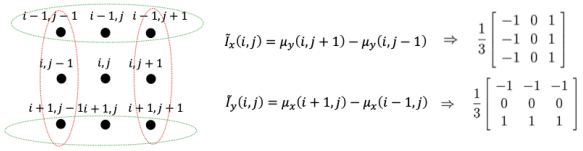
What about computing differences of averages? We join the two steps, and to try to avoid smoothing across edges the two operations are carried out along orthogonal directions.



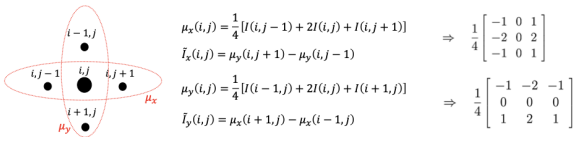
(nb → the numerical factor can be considered in thresholding phase).

Prewitt and Sobel Operators

Prewitt operator → approximate partial derivatives by central differences.



Sobel operator → inspired by the central pixel assumption for Gaussian Filtering: increase the importance given to the central pixel close to the edge: this is more isotropic to the direction.

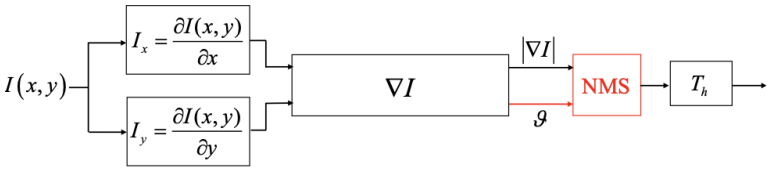


NMS (Non-Maxima Suppression)

Issue with edge detection, as using gradient thresholding to detect edges is inherently inaccurate. It is difficult to choose the right threshold, as the image contains meaningful edges characterized by different contrast/illumination and trying to detect weak edges implies poor localization of the stronger ones. We want sharp edges, so we want ideally a response of one pixel for an edge using just one threshold. A better approach then for edge detection would be looking for the local maxima of the absolute value of the derivative of the signal.

Use Non-Maxima Suppression (NMS). When dealing with images (2D signals) one should look for the maxima of the absolute value of the derivative (i.e. the gradient magnitude) along the gradient direction (i.e. orthogonal to the edge direction). The issue is that we don't know in advance the correct direction to carry out NMS. The direction has to be estimated locally.

OpenCV approximate direction of the gradient to a 45° grid field. This allows to consider direction in a discretized way, and then compute NMS. NMS will put at 1 only the points whose gradient is greater than the ones in the closest points in the discretized directions.
Instead of approximating one could do an interpolation that can take into account all the points involved in the NMS process, according to their weight.



Canny's Edge Detector

Canny proposed to set forth quantitative criteria to measure the performance of an edge detector and then to find the optimal filter with respect to such criteria:

1. Good Detection → the filter should correctly extract edges in noisy images (i.e. should be robust w.r.t. noise)
2. Good Localization → the distance between the found edge and the "true" edge should be minimum
3. One Response to One Edge → the filter should detect one single edge pixel at each "true" edge. Should provide no multiple response

Addressing the 1D case and modeling an edge as a noisy step, Canny shows that the optimal edge detection operation criteria can be mapped by computing local extrema of the convolution of the signal by a first order Gaussian derivative.

A straightforward Canny edge detector can be achieved by:

- Gaussian smoothing
- Gradient computation
- NMS along the gradient direction

To speed up the calculations, we can exploit the separability of the Gaussian function, in addition to the derivability properties of the Convolution operation. So it becomes computationally efficient and doesn't actually touch the image.

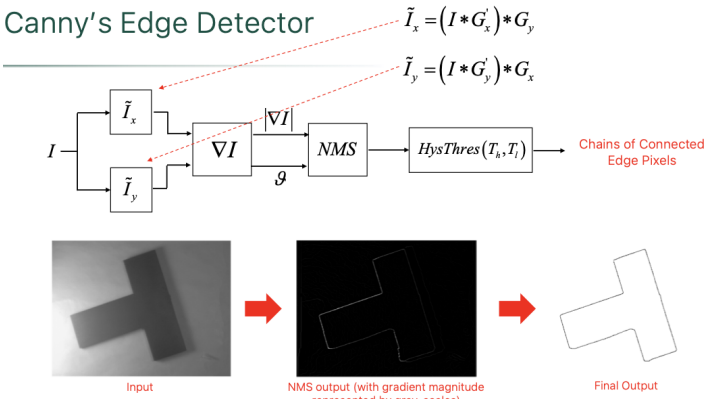
NMS is often followed by a thresholding of the gradient magnitude to help distinguish between true "semantic" edges and unwanted ones (phenomenon of edge streaking: I'm losing weak edges). Edge streaking may occur when magnitude varies along object contours.

Hysteresis approach relying on a higher and lower threshold: a pixel is taken as an edge if either the gradient magnitude is higher than T_h or higher than T_l and neighbor of an already detected edge.

@March 20, 2023

The pipeline gets modified:

- The partial derivative implies also a gaussian smoothing
- NMS output will be thresholded using the hysteresis Threshold

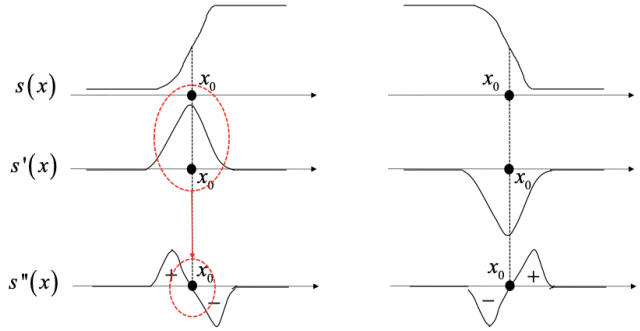


Final pipeline for Canny Edge Detector

In industrial settings, where you can actually control your environment, it is better not to fall in the case shown above: do not create shadows. Anyway, Canny can get you out of the issue.

Zero-Crossing

We have said that we can have both positive and negative edges and this depends on the polarity. We can use first derivatives, but also use the second.



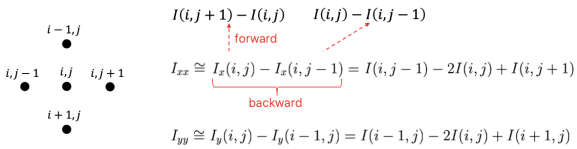
We consider an edge where the second derivative crosses the zero (zero-crossing point). Little issue: we have to compute the Hessian Matrix, aka a lot more operators, and this means the method is expensive.

$$\mathbf{n}^T\mathbf{H}\mathbf{n} \text{ with } \mathbf{n} = \frac{\nabla I(x,y)}{\|\nabla I(x,y)\|} \text{ (unit vector along gradient's direction)}$$
$$\mathbf{H} \text{ as the Hessian Matrix}$$

We use an approximation: the Laplacian (Marr&Hildreth):

$$\nabla^2 I(x,y) = \frac{\delta^2 I(x,y)}{\delta x^2} + \frac{\delta^2 I(x,y)}{\delta y^2} = I_{xx} + I_{yy}$$

We can use differences of intensities to approximate first and second order derivatives again to the discrete case (in particular, differences of differences). We will use both forward (first order) and backward (second order) differences.



The overall second order term then can be approximated to one kernel. It can be shown that the zero-crossing of the Laplacian typically lay close to those of the second derivative along the gradient and it is much faster to compute.

We still miss a de-noise operation: second-order derivatives STILL amplify noise.

Laplacian of Gaussian (LOG)

Before using the laplacian, use a smoothing step (Marr&Hildreth proposed a Gaussian Filter).

- Gaussian Smoothing: $\tilde{I}(x,y) = I(x,y) * G(x,y)$
- Laplacian to compute second order derivatives
- Extraction of zero-crossing: $\nabla^2 \tilde{I}(x,y)$

with practical implementation deploying properties of convolution to speed-up the computation.

Observing exactly zero will be actually improbable, verging on impossible. It will be more probable to observe changes of sign between consecutive pixels. At this point the actual edge will be the one where the absolute value of the pixel is smaller (closer to the "true" zero-crossing). In principle you can take the criterion you like best though.

Usually, you enforce a threshold: a limit after which you cut the values, to help discarding spurious edges.

Using Gaussian step, we can control σ, which influences the smoothing (the higher the σ, the lower the number of edges). The best thing to do will be to threshold the response of a small σ. Instead of working with smoothed differences, it is more useful to exploit the LOG and deal with noise through σ (thus controlling the level of noise). Another thing, somehow we can use σ to build a scale space: extract fine or less fine features.

Local Invariant Features

We want to look for correspondences between two or more images for various tasks (stereo images, disparity maps). Our first algorithm was based partially on geometry and partially on pixels. But removing all the basic assumptions we used, we can undergo rotations, translations, scale changes, illuminations…

Finding correspondences means finding which two points in two different images represent the same pixel. Establishing the correspondences can be difficult as the views may look different (lighting, viewpoint changes…).

Panorama Stitching

If you have four points in two images, you can compute a transformation to obtain the correspondences (homography). We need to look for salient points (distinctive points in the image - e.g. the peaks or distinctive sections in the region). Therefore you firstly need a detector. Then I need to encode these details to see what I should keep: I will need a second algorithm for descriptions (take into account also the neighbors of the point). Now we need Matching.

Various applications (*cries in COLMAP*)

We need:

- **Detection**: we need to detect the salient/feature/interest/key points
- **Description**: compute a suitable descriptor based on pixels in the keypoint neighborhood
- **Matching** descriptors between images.

Descriptors must be invariant to as many transformations as possible

The two main components are Detectors and Descriptors, which are used to achieve matching. Detectors properties:

- Repeatability: we want to find the same keypoints in different views of the scene despite the transformations undergone by the images.
- Saliency: work in informative regions, not in uniform regions

The Descriptor:

- Distinctiveness vs Robustness → the algorithm should capture the salient information around a keypoint, to keep important tokens and disregard changes due to noise
- Compactness → we detect thousands of pixels in an image and we need to match all of them. We want our description to be as compact as we can manage

The detector is applied to the whole image, so we need it to be the faster; Descriptors work on a subset.

Edges are not necessarily distinctive. They’re not discriminative enough. However, corners are indeed more discriminative, as they vary among all directions.

Moravec Interest Point Detector

$$C(\mathbf{p}) = \min_{\mathbf{q} \in n_s(\mathbf{p})} \|N(\mathbf{p}) - N(\mathbf{q})\|^2$$

The cornerness at $\mathbf{p}$ is given by the minimum squared difference between the patch centered at $\mathbf{p}$ and those centered at its eight neighbors.

In a uniform region, our C will be small, as I have no actual change in any direction. Same for edges, there won’t be change along the edge direction (will be higher than before). For corners, instead, we will observe a significant change!

Moravec Interest Point Detector is the first discrete approximation to compute cornerness

Harris Corner Detector

Defined mathematical framework descending from Moravec. If I take my image and I take a point, then do an infinitesimal shift, the error function will be computed as the difference between the original point and the infinitesimal shifted point, constrained to a specific region by a window. This way, I can exploit the Taylor Expansion and can approximate to the product of the partial derivatives and the infinitesimal shifts themselves.

$$E(\Delta x, \Delta y) = \sum_{x,y} w(x,y)(I(x+\Delta x, y+\Delta y) - I(x,y))^2$$

$$f(x+\Delta x) = f(x) + f'(x)\Delta x \rightarrow I(x+\Delta x, y+\Delta y) \cong\ I(x,y) + \frac{\delta I(x,y)}{\delta x}\Delta x + \frac{\delta I(x,y)}{\delta y}\Delta y$$

Harris was able to isolate the structure matrix, that tells us how much the pixels change. There's a score (we don't implement this).

M is called structure matrix. We can assume that we can diagonalize M. If we assume that M is a diagonal matrix, we can say that the final measure of cornerness is given by the value of the eigenvalues λ_1, λ_2. Both small (corner), one is a lot higher than the other (i see change in only one direction)-edge, else corner

Computation is a little costly. Approximation: as the eigenvalues are invariant, the final corner measure will be given by the determinant of M and the trace multiplied by a hyperparameter (original paper = 0.004).

$$C = \det(M) - k \cdot \mathrm{tr}(M)^2 = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

@March 27, 2023

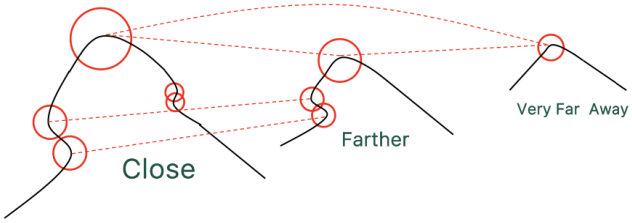
We want three invariances: Rotation, Illumination and Scale. In this case, by using eigenvalues I achieve Rotation Invariance

The overall framework from Harris does not stop to the computation of cornerness. It requires a thresholding step and after, we do NMS, because we still want, as for edges, a single pixel for corners instead of a region (slide cornerness response as a 3x3 and take maximum should be the one implemented by OpenCV). The last element is the weighting function, which follows a gaussian distribution, because of its properties (weight majorly closer pixels w.r.t those farther away)

Invariance Properties

Harris Cornerness function is invariant to rotation, due to the eigenvalues (eigenvalues of M are invariant to a rotation of the image axes). However no invariance to illumination. What we're doing is an Affine Illumination Change: $I' = aI + b$. We're computing differences, with both a multiplicative gain and an addictive bias. Are we invariant w.r.t. multiplicative gain? No, because we're changing the intensity of the difference, we can factorize it but it is still there. It amplifies values, meaning that some pixels that wouldn't normally pass the threshold can now pass it. Are we invariant w.r.t. addictive bias? Yeah, the bias removes itself out. Scale Invariance? No, but why it is so important to be scale invariant? We aim for scale invariance because we want to be able to perform in the wild. Observing objects at a different scale using the same detection window size yields a different outcome. It makes it impossible to repeatably detect homologous features when they appear at different scales in images.

Using one fixed window is not sufficient to deal with different scales. Large Scale Feature: I can tell it is a discriminative feature only by using a "big" window. So the size of the window influences the features we will observe. It is empirical. We detect features in order to match their descriptors.



Smoothing images to compute similar descriptors at different scales → the description loses details. Cancel out all the details that do not appear across the range of scales.

Instead of enlarging the window, use fixed-size window but increasingly downsample and smooth the image. This is the basis of Multi-Scale Feature Detection.

This representation is the Scale Space (scale is a third dimension). As you move along the scale dimension, small details should continuously disappear and no structure should be introduced. Gaussian filter can be demonstrated to be the only linear filter (we know) that does not introduce any artifacts when applied, thus achieving Scale Space.

A Scale-Space is a one-parameter family of images created from the original one so that the structures at smaller scales are successively suppressed by smoothing operation (which is performed using Gaussian Smoothing).

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

Scale space is 3D, with two coordinates (x, y) for the position and σ which is related to the scale (it is the sigma of the Gaussian).

The Gaussian Scale-Space is only a tool to represent the input image at different scales, as it neither includes any criterion to detect features, nor to select their characteristic scale. So how do we establish at which scale a feature turns out maximally interesting and should therefore be described? Research on Multi-Scale Feature Detection and Automatic Scale Selection goes in the direction of computing suitable combinations of scale-normalized derivatives of the Gaussian Scale-Space and find their extrema.

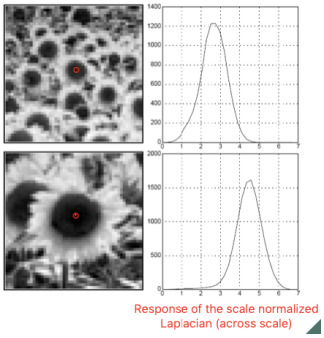
Scale-Normalized LOG (Laplacian of Gaussian)

Lindeberg → Multi-Scale Feature Detector that exploits the Scale Space Definition. Use of scale normalized derivatives of Gaussian (new detector is something already defined over the scale space). We talk of scale-normalized because the differences become weaker and weaker the more you increase the sigma (multiplies depth values by the value of the sigma to amplify the response of the gradient).

It is a second-derivative operator applied to the gaussian filter convolved with the image, multiplied by σ^2 to avoid the increasing weakness.

$$F(x, y, \sigma) = \sigma^2 \nabla^2 L(x, y, \sigma) = \sigma^2 (\nabla^2 G(x, y, \sigma) * I(x, y))$$

Sigma changes proportionally to the size of the region we're observing.



The ratios between the image and its scale are related

We no longer look for maximum values. We look for extrema (minimum and maximum depending on the contrast), because we can have different responses across scales.
Features (Blob-like) and scales detected as extrema of the scale-normalized LOG: we can have:

- Maxima = dark on light background
- Minima = light on dark background

Difference of Gaussian (DoG)

@April 3, 2023

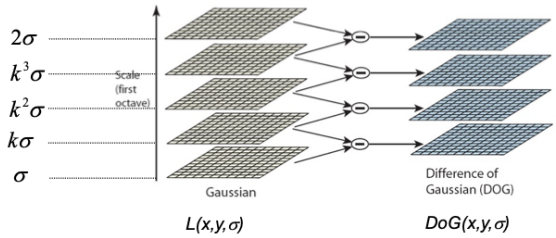
Lowe: change the Multi Scale Feature Detector with Difference of Gaussians instead of the Laplacian of Gaussian (he also introduced the SIFT, which is a descriptor). DoG exploits distributive property from convolution. The result is two smoothed images in the scale space. It obtains directly the difference.

$$DoG(x, y, z) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$

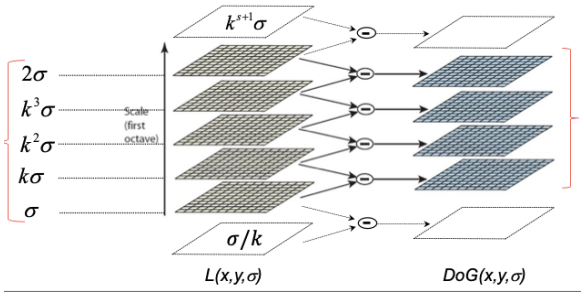
He shows that this is the same apart for a scale factor of applying the LoG. Extremes will not be influenced, so the DoG is similar anyway. It is a computationally efficient approximation.

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2 \nabla^2 G(x, y, \sigma)$$

Both Lindeberg and DoG are rotation invariant, because the filter itself is circularly symmetric, therefore rotation invariant (we still miss something). They also find blob-like features.



We have a stack of images with different σs (set of smoothed images = octave). We take differences of couples of nearby images at different scales, as they will contain features. From five levels of smoothing (e.g.) we obtain four levels of DoG, in which we then look for extrema. The octave contains all smoothed images from σ to $k\sigma$ (in our example, $k = 2$). We need a relation between the levels → $k = 2^{1/s}$ → in the upper bound, $s = 4$ (number of levels of DoG we obtain). In order to compute extrema on the DoG, we use NMS. What is the neighborhood here? Differently, we need to take into account also scale, so it will be along (x, y, σ); therefore, I need at least 3 levels (as it is now, I could compute NMS only for the middle levels, which is limiting). We add a couple layer more, so I can compute DoG all over the response we want to reach.

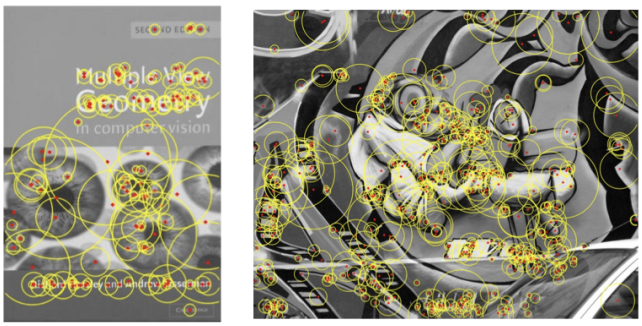


This addition allows me to compute the DoG on all levels.

What is the problem with increasing σ? The dimension of the kernel depends on k, which depends on σ. Increasing σ means enlarging the kernel, which means it quickly becomes unfeasible. We can deal with this by halving the dimension of the images (keep the same set of kernels, but removing some rows and columns). This way I still compute Larger Scale Features, but we don't explode computationally.

When we look for the maxima, we will look at other 8 at the same level, 9 up-scale and 9 down-scale = 26 values. The best setting that Lowe found out was 4 octaves $s = 3$, he doubles the input by replicating rows and columns (too many downsample would make the image a lot smaller). 5 levels of filtering for $s = 3$ and $\sigma = 1.6$ validated on datasets.

To obtain clearer results, we can threshold the DoG (prune it away). Plus, there are some edge-responses we do not really want (for this kind of applications, we don't like edges (difficult to match between different images)). Solution to clear also them up: another threshold.



The size of the circle is proportional to the scale of the feature (σ). My detector provides me also with the scale of the feature (the level at which we found the extrema).

Scale and Rotation Invariant Description

The detector provides us (x, y, σ). We need the rotation information though.

We have patches, they are the same, just rotated in different directions. Considered the direction of the gradient, they can look the same. So I have a sort of reference for the rotation. If I use the reference system of the image I cannot see the pattern. I need a new reference system (

x, y, σ, θ), with θ orientation reference of my keypoints. The canonical orientation of the keypoints can be exploited to find a local reference frame for each patch. How do we choose? The direction of maximum variation of my gradient? We need something more precise.

I'm centered on my keypoint. The canonical orientation can be obtained by computing the magnitude using the smoothed images (partial derivatives along the two directions) and the orientation

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$
$$\theta(x,y) = \tan^{-1}((L(x+1,y) - L(x-1,y))/(L(x,y+1) - L(x,y-1)))$$

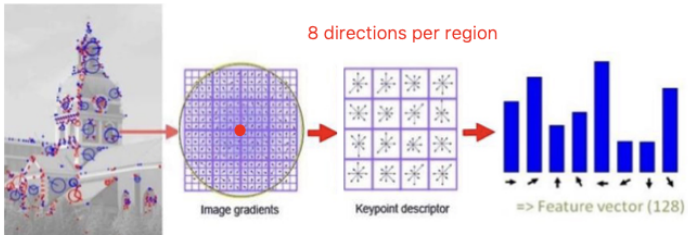
To obtain the canonical, Lowe proposed to construct a histogram (bin size of 10°) that contained the contributions of the pixels belonging to a neighborhood of the keypoint location (sort of counting orientations). We weight them using the magnitudes. Usually, you use a gaussian to weight the magnitudes. What comes out would be a nice histogram with different bins. You take the maximum.

Little problem: you can have multiple peaks. Lowe suggests that all those peaks higher than 80% of the maximum θ_n should be considered as canonical reference, as the difference might have been influenced by noise/illumination etc → We can have two canonical orientations (by keeping both I can have higher match probability).

Also, one could use a parabola fitting algorithm to obtain a θ value that takes into account also the neighborhood of the peak(s).

SIFT Descriptor

Scale Invariant Feature Transform (SIFT) → I take a patch, always of the same dimension, for each keypoint (16x16, oriented). I will compute the descriptor at the level in which the keypoint was found to be an extrema. This patch is further divided into 4x4 regions each of size 4x4. I will orient the patches according to the canonical orientations. I will need to interpolate levels. I use θ to rotate the window → everytime I rotate, I compute a meaningful value. We use 45° bins, therefore 8 directions per region. So, for each region, I have a 8 bin histogram. The SIFT will be composed of 16 histograms, each of 8 bins (rather coarse).



Again, there's a Gaussian, to weight the orientations closer to my key points

Why? Noise and complexity issues.

For both the computation of the histogram of the canonical orientation and the computation of the 8-bins histogram, he also used another weight (distance from the bin center).

We are scale/rotation invariant. We miss illumination? If you normalize with $L2$ norm, it will bring invariance to illumination changes → normalizing to a unit-hypersphere provides invariance to linear affine transformations. What they also found out was that if you have your descriptor and you clip the histogram to a threshold, and you normalize again after clipping, you get a bit of robustness (but not invariance) to non-linear intensity change.

So we have all the three invariances.

Matching Process

What I'm going to use for the matching process? We use the information from the keypoints to compute the descriptors, so I'll use the descriptors to compute Nearest Neighbor Search. Most simple: brute-force matching based on the distance.

@April 13, 2023

Now, I'm computing the matches between the descriptors, but the correspondences may not be correct. We need a form of match validation.

If I match my query descriptor with a set of reference descriptors, how can I choose if a match is good? Enforce a threshold to filter distances that are too high. We now have a 128-floating-values vector, therefore thresholding might be too difficult. Lowe proposed a ratio of distances (Lowe's ratio). Ratio between the two nearest neighbors.

$$\frac{d_{NN}}{d_{2-NN}} \leq T$$

We want it to be small, so we can discriminate better. On the numerator I have the closest point (small ap) and setting T=0.8 demonstrated that we can keep up to 95% of the good matches, while discarding almost 90% of the wrong matches. One of the ideas to validate matches which is used is this.

Up until now, we worked in Brute-Force Approaches. Slow. Use of indexing techniques to store matches in databases to speed up the process. Use tree-based structure (k-d tree = derived from binary tree structures, but multi-dimensional). Due to the increasing dimension of the descriptor, k-d tree doesn't scale well in this descriptor space, so we use an approximation, which is Best-Bin-First (BBF), which is efficient in high-dimensional spaces.

Alternative proposals

- SURF (Speeded-Up Robust Features) → blob-like features are detected through efficiently computable filters inspired by Lindberg's Gaussian derivatives. Scale-Space is achieved more efficiently by mean filtering
- MSER (Maximally Stable Extremal Regions) → detect regions of interest of arbitrary shapes, in particular approximately uniform areas either brighter or darker than their surroundings. Descriptions can be carried out using SIFT
- FAST (Features from Accelerated Segment Test) → efficient detector of corner-like features
- Binary descriptors to speed up matches (BRIEF, ORB and BRISK) → binary representation is very fast to compute using Hamming Distance.

Instance-Level Object Detection

@April 17, 2023

Given a reference image (called "model image") of a specific object, determine whether the object is present or not in the image under analysis (aka "target image") and, in case of detection, estimate the pose of the object. Depending on the application, the pose may often be given by a translation, a rototranslation or a similarity.

Faces, chairs, etc… are all classes, and you can acquire many examples of them at different dimensions/scales to train a NN. Instance-level OD deals with a specific object (not every possible one). You have at your disposal one image of the object (industrial setting), which is called the "model image". The image in which we're looking for one or multiple appearances is the target image. You can use the so-called "Finder Pattern" as a secondary component to align to and use as a reference.

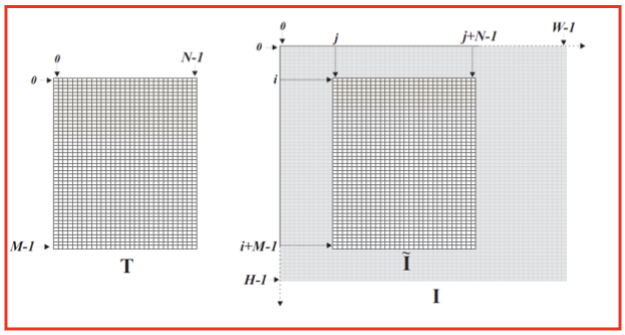
In general, the problem has a number of diverse facets. For the sake of simplicity, we refer to the basic setting: detecting a single object that may appear once in the target image.

Idea: is the object there? Where is the object? Rotation of the object? Scale of the object? The object can undergo similarity transformations (position, rotation and scale). We can encounter, even in an industrial setting, which we should have complete control on, nuisances like intensity changes, occlusions and clutter. At least for these applications, we expect limited variability (not in the wild). We need to be efficient (computational efficiency is a major requirement). So this family of algorithms - template matching - belongs in general to classical CV techniques: I have a model image and a target image and I slide the template over the image at each possible candidate position and at each candidate position I compute a similarity function (we deal with pixels). What can we change in this setting? The way we compute the (dis)similarity function.

On the opposite side, we have Category-Level Object Detection, which aims at detecting certain kinds of objects, regardless of their appearance and pose. Due to this high-variability, this problem is addressed by ML or DL techniques.

The model image is slid across the target image to be compared at each position to an equally sized window, by means of a suitable (dis)similarity function.

(Dis)Similarity Functions



We don’t usually do padding here because of the template matching (it’s not gonna be efficient). Also it’s not convenient to compute similarity when you don’t have enough pixels. Both $\tilde{I}(i,j)$, the window at position (i,j) of the target image, having the same size as T, as well as T can be thought of as a MN-dimensional vectors (flattened).

Pixel-wise distance between pixels at the same position in the Target and Sub-Image: SSD → I obtain a set of dissimilarities and I get the smallest. Should be an L2, but as we want to be the most efficient possible, we ditch the square root, as it doesn’t influence the results.

$$SSD(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(I(i+m,j+n) - T(m,n))^2$$

SAD (Sum of Absolute Differences) → I take the minimum.

$$SAD(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}|I(i+m,j+n) - T(m,n)|$$

Careful, doesn’t work for intensity changes. We want to deal with affine intensity changes. SAD and SSD are not invariant to linear transformations. If we have perfect control of the environment, they are perfect as they are efficient.

Changing the magnitude, though, doesn’t change the angle between the two vectors (target and sub-image), so we can use a measure of the cosine of the angle between the vector (the dot product). Normalized pixel-wise dot product (a.k.a Normalized Cross-Correlation). This time I have a measure of similarity. One of the most used in industry settings. But the bias changes the direction of the vector, so I’m not invariant to both components of the affine intensity changes (only to the linear). It is also slower!! (computationally more expensive)

$$NCC(i,j) = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m,j+n)\cdot T(m,n)}{\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m,j+n)^2}\cdot\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} T(m,n)^2}}$$

A way to deal with the bias term is introducing Zero Mean Cross Correlation. What’s the “damage”? We lose computation

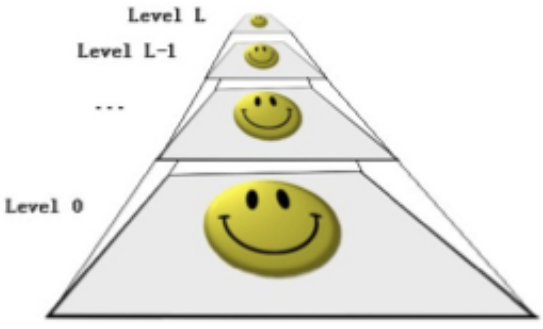
$$\mu(\tilde{I}) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m,j+n) \quad \mu(T) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} T(m,n)$$

The NNC is then computed after the subtraction of the means.

$$ZNCC(i,j) = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(I(i+m,j+n) - \mu(\tilde{I}))\cdot(T(m,n)-\mu(T))}{\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(I(i+m,j+n) - \mu(\tilde{I}))^2}\cdot\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(T(m,n)-\mu(T))^2}}$$

Fast Template Matching

Template Matching Complexity is given by the dimension of the template and the dimension of the whole image, so this means that it might be exceedingly slow whenever the model and/or target images have a large size. A way to reduce this proposed the concept of image pyramid to then start from an image which is smaller and smoother. I do the same level of shrinking and smoothing for the template and the target image - this is typically a 1/2 smoothing and sub-sampling on both sides at each level. Then I start a full search from the lowest level(small complexity) and I choose multiple points in which to look on the larger images (higher resolution).



Takes inspiration from the scale space. The development of Fast Template Matching is an active research topic

It is a very fast approach, though the number of levels needs to be chosen carefully, and empirically, to avoid loss of information, which would negatively affect the detection.

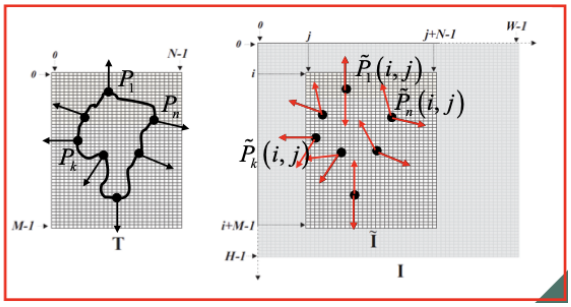
Shape-Based Matching

Focused on shape-matching. Edge-Based template matching approach. Industrial setting for robot guidance. We used SIFT in the lab 02, but we used with an image with a deeply recognizable and textured pattern. We can't detect corners and specific edges using SIFT. So we need another way, because we want to extract shape using edges. Exploit orientation of the control points.

We extract a set of control points P_k from the model image by an Edge Detector, and we store the gradient direction at each P_k. The template is composed by offsets and gradient directions. Then, at each position (i, j) of the target image, the recorded gradient directions associated with control points are compared to those at their corresponding image points $P_k(i, j)$.

The more the red arrows are aligned to the black arrows, the more similar the sub-image is to the template.

We do not perform edge detection on the target image, just on the template.



Normalized => unit vector

$$\mathbf{G}_k(P_k) = \begin{bmatrix} I_x(P_k) \\ I_y(P_k) \end{bmatrix}, \mathbf{u}_k(P_k) = \frac{1}{\|\mathbf{G}_k(P_k)\|} \begin{bmatrix} I_x(P_k) \\ I_y(P_k) \end{bmatrix}, k = 1..n \quad \text{Template}$$

$$\tilde{\mathbf{G}}_k(\tilde{P}_k) = \begin{bmatrix} I_x(\tilde{P}_k) \\ I_y(\tilde{P}_k) \end{bmatrix}, \tilde{\mathbf{u}}_k(\tilde{P}_k) = \frac{1}{\|\tilde{\mathbf{G}}_k(\tilde{P}_k)\|} \begin{bmatrix} I_x(\tilde{P}_k) \\ I_y(\tilde{P}_k) \end{bmatrix}, k = 1..n \quad \text{Target}$$

The similarity function spans the interval $[-1, 1]$. It takes the maximum value when all the gradients at the control points in the current window of the target image are perfectly aligned to those at the control points of the model image. If they are perfectly aligned the angle is 0, the cosine is 1 and the sum is n, which divided by n is 1, and vice versa

$$S(i,j) = \frac{1}{n} \sum_{k=1}^{n} \mathbf{u}_k(P_k) \cdot \tilde{\mathbf{u}}_k(\tilde{P}_k) = \frac{1}{n} \sum_{k=1}^{n} cos\theta_k$$

Choosing a detection threshold $S_{\min}$ can be thought of as specifying the fraction of model points which must be seen in the image to trigger a detection.

Certain application settings call for invariance to global inversion of contrast polarity along object's contours, as the object may appear either darker or brighter than the background in the target image. This kind of invariance can be achieved by a slight modification to the similarity function defined previously:

$$S(i,j) = \frac{1}{n} \left| \sum_{k=1}^{n} \mathbf{u}_k(P_k) \cdot \tilde{\mathbf{u}}_k(\tilde{P}_k) \right| = \frac{1}{n} \left| \sum_{k=1}^{n} cos\theta_k \right|$$

The following is even more robust due to the ability to withstand local contrast polarity inversions (local):

$$S(i,j) = \frac{1}{n} \sum_{k=1}^{n} \left| \mathbf{u}_k(P_k) \cdot \tilde{\mathbf{u}}_k(\tilde{P}_k) \right| = \frac{1}{n} \sum_{k=1}^{n} |cos\theta_k|$$

Hough Transform

The Hough Transform (HT) enables to detect objects having a known shape that can be expressed by an equation, based on projection of the input data into a suitable space referred to as parameter or Hough Space (which is different from the image space). This turns a global detection problem into a local one: we look for feature points into the parameter space instead of looking for the whole shape in the image space. This is usually applied after an edge detection process → the actual input data consist of the edge pixels extracted from the original image.

The Hough Transform is robust to noise and allows for detecting the sought shape even though it is partially occluded into the image (up to a certain user-selectable degree of occlusion).

The HT was invented to detect lines and was later extended to other analytical shapes (circle, ellipses), as well as to arbitrary shapes (Generalized Hough Transform - GHT). The GHT principle is widely deployed also within object detection pipelines relying on local invariant features such as SIFT.

Basic Principle

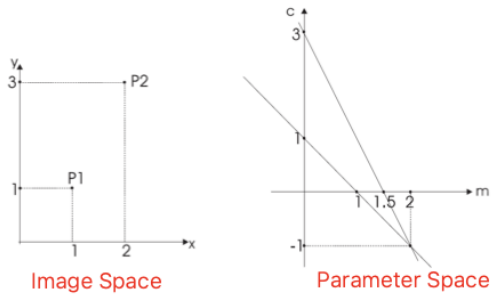
Consider the equation of a line: $y - mx - c = 0$. In the usual image space interpretation of the line equation, the parameters $(\hat{m}, \hat{c})$ are fixed → $y - \hat{m}x - \hat{c} = 0$ → so that the equation represents the mapping from the point $(\hat{m}, \hat{c})$ of the parameter space to the image points belonging to the line.

However, we may instead fix $(\hat{x}, \hat{y})$, which gives $\hat{y} - m\hat{x} - c = 0$: this equation represents the mapping from the image point $(\hat{x}, \hat{y})$ to the parameter space providing all the lines through the image point.

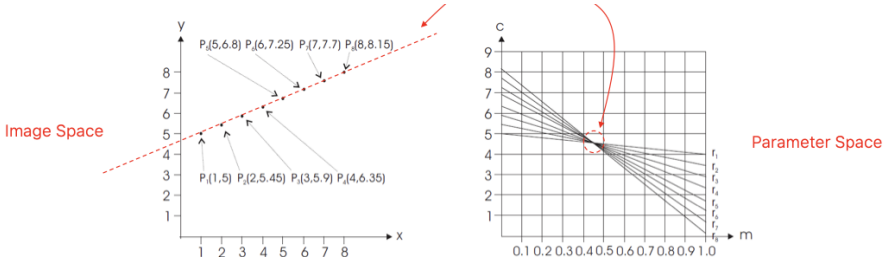
Now, consider two image points P_1, P_2 and map both into the parameter space, we get two lines intersecting at the parameter space point representing the image line through P_1, P_2.

$$\begin{cases} \hat{y}_1 - m\hat{x}_1 - c = 0 \\ \hat{y}_2 - m\hat{x}_2 - c = 0 \end{cases} \Longrightarrow \begin{cases} m = \frac{\hat{y}_2 - \hat{y}_1}{\hat{x}_2 - \hat{x}_1} \\ c = \frac{\hat{x}_2\hat{y}_1 - \hat{x}_1\hat{y}_2}{\hat{x}_2 - \hat{x}_1} \end{cases}$$

m and c represent the parameters of the line passing through P_1 and P_2. More generally, if we map n image points we get as many intersections as $n(n-1)/2$ (which is the number of lines through the n image points)



Considering n collinear image points, we can notice that their corresponding transforms (parameter space lines) will intersect at a single parameter space point representing the image line along which such n points lay. Rather than looking at an extended shape into the image, we look for a specific feature (where the lines intersect) in the parameter space of lines.



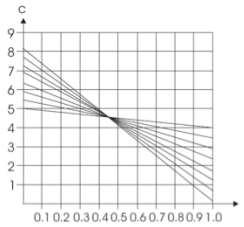
Therefore, given a sought analytic shape represented by a set of parameters, the HT consists in mapping image points (usually edges) so as to create curves into the parameter space of the shape. Intersections of parameter space curves indicate the presence of image points explained by a certain instance of the shape. The more the intersecting curves the more are such image points and thus, the higher the evidence of the presence of that instance in the image.
Object detection through the HT consists in finding the parameter space points through which many curves do intersect (local rather than global detection problem).

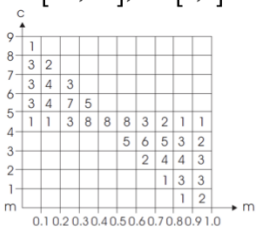
To make it work in practice, the parameter space needs to be quantized and allocated as a memory array, which is often referred to as Accumulator Array (AA). Curves are drawn into the AA by a “voting process”:

1. The transform equation is repeatedly computed to increment the bins satisfying the equation
2. A high number of intersecting curves at a point of the parameter space will provide a high number of votes into a bin of the AA
3. Finding parameter space points through which many curves do intersect is thus implemented in practice by finding peaks of the AA (i.e. local maxima showing a high number of votes).

The AA highlights the presence of a line with $m \in [0.3, 0.6],\ c \in [4, 5]$

Continuous Parameter Space





Quantized AA

To detect the line more accurately, the AA should be quantized more finely. The HT is robust to noise because spurious votes due to noise unlikely accumulate into a bin so as to trigger a false detection. A partially occluded object can be detected, provided that the threshold on the minimum number of votes required to declare a detection is lowered according to the degree of occlusion to be handled

The usual line parametrization we have considered so far is impractical (m spans an infinite range). So, in HT for lines we use the "normal parametrization" → $\rho = x \cos\vartheta + y \sin\vartheta$.
Image points $(\hat{x}, \hat{y})$ are mapped into sinusoidal curves of the (ϑ, ρ) parameter space: $\rho = \hat{x} \cos\vartheta + \hat{y} \sin\vartheta$.

With this parametrization, we have $\vartheta \in [-\pi/2, \pi/2]$ and $\rho \in [-\rho_{max}, \rho_{max}]$, where ρ_{max} is usually taken as large as the image diagonal $N\sqrt{2}$.