Image Processing and Computer Vision (Module 1)

Last update: 09 April 2024

Academic Year 2023 – 2024
Alma Mater Studiorum · University of Bologna

Contents

1 Image acquisition and formation

1.1 Pinhole camera

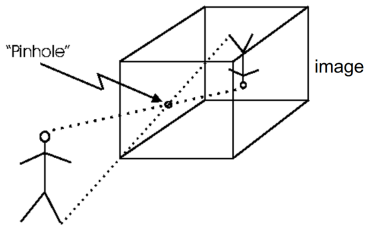
Imaging device Gathers the light reflected by 3D objects in a scene and creates a 2D representation of them.

Computer vision Infer knowledge of the 3D scene from 2D digital images.

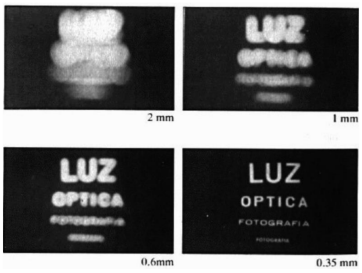
Pinhole camera Imaging device where the light passes through a small pinhole and hits the image plane. Geometrically, the image is obtained by drawing straight rays from the scene to the image plane passing through the pinhole.

> **Remark.** Larger aperture size of the pinhole results in blurry images (circle of confusion), while smaller aperture results in sharper images but requires longer exposure time (as less light passes through).

> **Remark.** The pinhole camera is a good approximation of the geometry of the image formation mechanism of modern imaging devices.



(a) Pinhole camera model



(b) Images with varying pinhole aperture size

1.2 Perspective projection

Geometric model of a pinhole camera.

Perspective projection

Scene point M (the object in the real world).

Image point m (the object in the image).

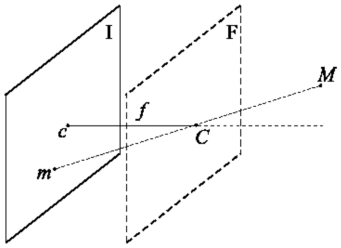
Image plane I.

Optical center C (the pinhole).

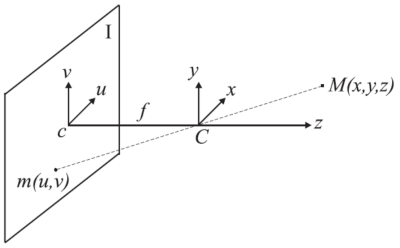
Image center/piercing point c (intersection between the optical axis – the line orthogonal to I passing through C – and I).

Focal length f.

Focal plane F.



- u and v are the horizontal and vertical axis of the image plane, respectively.
- x and y are the horizontal and vertical axis of the 3D reference system, respectively, and form the **camera reference system**.



Remark. For the perspective model, the coordinate systems (U, V) and (X, Y) must be parallel.

Scene–image mapping The equations to map scene points into image points are the following:

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

Proof. This is the consequence of the triangle similarity theorems.

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

The minus is needed as the axes are inverted

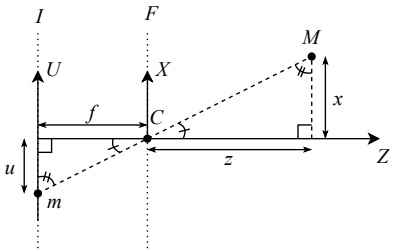


Figure 1.2: Visualization of the horizontal axis. The same holds on the vertical axis.

By inverting the axis horizontally and vertically (i.e. inverting the sign), the image plane can be adjusted to have the same orientation of the scene:

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

□

Remark. The image coordinates are a scaled version of the scene coordinates. The scaling is inversely proportioned with respect to the depth.

- The farther the point, the smaller the coordinates.
- The larger the focal length, the bigger the object is in the image.

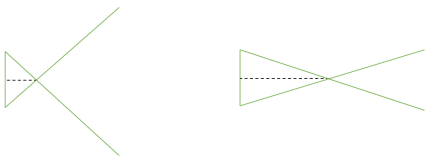


Figure 1.3: Coordinate space by varying focal length

Remark. The perspective projection mapping is not a bijection:

- A scene point is mapped into a unique image point.
- An image point is mapped onto a 3D line.

Therefore, reconstructing the 3D structure of a single image is an ill-posed problem

(i.e. it has multiple solutions).

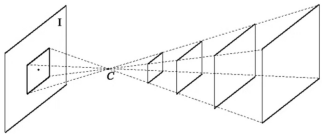


Figure 1.4: Projection from scene and image points

1.2.1 Stereo geometry

Stereo vision Use multiple images to triangulate the 3D position of an object.

Stereo correspondence Given a point L in an image, find the corresponding point R in another image.

Without any assumptions, an oracle is needed to determine the correspondences.

Standard stereo geometry Given two reference images, the following assumptions must hold:

- The X, Y, Z axes are parallel.
- The cameras that took the two images have the same focal length f (coplanar image planes) and the images have been taken at the same time.
- There is a horizontal translation b between the two cameras (baseline).
- The disparity d is the difference of the U coordinates of the object in the left and right image.

Theorem 1.2.1 (Fundamental relationship in stereo vision)**.** If the assumptions above hold, the following equation holds:

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

Proof. Let $P_L = \begin{pmatrix} x_L & y & z \end{pmatrix}$ and $P_R = \begin{pmatrix} x_R & y & z \end{pmatrix}$ be the coordinates of the object P with respect to the left and right camera reference system, respectively. Let $p_L = \begin{pmatrix} u_L & v \end{pmatrix}$ and $p_R = \begin{pmatrix} u_R & v \end{pmatrix}$ be the coordinates of the object P in the left and right image plane, respectively.

By assumption, we have that $P_L - P_R = \begin{pmatrix} b & 0 & 0 \end{pmatrix}$, where b is the baseline.

By the perspective projection equation, we have that:

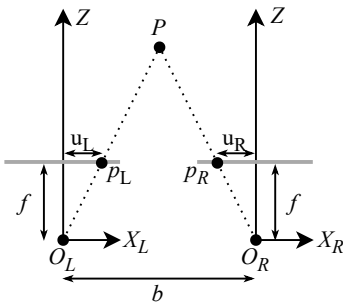
$$u_L = x_L\frac{f}{z} \qquad u_R = x_R\frac{f}{z}$$

Disparity is computed as follows:

$$d = u_L - u_R = x_L\frac{f}{z} - x_R\frac{f}{z} = b\frac{f}{z}$$

We can therefore obtain the Z coordinate of P as:

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



Note: the Y/V axes are not in figure.

□

Remark. Disparity and depth are inversely proportional: the disparity of two points decreases if the points are farther in depth.

Stereo matching If the assumptions for standard stereo geometry hold, to find the object corresponding to p_L in another image, it is sufficient to search along the horizontal axis of p_L looking for the same colors or patterns.

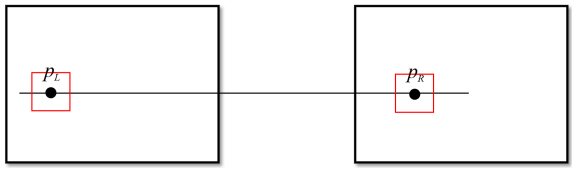


Figure 1.5: Example of stereo matching

Epipolar geometry Approach applied when the two cameras are no longer aligned according to the standard stereo geometry assumption. Still, the focal lengths and the roto-translation between the two cameras must be known.

Given two images, we can project the epipolar line related to the point p_L in the left plane onto the right plane to reduce the problem of correspondence search to a single dimension.

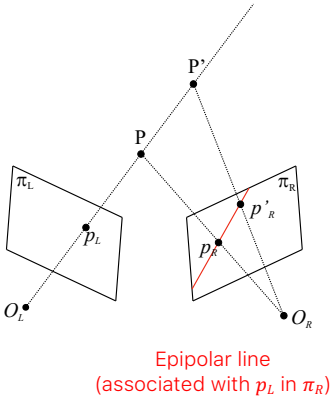
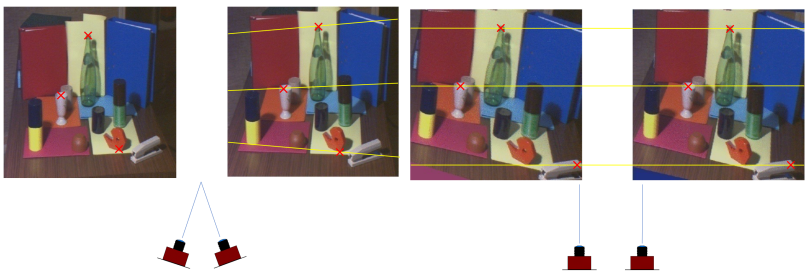


Figure 1.6: Example of epipolar geometry

Remark. It is nearly impossible to project horizontal epipolar lines and searching through oblique lines is awkward and computationally less efficient than straight lines.

Rectification Transformation applied to convert epipolar geometry to a standard stereo geometry.



(a) Images before rectification (b) Images after rectification

1.2.2 Ratios and parallelism

Given a 3D line of length L lying in a plane parallel to the image plane at distance z, its length l in the image plane is:

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

In all the other cases (i.e. when the line is not parallel to the image plane), the ratios of lengths and the parallelism of lines are not preserved.

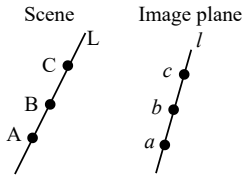


Figure 1.8: Example of not preserved ratios. It holds that $\frac{\overline{AB}}{\overline{BC}} \neq \frac{\overline{ab}}{\overline{bc}}$.

Vanishing point Intersection point of lines that are parallel in the scene but not in the image plane.

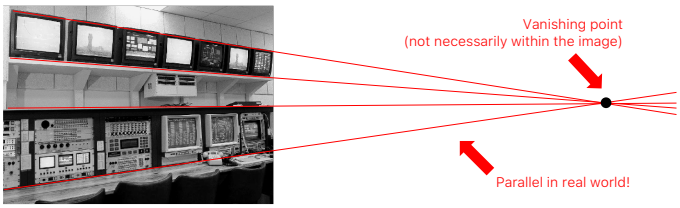


Figure 1.9: Example of vanishing point

1.3 Lens

Depth of field (DOF) Distance at which a scene point is in focus (i.e. when all its light rays gathered by the imaging device hit the image plane at the same point).

> **Remark.** Because of the small size of the aperture, a pinhole camera has infinite depth of field but requires a long exposure time making it only suitable for static scenes.

Lens A lens gathers more light from the scene point and focuses it on a single image point. Lens

This allows for a smaller exposure time but limits the depth of field (i.e. only a limited range of distances in the image can be in focus at the same time).

Thin lens Approximate model for lenses. Thin

Scene point P (the object in the real world).

Image point p (the object in the image).

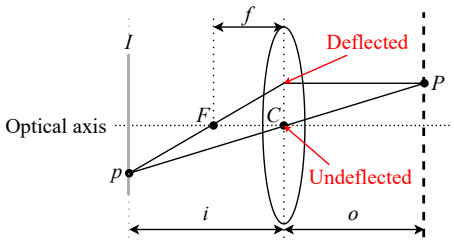
Object–lens distance o.

Image–lens distance i (i.e. focal length of the camera).

Center of the lens C.

Focal length of the lens f.

Focal plane/focus of the lens F.



A thin lens has the following properties:

- Rays hitting the lens parallel to the optical axis are deflected to pass through the focal plane of the lens F.
- Rays passing through the center of the lens C are undeflected.
- The following equation holds:

$$\frac{1}{o} + \frac{1}{i} = \frac{1}{f}$$

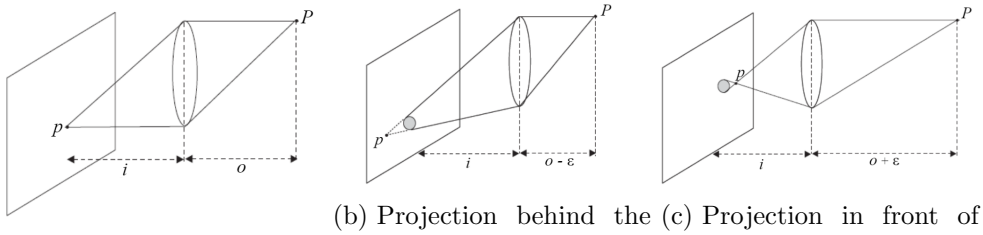
Image formation When the image is in focus, the image formation process follows the normal rules of the perspective projection model where:

- C is the optical center.
- i is the focal length of the camera.

By fixing the focal length of the lens (f), we can determine the distance of the scene point (o) or the image point (i) required to have the object in focus.

$$\frac{1}{o} + \frac{1}{i} = \frac{1}{f} \iff o = \frac{if}{i - f} \qquad \frac{1}{o} + \frac{1}{i} = \frac{1}{f} \iff i = \frac{of}{o - f}$$

Remark. Points projected in front or behind the image plane will create a circle of confusion (blur).



(a) Image in focus (b) Projection behind the image plane (c) Projection in front of the image plane

Adjustable diaphragm Device to control the light gathered by the effective aperture of the lens.

Reducing the aperture will result in less light and an increased depth of field.

Remark. On a theoretical level, images that are not in focus appear blurred (circles of confusion). Despite that, if the circle is smaller than the photo-sensing elements (i.e. pixels), it will appear in focus.

Focusing mechanism Allows the lens to translate along the optical axis to increase its distance to the image plane.

At the minimum extension (**??**), we have that:

$$i = f \text{ and } o = \infty \text{ as the thin lens equation states that } \frac{1}{o} + \frac{1}{i} = \frac{1}{f}$$

By increasing the extension (i.e. increase i), we have that the distance to the scene point o decreases. The maximum extension determines the minimum focusing distance.

(a) Minimum extension (b) Maximum extension

Figure 1.11: Extension of a focusing mechanism

1.4 Image digitalization

1.4.1 Sampling and quantization

The image plane of a camera converts the received irradiance into electrical signals.

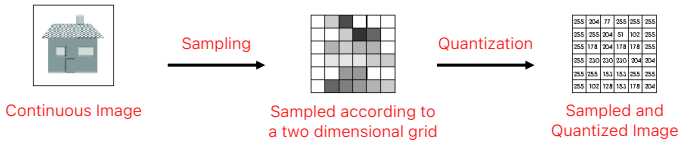


Figure 1.12: Image digitalization steps

Sampling The continuous electrical signal is sampled to produce a $N \times M$ matrix of pixels:

$$I(x,y) = \begin{pmatrix} I(0,0) & \dots & I(0,M-1) \\ \vdots & \ddots & \vdots \\ I(N-1,0) & \dots & I(N-1,M-1) \end{pmatrix}$$

Quantization Let m be the number of bits used to encode a pixel. The value of each pixel is quantized into 2^m discrete gray levels.

> **Remark.** A grayscale image usually uses 8 bits
> An RGB image usually uses $3 \cdot 8$ bits.

Remark. The more bits are used for the representation, the higher the quality of the image will be.

- Sampling with fewer bits will result in a lower resolution (aliasing).
- Quantization with fewer bits will result in less representable colors.

Figure 1.13: Sampling and quantization using fewer bits

1.4.2 Camera sensors

Photodetector Sensor that, during the exposure time, converts the light into a proportional electrical charge that will be processed by a circuit and converted into a digital or analog signal.

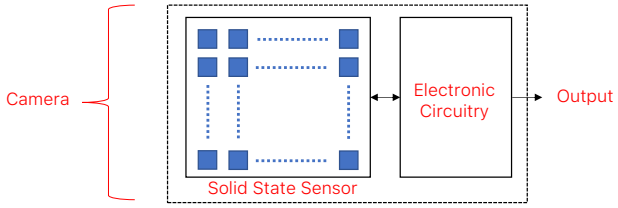


Figure 1.14: Components of a camera

The two main sensor technologies are:

Charge Coupled Device (CCD) Typically produces higher quality images but are more expensive.

Complementary Metal Oxide Semiconductor (CMOS) Generally produces lower quality images but is more compact and less expensive. Each sensor has integrated its own circuitry that allows to read an arbitrary window of the sensors.

Color sensors CCD and CMOS sensors are sensitive to a wide spectrum of light frequencies (both visible and invisible) but are unable to sense colors as they produce a single value per pixel.

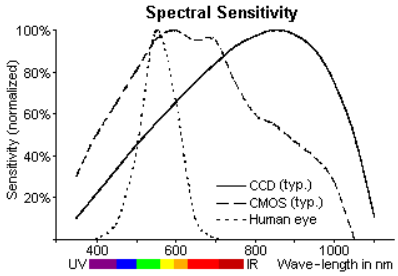


Figure 1.15: CCD and CMOS spectral sensitivity

Color Filter Array (CFA) Filter placed in front of a photodetector to allow it to detect colors.

Possible approaches are:

Bayer CFA A grid of green, blue, and red filters with the greens being twice as much as the others (the human eye is more sensible to the green range). To determine the RGB value of each pixel, missing color channels are sampled from neighboring pixels (demosaicking).

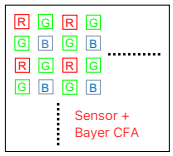


Figure 1.16: Example of Bayer filter

Optical prism A prism splits the incoming light into 3 RGB beams, each directed to a different sensor. It is more expensive than Bayer CFA.

1.4.3 Metrics

Signal to Noise Ratio (SNR) Quantifies the strength of the actual signal with respect to unwanted noise.

Sources of noise are:

Photon shot noise Number of photons captured during exposure time.

Elecronic circuitry noise Generated by the electronics that read the sensors.

Quantization noise Caused by the digitalization of the image (ADC conversion).

Dark current noise Random charge caused by thermal excitement.

SNR is usually expressed in decibels or bits:

$$\texttt{SNR}_{\text{db}} = 20 \cdot \log_{10}(\texttt{SNR}) \qquad \texttt{SNR}_{\text{bit}} = \log_2(\texttt{SNR})$$

Dynamic Range (DR) Measures the ability of a sensor to capture both the dark and bright structure of the scene.

Let:

- E_{min} be the minimum detectable irradiation. This value depends on the noise.
- E_{max} be the saturation irradiation (i.e. the maximum amount of light that fills the capacity of the photodetector).

DR is defined as:

$$\texttt{DR} = \frac{E_{\text{max}}}{E_{\text{mix}}}$$

As with SNR, DR can be expressed in decibels or bits.

2 Spatial filtering

2.1 Noise

The noise added to a pixel p is defined by $n_k(p)$, where k indicates the time step (i.e. noise changes depending on the moment the image is taken). It is assumed that $n_k(p)$ is i.i.d. and $n_k(p) \sim \mathcal{N}(0, \sigma)$.
The information of a pixel p is therefore defined as:

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

where $\tilde{I}(p)$ is the real information.

Temporal mean denoising Averaging N images taken at different time steps.

$$\begin{aligned} O(p) &= \frac{1}{N} \sum_{k=1}^{N} I_k(p) \\ &= \frac{1}{N} \sum_{k=1}^{N} \Big(\tilde{I}(p) + n_k(p) \Big) \\ &= \frac{1}{N} \sum_{k=1}^{N} \tilde{I}(p) + \overbrace{\frac{1}{N} \sum_{k=1}^{N} n_k(p)}^{\mu = 0} \\ &\approx \tilde{I}(p) \end{aligned}$$

> **Remark.** As multiple images of the same object are required, this method is only suited for static images.

Spatial mean denoising Given an image, average across neighboring pixels.

Let K_p be the pixels in a window around p (included):

$$\begin{aligned} O(p) &= \frac{1}{|K_p|} \sum_{q \in K_p} I(q) \\ &= \frac{1}{|K_p|} \sum_{q \in K_p} \Big(\tilde{I}(q) + n(q) \Big) \\ &= \frac{1}{|K_p|} \sum_{q \in K_p} \tilde{I}(q) + \frac{1}{|K_p|} \sum_{q \in K_p} n(q) \\ &\approx \frac{1}{|K_p|} \sum_{q \in K_p} \tilde{I}(q) \end{aligned}$$

> **Remark.** As the average of neighboring pixels is considered, this method is only suited for uniform regions.

2.2 Convolutions

2.2.1 Preliminaries

Convolution Given two functions f and g, their 1D convolution is defined as [**wiki:1d˙convolution**]:

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)\,\mathrm{d}\tau$$

In other words, at each t, a convolution can be interpreted as the area under $f(\tau)$ weighted by $g(t-\tau)$ (i.e. $g(\tau)$ flipped w.r.t. the y-axis and with the argument shifted by t).

Alternatively, it can be seen as the amount of overlap between $f(\tau)$ and $g(t-\tau)$.

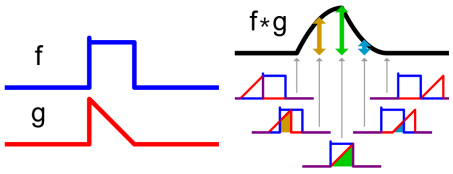


Figure 2.1: Example of convolution

Extended to the 2-dimensional case, the definition becomes:

$$(f * g)(x, y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(\alpha, \beta)g(x-\alpha, y-\beta)\,\mathrm{d}\alpha\,\mathrm{d}\beta$$

A convolution enjoys the following properties:

Associative $f * (g * h) = (f * g) * h$.

Commutative $f * g = g * f$.

Distributive w.r.t. sum $f * (g + h) = f * g + f * h$.

Commutative with differentiation $(f * g)' = f' * g = f * g'$

Dirac delta The Dirac delta "function" δ is defined as follows [**wiki:dirac**, **book:sonka**]:

$$\forall x \neq 0 : \delta(x) = 0, \text{ constrained to } \int_{-\infty}^{+\infty} \delta(x)\,\mathrm{d}x = 1$$

Extended to the 2-dimensional case, the definition is the following:

$$\forall (x, y) \neq (0, 0) : \delta(x, y) = 0, \text{ constrained to } \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \delta(x, y)\,\mathrm{d}x\,\mathrm{d}y = 1$$

Sifting property The following property holds:

$$\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x, y)\delta(\alpha - x, \beta - y)\,\mathrm{d}x\,\mathrm{d}y = f(\alpha, \beta)$$

> **Remark.** Exploiting the sifting property, the signal of an image can be expressed through an integral of Dirac deltas (i.e. a linear combination)

[**slides:filters**, **book:sonka**]:

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

Kronecker delta Discrete version of the Dirac delta [**wiki:kronecker**]:

$$\delta(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ 1 & \text{if } x = 0 \end{cases}$$

Extended to the 2-dimensional case, the definition is the following:

$$\delta(x,y) = \begin{cases} 0 & \text{if } (x,y) \neq (0,0) \\ 1 & \text{if } (x,y) = (0,0) \end{cases}$$

Sifting property The following property holds:

$$i(x,y) = \sum_{\alpha=-\infty}^{+\infty} \sum_{\beta=-\infty}^{+\infty} i(\alpha,\beta)\delta(x-\alpha, y-\beta)$$

2.2.2 Continuous convolutions

Image filter Operator that computes the new intensity of a pixel p based on the intensities of a neighborhood of p.

Remark. Image filters are useful for denoising and sharpening operations.

Linear translation-equivariant (LTE) operator A 2D operator $T\{\cdot\}$ is denoted as:

$$T\{i(x,y)\} = o(x,y)$$

$T\{i(x,y)\}$ is LTE iff it is:

Linear Given two input 2D signals $i(x,y)$, $j(x,y)$ and two constants α, β, it holds that:

$$T\{\alpha \cdot i(x,y) + \beta \cdot j(x,y)\} = \alpha T\{i(x,y)\} + \beta T\{j(x,y)\}$$

Translation-equivariant Given an input 2D signal $i(x,y)$ and two offsets x_o, y_o, it holds that:

$$\text{if } T\{i(x,y)\} = o(x,y) \text{ then } T\{i(x-x_o, y-y_o)\} = o(x-x_o, y-y_o)$$

Impulse response/Point spread function/Kernel Given a 2D operator $T\{\cdot\}$, its impulse response, denoted with h, is the output of the operator when the input signal is a Dirac delta [**slides:filters**]:

$$h(x,y) \triangleq T\{\delta(x,y)\}$$

Theorem 2.2.1 (LTE operators as convolutions)**.** Applying an LTE operator on an image is equivalent to computing the convolution between the image and the impulse response h of the operator.

$$\begin{aligned} T\{i(x,y)\} &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} i(\alpha,\beta)h(x-\alpha, y-\beta)\,\mathrm{d}\alpha\,\mathrm{d}\beta \\ &= i(x,y) * h(x,y) \end{aligned}$$

In other words, the impulse response allows to compute the output of any input signal through a convolution.

Proof. Let $i(x, y)$ be an input signal and $T\{\cdot\}$ be a 2D operator. We have that:

$$\begin{aligned}
T\{i(x,y)\} &= T\left\{\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(\alpha,\beta)\delta(x-\alpha,y-\beta)\,\mathrm{d}\alpha\,\mathrm{d}\beta\right\} && \text{sifting property}\\
&= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} T\left\{i(\alpha,\beta)\delta(x-\alpha,y-\beta)\right\}\,\mathrm{d}\alpha\,\mathrm{d}\beta && \text{linearity of } T\{\cdot\}\\
&= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(\alpha,\beta)T\left\{\delta(x-\alpha,y-\beta)\right\}\,\mathrm{d}\alpha\,\mathrm{d}\beta && \text{linearity of } T\{\cdot\}\\
&= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(\alpha,\beta)h(x-\alpha,y-\beta)\,\mathrm{d}\alpha\,\mathrm{d}\beta && \text{impulse response}\\
&= i(x,y) * h(x,y) && \text{definition of convolution}
\end{aligned}$$

□

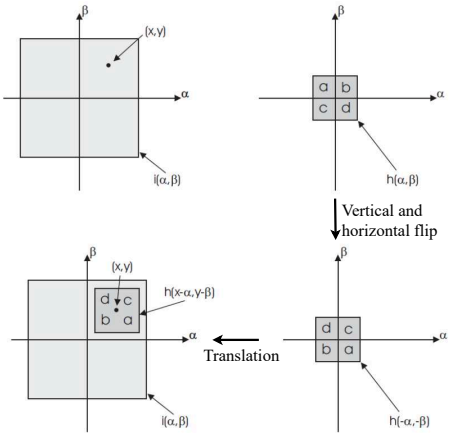


Figure 2.2: Visualization of a convolution

Cross-correlation Given two signals $i(x, y)$ and $h(x, y)$, their cross-correlation computes their similarity and is defined as follows [**wiki:crosscorrelation**]:

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

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

Remark. Cross-correlation is not commutative.

Remark. The cross-correlation $h \circ i$ is similar to a convolution without flipping the kernel.
If h is an even function (i.e. $h(x, y) = h(-x, -y)$), we have that $h \circ i$ has the same result of a convolution:

$$\begin{aligned}
h(x,y) * i(x,y) &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(\alpha,\beta)h(x-\alpha,y-\beta)\,\mathrm{d}\alpha\,\mathrm{d}\beta\\
&= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(\alpha,\beta)h(\alpha-x,\beta-y)\,\mathrm{d}\alpha\,\mathrm{d}\beta\\
&= h(x,y)\circ i(x,y)
\end{aligned}$$

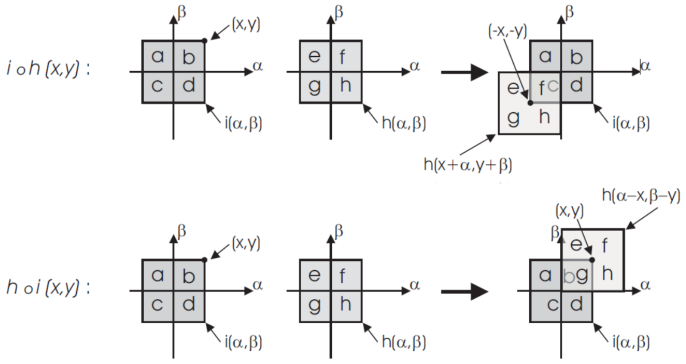


Figure 2.3: Visualization of cross-correlation

2.2.3 Discrete convolutions

Discrete convolution Given an input 2D signal $I(i,j)$ and the kernel $H(i,j) = T\{\delta(i,j)\}$ of a discrete LTE operator (where $\delta(i,j)$ is the Kronecker delta), a discrete convolution is defined as:

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

In practice, the kernel is finitely defined and is applied to each pixel of the image:

$$T\{I(i,j)\} = \sum_{m=-k}^{k} \sum_{n=-k}^{k} K(m,n)I(i-m,j-n) = O(i,j)$$

Example. For simplicity, a kernel of size 3 is considered. Given an image I and a kernel K, the output $O(1,1)$ of the pixel $(1,1)$ is computed as:

$$\begin{aligned} O(1,1) &= \begin{pmatrix} I(0,0) & I(0,1) & I(0,2) \\ I(1,0) & I(1,1) & I(1,2) \\ I(2,0) & I(2,1) & I(2,2) \end{pmatrix} * \begin{pmatrix} K(0,0) & K(0,1) & K(0,2) \\ K(1,0) & K(1,1) & K(1,2) \\ K(2,0) & K(2,1) & K(2,2) \end{pmatrix} \\ &= I(0,0)K(2,2) + I(0,1)K(2,1) + I(0,2)K(2,0) + \\ &\quad + I(1,0)K(1,2) + I(1,1)K(1,1) + I(1,2)K(1,0) + \\ &\quad + I(2,0)K(0,2) + I(2,1)K(0,1) + I(2,2)K(0,0) \end{aligned}$$

Note that by definition, K has to be flipped.

Remark. In convolutional neural networks, the flip of the learned kernels can be considered implicit.

Border handling Computing the convolution of the pixels at the borders of the image might be an issue as it goes out-of-bounds, possible solutions are:

Crop Ignore border pixels on which the convolution overflows.

Pad Add a padding to the image:

Zero-padding Add zeros (e.g. `000`$|a \dots d|$`000`).

Replicate Repeat the bordering pixel (e.g. `aaa`$|a \dots d|$`ddd`).

Reflect Use the n pixels closest to the border (e.g. `cba`$|abc \dots dfg|$`gfd`).

Reflect_101 Use the n pixels closest to the border, skipping the first/last one (e.g. `dcb`$|abcd \dots efgh|$`gfe`).

2.2.4 Common linear kernels

Mean filter LTE operator that computes the intensity of a pixel as the average intensity of the pixels in its neighborhood.

The kernel has the form (example with a 3×3 kernel):

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix} = \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Remark. The mean filter has a low-pass effect which allows the removal of details from the signal. This allows for image smoothing and, to some extent, denoising (but adds blur).

Remark. As the intensity of a pixel is computed by averaging its neighborhood, the results for pixels located between low-intensity and high-intensity areas might not be ideal.

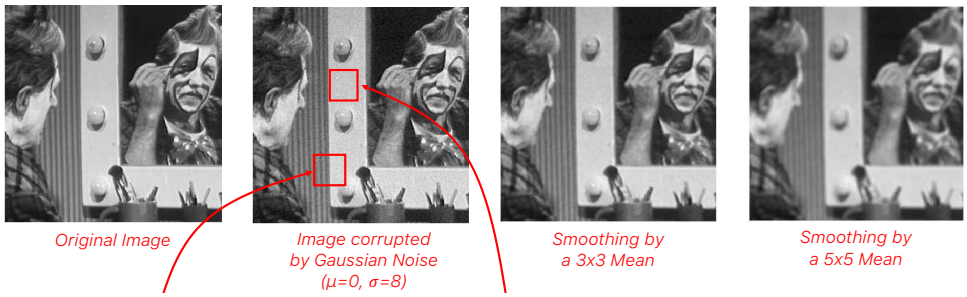


Figure 2.4: Example of mean filter application

Gaussian filter LTE operator whose kernel follows a 2D Gaussian distribution with $\mu = 0$ and given σ.

Remark. The smoothing strength of the filter grows with σ.

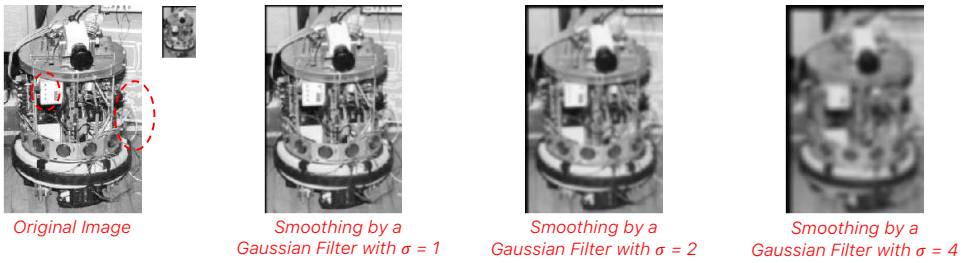


Figure 2.5: Example of Gaussian filter application

Sampling In practice, the kernel is created by sampling from the wanted Gaussian distribution. One can notice that a higher σ results in a more spread distribution and therefore a larger kernel is more suited. On the other hand, a smaller σ can be represented using a smaller kernel as it is more concentrated around the origin.

As a rule-of-thumb, given σ, an ideal kernel is of size $(2\lceil 3\sigma \rceil + 1) \times (2\lceil 3\sigma \rceil + 1)$.

Separability As a 2D Gaussian $G(x, y)$ can be decomposed into a product of two 1D Gaussians $G(x, y) = G_1(x)G_2(y)$, it is possible to split the convolution into two 1D convolutions.

$$\begin{aligned} I(x,y) * G(x,y) &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} I(\alpha,\beta) G(x-\alpha, y-\beta)\, \mathrm{d}\alpha\, \mathrm{d}\beta \\ &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} I(\alpha,\beta) G_1(x-\alpha) G_2(y-\beta)\, \mathrm{d}\alpha\, \mathrm{d}\beta \\ &= \int_{-\infty}^{+\infty} G_2(y-\beta) \left(\int_{-\infty}^{+\infty} I(\alpha,\beta) G_1(x-\alpha)\, \mathrm{d}\alpha \right) \mathrm{d}\beta \\ &= (I(x,y) * G_1(x)) * G_2(y) \end{aligned}$$

Remark. The speed-up in number-of-operations is linear.

2.2.5 Common non-linear kernels

Remark. Linear filters are ineffective when dealing with impulse noise and have the side effect of blurring the image.

Figure 2.6: Example of impulse noise and denoising with mean filter

Remark. As they lose linearity, non-linear filters are technically not convolutions anymore.

Median filter The intensity of a pixel is obtained as the median intensity of its neighborhood.

Remark. Median filters are effective in removing impulse noise (as outliers are excluded) without introducing significant blur. It also tends to result in sharper edges.

Remark. Median filters are not suited for Gaussian noise. It might be useful to apply a linear filter after a median filter.

Figure 2.7: Example of median filter application

Bilateral filter Given two pixels p and q, the following can be computed:

Spatial distance $d_s(p,q) = \|p - q\|_2$

Range/intensity distance $d_r(p,q) = |\texttt{intensity}(p) - \texttt{intensity}(q)|$

Given a pixel p, its neighborhood $\mathcal{N}(p)$ and the variances σ_s, σ_r of two Gaussians, the bilateral filter applied on p is computed as follows:

$$O(p) = \sum_{q \in \mathcal{N}(p)} H(p,q) \cdot \texttt{intensity}(q)$$

$$\text{where } H(p,q) = \frac{G_{\sigma_s}(d_s(p,q)) G_{\sigma_r}(d_r(p,q))}{\sum_{z \in \mathcal{N}(p)} G_{\sigma_s}(d_s(p,z)) G_{\sigma_r}(d_r(p,z))}$$

where the denominator of H is a normalization factor.

Remark. Bilateral filters allow to deal with Gaussian noise without the introduction of blur.

Remark. Neighboring pixels with similar intensities result in larger weights in the filter, while pixels with different intensities (i.e. near an edge) result in smaller weights. This allows to effectively ignore pixels that belong to a different object from being considered when computing the intensity of a pixel.

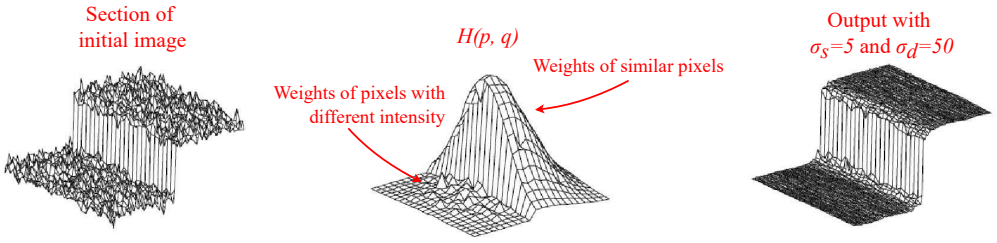
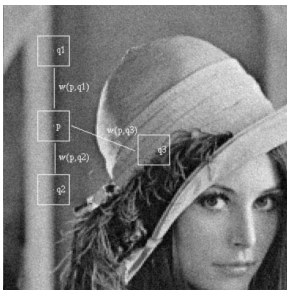


Figure 2.8: Example of bilateral filter application

Non-local means filter Exploits patches with similar pixels to denoise the image.

Let:

- I be the image plane.
- $\mathcal{N}_i$ be the intensity matrix of a patch centered on the pixel i.
- S_i be a neighborhood of the pixel i.
- h be the bandwidth (a hyperparameter).



The intensity of a pixel p is computed as follows:

$$O(p) = \sum_{q \in S_p} w(p,q) \cdot \texttt{intensity}(q)$$

$$\text{where } w(p,q) = \frac{1}{Z(p)} e^{-\frac{\|\mathcal{N}_p - \mathcal{N}_q\|_2^2}{h^2}}$$

$$Z(p) = \sum_{v \in I} e^{\frac{\|\mathcal{N}_p - \mathcal{N}_v\|_2^2}{h^2}}$$

Instead of using the full image plane I, a neighborhood S_p is used for computational purposes. $Z(p)$ is a normalization factor.

3 Edge detection

Edge Pixel lying in between regions of the image with different intensities. Edge

3.1 Gradient thresholding

3.1.1 1D step-edge

In the transition region, the absolute value of the first derivative grows (the absolute value is used as the polarity is not relevant). By fixing a threshold, an edge can be detected. 1D step-edge

(a) Input signal (b) Derivative of the signal

3.1.2 2D step-edge

In a 2D signal (e.g. an image), the gradient allows to determine the magnitude and the direction of the edge. 2D step-edge

$$\nabla I(x,y) = \begin{pmatrix} \frac{\partial I(x,y)}{\partial x} & \frac{\partial I(x,y)}{\partial y} \end{pmatrix} = \begin{pmatrix} \partial_x I & \partial_y I \end{pmatrix}$$

$$\text{Magnitude: } \|\nabla I(x,y)\|$$

$$\text{Direction: } \arctan\left(\frac{\partial_y I}{\partial_x I}\right) \in [-\frac{\pi}{2}, \frac{\pi}{2}]$$

$$\text{Direction and sign: } \arctan 2(\partial_x I, \partial_y I) \in [0, 2\pi]$$

Discrete gradient approximation Approximation of the partial derivatives as a difference.

Backward difference

$$\partial_x I(i,j) \approx I(i,j) - I(i,j-1) \qquad \partial_y I(i,j) \approx I(i,j) - I(i-1,j)$$

Forward difference

$$\partial_x I(i,j) \approx I(i,j+1) - I(i,j) \qquad \partial_y I(i,j) \approx I(i+1,j) - I(i,j)$$

> **Remark.** Forward and backward differences are equivalent to applying two cross-correlations with kernels $\begin{pmatrix} -1 & 1 \end{pmatrix}$ and $\begin{pmatrix} -1 \\ 1 \end{pmatrix}$.

Central difference

$$\partial_x I(i,j) \approx I(i,j+1) - I(i,j-1) \qquad \partial_y I(i,j) \approx I(i+1,j) - I(i-1,j)$$

Remark. Central difference is equivalent to applying two cross-correlations with kernels $\begin{pmatrix}-1 & 0 & 1\end{pmatrix}$ and $\begin{pmatrix}-1\\0\\1\end{pmatrix}$.

Discrete magnitude approximation The gradient magnitude can be approximated in different ways by using the approximated partial derivatives:

$$\|\nabla I\| = \sqrt{(\partial_x I)^2 + (\partial_y I)^2} \qquad \|\nabla I\|_+ = |\partial_x I| + |\partial_y I| \qquad \|\nabla I\|_{\max} = \max(|\partial_x I|, |\partial_y I|)$$

Among all, $\|\nabla I\|_{\max}$ is the most isotropic (i.e. gives a more consistent response).

Example. Given the following images:

$$E_v = \begin{pmatrix}0 & 0 & h & h\\0 & 0 & h & h\\0 & 0 & h & h\\0 & 0 & h & h\end{pmatrix} \quad E_h = \begin{pmatrix}0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\h & h & h & h\\h & h & h & h\end{pmatrix} \quad E_d = \begin{pmatrix}0 & 0 & 0 & 0 & h\\0 & 0 & 0 & h & h\\0 & 0 & h & h & h\\0 & h & h & h & h\end{pmatrix}$$

The magnitudes are:

	$\|\nabla I\|$	$\|\nabla I\|_+$	$\|\nabla I\|_{\max}$
E_h	h	h	h
E_v	h	h	h
E_d	$\sqrt{2}h$	$2h$	h

Remark. In practice, the signal of an image is not always smooth due to noise. Derivatives amplify noise and are therefore unable to recognize edges.
Smoothing the signal before computing the derivative allows to reduce the noise but also blurs the edges, making it more difficult to localize them.
A solution is to smooth and differentiate in a single operation by approximating the gradient as a difference of averages.

Smooth derivative Compute the approximation of a partial derivative as the difference of the pixels in a given window. For instance, considering a window of 3 pixels, the cross-correlation kernels are:

$$\frac{1}{3}\begin{pmatrix}-1 & 1\\-1 & 1\\-1 & 1\end{pmatrix} \qquad \frac{1}{3}\begin{pmatrix}-1 & -1 & -1\\1 & 1 & 1\end{pmatrix}$$

Prewitt operator Derivative approximation using central differences. The cross-correlation kernels are:

$$\frac{1}{3}\begin{pmatrix}-1 & 0 & 1\\-1 & 0 & 1\\-1 & 0 & 1\end{pmatrix} \qquad \frac{1}{3}\begin{pmatrix}-1 & -1 & -1\\0 & 0 & 0\\1 & 1 & 1\end{pmatrix}$$

Sobel operator Prewitt operator where the central pixels have a higher weight. The cross-correlation kernels are:

$$\frac{1}{4}\begin{pmatrix}-1 & 0 & 1\\-2 & 0 & 2\\-1 & 0 & 1\end{pmatrix} \qquad \frac{1}{4}\begin{pmatrix}-1 & -2 & -1\\0 & 0 & 0\\1 & 2 & 1\end{pmatrix}$$

Remark. Thresholding is inaccurate as choosing the threshold is not straightforward. An image has strong and weak edges. Trying to detect one type might lead to poor detection of the other.
A better solution is to find a local maxima of the absolute value of the derivatives.

3.2 Non-maxima suppression (NMS)

Non-maxima suppression (NMS

Algorithm that looks for local maxima of the absolute value of the gradient along the gradient direction.
The algorithm works as follows:

1. Given a pixel at coordinates (i, j), estimate the magnitude $G = \|\nabla I(i, j)\|$ and the direction θ of the gradient.
2. Consider two points A and B along the direction θ passing through (i, j) and compute their gradient magnitudes G_A and G_B.
3. Substitute the pixel (i, j) as follows:

$$\texttt{NMS}(i,j) = \begin{cases} 1 & (G > G_A) \wedge (G > G_B) \text{ (i.e. local maximum)} \\ 0 & \text{otherwise} \end{cases}$$

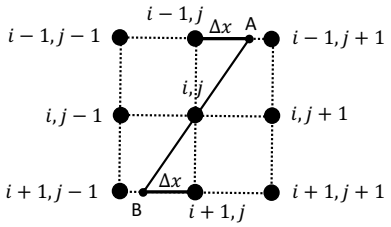
Remark. After applying NMS, the resulting signal (now composed of 0s and 1s) is converted back to the original gradient magnitudes in such a way that pixels 0ed by NMS remain 0 and pixels set at 1 by NMS return to their original value.
After the conversion, a thresholding step might be applied to filter out unwanted edges that are either due to noise or not relevant.

3.2.1 Linear interpolation

As there might not be two points A and B along the direction θ belonging the the discrete pixel grid and because one doesn't want to consider two points too far from (i, j), it is possible to use linear interpolation to estimate the gradients of A and B even if off-grid by an offset Δx:

$$G_1 = \|\nabla I(i-1, j)\| \qquad G_2 = \|\nabla I(i-1, j+1)\|$$
$$G_3 = \|\nabla I(i+1, j)\| \qquad G_4 = \|\nabla I(i+1, j-1)\|$$

$$G_A \approx G1 + (G2 - G1)\Delta x$$
$$G_B \approx G3 - (G3 - G4)\Delta x$$



3.3 Canny's edge detector

Method based on three criteria:

Good detection Correctly detect edges in noisy images.

Good localization Minimize the distance between the found edges and the true edges.

One response to one edge Detect only one pixel at each true edge.

In the 1D case, the optimal operator consists in finding the local extrema of the signal obtained by convolving the image and the Gaussian first-order derivative.
Generalized for the 2D case, Canny's edge detector does the following:

1. Gaussian smoothing.
2. Gradient computation.
3. NMS and thresholding.

It is possible to exploit the convolutions property of being commutative w.r.t. differentiation and simplify the smoothing and gradient computation:

$$\partial_x I(x,y) = \frac{\partial}{\partial x}(I(x,y) * G(x,y)) = I(x,y) * \frac{\partial G(x,y)}{\partial x}$$
$$\partial_y I(x,y) = \frac{\partial}{\partial y}(I(x,y) * G(x,y)) = I(x,y) * \frac{\partial G(x,y)}{\partial y}$$

By leveraging the separability of a 2D Gaussian ($G(x,y) = G_1(x)G_2(y)$), the computation can be reduced to 1D convolutions:

$$\partial_x I(x,y) = I(x,y) * (G_1'(x)G_2(y)) = (I(x,y) * G_1'(x)) * G_2(y)$$
$$\partial_y I(x,y) = I(x,y) * (G_1(x)G_2'(y)) = (I(x,y) * G_1(x)) * G_2'(y)$$

Remark. When magnitude varies along the object contour, thresholding might remove true edges (edge streaking).

Hysteresis thresholding Given a high threshold T_h and a low threshold T_l, depending on its magnitude, a pixel (i,j) can be considered a:

Strong edge $\nabla I(i,j) > T_h$.

Weak edge $\nabla I(i,j) > T_l$ and the pixel (i,j) is a neighbor of a strong/weak edge.

In practice, the algorithm starts from strong edges and "propagates" them.

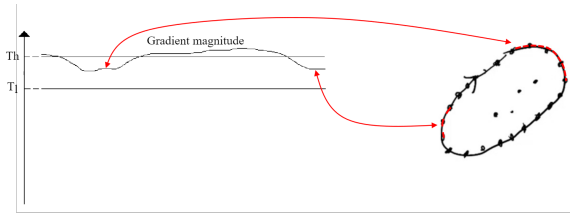


Figure 3.2: Application of hysteresis thresholding

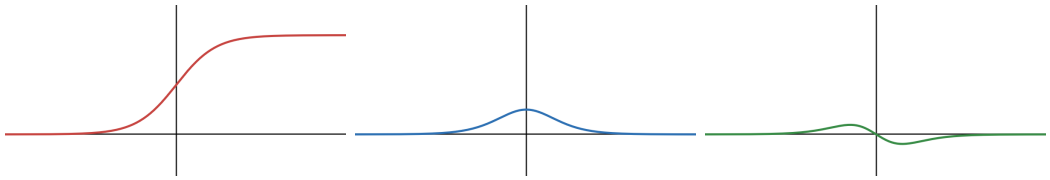
Remark. The output of Canny's edge detector is not an image with edges but a list of edges.
Note that if the edges of two objects intersect, this will be recognized as a single edge.

3.4 Zero-crossing edge detector

Zero-crossing edge detector Detect edges by finding zero-crossing of the second derivative of the signal.

Remark. A zero-crossing is a point at 0 where the function changes sign.

Remark. This approach does not require a threshold anymore but is computationally more expensive.



(a) Input signal (b) First-order derivative (c) Second-order derivative

Laplacian Approximation of the second-order derivative:

$$\nabla^2 I(x,y) \approx \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2} = \partial_{x,x} I + \partial_{y,y} I$$

Discrete Laplacian Use forward difference to compute first-order derivatives, followed by backward difference to compute second-order derivatives.

$$\begin{aligned}
\partial_{x,x} I(i,j) &\approx \partial_x I(i,j) - \partial_x I(i,j-1) \\
&= \big(I(i,j+1) - I(i,j)\big) - \big(I(i,j) - I(i,j-1)\big) \\
&= I(i,j+1) - 2I(i,j) + I(i,j-1)
\end{aligned}$$

$$\begin{aligned}
\partial_{y,y} I(i,j) &\approx \partial_y I(i,j) - \partial_y I(i-1,j) \\
&= \big(I(i+1,j) - I(i,j)\big) - \big(I(i,j) - I(i-1,j)\big) \\
&= I(i+1,j) - 2I(i,j) + I(i-1,j)
\end{aligned}$$

This is equivalent to applying the cross-correlation kernel:

$$\nabla^2 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Remark. It can be shown that zero-crossings of the Laplacian typically lay close to those of the second-order derivative.

3.4.1 Laplacian of Gaussian (LoG)

Laplacian of Gaussian (LoG) does the following:

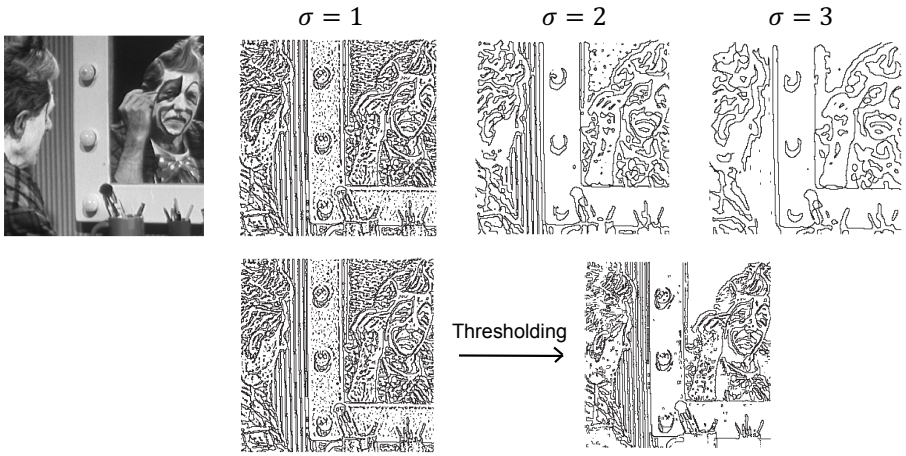
1. Gaussian smoothing.
2. Second-order differentiation using the Laplacian filter.
3. Zero-crossings extraction.

Discrete zero-crossing As the image consists of a discrete grid of pixels, finding a zero-crossing as per definition is not always possible. Instead, an edge is detected when there is a change of sign in the magnitude. The edge pixel can be identified as:

- The pixel where the magnitude is positive.
- The pixel where the magnitude is negative.
- The pixel where the absolute value of the magnitude is smaller. This is usually the best choice as it is closer to the true zero-crossing.

Remark. A final thresholding might still be useful to remove uninteresting edges.

Remark. Smaller values of σ of the smoothing operator detect more detailed edges, while higher σ captures more general edges.



4 Local features

Correspondence points Image points projected from the same 3D point from different views of the scene.

> **Example** (Homography)**.** Align two images of the same scene to create a larger image. An homography requires at least 4 correspondences. To find them, it does the following:
>
> - Independently find salient points in the two images.
> - Compute a local description of the salient points.
> - Compare descriptions to find matching points.

Local invariant features Find correspondences in three steps:

Detection Find salient points (keypoints).

The detector should have the following properties:

Repeatability Find the same keypoints across different images.

Saliency Find keypoints surrounded by informative patterns.

Fast As it must scan the entire image.

Description Compute a descriptor for each salient point based on its neighborhood.

A descriptor should have the following properties:

Invariant Robust to as many transformations as possible (i.e. illumination, weather, scaling, viewpoint, ...).

Distinctiveness/robustness trade-off The description should only capture important information around a keypoint and ignore irrelevant features or noise.

Compactness The description should be concise.

Matching Identify the same descriptor across different images.

> **Remark.** Edges are not good interest points as they are locally ambiguous (i.e. pixels are very similar along the direction of the gradient).
> Corners on the other hand are more suited as they have a larger variation along all directions.

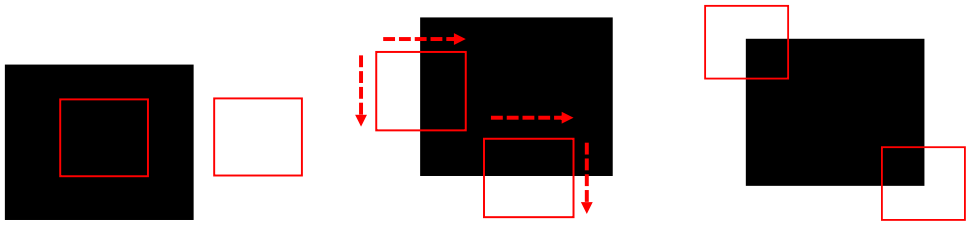
4.1 Moravec's corner detector

Moravec's corner detector

Given a window W of size $n \times n$, the cornerness of a pixel p is given by the minimum squared difference between the intensity of W centered on p and the intensity of W centered on each of its neighbors:

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

After computing the cornerness of each pixel, one can apply thresholding and then NMS to obtain a matrix where 1 indicates a corner.



(a) Flat region: $C(p)$ is low. (b) Edge: $C(p)$ is low. (c) Corner: $C(p)$ is high.

Remark. Moravec corner detector is isotropic (i.e. independent of the direction).

4.2 Harris' corner detector

4.2.1 Structure matrix

Harris' corner detector uses an error function formulated as the continuous version of Moravec's detector and assumes an infinitesimal shift $(\Delta x, \Delta y)$ of the image:

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

where $w(x,y)$ represents a window and can be seen as a mask with 1 when the pixel belongs to the window and 0 otherwise.
By employing the Taylor's series $f(x+\Delta x) = f(x) + f'(x)\Delta x$, we can expand the intensity difference as:

$$\begin{aligned} I(x+\Delta x, y+\Delta y) - I(x,y) &\approx \big(I(x,y) + \partial_x I(x,y)\Delta x + \partial_y I(x,y)\Delta y\big) - I(x,y) \\ &= \partial_x I(x,y)\Delta x + \partial_y I(x,y)\Delta y \end{aligned}$$

By developing the error function into matrix form, we obtain the following:

$$\begin{aligned} E(\Delta x, \Delta y) &= \sum_{x,y} w(x,y)\big(I(x+\Delta x, y+\Delta y) - I(x,y)\big)^2 \\ &= \sum_{x,y} w(x,y)\big(\partial_x I(x,y)\Delta x + \partial_y I(x,y)\Delta y\big)^2 \\ &= \sum_{x,y} w(x,y)\big(\partial_x^2 I(x,y)\Delta x^2 + 2\partial_x I(x,y)\partial_y I(x,y)\Delta x\Delta y + \partial_y^2 I(x,y)\Delta y^2\big) \\ &= \sum_{x,y} w(x,y)\left(\begin{bmatrix}\Delta x & \Delta y\end{bmatrix}\begin{bmatrix}\partial_x^2 I(x,y) & \partial_x I(x,y)\partial_y I(x,y) \\ \partial_x I(x,y)\partial_y I(x,y) & \partial_y^2 I(x,y)\end{bmatrix}\begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix}\right) \\ &= \begin{bmatrix}\Delta x & \Delta y\end{bmatrix}\begin{bmatrix}\sum_{x,y} w(x,y)\partial_x^2 I(x,y) & \sum_{x,y} w(x,y)(\partial_x I(x,y)\partial_y I(x,y)) \\ \sum_{x,y} w(x,y)(\partial_x I(x,y)\partial_y I(x,y)) & \sum_{x,y} w(x,y)\partial_y^2 I(x,y)\end{bmatrix}\begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix} \\ &= \begin{bmatrix}\Delta x & \Delta y\end{bmatrix}\boldsymbol{M}\begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix} \end{aligned}$$

Structure matrix Matrix $\boldsymbol{M}_w$ that encodes the local structure of the image at the pixels within a window w.

$$\boldsymbol{M}_w = \begin{pmatrix}\sum_{x,y} w(x,y)\partial_x^2 I(x,y) & \sum_{x,y} w(x,y)(\partial_x I(x,y)\partial_y I(x,y)) \\ \sum_{x,y} w(x,y)(\partial_x I(x,y)\partial_y I(x,y)) & \sum_{x,y} w(x,y)\partial_y^2 I(x,y)\end{pmatrix}$$

$\boldsymbol{M}_w$ is real and symmetric, thus it is diagonalizable through an orthogonal matrix $\boldsymbol{R}$:

$$\boldsymbol{M}_w = \boldsymbol{R} \begin{pmatrix} \lambda_1^{(w)} & 0 \\ 0 & \lambda_2^{(w)} \end{pmatrix} \boldsymbol{R}^T$$

$\boldsymbol{R}^T$ is the rotation matrix that aligns the image to the eigenvectors of $\boldsymbol{M}_w$, while the eigenvalues remain the same for any rotation of the same patch.

Therefore, the eigenvalues $\lambda_1^{(w)}, \lambda_2^{(w)}$ of $\boldsymbol{M}_w$ allow to detect intensity changes along the shift direction:

$$\begin{aligned} E(\Delta x, \Delta y) &= \begin{pmatrix} \Delta x & \Delta y \end{pmatrix} \begin{pmatrix} \lambda_1^{(w)} & 0 \\ 0 & \lambda_2^{(w)} \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \\ &= \lambda_1^{(w)} \Delta x^2 + \lambda_2^{(w)} \Delta y^2 \end{aligned}$$

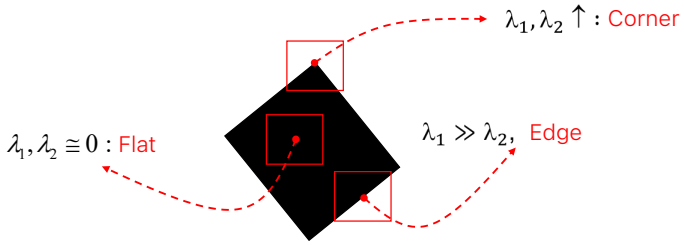


Figure 4.2: Eigenvalues relationship at different regions of an image.

4.2.2 Algorithm

Harris' corner detector

As computing the eigenvalues of $\boldsymbol{M}_w$ at each pixel is expensive, a more efficient cornerness function is the following:

$$\begin{aligned} C(x, y) &= \lambda_1^{(w)} \lambda_2^{(w)} - k(\lambda_1^{(w)} + \lambda_2^{(w)})^2 \\ &= \det(\boldsymbol{M}_{w(x,y)}) - k \cdot \operatorname{trace}(\boldsymbol{M}_{w(x,y)})^2 \end{aligned}$$

where k is a hyperparameter (empirically in $[0.04, 0.06]$).

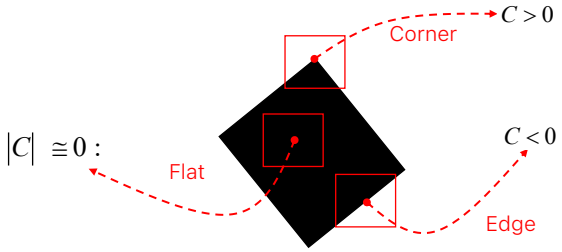


Figure 4.3: Cornerness at different regions.

After computing the cornerness of each pixel, one can apply thresholding and then NMS.

Remark. The window function $w(x, y)$ in the original work follows a Gaussian distribution.

4.2.3 Properties

Harris' corner detector enjoys the following properties:

Rotation invariance The eigenvalues are invariant to a rotation of the image.

No affine intensity change invariance An affine intensity change of a signal consists of a gain factor and the addition of a bias (i.e. $I' = \alpha I + \beta$).

Invariance to bias Harris' detector is invariant to an additive bias ($I' = I + \beta$) as a consequence of the approximate derivative computation:

$$\partial_x I'(i,j) = I'(i,j+1) - I'(i,j) = (I(i,j+1) + \cancel{\beta}) - (I(i,j) + \cancel{\beta})$$

No invariance to gain factor Harris' detector is not invariant to a gain factor ($I' = \alpha I$) as the multiplicative factor is carried in the derivatives.

Remark. In other words, Harris' detector is not illumination invariant.

No scale invariance Harris' detector is not scale invariant as the use of a fixed window size makes it impossible to recognize the same features when the image is scaled.

4.3 Multi-scale feature detector

Depending on the scale, an image may exhibit more or less details. A naive approach consists of using a smaller window size for images with a smaller scale, but this is not always able to capture the same features due to the details difference.

Scale-space One-parameter family of images obtained by increasingly smoothing the input image.

Remark. When smoothing, small details should disappear and no new structures should be introduced.

Remark. It is possible to use the same window size when working with scale-space images.

Gaussian scale-space Scale-space obtained using Gaussian smoothing:

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

where σ is the standard deviation but also the level of scaling.

Figure 4.4: Gaussian scale-space example

4.3.1 Scale-normalized LoG blob detection

Scale-normalized Laplacian of Gaussian LoG scaled by a factor of σ^2:

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

σ^2 avoids small derivatives when the scaling (σ) is large.

Characteristic scale Scale σ that produces a peak in the Laplacian response at a given pixel [**slides:scale˙normalized˙log**].

Algorithm Blob (circle) detection using scale-normalized LoG works as follows [**slides:scale˙normalized˙log**]:

Scal
LoG

1. Create a Gaussian scale-space by applying the scale-normalized Laplacian of Gaussian with different values of σ.
2. For each pixel, find the characteristic scale and its corresponding Laplacian response across the scale-space (automatic scale selection).
3. Filter out the pixels whose response is lower than a threshold and find the peaks.
4. The found pixels are the centers of the blobs. It can be shown that the radius is given by $r = \sigma\sqrt{2}$.

When detecting a peak, there are two cases:

Maximum Dark blobs on a light background.

Minimum Light blobs on a dark background.

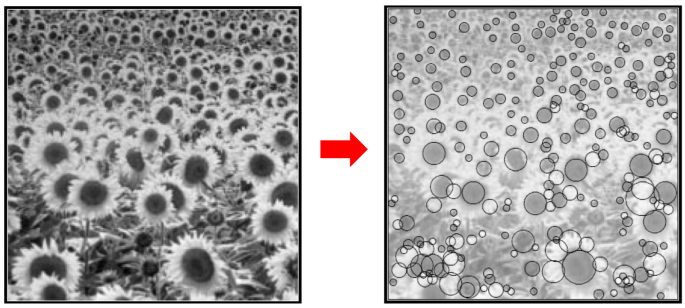


Figure 4.5: Example of application of the algorithm

Remark. The intuitive idea of the detection algorithm is the following:

- $F(x, y, \sigma)$ keeps growing for σs that capture areas within the blob (i.e. with similar intensity).
- The Laplacian reaches its peak when its weights capture the entire blob (virtually it detects an edge).
- After the peak, the LoG filter will also capture intensities outside the blob and therefore decrease.

Remark. Using different scales creates the effect of searching in a 3D space.

Remark. Scale-normalized LoG blob detection is scale and rotation invariant.

Remark. It empirically holds that, given two points representing the centers of two blobs, the ratio between the two characteristic scales is approximately the ratio between the diameters of the two blobs.

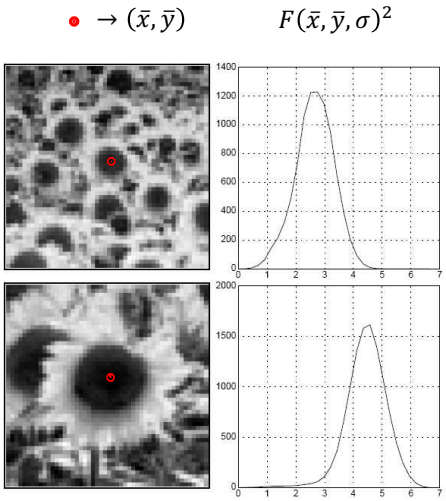


Figure 4.6: Scale-normalized LoG computed on varying σ. Note that, in the second image, the characteristic scale is higher as the scale is larger.

4.3.2 Difference of Gaussians blob detection

Difference of Gaussians (DoG) Approximation of the scale-normalized LoG computed as:

$$\begin{aligned} \texttt{DoG}(x, y, \sigma) &= \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned}$$

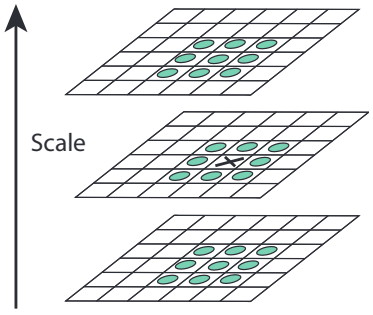
Theorem 4.3.1. It can be proved that the DoG kernel is a scaled version of the LoG kernel:

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

Remark. As we are interested in extrema, the scaling factor is irrelevant.

Extrema detection Given three DoG images with scales σ_{i-1}, σ_i and σ_{i+1}, a pixel (x, y, σ_i) is an extrema (i.e. keypoint) iff:

- It is an extrema in a 3×3 patch centered on it (8 pixels as (x, y, σ_i) is excluded).
- It is an extrema in a 3×3 patch centered on the pixels at (x, y, σ_{i-1}) and at (x, y, σ_{i+1}) $(9 + 9$ pixels).



Algorithm To detect blob centers (i.e. a DoG extrema), an octave of s DoG images is computed as follows:

1. Compute a scale-space L of $s+1$ Gaussian smoothed images with σ varying by a factor $k = 2^{1/s}$. As the extrema detection method requires checking the DoG images above and below, two additional Gaussians (with scales $k^{-1}\sigma$ and $k^{s+1}\sigma$) are computed.
2. The DoG image $\texttt{DoG}(\cdot, \cdot, k^i\sigma)$ is obtained as the difference between the images $L(\cdot, \cdot, k^{i+1}\sigma)$ and $L(\cdot, \cdot, k^i\sigma)$ of the Gaussian scale-space.

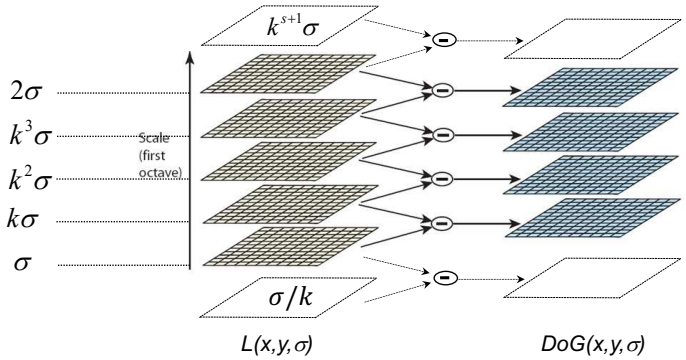


Figure 4.7: Example of octave computation with $s = 4$

3. Extrema detection is done as described above across the s DoG images.
4. Points with a weak DoG response can be pruned through thresholding. Furthermore, it has been observed that strong DoG points along edges are unstable and can be also pruned.

Octaves should be computed using different starting σs. Instead of recomputing the Gaussian scale-space, it is possible to simply down-sample the already computed Gaussians and compute the DoG images starting from shrunk smoothed images.

Remark. In the original work, the input image is first enlarged by a factor of 2. Then, four octaves are computed starting from the enlarged image (i.e. images of size factor $\times 2$, $\times 1$, $\times \frac{1}{2}$ and $\times \frac{1}{4}$ are considered).

Remark. The original work found out that the best hyperparameters are $s = 3$ and $\sigma = 1.6$.

Remark. DoG blob detection is scale and rotation invariant.

4.4 Descriptor

After finding the keypoints, a descriptor of a keypoint is computed from the pixels within a patch centered on it.

4.4.1 DoG descriptor

Canonical / Characteristic orientation Direction along which the magnitudes of the gradients of the neighboring pixels of a keypoint are the highest.

Given a pixel (x, y), its gradient magnitude and direction is computed from the Gaussian smoothed image L:

$$\|\nabla L(x,y)\| = \sqrt{\big(L(x+1,y) - L(x-1,y)\big)^2 + \big(L(x,y+1) - L(x,y-1)\big)^2}$$
$$\theta_L(x,y) = \arctan\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right)$$

Orientation histogram By dividing the directions into bins (e.g. bins of size 10°), it is possible to define for each keypoint a histogram by considering its neighboring pixels within a patch. For each pixel (x, y) neighboring a keypoint (x_k, y_k), its contribution to the histogram along the direction $\theta_L(x, y)$ is given by:

$$G_{(x_k,y_k)}\left(x, y, \frac{3}{2}\sigma_s(x_k, y_k)\right) \cdot \|\nabla L(x,y)\|$$

where $G_{(x_k,y_k)}$ is a Gaussian centered on the keypoint and $\sigma_s(x_k, y_k)$ is the scale of the keypoint.

The characteristic orientation of a keypoint is given by the highest peak of the orientation histogram. Other peaks that are higher than at least 80% of the main one are also considered characteristic orientations (i.e. a keypoint might have multiple canonical orientations and, therefore, multiple descriptors).

For a more accurate estimation, a parabola is interpolated on the neighborhood of each peak and the two bins adjacent to the peak of the parabola are considered.

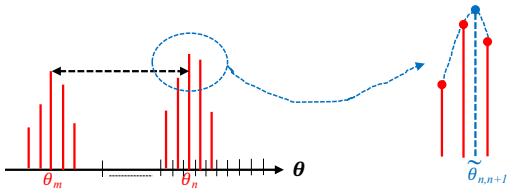


Figure 4.8: Orientation histogram and parabola interpolation

DoG descriptor Keypoints are found using the DoG detector and the descriptors are computed through patches along the canonical orientations.

| **Remark.** DoG descriptor is scale and rotation invariant.

4.4.2 Scale invariant feature transform (SIFT) descriptor

Given a keypoint, SIFT detector works as follows: SIFT descriptor

1. Center on the keypoint a 16×16 grid divided into 4×4 regions.
2. Compute, for each region, its orientation histogram with eight bins (i.e. bins of size 45°). The Gaussian weighting function is centered on the keypoint and has σ equal to half the grid size.
3. The descriptor is obtained by concatenating the histograms of each region. This results in a feature vector with 128 elements $((4 \cdot 4) \cdot 8)$.

4. Normalize the descriptor to unit length. Pixels larger than 0.2 are saturated and normalized again (for illumination invariance).

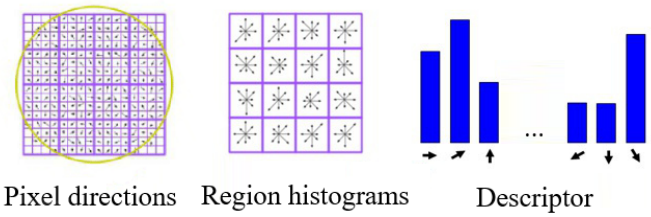
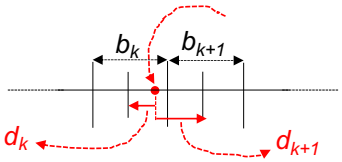


Figure 4.9: SIFT detector example

Trilinear interpolation Bins are assigned in a soft manner to avoid boundary effects. The contribution of a pixel is spread between its two adjacent bins weighted by the distance to the bin centers:



$$\begin{cases} \text{weight}_k = 1 - d_k \\ \text{weight}_{k+1} = 1 - d_{k+1} \end{cases}$$

This is done both on the histogram within a region and on the histograms between the four neighboring regions.

| **Remark.** SIFT descriptor is scale, rotation and affine intensity change invariant.

4.5 Matching

Matching the keypoints across different views is a nearest-neighbor search problem. Given a target image T and a reference image R, we want to match each keypoint in T to the most similar one in R (usually using the Euclidean distance).

Matching criteria Given the distance d_{NN} to the nearest-neighbor of a keypoint of T in R, the match is accepted by respecting one of the following criteria:

Threshold Given a threshold T, the match is accepted iff:

$$d_{\text{NN}} \leq T$$

Ratio of distances Given a threshold T and the distance to the second nearest-neighbor d_{NN2}, the match is accepted iff:

$$\frac{d_{\text{NN}}}{d_{\text{NN2}}} \leq T$$

This method allows to avoid ambiguity if two neighbors are too close.

(a) Non ambiguous match

(b) Ambiguous match

Remark. It has been empirically shown that a threshold $T = 0.8$ allows to reject 90% of wrong matches while missing 5% of correct matches.

Efficient NN search As an exhaustive search is inefficient, indexing techniques may be employed.

The main indexing technique applied for feature matching is the k-d tree in the best bin first (BBF) variant.

Remark. Best bin first is not optimal.

5 Instance-level object detection

Instance-level object detection Given a reference/model image of a specific object, determine if the object is present in the target image and estimate its pose.

> **Remark.** This is different from category-level object detection which deals with detecting classes of objects independent of appearance and pose.

5.1 Template matching

Slide the model image (template) across the target image and use a similarity/dissimilarity function with a threshold to determine if a match has been found.

Template matchin

5.1.1 Similarity/dissimilarity functions

Let I be the target image and T the template image of shape $M \times N$. Possible similarity/dissimilarity functions are:

Pixel-wise intensity differences Squared difference between the intensities of the template image and the patch in the target image:

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

> **Remark.** `SSD` is fast but not intensity invariant.

Sum of absolute differences Difference in absolute value between the intensities of the template image and the patch in the target image:

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

> **Remark.** `SAD` is fast but not intensity invariant.

Normalized cross-correlation Cosine similarity between the template image and the target image patch (which are seen as flattened vectors):

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

In other words, let $\tilde{I}_{i,j}$ be a $M \times N$ patch of the target image I starting at the coordinates (i,j). In vector form, $\texttt{NCC}(i,j)$ computes the cosine of the angle θ between $\tilde{I}_{i,j}$ and T.

$$\texttt{NCC}(i,j) = \frac{\tilde{I}_{i,j}\cdot T}{\|\tilde{I}_{i,j}\|\cdot\|T\|} = \cos\theta$$

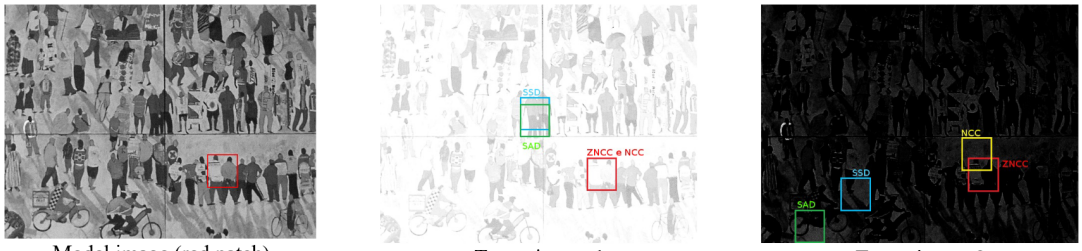
| **Remark.** `NCC` is invariant to a linear intensity change but not to an additive bias.

Zero-mean normalized cross-correlation Let $\tilde{I}_{i,j}$ be a $M \times N$ patch of I starting at the coordinates (i, j). Zero-mean normalized cross-correlation subtracts the mean of the template image and the patch of the target image before computing `NCC`:

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

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

| **Remark.** `ZNCC` is invariant to an affine intensity change.



Model image (red patch) Target image 1 Target image 2

Figure 5.1: Examples of template matching

5.2 Shape-based matching

Shape-based matching

Edge-based template matching that works as follows:

1. Use an edge detector to extract a set of control points $\{P_1, \ldots, P_n\}$ in the template image T.

2. Compute the gradient normalized to unit vector at each point P_k:

$$\nabla T(P_k) = \begin{pmatrix} \partial_x T(P_k) \\ \partial_y T(P_k) \end{pmatrix} \qquad \mathbf{u}_k(P_k) = \frac{\nabla T(P_k)}{\|\nabla T(P_k)\|}$$

3. Given a patch $\tilde{I}_{i,j}$ of the target image, compute the gradient normalized to unit vector at the points $\{\tilde{P}_1(i,y), \ldots, \tilde{P}_n(i,y)\}$ corresponding to the control points of the template image:

$$\nabla \tilde{I}_{i,j}(\tilde{P}_k) = \begin{pmatrix} \partial_x \tilde{I}_{i,j}(\tilde{P}_k) \\ \partial_y \tilde{I}_{i,j}(\tilde{P}_k) \end{pmatrix} \qquad \tilde{\mathbf{u}}_k(\tilde{P}_k) = \frac{\nabla \tilde{I}_{i,j}(\tilde{P}_k)}{\|\nabla \tilde{I}_{i,j}(\tilde{P}_k)\|}$$

4. Compute the similarity as the mean cosine similarity of each pair of gradients:

$$S(T, \tilde{I}_{i,j}) = \frac{1}{n} \sum_{k=1}^{n} \mathbf{u}_k(P_k) \cdot \tilde{\mathbf{u}}_k(\tilde{P}_k) = \frac{1}{n} \sum_{k=1}^{n} \cos\theta_k \in [-1, 1]$$

$S(T, \tilde{I}_{i,j}) = 1$ when the gradients perfectly match. A minimum threshold $S_{\min}$ is used to determine if there is a match.

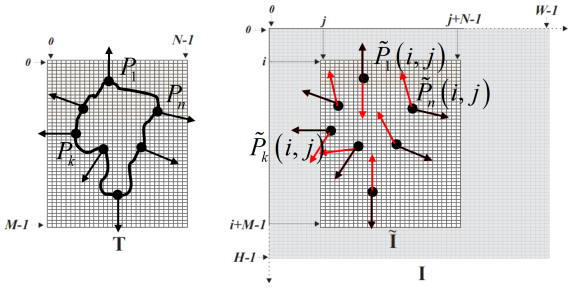


Figure 5.2: Example of control points matching

5.2.1 Invariance to contrast polarity inversion

As an object might appear on a darker or brighter background, more robust similarity functions can be employed:

Global contrast polarity inversion

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

Local contrast polarity inversion

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

Remark. This is the most robust one.

5.3 Hough transform

Detect objects of a known shape that can be expressed through an analytic equation by means of a projection from the image space to a parameter space.

Parameter space Euclidean space parametrized on the parameters ϕ of an analytic shape $\mathcal{S}_\phi$ (e.g. line, sphere, ...). A point (x, y) in the image space is projected into the curve (which also intends lines) that contains the set of parameters $\hat{\phi}$ such that the shape $\mathcal{S}_{\hat{\phi}}$ defined on those parameters passes through (x, y) in the image space.

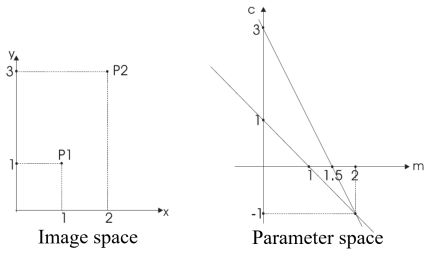
Remark. If many curves intersect at $\hat{\phi}$ in the parameter space of a shape S_ϕ, then, there is high evidence of the fact that the image points that were projected into those curves are part of the shape $S_{\hat{\phi}}$.

Example (Space of straight lines)**.** Consider the line equation:

$$\hat{y} - m\hat{x} - c = 0$$

where $(\hat{x}, \hat{y})$ are fixed while (m, c) vary (in the usual equation, it is the opposite).
This can be seen as a mapping of points $(\hat{x}, \hat{y})$ into the space of points (m, c) such that the straight line parametrized on m and c passes through $(\hat{x}, \hat{y})$ (i.e. a point in the image space is mapped into a line in the parameter space as infinite lines pass through a point).
For instance, consider two points p_1, p_2 in the image space and their projection in

the parameter space. If the two lines intersect at the point $(\tilde{m}, \tilde{c})$, then the line parametrized on $\tilde{m}$ and $\tilde{c}$ passes through both p_1 and p_2 in the image space.



Remark. By projecting n points of the image space, there are at most $\frac{n(n-1)}{2}$ intersections in the parameter space.

Algorithm Given an analytic shape that we want to detect, object detection using the Hough transform works as follows:

1. Map image points into curves of the parameter space.
2. The parameter space is quantized into $M \times N$ cells and an accumulator array A of the same shape is initialized to 0.
3. For each cell (i, j) of the discretized grid, the corresponding cell $A(i, j)$ in the accumulator array counts how many curves lie in that cell (i.e. voting process).
4. Find the local maxima of A and apply a threshold if needed. The points that were projected into the curves passing through a maximum cell are points belonging to an object of the sought shape.

Remark. The Hough transform reduces a global detection problem into a local detection in the parameter space.

Remark. The Hough transform is usually preceded by an edge detection phase so that the input consists of the edge pixels of the image.

Remark. The Hough transform is robust to noise and can detect partially occluded images (with a suitable threshold).

5.3.1 Hough transform for line detection

For line detection, points in the image space are projected into lines of the parameter space.

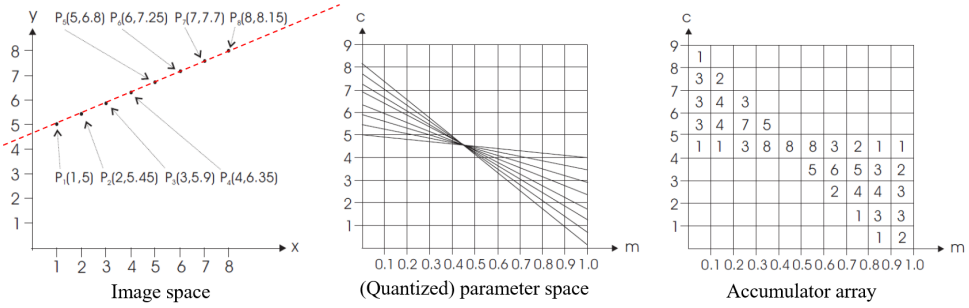


Figure 5.3: Example of line detection

In practice, the parametrization on (m, c) of the equation $y - mx - c = 0$ is impractical as m and c have an infinite range. An alternative approach is to describe straight lines as

linear trigonometric equations parametrized on (θ, ρ):

$$x \cos\theta + y \sin\theta - \rho = 0$$

In this form, $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ while $\rho \in [-\rho_{\max}, \rho_{\max}]$ where $\rho_{\max}$ is usually taken of the same size as the diagonal of the image (e.g. for square $N \times N$ images, $\rho_{\max} = N\sqrt{2}$).

5.4 Generalized Hough transform

Hough transform extended to detect an arbitrary shape.

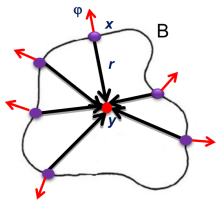
5.4.1 Naive approach

Off-line phase Phase in which the model of the template object is built. Given a template shape, the algorithm works as follows:

1. Fix a reference point $\mathbf{y}$ (barycenter). $\mathbf{y}$ is typically within the shape of the object.
2. For each point $\mathbf{x}$ at the border B of the object:
   a) Compute its gradient direction $\varphi(\mathbf{x})$ and discretize it according to a chosen step $\Delta\varphi$.
   b) Compute the vector $\mathbf{r} = \mathbf{y} - \mathbf{x}$ as the distance of $\mathbf{x}$ to the barycenter.
   c) Store $\mathbf{r}$ as a function of $\Delta\varphi$ in a table (R-table). Note that more than one vector might be associated with the same $\Delta\varphi$.

Example.

i	φ_i	R_{φ_i}
0	0	$\{\mathbf{r} \mid \mathbf{r} = \mathbf{y} - \mathbf{x}. \forall \mathbf{x} \in B : \varphi(\mathbf{x}) = 0\}$
1	$\Delta\varphi$	$\{\mathbf{r} \mid \mathbf{r} = \mathbf{y} - \mathbf{x}. \forall \mathbf{x} \in B : \varphi(\mathbf{x}) = \Delta\varphi\}$
2	$2\Delta\varphi$	$\{\mathbf{r} \mid \mathbf{r} = \mathbf{y} - \mathbf{x}. \forall \mathbf{x} \in B : \varphi(\mathbf{x}) = 2\Delta\varphi\}$
⋮	⋮	⋮



On-line phase Phase in which the object is detected. Given a $M \times N$ image, the algorithm works as follows:

1. Find the edges E of the image.
2. Initialize an accumulator array A of the same shape of the image.
3. For each edge pixel $\mathbf{x} \in E$:
   a) Compute its gradient direction $\varphi(\mathbf{x})$ discretized to match the step $\Delta\varphi$ of the R-table.
   b) For each $\mathbf{r}_i$ in the corresponding row of the R-table:
      i. Compute an estimate of the barycenter as $\mathbf{y} = \mathbf{x} + \mathbf{r}_i$.
      ii. Cast a vote in the accumulator array $A[\mathbf{y}]$+=1.
4. Find the local maxima of the accumulator vector to estimate the barycenter. The shape can then be visually found by overlaying the template barycenter to the found barycenter.

Remark. This approach is not rotation and scale invariant.
Therefore, if rotation or scale is changed, the method should try different rotations or scales.

5.4.2 Star model

Generalized Hough transform based on local invariant features. Star model

Off-line phase Given a template, its model is obtained as follows:

1. Detect local invariant features $F = \{F_1, \dots, F_N\}$ and compute their descriptors. Each feature F_i is described by the tuple:

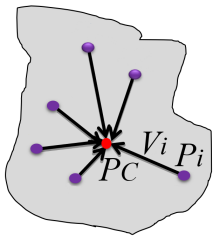
$$F_i = (\mathbf{P}_i, \varphi_i, S_i, \mathbf{D}_i) = \text{(position, canonical orientation, scale, descriptor)}$$

2. Compute the position of the barycenter $\mathbf{P}_C$ as:

$$\mathbf{P}_C = \frac{1}{N} \sum_{i=1}^{N} \mathbf{P}_i$$

3. For each feature F_i compute the joining vector $\mathbf{V}_i = \mathbf{P}_C - \mathbf{P}_i$ as the distance to the barycenter and add it to its associated tuple to obtain:

$$F_i = (\mathbf{P}_i, \varphi_i, S_i, \mathbf{D}_i, \mathbf{V}_i)$$



Remark. The R-table is not needed anymore.

On-line phase Given a target image, detection works as follows:

1. Extract the local invariant features $\tilde{F} = \{\tilde{F}_1, \dots, \tilde{F}_M\}$ of the target image. Each feature $\tilde{F}_i$ is describe by the tuple:

$$\tilde{F}_i = (\tilde{\mathbf{P}}_i, \tilde{\varphi}_i, \tilde{S}_i, \tilde{\mathbf{D}}_i)$$

2. Match the features $\tilde{F}$ of the target image to the features F of the template image through the descriptors.
3. Initialize a 4-dimensional accumulator array. Two dimensions match the image shape. The other two represent different rotations and scales.
4. For each target feature $\tilde{F}_i = (\tilde{\mathbf{P}}_i, \tilde{\varphi}_i, \tilde{S}_i, \tilde{\mathbf{D}}_i)$ with matching template feature $F_j = (\mathbf{P}_j, \varphi_j, S_j, \mathbf{D}_j, \mathbf{V}_j)$:
   a) Align the joining vector $\mathbf{V}_j$ to the scale and rotation of $\tilde{F}_i$.

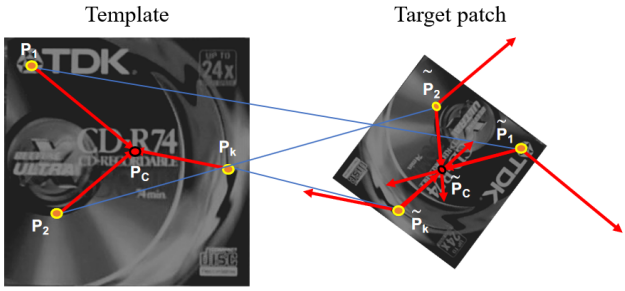
   Let $\boldsymbol{R}(\phi)$ be a ϕ degree rotation matrix. The aligned joining vector $\tilde{\mathbf{V}}_i$ is obtained as:

$$\Delta\varphi_i = \tilde{\varphi}_i - \varphi_j \qquad \Delta S_i = \frac{\tilde{S}_i}{S_j}$$

$$\tilde{\mathbf{V}}_i = \Delta S_i \cdot \boldsymbol{R}(\Delta\varphi_i)\mathbf{V}_i$$

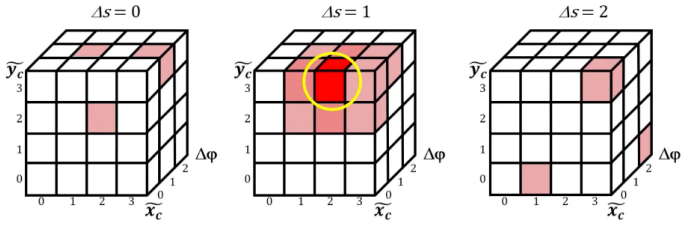
b) Estimate the barycenter $\tilde{\mathbf{P}}_{C_i}$ associated to the feature $\tilde{F}_i$ as:

$$\tilde{\mathbf{P}}_{C_i} = \tilde{\mathbf{P}}_i + \tilde{\mathbf{V}}_i$$



c) Cast a vote in the accumulator array $A[\tilde{\mathbf{P}}_{C_i}, \Delta S_i, \Delta\varphi_i]$+=1.

5. Find the local maxima of the accumulator vector to estimate the barycenters. The shape can then be visually found by overlaying the template barycenter to the found barycenters with the proper scaling and rotation.



<end of course>