# Image Processing and Computer Vision

Matteo Donati

July 9, 2021

# Contents

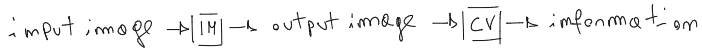
# Chapter 1

# Introduction

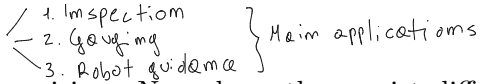
## 1.1 Image Processing and Computer Vision

- Image processing methods allow the manipulation of images. These methods are functions from pixels to pixels.
- Computer vision methods allow the extrapolation of informations from images. These methods are functions from pixels to informations.

Sometimes the two fields intersect. For example, one might need to transform an image before applying computer vision methods.




### 1.1.1 Computer Vision as Process Technology

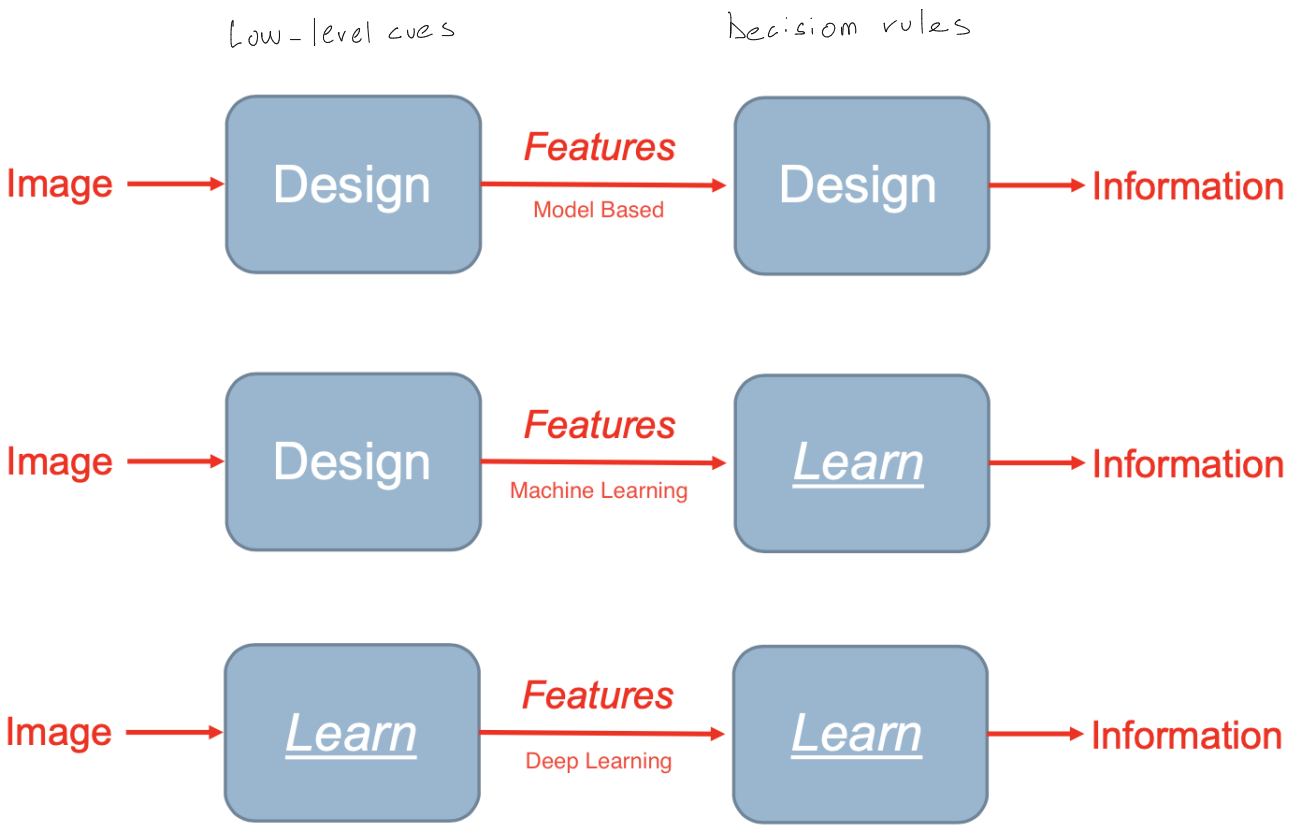



In factories, computer vision is usually referred as machine vision. Nowadays, there exist different types of machine vision systems:

- Vision sensors, which are cameras with a pre-programmed processor. They behave like sensors.
- Smart cameras, which are cameras with programmable embedded intelligence (usually CPU, DSP, FPGA).
- PC-based computer vision systems, which are composed by a simple camera connected to a computer.
- Application specific vision systems, which are systems designed for specific applications.

### 1.1.2 The Evolution of Computer Vision

Nowadays, computer vision methods are mostly based on deep learning techniques. Neural networks are able to learn how to extract meaningful features from the pixels and how to extract informations from those features.


Low-level cues
Decision rules
Image
Design
Features
Model Based
Design
Information
Image
Design
Features
Machine Learning
Learn
Information
Image
Learn
Features
Deep Learning
Learn
Information

# Chapter 2

# Image Formation and Acquisition

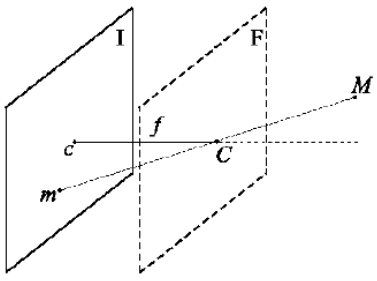
In order to be able to understand the image acquisition process, it is necessary to study:

- The geometric relationship between scene points and image points.
- The radiometric relationship between the brightness of image points and the light emitted by scene points.
- The image digitization process.

In the first place, it is necessary to find a device to capture the image with. The simplest device is called *pinhole camera*. This camera could be thought as just a hole in a box that reflects an image. Given a light ray emitted from a point in the scene, this ray will pass through the hole and hit the photosensitive material on the image plane, therefore creating an image. In particular, every point of the object will emit light in different directions, but because the box's hole is consider infinitesimally small, just one ray will be able to pass through the hole. Any other camera can be modelled as if it were a pinhole camera.

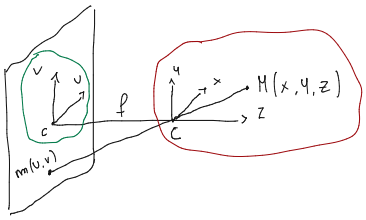
## 2.1 Perspective Projection — mapping using a pinhole camera.

The geometric model of image formation in a pinhole camera is known as *Perspective Projection*:




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

Can be seen as a function:
$m = G(M)$




— Reference frames:
CRF : camera reference frame (3D)
IRF : image " " (2D)

The equations to map scene points into their corresponding image points are the following:

$$\frac{u}{x} = \frac{v}{y} = -\frac{f}{z} \qquad \rightarrow \qquad u = -x \cdot \frac{f}{z} \qquad v = -y \cdot \frac{f}{z} \tag{2.1}$$

To get rid of the up-down and left-right inversions:

$$u = x \cdot \frac{f}{z} \qquad v = y \cdot \frac{f}{z} \tag{2.2}$$

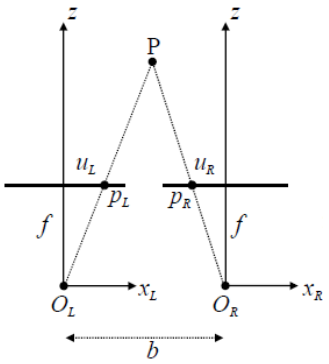
### 2.1.1 Properties of Perspective Projection

- 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 $l = L \cdot f/z$.
- Perspective projection maps 3D lines into image lines. — not if the camera has lens (pinhole camera has no lens)
- Ratios of lengths are not preserved, unless the scene is planar and parallel to the image plane. e.g. $\frac{L_1}{L_2} \neq \frac{l_1}{l_2}$
- Parallelism between 3D lines is not preserved, except for lines parallel to the image plane.

## 2.2 Standard Stereo Geometry

The image formation process deals with mapping a 3D space onto a 2D space. Thus, recovering the 3D structure of a scene from a single image is an ill posed problem (the solution is not unique). As a matter of fact, given a image point it is possible to only state that its corresponding scene point lays on a line. Stereo images allow to infer 3D. (triangulation)
Two cameras are horizontally placed with a distance $b$ (*baseline*). Therefore, they have the same focal length, coplanar image planes and parallel $(x, y, z)$ axes. The transformation between the two reference frames is just the translation $b$: $x_L - x_R = b$, $y_L = y_R = y$ and $z_L = z_R = z$.

two reference frames.
$CRF_L$, $CRF_R$
In particular:
$\vec{P}_L - \vec{P}_R = \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix}$




From equations 2.2, $v_L, v_R, u_L$ and $u_R$ are equal to:

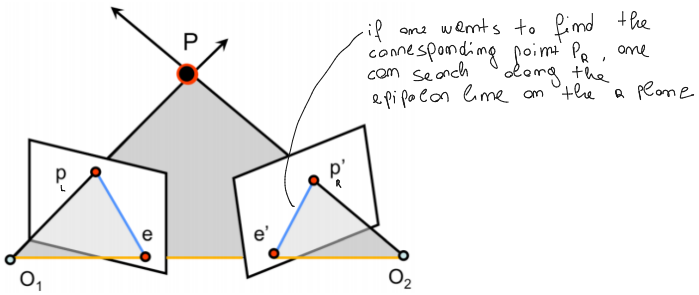
$$v_L = v_R = y \cdot \frac{f}{z}, \qquad u_L = x_L \cdot \frac{f}{z}, \qquad u_R = x_R \cdot \frac{f}{z} \tag{2.3}$$

The quantity $u_L - u_R = \underline{d} = b \cdot f/z$ is called *disparity*. The $z$ and $d$ values are proportional:

$$z = b \cdot \frac{f}{d} \tag{2.4}$$

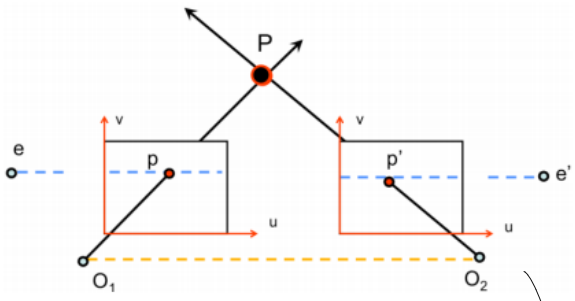
## 2.3 Epipolar Geometry

Most of the times, the two cameras are not perfectly aligned. The geometry that relates the cameras, points in 3D, and the corresponding observations is referred to as the epipolar geometry.




- The plane defined by the two camera centres and $P$ is known as the epipolar plane.
- The locations of where the baseline intersects the two image planes are known as the the epipoles ($e$ and $e'$).
- The intersection of the epipolar plane and the two image planes are known as the epipolar lines

The epipolar lines have the property that they intersect the baseline at the respective epipoles in the image plane. When the image planes are parallel to each other, then the epipoles $e$ and $e'$ will be located at infinity since the baseline joining the centers $O_1$, $O_2$ is parallel to the image planes:




it is also possible to go from epipolar geometry to standard stereo geometry through rectifications.
└ epipolar lines are horizontally aligned.

## 2.4 Vanishing Points

The vanishing point of a 3D line is the image of the point at infinity of the line. In particular, all parallel 3D lines will share the same vanishing point, which is the point on the image plane where the images of parallel 3D lines converge.
In order to find the vanishing point of a line, it is possible to use the parametric equation of the line:

$$\mathbf{M} = \mathbf{M}_0 + \lambda \mathbf{D} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \lambda \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{2.5}$$

where $\mathbf{M}_0$ is a point on the 3D line and $\mathbf{D}$ is the direction cosines vector.
Projecting a generic point of the 3D line onto the image place will produce:

$$\mathbf{m} = \begin{bmatrix} u \\ v \end{bmatrix}, \qquad u = f \cdot \frac{x_0 + \lambda a}{z_0 + \lambda c}, \qquad v = f \cdot \frac{y_0 + \lambda b}{z_0 + \lambda c} \tag{2.6}$$

To get the vanishing point of the 3D line, it is necessary to consider the infinitely distant point along the line. From equations 2.6:

$$\mathbf{m}_\infty = \begin{bmatrix} u_\infty \\ v_\infty \end{bmatrix}, \qquad u_\infty = \lim_{\lambda \to \infty} u = f \cdot \frac{\lambda a}{\lambda c} = f \cdot \frac{a}{c}, \qquad v_\infty = \lim_{\lambda \to \infty} v = f \cdot \frac{\lambda b}{\lambda c} = f \cdot \frac{b}{c} \tag{2.7}$$

Knowledge of the vanishing point of a sheaf of parallel lines (and of the focal length) allows for determining the unknown orientation of the lines. In particular, from:

$$c^2(u_\infty^2 + v_\infty^2) = f^2(1 - c^2) \tag{2.8}$$

it is possibile to calculate $a$, $b$ and $c$:

$$a = \frac{u_\infty}{\sqrt{u_\infty^2 + v_\infty^2 + f^2}}, \qquad b = \frac{v_\infty}{\sqrt{u_\infty^2 + v_\infty^2 + f^2}}, \qquad c = \frac{f}{\sqrt{u_\infty^2 + v_\infty^2 + f^2}} \tag{2.9}$$

Therefore, knowledge of the vanishing points of two orthogonal directions allows for determining camera orientation with respect to a scene plane:


From the vanishing point of the horizontal lines on the façade we get the unit vector, $\mathbf{i}_c$, which defines the orientation of such lines in the camera reference frame. Likewise, from the vanishing point of the vertical lines on the façade we get $\mathbf{j}_c$, the vector product between the two providing $\mathbf{k}_c$:

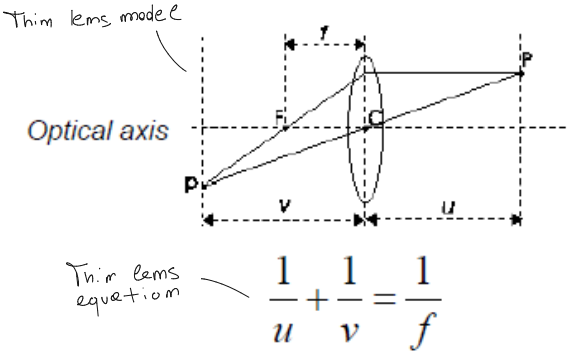
$$\mathbf{k}_c = \mathbf{i}_c \times \mathbf{j}_c \tag{2.10}$$

## 2.5 Weak Perspective

Perspective effects may be not so evident, this occurring whenever the framed subject is small compared to the distance from the camera. In such cases, perspective projection can be approximated by a scaled orthographic projection: $u = sx$, $v = sy$, where $s = f/z_0$ is the scaling factor.

## 2.6 Lenses

A scene point is on focus when all its light rays, gathered by the camera, hit the image plane at the same point. In a pinhole device this happens to all scene points because of the very small size of the hole, so that the camera features an infinite *Depth of Field* (DOF). The drawback is that such a small aperture allows gathering a very limited amount of light. Therefore, cameras rely on lenses to gather more light from a scene point and focus it on a single image point. However, the DOF is no longer infinite, for only points across a limited range of distances can be simultaneously on focus in a given image.
The following is an approximation of the more complex cameras' optical system:




*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

different from focal length of a pinhole camera.

In particular:

- Rays parallel to the optical axis are deflected to pass through $F$.
- Rays through $C$ are not deflected.

In the presented model, the distance $v$ is the effective focal length of the perspective projection, while $f$ is the camera focal length.

### 2.6.1 Circles of Confusion

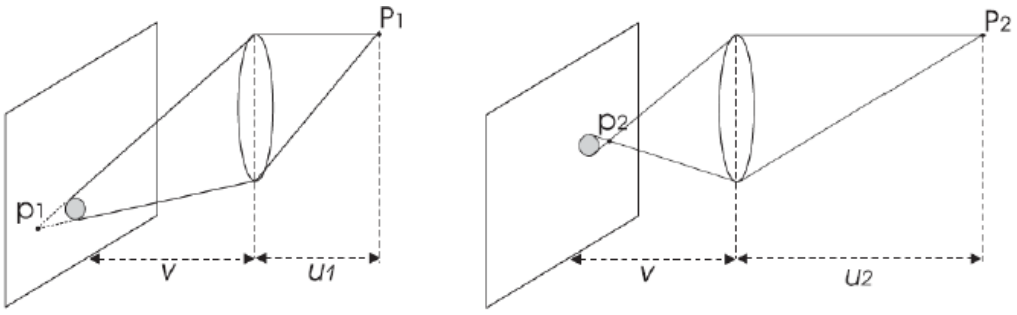
Due to the thin lens equation, choosing the distance of the image plane determines the distance at which scene points appear on focus in the image:

$$\frac{1}{u}+\frac{1}{v}=\frac{1}{f} \rightarrow u = \frac{vf}{v-f} \tag{2.11}$$

Likewise, to acquire scene points at a certain distance we must set the position of the image plane accordingly:

$$\frac{1}{u} + \frac{1}{v} = \frac{1}{f} \rightarrow v = \frac{uf}{u - f} \tag{2.12}$$

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 in the image as circles (Circles of Confusion):




As long as such circles are smaller than the size of the photo-sensing elements, the image will still look on-focus.

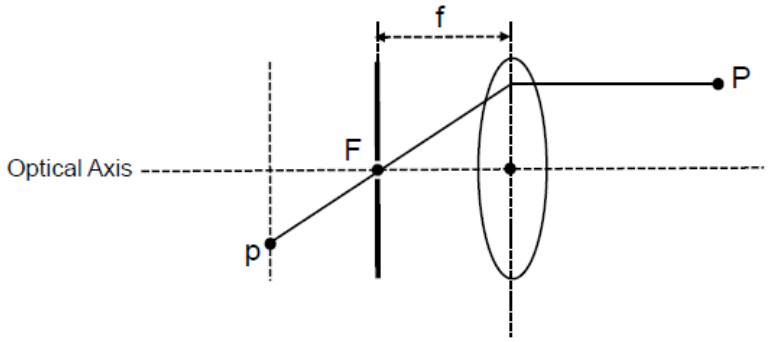
### 2.6.2 Focusing Mechanism

To focus on objects at diverse distances, another mechanism allow the lens to translate along the optical axis with respect to the fixed position of the image plane.

### 2.6.3 Telecentric Lens

the diaphragm allows to increase the DOF by reducing the amount of light which hits the length

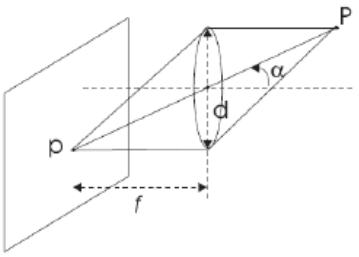
By placing a diaphragm with a small hole at the focal point of the lens we can block all light rays but those parallel to the optical axis, thereby realizing an ortographic projection. Telecentric lenses do not exhibit perspective distortion and thus are often used in machine vision to analyse, with high accuracy, 3D objects that might appear at different distances or off-axis.



## 2.7 Radiometric Relation

Another important relationship between scene points and image points involves irradiance $E$, i.e. the amount of light incident on a point, and radiance $L$, i.e. the amount of light emitted by a point in a direction. The fundamental radiometric relation shows that the irradiance is proportional to the radiance of the corresponding scene point:

$$E(p) = L(p) \cdot \frac{\pi}{4} \cdot \left(\frac{d}{f}\right)^2 \cdot \cos^4 \alpha$$




The radiance depends on the power and position of the light sources, as well as on the reflectance properties of the material. Such reflectance properties are usually described by a complex function called *Bi-Directional Reflectance Function*. Thus, it is possible to determine the amount of light emitted in a certain direction given the amount of light received from the sources.

## 2.8 Image Digitization

The image plane consists of a planar sensor which converts the irradiance into an electric quantity. These electrical quantities are put in a 2D $M \times N$ matrix, which is then sampled and quantized since the electrical quantities are continuous, and not discrete. In particular:

- **Sampling**: the planar continuous image is sampled evenly along the two axes.
- **Quantization**: the continuous values are quantized into $2^m$ discrete levels known as *grey levels*. Thus, $m$ is the number of bits used to represent a pixel. The total memory occupancy of a grey-scale image is $B = M \times N \times m$ [bit].

If the image is coloured, every point has three values, representing the RGB channels.

### 2.8.1 Digitization in Practice

The camera is composed of a first module called *Solid State Sensor*, which is a 2D array of photodetectors that convert incident light into a proportional electric charge, and of a second electronic module that eventually converts the analog signal to a digital signal.

### 2.8.2 Camera Parameters

- **Signal-to-Noise Ratio (SNR)**, which can be thought of as quantifying the strength of the "true" signal with respect to the unwanted fluctuations induced by noise (the higher, the better):

$$\text{SNR}_{dB} = 20 \cdot \log_{10}(\text{SNR}); \qquad \text{SNR}_{bit} = \log_2(\text{SNR}) \tag{2.13}$$

- **Dynamic Range (DR)**, which is the saturation irradiation (i.e. the amount of light that would fill up the capacity of a photodetector) to minimum detectable irradiation ratio (the higher, the better):

  └ This allows to capture both shadow parts and the highlighted parts of a sceme

$$\mathrm{DR} = \frac{E_{max}}{E_{min}} \tag{2.14}$$

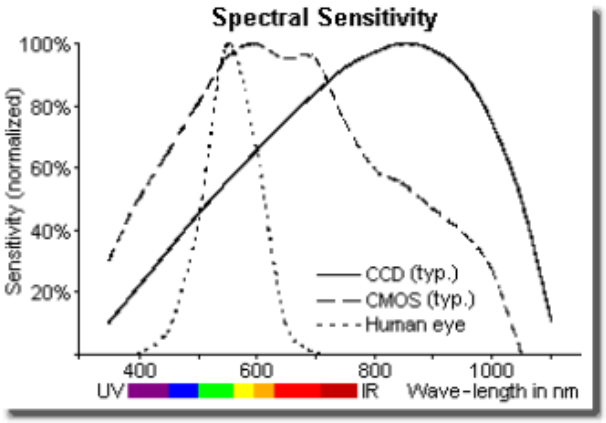
- **Sensitivity (Responsivity)**, which is related to the amount of signal that the sensor can deliver per unit of input optical energy.
- **Uniformity (spatial or pattern noise)**, which is related to the response to light of the various pixels.

### 2.8.3 CCD vs CMOS

- Unlike CCD, CMOS technology allows the electronic circuitry to be integrated within the same chip as the sensor (one chip camera): this provides compactness, less power consumption, lower costs.
- Unlike CCD, CMOS sensors allow an arbitrary window to be read out without having to receive the full image. This can be useful to inspect or track at a higher speed a small Region Of Interest (ROI) within the image.
- CCD technology typically provides higher SNR, higher DR and better uniformity.

### 2.8.4 Colour Sensors

CCD/CMOS sensors are able to sense intensity at a specific pixel by integrating over the range of wavelengths of the spectral distribution of the incoming light and multiplying this quantity by the spectral response function of the sensor. Hence, CCD/CMOS sensors can not sense colour.



In order to be able to sense colours, an array of optical filters is placed in front of the photodetectors, so as to render each pixel sensitive to a specific range of wavelengths. Some examples are listed below:

- Bayer CFA, which is a Colour Filter Array in which the number of green filters is twice the number of the red and blue ones, just to mimic the higher sensitivity of the human eye to the green colour. This implies a loss of resolution.
- A full resolution colour sensor can be achieved by deploying an optical prism to split the incoming light beam into three RGB beams sent to three distinct sensors equipped with corresponding filters.

## 2.9 Camera Calibration

Camera calibration is about measuring the important parameters of a camera. ~~Important~~, i.e. one wants to detect some quantitative measurements on the image. In order to simplify things, some mapping functions, like equations 2.2, need to be found. This time though, these new equations need to be linear in order to be more easily solvable. To do so, it is necessary to shift from 3D Euclidean space ($\mathbb{R}^3$) to projective space $\mathbb{P}^3$.
The generic vector $[x, y, z] \in \mathbb{R}^3$ becomes $[x, y, z, 1] \in \mathbb{P}^3$, i.e. a fourth dimension is added to the vector. Moreover, the new vector can be multiplied by a constant $k \neq 0$ and can still represent the same original point from $\mathbb{R}^3$. In general, if a $n$-dimensional projective space vector is given, this will just be represented as a vector of $n + 1$ elements: $[x_1, x_2, \ldots, x_{n+1}]$. To shift from $\mathbb{P}^3$ to $\mathbb{R}^3$ it is necessary to divide every element by $x_{n+1}$ (i.e. to get the $[\ldots, 1]$ representation) and than remove the last dimension.
The projective space allow the representation of points at infinity, i.e. when $x_{n+1} = 0$, while in $\mathbb{R}^3$ this is not possible.
As an example, considering the parametric equation of a 3D line (equation 2.5), if one wants to shift from $\mathbb{R}^3$ to $\mathbb{P}^3$ and multiply the resulting vector by $1/\lambda$, than one would obtain the following:

$$\underset{\in \mathbb{P}^3}{\tilde{\mathbf{M}}} = \begin{bmatrix} \underset{\in \mathbb{R}^3}{\mathbf{M}} \\ 1 \end{bmatrix} = \begin{bmatrix} x_0 + \lambda a \\ y_0 + \lambda b \\ z_0 + \lambda c \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} \frac{x_0}{\lambda} + a \\ \frac{y_0}{\lambda} + b \\ \frac{z_0}{\lambda} + c \\ \frac{1}{\lambda} \end{bmatrix} \quad \cdot \; k = \frac{1}{\lambda} \tag{2.15}$$

In this case, if $\lambda \to \infty$ than $x_{n+1} = 0$ obtaining a point at infinity. $\tilde{M}_\infty = \lim_{\lambda \to \infty} \tilde{M} = \begin{bmatrix} a \\ b \\ c \\ 0 \end{bmatrix}$
Let's denote a 3D point as $\tilde{\mathbf{M}}$ and the corresponding image point as $\tilde{\mathbf{m}}$:

$$\underset{\in \mathbb{P}^3}{\tilde{\mathbf{M}}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \qquad \underset{\in \mathbb{P}^2}{\tilde{\mathbf{m}}} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2.16}$$

Now it is possible to transform the perspective projection into a linear transformation by setting the constant $k$ equal to $z$:

$$\tilde{\mathbf{m}} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f\frac{x}{z} \\ f\frac{y}{z} \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} fx \\ fy \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \underbrace{\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}}_{=\tilde{M}} \tag{2.17}$$

In matrix notation:

$$k\tilde{\mathbf{m}} = \tilde{\mathbf{P}}\tilde{\mathbf{M}} \tag{2.18}$$

Mathematical model of the camera when mapping between projective spaces.

The matrix $\tilde{\mathbf{P}}$ is called *Perspective Projection Matrix* (PPM).
Let's assume that distances are measurable in focal length units, the PPM becomes:

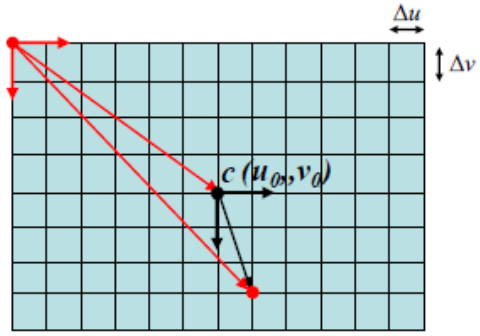
$$\tilde{\mathbf{P}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = [\mathbf{I}|\mathbf{0}] \tag{2.19}$$

↳ concatenation

This representation is called *Canonical Perspective Projection Matrix* or *Standard Perspective Projection Matrix*.
To come up with a really useful camera model it is necessary to take into account two additional issues:

- Image digitization.
- The six degrees-of-freedom (i.e. 3D rotation and translation) rigid motion between the camera reference frame (CRF) and the world reference frame (WRT).

Digitization can be accounted for by including into the projection equations the scaling factors along the two axis due to the quantization associated with the horizontal and vertical pixel size. Moreover, we need to model the translation of the piercing point (intersection between the optical axis and the image plane) with respect to the origin of the image coordinate system (top-left corner of the image):




$$u = \frac{f}{z}x \rightarrow u = \frac{1}{\Delta u}\frac{f}{z}x + u_0 = k_u\frac{f}{z}x + u_0 \tag{2.20}$$

$$v = \frac{f}{z}y \rightarrow v = \frac{1}{\Delta v}\frac{f}{z}y + v_0 = k_v\frac{f}{z}y + v_0 \tag{2.21}$$

The PPM can be written as:

$$\tilde{\mathbf{P}} = \begin{bmatrix} fk_u & 0 & u_0 & 0 \\ 0 & fk_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} fk_u & 0 & u_0 \\ 0 & fk_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \mathbf{A}[\mathbf{I}|\mathbf{0}] \tag{2.22}$$

The matrix **A** models the characteristics of the image sensing device, and is called *Intrinsic Parameter Matrix*. Sometimes $fk_u$ is denoted as $\alpha_u$ and $fk_v$ is denoted as $\alpha_v$.
3D coordinates are measured into a world reference frame (WRF) external to the camera, and not into the camera reference frame (CRF). The WRF will be related to the CRF by:

- A rotation around the optical centre, which is expressed by a $3 \times 3$ matrix (**R**).
- A translation, which is expressed by a $3 \times 1$ vector (**T**).

Therefore, the relation between the coordinates of a point in the two reference frames is:

$$\mathbf{W} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \quad \mathbf{M} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \rightarrow \quad \mathbf{M} = \mathbf{RW} + \mathbf{T} \tag{2.23}$$

where **R** is an orthogonal matrix. This equation can be rewritten in homogeneous coordinates ($\in \mathbb{P}^3$):

$$\tilde{\mathbf{W}} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \quad \tilde{\mathbf{M}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \rightarrow \quad \tilde{\mathbf{M}} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0} & 1 \end{bmatrix} \tilde{\mathbf{W}} = \mathbf{G}\tilde{\mathbf{W}} \tag{2.24}$$

where **R** is a $3 \times 3$ matrix (in which only three independent parameters, the ones related to the three angles, are unknown), **T** is a $3 \times 1$ matrix, **0** is a $1 \times 3$ vector, 1 is a $1 \times 1$ vector and **G** is a $4 \times 4$ matrix. Therefore, taking into consideration both digitization and the six DOF rigid motion:

$$k\tilde{\mathbf{m}} = \mathbf{A}[\mathbf{I}|\mathbf{0}]\mathbf{G}\tilde{\mathbf{W}} \tag{2.25}$$

Accordingly, the general form of the PPM can be expressed as follows:

$$\tilde{\mathbf{P}} = \mathbf{A}[\mathbf{I}|\mathbf{0}]\mathbf{G} \quad \text{or} \quad \tilde{\mathbf{P}} = \mathbf{A}[\mathbf{R}|\mathbf{T}] \tag{2.26}$$

### 2.9.1 Lens Distortion

Lens distort images because they are not ideal. If they were ideal, they would obey perspective projection. In particular, there exist two types of distortions:

- **Radial distortion**, which is related to the lens curvature.
- **Tangential distortion**, which is related to the misalignment of optical components and/or defects.

In order to take into account lens distortion, it is necessary to model this distortion:

$$x' = L(r)\tilde{x} + d\tilde{x} \tag{2.27}$$
$$y' = L(r)\tilde{y} + d\tilde{y} \tag{2.28}$$

where $x'$ and $y'$ are the distorted coordinates and $\tilde{x}$, $\tilde{y}$ are the undistorted ones.

In particular, the radius is given by:

$$r = \sqrt{(\tilde{x} - \tilde{x}_c)^2 + (\tilde{y} - \tilde{y}_c)^2} \tag{2.29}$$

and $L(r)$ is equal to:

$$L(r) = 1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + \dots \tag{2.30}$$

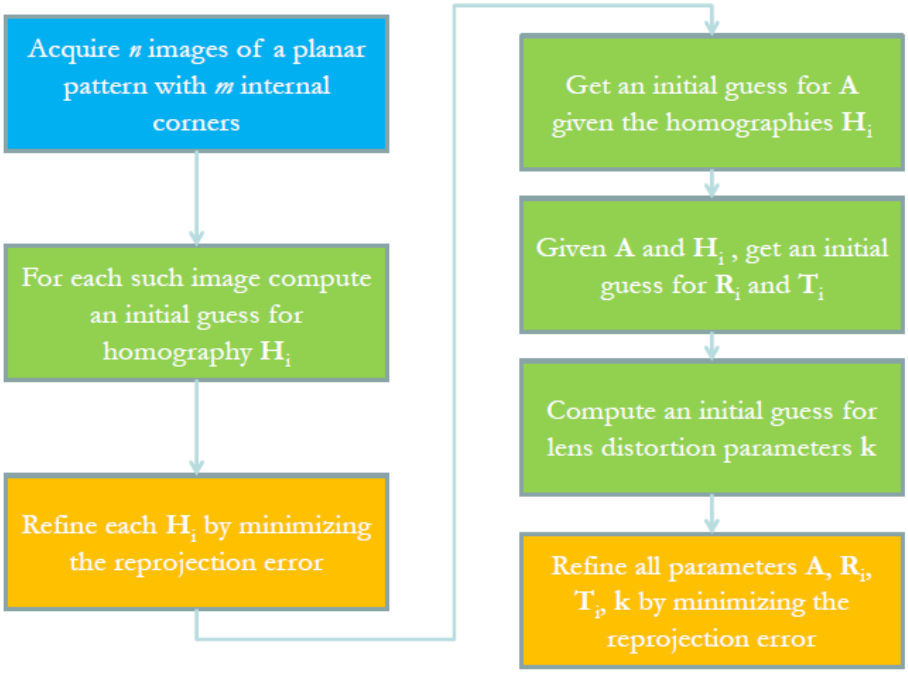
The tangential distortion factors, $d\tilde{x}$ and $d\tilde{y}$ are approximated as:

$$d\tilde{x} = 2p_1\tilde{x}\tilde{y} + p_2(r^2 + 2\tilde{x}^2) \tag{2.31}$$

$$d\tilde{y} = 2p_2\tilde{x}\tilde{y} + p_1(r^2 + 2\tilde{y}^2) \tag{2.32}$$

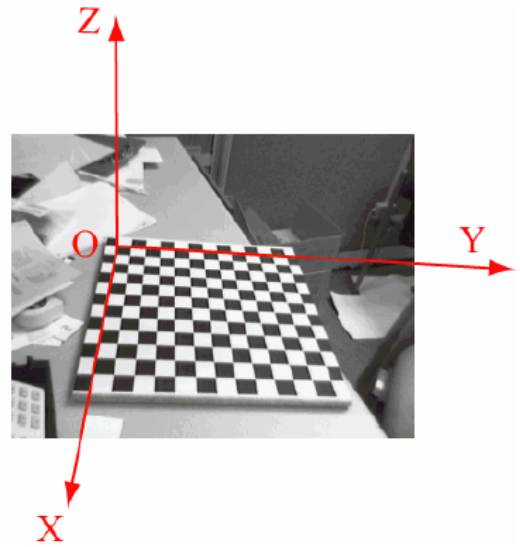
### 2.9.2 Calibration Process

Camera calibration is about finding intrinsics ($\mathbf{A}$), extrinsics ($\mathbf{R}$, $\mathbf{T}$) and lens distortion parameters ($k_1, k_2, \dots$ and $p_1, p_2$). In more mathematical terms, given that $\tilde{\mathbf{m}} = \tilde{\mathbf{P}}(\mathbf{A}, \mathbf{R}, \mathbf{T})\tilde{\mathbf{M}}$, where $\tilde{\mathbf{m}}$ are pixel coordinates and $\tilde{\mathbf{M}}$ are world coordinates, it is necessary to find the lens distortion parameters and $\tilde{\mathbf{P}}$. The basic process relies always on setting up a linear system of equations, given a set of known 3D-2D correspondences, so as to solve for the unknown camera parameters. In order to so it is necessary to use calibration targets, i.e. objects for which the real world coordinates are known, which have a set of control points. In particular, Zhang's method is one of the most used methods to estimate the unknown parameters:



For each of the $n$ images (e.g. chessboard images), a global 3D WRF is taken at the top-left corner of the pattern, with plane $Z = 0$ given by the pattern itself and the $X, Y$ axes aligned to the two orthogonal directions, so as to keep always the same association between axes and directions (i.e. $X =$ rows, $Y =$ columns). Thus, for a specific 3D point:

- The third coordinate is always 0. (i.e. $Z = 0$)
- $X, Y$ are determined by the known size of the squares forming the chessboard.




also called "control points"

The world coordinates ($\tilde{\mathbf{M}}$) of the $m$ internal corners are then immediately found. The corresponding pixel coordinates ($\tilde{\mathbf{m}}$) can be found using well-known algorithms (e.g. Harris algorithm). At this point, the correspondence between pixel coordinates and world coordinates is given by:

known with Harris — known world coordinates — PPH — homogeneous world coordinates — $p_{i,j}$ is a parameter of H

$$k\tilde{\mathbf{m}} = \tilde{\mathbf{P}}\tilde{\mathbf{w}} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,4} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{H}\tilde{\mathbf{w}}' \tag{2.33}$$

($\tilde{P}$, $\tilde{w}$, $H$, $\tilde{w}'$)

In particular, $\tilde{\mathbf{w}}$ and $\tilde{\mathbf{w}}'$ are world coordinates vector and the transformation $\mathbf{H}$ is called *homography* and represents a general linear transformation between planes.
Being $m$ the number of internal corners of a single image, one can set up $m$ systems of three linear equations as in 2.33.
To estimate $H_i$, where $i$ is the index of the considered image, the *Direct Linear Transform* algorithm (DLT) can be used:

in the Euclidian space. If I consider $\tilde{m}$ and $\tilde{w}$ in $\mathbb{R}^3$ instead of in $\mathbb{P}^2$, then $\tilde{m}$ and $\tilde{w}$ are parallel

$$k\tilde{\mathbf{m}} = \mathbf{H}\tilde{\mathbf{w}}' \xrightarrow[\text{since } \mathbf{H} \text{ is constant}]{\tilde{\mathbf{m}} \text{ and } \tilde{\mathbf{w}}' \text{ are parallel}} \tilde{\mathbf{m}} \times \mathbf{H}\tilde{\mathbf{w}}' = 0 \tag{2.34}$$

Being that:

$$\mathbf{H}\tilde{\mathbf{w}}' = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tilde{\mathbf{w}}' = \begin{bmatrix} \mathbf{h_1^T} \\ \mathbf{h_2^T} \\ \mathbf{h_3^T} \end{bmatrix} \tilde{\mathbf{w}}' = \begin{bmatrix} \mathbf{h_1^T}\tilde{\mathbf{w}}' \\ \mathbf{h_2^T}\tilde{\mathbf{w}}' \\ \mathbf{h_3^T}\tilde{\mathbf{w}}' \end{bmatrix} \tag{2.35}$$

after computing the ~~dot~~ product between $\tilde{\mathbf{m}}$ and the matrix of equation 2.35, the following result is obtained:

vector — algebraic manipulation

$$\tilde{\mathbf{m}} \times \mathbf{H}\tilde{\mathbf{w}}' = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \times \begin{bmatrix} \mathbf{h_1^T}\tilde{\mathbf{w}}' \\ \mathbf{h_2^T}\tilde{\mathbf{w}}' \\ \mathbf{h_3^T}\tilde{\mathbf{w}}' \end{bmatrix} = \begin{bmatrix} v\mathbf{h_3^T}\tilde{\mathbf{w}}' - \mathbf{h_2^T}\tilde{\mathbf{w}}' \\ \mathbf{h_1^T}\tilde{\mathbf{w}}' - u\mathbf{h_3^T}\tilde{\mathbf{w}}' \\ u\mathbf{u_2^T}\tilde{\mathbf{w}}' - v\mathbf{h_1^T}\tilde{\mathbf{w}}' \end{bmatrix} = \begin{bmatrix} \mathbf{0^T} & -\tilde{\mathbf{w}}'^{\mathbf{T}} & v\tilde{\mathbf{w}}'^{\mathbf{T}} \\ \tilde{\mathbf{w}}'^{\mathbf{T}} & \mathbf{0^T} & -u\tilde{\mathbf{w}}'^{\mathbf{T}} \\ -v\tilde{\mathbf{w}}'^{\mathbf{T}} & u\tilde{\mathbf{w}}'^{\mathbf{T}} & \mathbf{0^T} \end{bmatrix} \begin{bmatrix} \mathbf{h_1} \\ \mathbf{h_2} \\ \mathbf{h_3} \end{bmatrix} = \mathbf{A}\mathbf{h} = 0 \tag{2.36}$$

In particular $A$ is a coefficient matrix and $h$ is the unknown matrix. The result of 2.36 is a system of three equation, in which only two of these are linearly independent, in nine unknowns. Being that the considered image contains $m$ internal corners, and not just one, one can set up a system of $2m$ equations in nine unknowns. Moreover, being the homography a projective transformation, it is possible to scale the homography as to obtain one element of $h$ equal to 1. In this case, the number of unknowns is equal to just eight. One can only solve the system in a least squares sense, it's an overdetermined system!

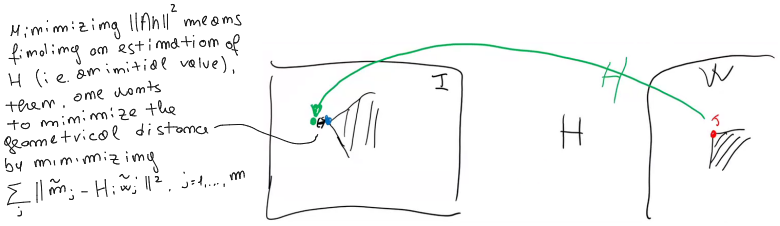
algebraic error

The problem is now formulated as finding a homography such that $\mathbf{h}^* = \text{argmin}(\overbrace{\|\mathbf{A}\mathbf{h}\|^2}^{\text{algebraic error}})$, which is the best approximation. The solution of this estimation problem can be obtained by computing the Singular Value Decomposition of $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathbf{T}$ with $\mathbf{A} \in \mathbb{R}^{2m\times 9}$, $\mathbf{U} \in \mathbb{R}^{2m\times 2m}$, $\mathbf{\Sigma} \in \mathbb{R}^{2m\times 9}$ and $\mathbf{V} \in \mathbb{R}^{9\times 9}$. It turns out that $\mathbf{h}^*$ is the last column of $\mathbf{V}$, $\mathbf{v_9}$, being $\mathbf{V}$ composed of the left singular vectors $[\mathbf{v_1}, \ldots, \mathbf{v_9}]$.

After finding this first estimation of the homography, it is necessary to apply an iterative numerical method (e.g. the Levenberg-Marquardt algorithm) in order to minimize the geometrical error:

geometrical error

$$\min_{\mathbf{H_i}} \sum_j \overbrace{\|\tilde{\mathbf{m}}_j - \mathbf{H}_i\tilde{\mathbf{w}}'_j\|^2}^{\text{geometrical error}}, \quad j = 1, 2, \ldots, m \tag{2.37}$$




Minimizing $\|f(h)\|^2$ means finding an estimation of H (i.e. an initial value), then one wants to minimize the geometrical distance by minimizing $\sum_j \|\tilde{m}_j - H_i\tilde{w}_j\|^2$, $j=1,\ldots,m$

Since $P = A[R\,|\,T]$ (extrinsics)

Intrinsics shared by every homography

At this point, it is possible to estimate $n$ homographies which share the same intrinsic parameters. In particular, it is possible to establish the following relation between $\mathbf{H_i}$ and the PPM:

$$\mathbf{H} = \begin{bmatrix} \mathbf{h_1} & \mathbf{h_2} & \mathbf{h_3} \end{bmatrix} = \begin{bmatrix} \mathbf{p_1} & \mathbf{p_2} & \mathbf{p_4} \end{bmatrix} = \lambda\mathbf{A}\begin{bmatrix} \mathbf{r_1} & \mathbf{r_2} & \mathbf{T} \end{bmatrix} \tag{2.38}$$

Since $\mathbf{r_1}$ and $\mathbf{r_2}$ are vectors of a rotation matrix, they are a base of considered space and so they are orthonormal vectors, i.e. they are perpendicular and they obey to the following relations:

re-written as $A^{-T}$

$r_1^T r_2 = 0 \Rightarrow \frac{1}{\lambda^2}\left(A^{-1}h_1\right)^T A^{-1} h_2 = h_1^T \left(A^{-1}\right)^T A^{-1} h_2 = 0$

$1 = r_1^T r_1 = r_2^T r_2 \Rightarrow h_1^T A^{-T} A^{-1} h_1 = h_2 A^{-T} A^{-1} h_2$

$\equiv$

$\|r_1\|_2 = \|r_2\|_2 = 1$

$r_1$ and $r_2$ are orthogonal

$r_1$ and $r_2$ are unit vectors

$$\mathbf{r_1^T r_2} = 0 \quad \Rightarrow \quad \mathbf{h_1^T A^{-T} A^{-1} h_2} = 0 \tag{2.39}$$

This system allows one to get rid of the extrinsics (rotation and traslation)

$$1 = \mathbf{r_1^T r_1} = \mathbf{r_2^T r_2} \quad \Rightarrow \quad \mathbf{h_1^T A^{-T} A^{-1} h_1} = \mathbf{h_2^T A^{-T} A^{-1} h_2} \tag{2.40}$$

B is symmetric since A is upper triangular

At this point one has $2n$ equations in 6 unknowns ($\mathbf{B} = \mathbf{A^{-T}A^{-1}}$ elements are the unknowns). The system is then solvable, in closed form, using at least three calibration images ($n = 3$), but it is always preferable to use every possible equation and solve the system in a least squares sense. Once $\mathbf{B}$ has been calculated, the intrinsics parameters (i.e. $\mathbf{A}$) can be obtained in closed form. (i.e. if one selects only three images)
Once $\mathbf{A}$ has been estimated, for each image it is possible to compute $\mathbf{R}$ and then $\mathbf{T}$ given the previously computed homography $\mathbf{H}$: — This is done for each H:

$$\mathbf{H} = \begin{bmatrix}\mathbf{h_1} & \mathbf{h_2} & \mathbf{h_3}\end{bmatrix} = \lambda \mathbf{A} \begin{bmatrix}\mathbf{r_1} & \mathbf{r_2} & \mathbf{T}\end{bmatrix} \tag{2.41}$$

As $\mathbf{r_1}$ is a unit vectors, $\lambda = \|\mathbf{A^{-1}h_1}\|$. It is possible to calculate $\mathbf{r_1}$ and $\mathbf{r_2}$:

$$\mathbf{h_1} = \lambda \mathbf{A r_1} \quad \Rightarrow \quad \mathbf{r_1} = \frac{1}{\lambda}\mathbf{A^{-1}h_1} = \frac{A^{-1}h_1}{\|A^{-1}h_1\|} \tag{2.42}$$

$$\mathbf{h_2} = \lambda \mathbf{A r_2} \quad \Rightarrow \quad \mathbf{r_2} = \frac{1}{\lambda}\mathbf{A^{-1}h_2} = \frac{A^{-1}h_2}{\|A^{-1}h_2\|} \tag{2.43}$$

$\lambda$ ($r_2$ is not normalized correctly)

$$\tag{2.44}$$

Since $\lambda$ is calculated only once, using $\mathbf{r_1}$, it does not perfectly normalize $\mathbf{r_2}$ and the rotation matrix, $\mathbf{R}$, needs to be turned into an orthogonal matrix.
$\mathbf{r_3}$ can be derived by exploiting orthonormality:

$$\mathbf{r_3} = \mathbf{r_1} \times \mathbf{r_2} \tag{2.45}$$

Since $r_2$ and $r_3$ have not been normalized correctly (i.e. they are not unit vectors) algebraic methods can be used to compute the real rotation matrix R.

Lastly, $\mathbf{T}$ is computed as follows:

$$\mathbf{T} = \frac{1}{\lambda}\mathbf{A^{-1}h_3} \tag{2.46}$$

The lens distortion equations are the following:

considering only radial distortion

$$x' = L(r)\tilde{x} = (1 + k_1 r^2 + k_2 r^4)\tilde{x} \tag{2.47}$$

$$y' = L(r)\tilde{y} = (1 + k_1 r^2 + k_2 r^4)\tilde{y} \tag{2.48}$$

In particular, $k_1$ and $k_2$ are the unknowns. In order to find the unknowns, it is necessary to know the distort coordinates $(x', y')$ and the undistorted coordinates $(\tilde{x}, \tilde{y})$. The former are the coordinates of the corners (i.e. found by the Harris algorithm), while the latter are computed establishing, in the first place, the relationship between distorted $(u', v')$ and ideal $(\tilde{u}, \tilde{v})$ pixel coordinates. In order to pass from image coordinates to pixel coordinates, the matrix $\mathbf{A}$ is used:

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = A \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \rightarrow \begin{cases} x' = \dfrac{u' - u_0}{\alpha_u} \\ y' = \dfrac{v' - v_0}{\alpha_v} \end{cases} \qquad \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ 1 \end{bmatrix} = A \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ 1 \end{bmatrix} \rightarrow \begin{cases} \tilde{x} = \dfrac{\tilde{u} - u_0}{\alpha_u} \\ \tilde{y} = \dfrac{\tilde{v} - v_0}{\alpha_v} \end{cases}$$

From these equations it is possible to derive $u'$ and $v'$:

$$\begin{cases} u' = \tilde{u} + \left(k_1 r^2 + k_2 r^4\right)\left(\tilde{u} - u_0\right) \\ v' = \tilde{v} + \left(k_1 r^2 + k_2 r^4\right)\left(\tilde{v} - v_0\right) \end{cases}$$

In matrix form:

$$\begin{bmatrix} \left(\tilde{u} - u_0\right) r^2 & \left(\tilde{u} - u_0\right) r^4 \\ \left(\tilde{v} - v_0\right) r^2 & \left(\tilde{v} - v_0\right) r^4 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} u' - \tilde{u} \\ v' - \tilde{v} \end{bmatrix}$$

At this point, one has a system with $2mn$ equations in 2 unknowns solvable in a least squares sense. At this point, intrinsic ($\mathbf{A}$), extrinsic ($\mathbf{R}$, $\mathbf{T}$) and lens distortion ($K_1, k_2$) parameters are known. At this point, in order to minimize the geometric error between observed and predicted pixel positions in the image, a final non-linear refinement is needed (Levenberg-Marquardt algorithm):

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \|\mathbf{m}_{i,j} - \tilde{\mathbf{m}}(\mathbf{A}, \mathbf{k}, \mathbf{R}_i, \mathbf{T}_i, \mathbf{w}_j)\|^2 \tag{2.49}$$

### 2.9.3 From Pixels to 3D Coordinates

In the camera is calibrated, i.e. matrix $\mathbf{A}$ is known, and depth is also known (e.g. by using depth sensors), one can estimate the 3D coordinates of pixels in the CRF in order to obtain a so called *point cloud*.
Given an image coordinates, $p*$, and depth, $z$, one can calculate the corresponding camera points, $\mathbf{P}$:

$$\underbrace{\begin{bmatrix} \alpha_u x + u_0 z \\ \alpha_v y + v_0 z \\ z \end{bmatrix}}_{p*} = \underbrace{\begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x \\ y \\ z \end{bmatrix}}_{P} \Rightarrow \mathbf{p*} = \mathbf{AP} \Rightarrow \mathbf{P} = \mathbf{A^{-1}p*} \Rightarrow \mathbf{P} = z\mathbf{A^{-1}}\frac{\mathbf{p*}}{z} \tag{2.50}$$

Where:

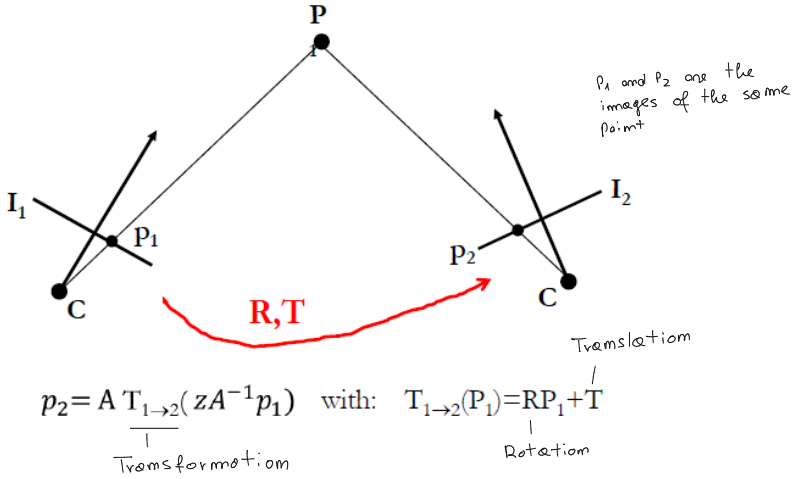
$$\frac{\mathbf{p*}}{z} = \begin{bmatrix} \alpha_u \frac{x}{z} + u_0 \\ \alpha_v \frac{y}{z} + v_0 \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{p} \tag{2.51}$$

Thus:

$$\mathbf{P} = z\mathbf{A}^{-1}\mathbf{p} \tag{2.52}$$

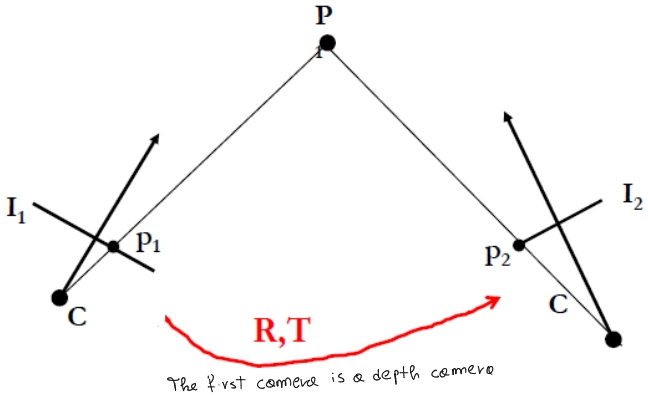
Knowing $z$, thanks to a depth sensor allows to reconstruct the 3D informations in the CRF. In the last years, machine learning has been used to predict depth, from a single image (depth from mono, in contrast with stereo theory).

If one wants to project an image point from image plane $I_1$ to another image plane, $I_2$, knowing the distance from the world point, $P$, and the vectors $\mathbf{R}, \mathbf{T}$ between the two positions of the same camera ($C$), one could apply the following equation:




$$p_2 = \mathrm{A}\, \mathrm{T}_{1\to 2}(zA^{-1}p_1) \quad \text{with:} \quad \mathrm{T}_{1\to 2}(\mathrm{P}_1) = \mathrm{RP}_1 + \mathrm{T}$$

One could also do the same things when the cameras are different:




$$p_2 = A_2\, \mathrm{T}_{1\to 2}(zA_1^{-1}p_1) \quad \text{with:} \quad \mathrm{T}_{1\to 2}(\mathrm{P}_1) = \mathrm{RP}_1 + \mathrm{T}$$

For example, in RGB-D cameras, the depth and the colours are given by different cameras, where the $\mathbf{R}$ and $\mathbf{T}$ vectors between the cameras are known.

Moreover:

- Any two images of a planar scene are related by a homography.
- Any two images taken by a camera rotating with respect to the optical centre are related by a homography.
- Any two images taken by different cameras (i.e. with different $\mathbf{A}$) in a fixed pose (i.e. with the same CRF) are related by a homography.
- Any two images taken by a camera rotating with respect to the optical centre and changing $\mathbf{A}$ are related by a homography.

This mean that there exist two homographies $\mathbf{H}_{12} = \mathbf{H}_1\mathbf{H}_2$ and $\mathbf{H}_{21} = \mathbf{H}_2\mathbf{H}_1$ such that $\tilde{\mathbf{m}}_1 = \mathbf{H}_{12}\tilde{\mathbf{m}}_2$ and $\tilde{\mathbf{m}}_1 = \mathbf{H}_{21}\tilde{\mathbf{m}}_2$.

### 2.9.4 Stereo Calibration

Given a stereo system, one is able to calibrate separately the two cameras. This would provide one with:

- Intrinsic parameters and lens distortion coefficients.
- Extrinsic parameters, i.e. the rigid motion between the CRF and a given WRF.

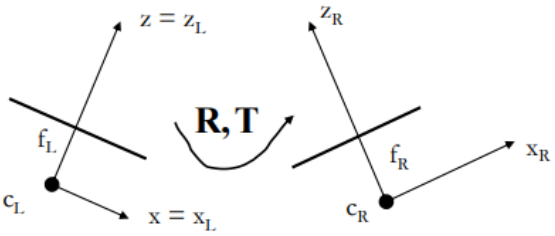
However, calibrating a stereo system requires also determining the rigid motion, $\mathbf{R}$ and $\mathbf{T}$, between the two cameras (i.e. between $\text{CRF}_L$ and $\text{CRF}_R$). In order to know such rigid motion information, one could start from an initial guess and refine such guess via numerical optimization. In particular:

- The initial guess is obtained by considering the same WRF (i.e. by showing to both cameras the same pattern). This allows one to compute the rigid motion between the two cameras, for each calibration image, by simple chaining of the transformations $\mathbf{G}_i(\mathbf{G}_j)^{-1}$, with either $i = L, j = R$ or $i = R, j = L$ and $\mathbf{G}$ being the extrinsic parameters matrix. In this way, $\mathbf{P}_L = \mathbf{G}_L\mathbf{P}_W$, $\mathbf{P}_R = \mathbf{G}_R\mathbf{P}_W$, thus $\mathbf{P}_W = (\mathbf{G}_L)^{-1}\mathbf{P}_L$ and $\mathbf{P}_W = (\mathbf{G}_R)^{-1}\mathbf{P}_R$. Therefore, $\mathbf{P}_R = \mathbf{G}_R\mathbf{G}_L\mathbf{P}_L$ and $\mathbf{P}_L = \mathbf{G}_L\mathbf{G}_R\mathbf{P}_R$. Once one has all the $\mathbf{R}_i$ and $\mathbf{T}_i$ matrices, then one can take the median of these matrices, selecting one $\mathbf{R}$ and one $\mathbf{T}$.
- Finally, the estimation is refined by a non-linear minimization of the projection error in both images of each pair by a numerical method:

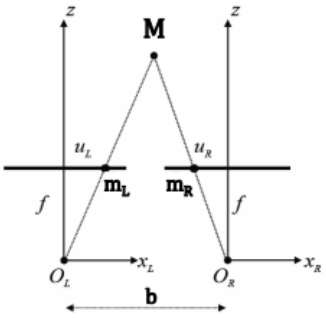
$$\sum_{k=L,R}\sum_{i=1}^{n}\sum_{j=1}^{m}\|\mathbf{m}_{ij}^{k} - \tilde{\mathbf{m}}(\mathbf{A}, \mathbf{k}, \mathbf{R}, \mathbf{T}, \mathbf{w}_j)\|^2 \tag{2.53}$$

Once the stereo system has been calibrated, it is often useful to define a reference frame attached to the whole stereo system, which is usually denoted as **Stereo Reference Frame** (SRF). This is achieved by setting such reference frame equal to the CRF of the left camera, such that:

$$\tilde{\mathbf{P}}_L = \mathbf{A}_L\left[\mathbf{I}|\mathbf{0}\right] \quad \tilde{\mathbf{P}}_R = \mathbf{A}_R\left[\mathbf{R}|\mathbf{T}\right]$$




It is well known that a standard stereo geometry is the most convenient choice to search for corresponding pixels:




$$M_L = M_R + \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix} \tag{2.54}$$

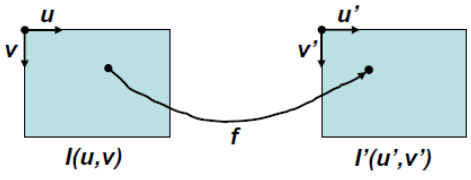
$$f_L = f_R = f \rightarrow A_L = A_R = A \tag{2.55}$$

Unfortunately, it is impossible to achieve such geometry by mechanical alignment. In order to solve this problem, once calibration has been performed, one can define two new cameras (i.e. two new PPMs) in standard geometry by virtually rotating the original ones about their optical centres and constraining the intrinsic parameters of the new cameras to be the same. Then, the images acquired by the original cameras can be **rectified** (i.e. pixels from the left camera images corresponds to pixels from the right camera).

└ rectification consists in changing intrinsics and the rotation matrix of both cameras. This means that rectification is obtained by applying homographies.

## 2.10 Image Warping

Image warping is the process of digitally manipulating an image. In particular, one could have some function, $f$, which maps coordinates of a first image into coordinates of a second image:




forward mapping

$$\begin{cases} u' = f_u(u,v) \\ v' = f_v(u,v) \end{cases}$$

copying the intensity

$$I'\left(f_u(u,v), f_v(u,v)\right) = I(u,v)$$

examples of image warping are:
- Removal of lens distortion
- Stereo rectification

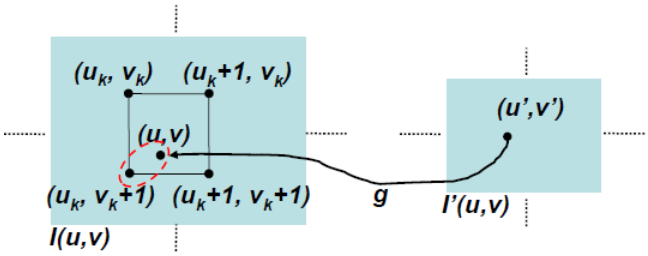
### 2.10.1 Forward and Backward Mapping

When doing forward mapping, it is not guaranteed that the output image gets filled correctly. This is due to the fact that the mapping function, $\mathbf{f}$, is usually a real function which outputs real numbers. In this case, these real numbers have to be transformed into integers in order to get the output pixel coordinates. This problem produces holes (the output image is not completely filled) and folds (a single pixel, in the output image, is selected multiple times due to real-to-integer transformation).
To solve the problem, a backward mapping is applied. In particular:

$$\begin{cases} u = g_u(u',v') \\ v = g_v(u',v') \end{cases} \Rightarrow \forall (u',v'):\ I'(u',v') = I\left(g_u(u',v'), g_v(u',v')\right)$$

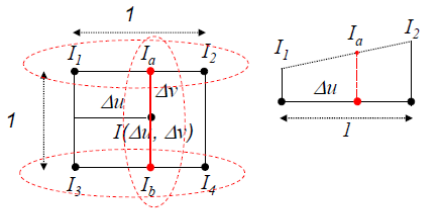
backward mapping

Since the backward mapping produces real numbers, in order to obtain the intensity of the pixel in the output image, $I(u', v')$, one could interpolate between the intensity of the four closest points of the input image.




After defining $\Delta u = u - u_k$ and $\Delta v = v - v_k$ and considering the intensity at the four closest points as $I_1 = I(u_k, v_k), I_2 = I(u_k + 1, v_k), I_3 = I(u_k, v_k + 1), I_4 = I(u_k + 1, v_k + 1)$, an example of bilinear interpolation is the following:

weighted sum







$$\frac{I_a - I_1}{\Delta u} = I_2 - I_1$$

$$I_a = (I_2 - I_1)\Delta u + I_1$$

$$I_b = (I_4 - I_3)\Delta u + I_3$$

$$I(\Delta u, \Delta v) = (I_b - I_a)\Delta v + I_a$$

$$I(\Delta u, \Delta v) = ((I_4 - I_3)\Delta u + I_3 - ((I_2 - I_1)\Delta u + I_1))\Delta v + (I_2 - I_1)\Delta u + I_1$$

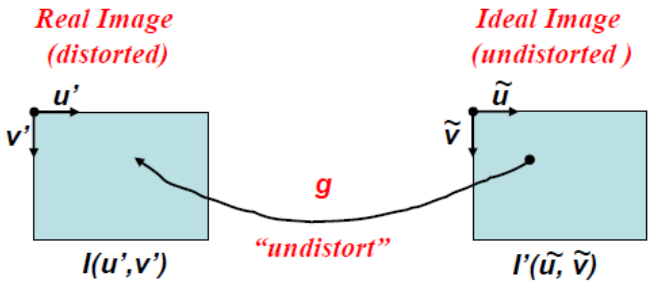
$$\Downarrow$$

$$I'(u',v') = (1 - \Delta u)(1 - \Delta v)I_1 + \Delta u(1 - \Delta v)I_2 + (1 - \Delta u)\Delta v I_3 + \Delta u \Delta v I_4$$

In particular, $I'(u', v')$ is calculated using a weighted sum of $I_1, I_2, I_3, I_4$.

### 2.10.2 Warping to Compensate Lens Distortion

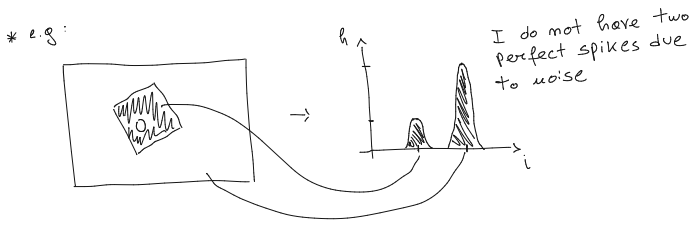
Warping can be used to even compensate lens distortion:




Once the lens distortion parameters, $k_1$ and $k_2$, have been computed by camera calibration, the image can be corrected by a backward warp from the undistorted image:

$$u' = \tilde{u} + (k_1 r^2 + k_2 r^4)(\tilde{u} - u_0) \tag{2.56}$$

$$v' = \tilde{v} + (k_1 r^2 + k_2 r^4)(\tilde{v} - v_0) \tag{2.57}$$




# Chapter 3

# Intensity Transformations

Intensity transformations are image processing operators which allow the improvement of an image so that informations can be extracted more easily from these images, using computer vision methods.

## 3.1 The Grey-Level Histogram

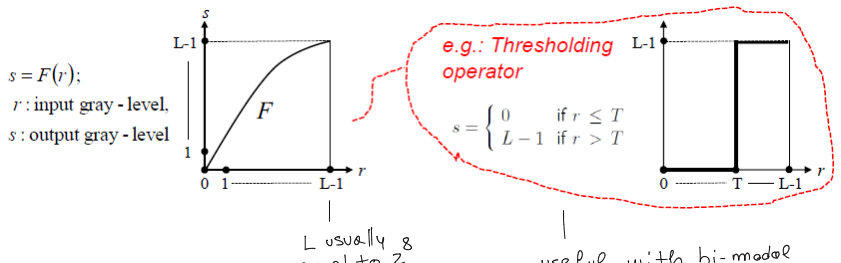
Most of these operators rely on the computation of the grey-level histogram (intensity histogram) of the input image. The grey level histogram of an image is simply a function associating to each grey level the number of pixels in the image which take that particular level:

$$\mathbf{p} = \begin{bmatrix} u \\ v \end{bmatrix}, \quad \forall \mathbf{p} \in \mathbf{I}: \quad \mathbf{H}[\mathbf{I}[\mathbf{p}]] + + \tag{3.1}$$

The histogram provides useful information on image content but not dot include any information related to the spatial distribution of intensities.
Normalization of histogram entries by the total number of pixels yields relative frequencies of occurrence of grey levels, which can be interpreted as their probabilities. Accordingly, the normalized histogram can be thought of as the *pmf* (probability mass function) of the discrete random variable given by the grey level of a randomly picked pixel in the image.

## 3.2 Point Operator

A point operator is an image processing operator whereby the intensity of a pixel of the output image is computed based on the intensity of the corresponding pixel of the input image:

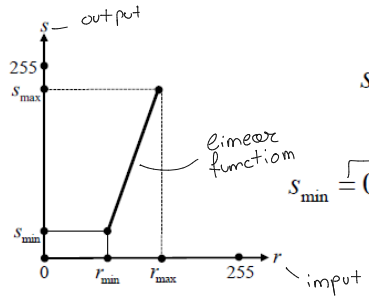


These operators can be implemented as a look-up table (LUT):

$$\mathbf{p} = \begin{bmatrix} u \\ v \end{bmatrix}, \quad \forall \mathbf{p} \in \mathbf{I}: \quad \mathbf{O}[\mathbf{p}] = \mathbf{LUT}[\mathbf{I}[\mathbf{p}]] \tag{3.2}$$

### 3.2.1 Linear Contrast Stretching

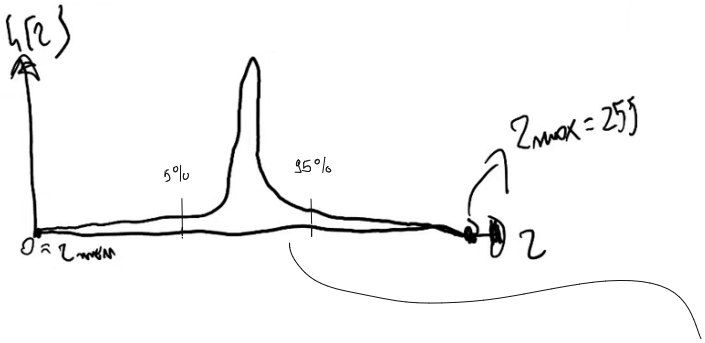
By using the linear contrast stretching method it is possible to enhance the contrast of an image which features a small grey-level range. In particular:




$$s = \frac{s_{\max} - s_{\min}}{r_{\max} - r_{\min}}\left(r - r_{\min}\right) + s_{\min}$$

$$s_{\min} = 0,\ s_{\max} = 255 \ \text{(if)} \rightarrow s = \frac{255}{r_{\max} - r_{\min}}\left(r - r_{\min}\right)$$

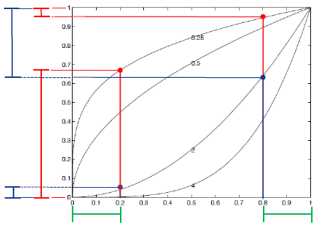
However, whenever $r_{min} = 0$ and $r_{max} = 255$ this method is not useful at all, since $s_{min}$ can not be less than 0 and $s_{max}$ can not be more than 255:




To avoid the problem one can arbitrary set $r_{min}$ and $r_{max}$, for example, at 5% and 95%. Everything below $r_{min}$ would then be set to 0 and everything above $r_{max}$ would then be set to 255. At this point one can apply the linear contrast stretching.

### 3.2.2 Exponential Operator

The exponential operator allows to selectively enhance the contrast in either dark or bright areas of the image:



$y = x^r$

$x \in [0,1], r = 0.25,\ 0.5,\ 2,\ 4$

Thus, $r < 1$ would stretch the intensity dynamics of dark areas and shrink that of bright ones. The opposite behaviour is achieved with $r > 1$.
As the grey levels of an image range within $[0, 255]$:

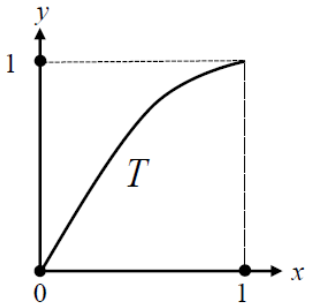
$$y \to \frac{y}{255}, \quad x \to \frac{x}{255} \quad \Rightarrow \quad \frac{y}{255} = \left(\frac{x}{255}\right)^r \to y = 255^{1-r}x^r \tag{3.3}$$

### 3.2.3 Histogram Equalization — applies a non-linear stretching

Histogram equalization is another contrast improvement operator which tries to spread uniformly pixel intensities across the whole available range. This method is used without having to tune any parameter and it is based on probability theory. In particular, histogram equalization transform the grey-level of the input image so that the histogram of the output image turns out flat.
From probability theory, given a continuous random variable, $x$, and a strictly monotonically increasing (i.e. invertible) function, $T$, such that:

this hypothesis will be dropped since one is processing on a discrete random variable

$$x \in [0,1] \to y = T(x) \in [0,1] \tag{3.4}$$




one wants to find a specific function $T$, such that the probability density function of $y$ is flat. If such a function exists than it is possible to spread the image histogram:

Denoting as $p_x(x)$ and $p_y(y)$ the probability density functions of $x$ and $y$ respectively, as $T$ is monotonically increasing:

$$\forall \tilde{x} \in [x, x+dx] \rightarrow \tilde{y} = T(\tilde{x}) \in [y, y+dy], \quad \text{with } y = T(x), y+dy = T(x+dx) \tag{3.5}$$

Therefore, the probability of $x$ and $y$ to belong to their infinitesimal intervals is exactly the same, which allows the derivation of the probability density function of $y$ as a function of $T$ and the probability density function of $x$:

extreme case of $\int_{y_1}^{y_2} p_y(y)\,dy$ — extreme case of $\int_{x_1}^{x_2} p_x(x)\,dx$

$$p_y(y)dy = p_x(x)dx \rightarrow p_y(y) = p_x(x)\frac{dx}{dy} \tag{3.6}$$

In particular, $p_x(x)dx$ and $p_y(y)dy$ are the probability of $x$ and $y$ to belong to their infinitesimal intervals, and $dx/dy$ is the derivative of the inverse function $x = T^{-1}(y)$.
At this point, $T$ is chosen to be the cumulative distribution function of $x$ which maps into $[0, 1]$ and, indeed, is monotonically increasing:

$$y = T(x) = \int_0^x p_x(\xi)d\xi \tag{3.7}$$

$\frac{dy}{dx} = p_x(x)$

Assuming that this function is strictly monotonically increasing, and since the derivative of $T$ (i.e. $dy/dx$) is equal to $p_x(x)$:

$p_x(x)\frac{1}{\frac{dy}{dx}} = p_x(x)$

$$p_y(y) = p_x(d)\frac{dx}{dy} = \cancel{p_x(x) = \frac{1}{\frac{dy}{dx}}} = \frac{p_x(x)}{p_x(x)} = 1 \tag{3.8}$$

In the end, turns out that, using the specific cumulative distribution function of $x$, the probability density function of $y$ is always 1.
At this point, it is necessary to discretize the previous result. In this case, the cumulative probability mass function of the discrete random variable associated with the grey level of a pixel:

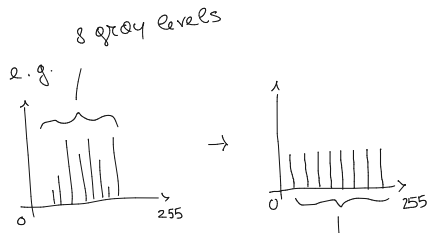
total number of pixels

$$\begin{cases} N = \sum_{i=0}^{L-1} h(i) \\ p(i) = \frac{h(i)}{N} \end{cases} \rightarrow j = T(i) = \sum_{k=0}^{i} p(k) = \frac{1}{N}\sum_{k=0}^{i} h(k) \rightarrow j = \frac{L-1}{N}\sum_{k=0}^{i} h(k)$$

to map $j$ in [0..L-1] instead of $j \in [0,1]$

proof of the image

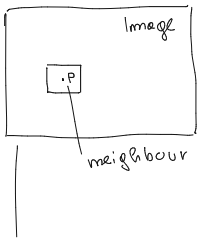
from 3.7 equation




8 gray levels however, one cannot have more gray levels than the original image, so it is impossible to obtain an ideal flat histogram. However, equalization is useful in order to make use of the whole gray level spectrum.

# Chapter 4

# Spatial Filtering




Spatial filters compute the new intensity of a pixel, $p$, based on the intensities of those belonging to a neighbourhood of $p$. These filters accomplish, for example, denoising and sharpening.

## 4.1 Linear Shift-Invariant Operators

Linear Shift-Invariant operators (LSI) applies 2D convolution between the input image and the *impulse response function* (*point spread function* or *kernel*) of the specific LSI operator.
Given an input 2D signal, $i(x, y)$ and two constants, $a, b$, a 2D operator, $T[\cdot]: o(x, y) = T[i(x, y)]$ is said to be linear if and only if:

$$T\{ai_1(x, y) + bi_2(x, y)\} = ao_1(x, y) + bo_2(x, y), \quad \text{with } o_1(\cdot) = T\{i_1(\cdot)\}, o_2(\cdot) = T\{i_2(\cdot)\} \tag{4.1}$$

The specific operator is said to be shift-invariant if and only if:

$$T\{i(x - x_0, y - y_0)\} = o(x - x_0, y - y_0) \tag{4.2}$$

Assuming $i(x, y) = \sum_k w_k e_k(x - x_k, y - y_k)$ and posing $h_k(\cdot) = T\{e_k(\cdot)\}$, it follows that:

$$\begin{aligned} o(x, y) &= T\left\{\sum_k w_k e_k(x - x_k, y - y_k)\right\} \\ &= \sum_k w_k T\{e_k(x - x_k, y - y_k)\} && \text{(Linearity)} \\ &= \sum_k w_k h_k(x - x_k, y - y_k) && \text{(Shift-invariance)} \end{aligned}$$

That is, if the input is a weighted sum of displaced elementary functions, the output is given by the same weighted sum of the displaced responses to the elementary functions.

## 4.2 Convolution

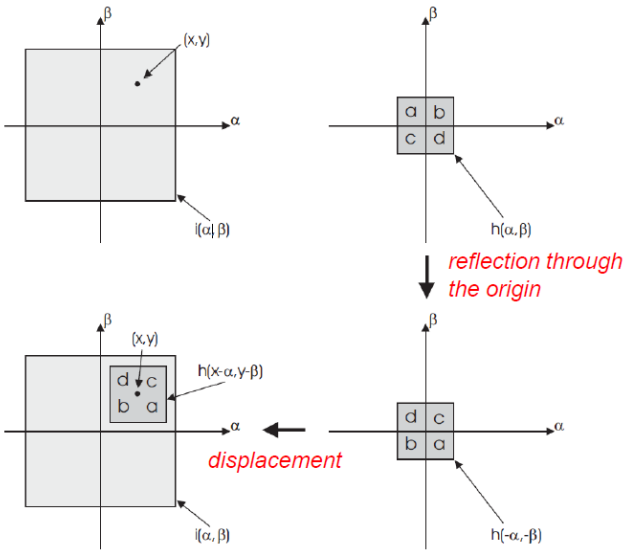
Any 2D signal can be expressed as a (infinite) weighted sum of displaced unit impulses (*Dirac delta function*, $\delta$):

$$i(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(\alpha,\beta)\delta(x-\alpha,y-\beta)d\alpha d\beta \tag{4.3}$$

Due to linearity and shift-invariance, the output signal can be expressed as the same (infinite) weighted sum of the displaced responses to the unit impulses:

$$o(x,y) = T\{i(x,y)\} = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(\alpha,\beta)h(x-\alpha,y-\beta)d\alpha d\beta \tag{4.4}$$

In particular, $h(x,y) = T\{\delta(x,y)\}$ is the impulse response (*point spread function* or *kernel*) of the operator. Equation 4.4 is called *continuous 2D convolution* and is denoted as $o(x,y) = i(x,y) * h(x,y)$.



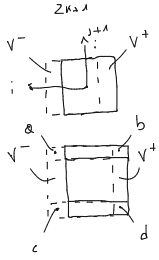

### 4.2.1 Discrete Convolution

Let $T\{\cdot\}$ be a discrete 2D LSI operator whose response to the 2D discrete unit impulse (*Kronecker delta function*) is denoted as $H(i,j)$:

$$H(i,j) = T\{\delta(i,j)\}, \quad \text{with} \quad \begin{cases}\delta(i,j) = 1 & \text{at } (0,0)\\ \delta(i,j) = 0 & \text{elsewhere}\end{cases} \tag{4.5}$$

Given a discrete 2D input signal, $I(i,j)$, the output signal, $O(i,j)$, is given by the *discrete 2D convolution* between $I(i,j)$ and $H(i,j)$:

Box filtering : $\mu(i,j) = \frac{\sum_{-k}^{k}\sum_{-k}^{k} I(i+m, j+m)}{(2k+1)^2} = \frac{s(i,j)}{(2k+1)^2}$ $O(k^2)$




$s(i,j+1) = s(i,j) + \underbrace{V^+(i,j+1) - V^-(i,j+1)}_{\Delta(i,j+1)}$ $O(k)$

$V^+(i-1,j+1) + a - b$ , $V^-(i-1,j+1) + c - d$ $\rightarrow$ $s(i,j+1) = s(i,j) + \Delta(i-1,j+1) + a - b - c + d$ $O(1)$

five sums per pixel independently from filter size

$$O(i,j) = T\{I(i,j)\} = \sum_{m=-\infty}^{+\infty}\sum_{n=-\infty}^{+\infty} I(m,n)H(i-m, j-n) \tag{4.6}$$

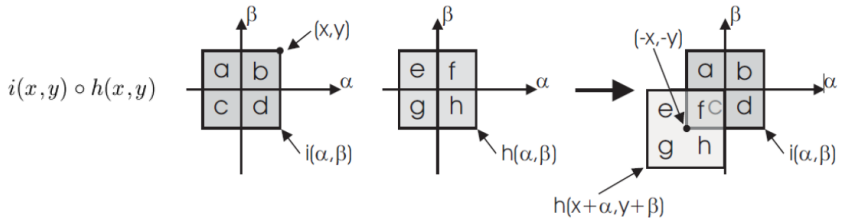
In particular, in image processing, bot the input signal (image) and the impulse response (kernel) are stored into matrices. Conceptually, in order to apply discrete convolution, it is necessary to slide the kernel across the whole image to compute the new intensity at each pixel. – im border point one needs to pad the image. Instead of padding one can also crop the image

## 4.3 Correlation

unlike convolution $h \circ i \neq i \circ h$

The correlation of signal $i(x,y)$ with signal $h(x,y)$ is defined as:

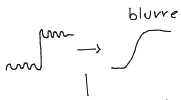
$$i(x,y) \circ h(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} i(\alpha,\beta)h(x+\alpha, y+\beta) \tag{4.7}$$




then, $i * h = h \circ i$

In particular, if $h$ is an even function, i.e. $h(x,y) = h(-x,-y)$, ~~correlation and convolution correspond~~.

## 4.4 Mean Filter




high frequencies are responsible for fast changes of intensity in the image

Mean filtering is the simplest way to carry out an image smoothing (i.e. low-pass filtering) operation. The aim of mean filtering is to de-noise an image, though sometimes the purpose is to cancel out small-sized unwanted details. This operator replaces each pixel's intensity by the average intensity over a given neighbourhood. In particular, this operator can be defined through a kernel, hence is a LSI operator. For example:

Instead of performing convolution one usually take the mean of the intensities of the neighbour. For this reason, mean filtering is really fast, especially if box-filtering is applied (i.e. incremental calculation scheme)

Kernel

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \end{bmatrix} = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

mean filtering allows to de-noise because noise can be modelled using a gaussian distribution with zero mean. → In order to remove the noise one should take a window as large as possible *

## 4.5 Gaussian Filter

A Gaussian filter is a LSI operator whose impulse response is a 2D Gaussian function (with zero mean and constant diagonal covariance matrix).

However by taking a large window one usually considers pixels of different intensities. This causes blurring.

* the mean of the sample should approximate the true mean (zero) of the noise distribution

* Since $G(x,y) = G(x)G(y)$, in order to speed-up the computation one can compute the chain of two 1D convolutions:

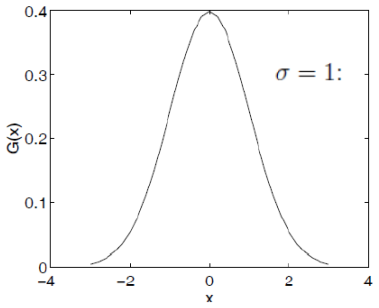
$$I(x,y) * G(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(\alpha,\beta)\, G(x-\alpha)\, G(y-\beta)\, d\alpha\, d\beta$$

$$= \int_{-\infty}^{\infty} G(y-\beta) \left( \int_{-\infty}^{\infty} I(\alpha,\beta)\, G(x-\alpha)\, d\alpha \right) d\beta$$

or viceversa → from $O(k^2)$ to $O(k)$ (always slower than mean filter but more accurate)

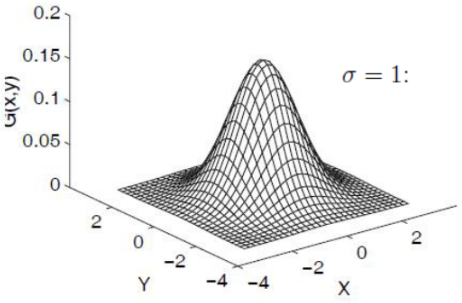
**1D Gaussian**

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

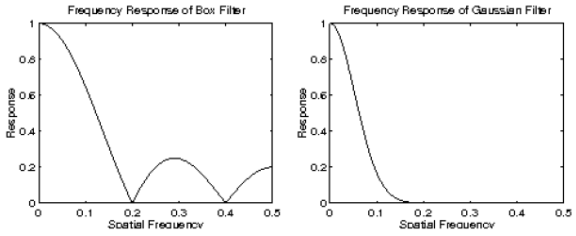



**2D Gaussian**

$$G(x,y) = G(x)G(y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



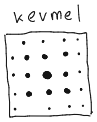

In particular, this filter is used to smooth the image; the higher $\sigma$, the stronger the smoothing caused by the filter. As a matter of fact, the Fourier transform of a Gaussian is a Gaussian with $\sigma_\omega = 1/\sigma$, i.e. the higher $\sigma$, the narrower the bandwidth of the filter. In general, a Gaussian filter is a more effective low-pass operator than a mean filter:




mean filter — Gaussian filter

Gaussian filters give more weights to pixels near the selected one → Pixels near the selected one should belong to the same object (i.e. should have the same intensity)

kernel



In order to obtain a discrete Gaussian kernel, it is possible to sample the corresponding continuous signal, which, however, is of infinite extent. To obtain a finite signal, one can consider only the interval $[-3\sigma, 3\sigma]$, which captures the 99% of the area (i.e. the energy) of the continuous Gaussian. Therefore, the kernel matrix would be of size $(2k+1) \times (2k+1)$ with $k = \lceil 3\sigma \rceil$. e.g. $\sigma = 2$ ⇒ $13 \times 13$ kernel

✳

## 4.6 Median Filter

The median filter is a non-linear filter whereby each pixel's intensity is replaced by the median over a given neighbourhood:

This filter counteracts impulse noise (noise which introduces black and white pixels randomly in space) effectively. As a matter of fact, noisy pixels tend to fall at either the top or bottom end of the sorted intensities. Moreover, median filtering tends to keep sharper edges than linear filters.

Median filters do not cause blurring. → it is able to handle outliers

| | | | | |
|---|---|---|---|---|
| 123 | 125 | 126 | 130 | 140 |
| 122 | 124 | 126 | 127 | 135 |
| 118 | 120 | 150 | 125 | 134 |
| 119 | 115 | 119 | 123 | 133 |
| 111 | 116 | 110 | 120 | 130 |

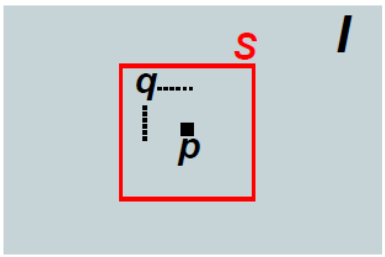
Neighbourhood values:

115, 119, 120, 123, 124, 125, 126, 127, 150

Median value: 124

## 4.7 Bilateral Filter

The bilateral filter is an advanced non-linear filter which is used to de-noise Gaussian-like noise without blurring the image (i.e. preserving smoothing). In particular, considering the following scenario:




Avoids blurring by considering spatial relationship between pixels and intensity of pixels. Higher weights will be given to similar pixels to the one considered, and to less distant pixels to the one considered.

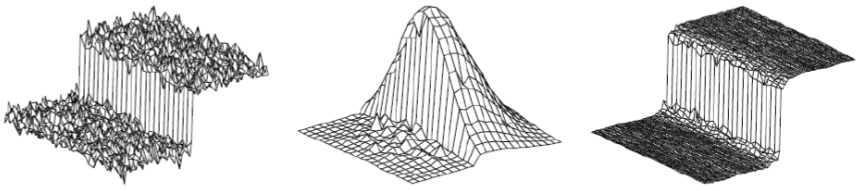
the output of the filter is given by:

Gaussian — standard Gaussian filter does not consider the $G_{\sigma_r}$ term

$$O(p) = \sum_{q \in S} I_q H(p,q), \quad \text{with} \quad H(p,q) = \frac{1}{W(p)} G_{\sigma_s}(d_s(p,q)) G_{\sigma_r}(d_r(I_p, I_q)) \tag{4.8}$$

In particular: $d_s(p,q) = \|p-q\|_2 = \sqrt{(u_p-u_q)^2+(v_p-v_q)^2}$ is the spatial distance between $p$ and $q$; $d_r(I_p, I_q) = |I_p - I_q|$ is the intensity distance between $p$ and $q$; $W(p) = \sum_{q \in S} G_{\sigma_s}(d_s(p,q)) G_{\sigma_r}(d_r(I_p, I_q))$ is a normalization factor.

***Step-edge as wide as 100 gray-levels*** — ***H(p,q) at a pixel just across the edge in the brighter region*** — ***Output provided by the filter*** $\sigma_s = 5, \sigma_r = 50$

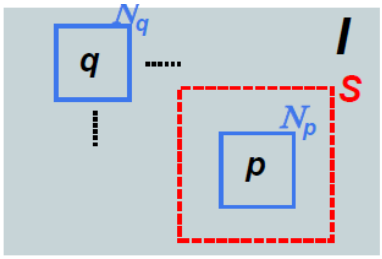


(a) (b) (c)

Due to the mathematical nature of the filter, neighbouring pixels take a larger weight as they are both closer and more similar to the central pixel.

## 4.8 Non-Local Mean Filter

The non-local mean filter is a more recent edge preserving smoothing filter. The key idea is that the similarity among patches spread over the image can be deployed to achieve de-noising. In particular, considering the following scenario:




the output of the filter is given by:

every pixel gets weighted — weight — normalizer — parameter

$$O(p) = \sum_{q \in I} I(q) w(p,q), \quad \text{with} \quad w(p,q) = \frac{1}{Z(p)} e^{-\frac{\|N_p - N_q\|_2^2}{h^2}}, \quad Z(p) = \sum_{q \in I} \bar{e}^{\frac{\|N_p - N_q\|_2^2}{h^2}} \tag{4.9}$$

To complexity of this operator is $\mathbb{O}(N)^2$. To simplify the filtering process, instead of considering the entire image (i.e. $q \in I$), it is possible to consider a smaller fraction of the image, $S$, centred in $p$ (i.e. $q \in S$).

# Chapter 5

# Image Segmentation

Image segmentation is a computer vision topic, since images are represented using a symbolic notation more prone to further processing.
Denoted as $P(x, y)$ a vector-valued function encoding a set of image properties, the goal of segmentation is to partition the image into disjoint homogeneous regions according to $P$. In particular, a good segmentation should:

- preserve spatial proximity (i.e. two nearby pixels must belong to the same region unless they exhibit significantly different $P$ values);
- provide relatively large regions with smooth boundaries.

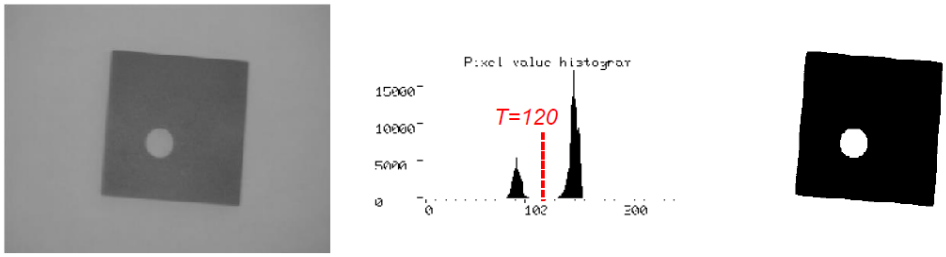
Segmentation brings in key *semantic knowledge* on the scene, as it splits the image into semantically meaningful parts (e.g. foreground and background, individual objects, moving and static pixels, ...).
In many computer vision / machine vision tasks, one usually works with *inherently binary images*, which are images that can be segmented into two different regions.

## 5.1 Image Binarization

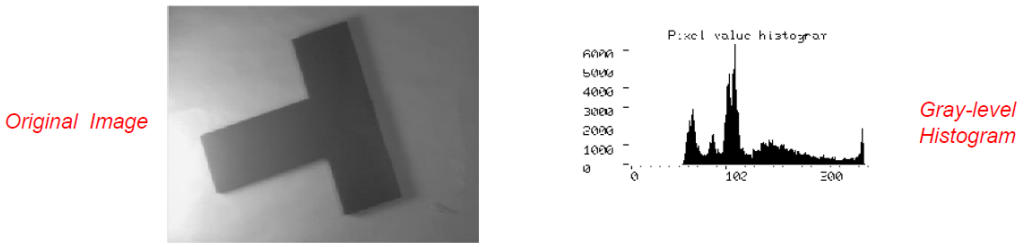
In many applications, the objects of interest (foreground) are neatly darker or brighter than the irrelevant areas of the scene (background). This can be achieved by *backlighting*. In such circumstances, it is possible to apply image segmentation, creating two disjoint regions corresponding to foreground and background. The foreground region can be further split into sub-regions corresponding to individual objects (*image labelling / connected components labelling*).

### 5.1.1 Binarization by Intensity Thresholding

Inherently binary images exhibit a clearly bimodal grey-level histogram, with two well-separated peaks corresponding to foreground and background pixels. In this case, binarization can be achieved by means of a thresholding operator. In the example below, $T = 120$ is the chosen threshold:



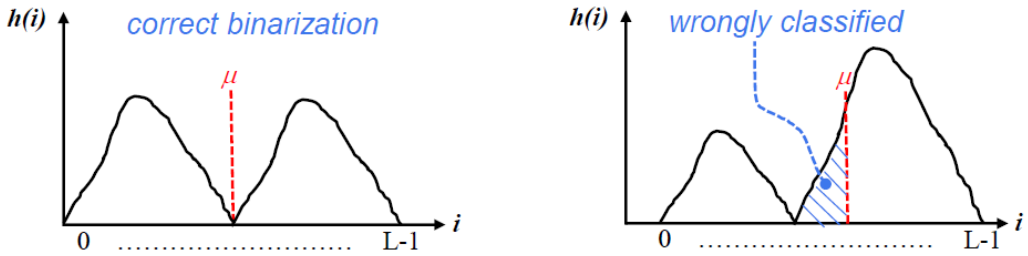

However, whenever the histogram is not clearly bimodal, binarization by intensity thresholding fails to provide the correct segmentation:




### 5.1.2 Automatic Threshold Selection

There exist algorithms which are able to aumatically select a suitable binarization threshold, per each image under analysis. Some examples of simple algorithms are the following:

- $T = \mu$, which works as long as pixels are equally distributed between the two classes, or, if one knows the percentile of dark/bright objects, it is possible to move $\mu$ according to this percentile:




- $T = \underline{\text{argmin}\{h(i)|i \in [i_1, i_2]\}}$, which requires to find the two main peaks of the histogram:

i contained between the two peaks such that h(i) is the minimum possible value.

- **Otsu's Algorithm**, in which the key intuition is to segment the image into two maximally homogeneous regions. The optimal threshold is chosen so as to minimize, across the grey-level range, the *within-group variance* of the regions. Given the possible grey-levels of the image, $i = 1, \ldots, L$, the number of pixels in the image, $N$, ~~the entries of the image histogram, $h(i) : i^{th}$~~ and the probability of grey-level $i$, $p(i) = h(i)/N$, the mean and variance of the *pmf* associated with image grey-levels can be expressed as:

$$\mu = \sum_{i=1}^{L} ip(i), \quad \sigma^2 = \sum_{i=1}^{L}(i-\mu)^2 p(i) \tag{5.1}$$

Any threshold value, $t$, would split pixels into two disjoint regions whose associated *pmf*s have mean and variance given by:

$$\mu_1(t) = \sum_{i=1}^{t} ip(i)/q_1(t), \quad \sigma_1^2(t) = \sum_{i=1}^{t}(i-\mu_1(t))^2 p(i)/q_1(t), \quad \text{with} \quad q_1(t) = \sum_{i=1}^{t} p(i) \tag{5.2}$$

$$\mu_2(t) = \sum_{i=t+1}^{L} ip(i)/q_2(t), \quad \sigma_2^2(t) = \sum_{i=t+1}^{L}(i-\mu_2(t))^2 p(i)/q_2(t), \quad \text{with} \quad q_2(t) = \sum_{i=t+1}^{L} p(i) \tag{5.3}$$

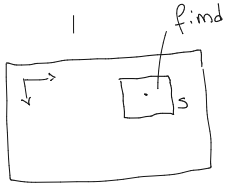
In particular, the *within-group variance* is given by:

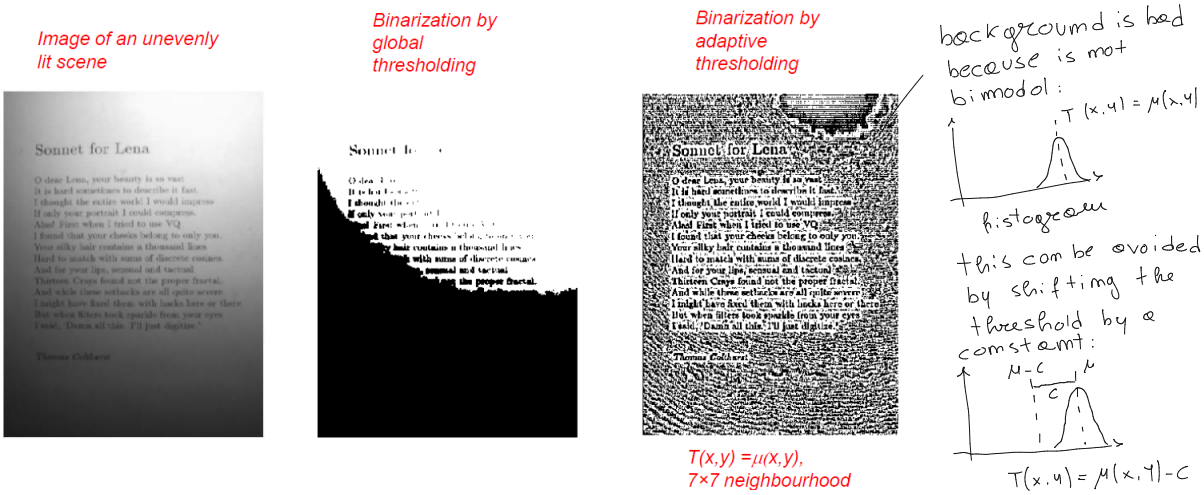
$$\sigma_W^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t) \tag{5.4}$$

If this variance is small then the classes are homogeneous. One ~~have~~ has to try every possible $t$ and find the one that minimize the quantity $\sigma_W^2$. *

## 5.2 Adaptive Thresholding

When the lighting is not uniform, a spatially varying (i.e. adaptive) threshold should be adopted. Usually, adaptive methods compute a specific threshold at each image pixel ($T \to T(x, y)$), based on the intensities within a small neighbourhood. However, too small of a neighbourhood might lack either background or foreground pixels, which would imply segmentation errors. For example:






## 5.3 Colour-Based Segmentation

In several applications, the objects of interest exhibit a known colour quite different from that of background structures. Hence, it is reasonable to try to segment-out foreground from background based on colour information.
In particular, the colour of a pixel is denoted as:

vector-valued function

$$\mathbf{I}(p) = \begin{bmatrix} I_r(p) \\ I_g(p) \\ I_b(p) \end{bmatrix} \tag{5.5}$$

Segmentation can be achieved by computing and thresholding the distance (e.g. Euclidean) between each pixel's colour and the reference background (foreground) colour $\boldsymbol{\mu}$:

$$d(\mathbf{I}(p), \boldsymbol{\mu}) = ((\mathbf{I}(p) - \boldsymbol{\mu})^T (\mathbf{I}(p) - \boldsymbol{\mu}))^{\frac{1}{2}} = ((I_r(p) - \mu_r)^2 + (I_g(p) - \mu_g)^2 + (I_b(p) - \mu_b)^2)^{\frac{1}{2}} \tag{5.6}$$

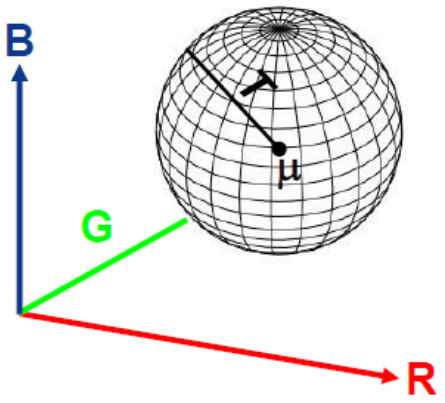
threshold, background, foreground

$$\forall p \in \mathbf{I} : \begin{cases} d(\mathbf{I}(p), \boldsymbol{\mu}) \leq T \rightarrow O(p) = B \ (F) \\ d(\mathbf{I}(p), \boldsymbol{\mu}) > T \rightarrow O(p) = F \ (B) \end{cases} \tag{5.7}$$

Therefore, it is necessary to know the reference colour $\boldsymbol{\mu}$, which can be estimated from one or more training images. For example, given a set of training samples of $N$ images:

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_r \\ \mu_g \\ \mu_b \end{bmatrix} = \frac{1}{N} \sum_{k=1}^{N} \mathbf{I}(p_k) \tag{5.8}$$

Accordingly, segmentation consists in classifying as background (foreground) all pixels lying within a 3D sphere of the RGB colour space centred at $\mu$ and having radius as large as the threshold $T$:



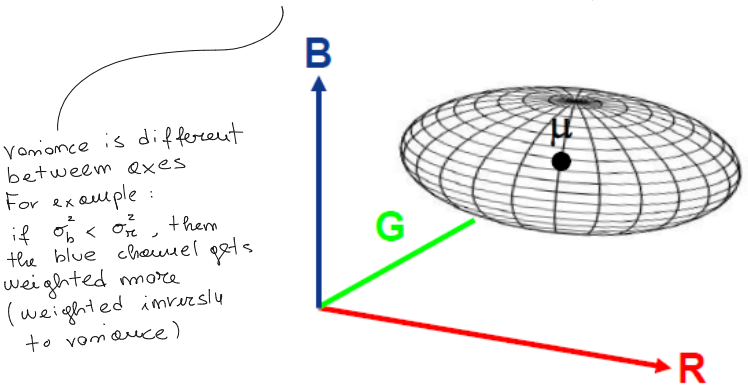

Other than Euclidean distance, Mahalanobis distance can be used instead which, given a set of training samples, uses the specific covariance matrix instead of the mean.

$$\Sigma = \begin{bmatrix} \sigma_{rr}^2 & \sigma_{rg}^2 & \sigma_{rb}^2 \\ \sigma_{gr}^2 & \sigma_{gg}^2 & \sigma_{gb}^2 \\ \sigma_{br}^2 & \sigma_{bg}^2 & \sigma_{bb}^2 \end{bmatrix}, \quad \text{with} \quad \begin{cases} \sigma_{ij}^2 = \frac{1}{N}\sum_{k=1}^{N}(I_i(p_k) - \mu_i)(I_j(p_k) - \mu_j) \\ i, j \in \{r, g, b\} \end{cases} \tag{5.9}$$

The Mahalanobis distance is given by:

$$^{*}\ d_M(\mathbf{I}(p), \boldsymbol{\mu}) = ((\mathbf{I}(p) - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{I}(p) - \boldsymbol{\mu}))^{\frac{1}{2}} \tag{5.10}$$

Contrary to the Euclidean distance, the Mahalanobis distance weights unequally the differences along the axes, i.e. there exist more variation along one axis (e.g. the circle has to be an ellipsoid).




* If $\Sigma$ is diagonal, $d_M = \left( \frac{(I_r(p) - \mu_r)^2}{\sigma_{rr}^2} + \frac{(I_g(p) - \mu_g)^2}{\sigma_{gg}^2} + \frac{(I_b(p) - \mu_b)^2}{\sigma_{bb}^2} \right)^{1/2}$

$$\begin{bmatrix} \sigma_{rr}^2 & 0 & 0 \\ 0 & \sigma_{gg}^2 & 0 \\ 0 & 0 & \sigma_{bb}^2 \end{bmatrix}$$

Since $\Sigma$ is real and and symmetric it can be diagonalized using the eigendecomposition.

$\Sigma = RDR^{-1}$, $R = (\overline{e_1}\ \overline{e_2}\ \overline{e_3})$ (eigenvectors), $D = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}$ (eigenvalues)

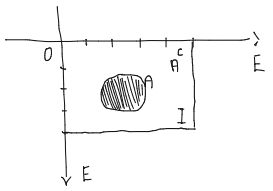
# Chapter 6

# Binary Morphology

correcting false positives and false negatives produced by segmentation

Binary morphology is about correcting, improving and performing analysis on binarized images. Binary morphology operators manipulate sets defined over the binary image. Given $I \subset E^2 = E \times E$, where $I$ is the image and $E$ is the set of integers number, the set of foreground pixels will be referred to as $A$, while background pixels as $A^c$. Binary morphology operators manipulates $A$ or $A^c$ through a second set, $B \subset E^2$, known as *structuring element*.

Similar to convolution where B is similare to a kernel




discrete Cartesian plane

## 6.1 Dilation (Minkowsky Sum)

The dilation operator is defined as follows:

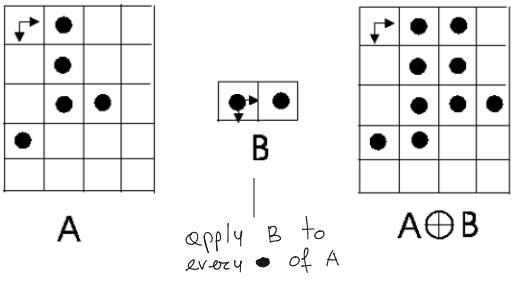
$$A \oplus B = \{c \in E^2 : c = a + b, a \in A, b \in B\} \tag{6.1}$$

$A = \{(0,1), (1,1), (2,1), (2,2), (3,0)\}$

$B = \{(0,0), (0,1)\}$

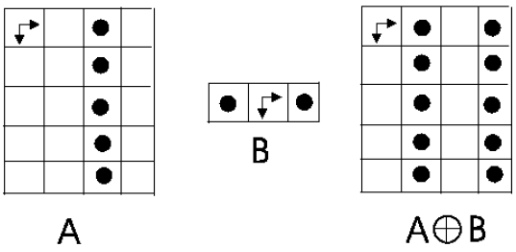
$A \oplus B = \{(0,1)+(0,0), (1,1)+(0,0), (2,1)+(0,0), (2,2)+(0,0), (3,0)+(0,0), (0,1)+(0,1), (1,1)+(0,1), (2,1)+(0,1), (2,2)+(0,1), (3,0)+(0,1)\}$

$= \{(0,1), (1,1), (2,1), (2,2), (3,0), (0,2), (1,2), (2,3), (3,1)\}$




apply B to every ● of A

the arrows represent the origin



* dilation changes the shape of foreground objects.

This operator can be used to correct segmentation errors dealing with foreground pixels falsely classified as background (e.g. to connect an object's parts of fill holes).

↳ false negatives

### 6.1.1 Properties of Dilation

The properties of dilation are the following:

translation

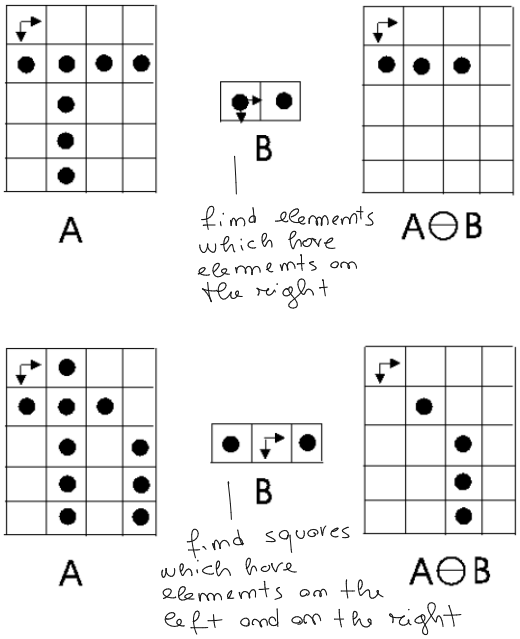
- Since translation $A_t$ of set $A$ is defined as: $A_t = \{c \in E^2 : c = a + t, a \in A\}$, dilation can be expressed as the union of the translations of either of the two sets by the elements of the other one: $A \oplus B = \bigcup_{b \in B} A_b = \bigcup_{a \in A} B_a$
- Dilation is commutative: $A \oplus B = B \oplus A$.
- Dilation is associative: $A \oplus (B \oplus C) = (A \oplus B) \oplus C$.
- If $\mathcal{O} \in B$, where $\mathcal{O}$ is the origin of the plane, then dilation is extensive: $A \subseteq A \oplus B$.
- Dilation is an increasing transformation: $A \subseteq C \Rightarrow A \oplus B \subseteq C \oplus B$.

*

## 6.2 Erosion (Minkowsky Subtraction)

The erosion operator is defined as follows:

$$A \ominus B = \{c \in E^2 : c + b \in A, \forall b \in B\} \tag{6.2}$$




This operator can be used to correct segmentation errors dealing with background pixels falsely classified as foreground (e.g. to split wrongly connected objects).

* erosion changes the shape of foreground object.

### 6.2.1 Properties of Dilation

The properties of erosion are the following:

- Erosion can be expressed in terms of translations: $A \ominus B = \{c \in E^2 : B_c \subseteq A\}$.
- Erosion involves subtraction of the elements of one set from those of the other: $A \ominus B = \{c \in E^2 : \forall b \in B \exists a \in A : c = a - b\}$.
- Erosion is not commutative: $A \ominus B \neq B \ominus A$.
- If the structuring element can be decomposed in terms of dilation, then erosion is associative: $A \ominus (B \oplus C) = (A \ominus B) \ominus C$.
- If $\mathcal{O} \in B$, then erosion is anti-extensive: $A \ominus B \subseteq A$.
- Erosion is an increasing transformation: $A \subseteq C \Rightarrow A \ominus B \subseteq C \ominus B$.

*

## 6.3 Duality Between Dilation and Erosion

flip b about the origin

Dilation and erosion have a great duality. As a matter of fact, given $\breve{B} = \{\breve{b} : \breve{b} = -b, b \in B\}$, it can be shown that:

$$(A \oplus B)^c = A^c \ominus \breve{B} \tag{6.3}$$
$$(A \ominus B)^c = A^c \oplus \breve{B} \tag{6.4}$$

If $B$ is symmetric ($B = \breve{B}$):

$$(A \oplus B)^c = A^c \ominus B \tag{6.5}$$
$$(A \ominus B)^c = A^c \oplus B \tag{6.6}$$

That is, dilation of foreground is equivalent to erosion of background, and vice versa.

## 6.4 Opening and Closing

Erosion and dilation by the same structuring element can be chained to remove selectively from either foreground or background the parts that do not match exactly the structuring element without causing any distortion to the other parts.

shape matching between structuring element and A (more smart than erosion). Similar to convolution

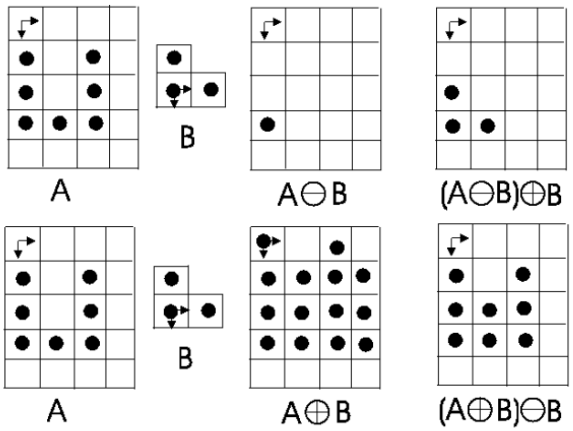
- **Erosion followed by Dilation is known as Opening:**

$$A \circ B = (A \ominus B) \oplus B$$

Add to the foreground all parts of the background which do not match the structuring elements

- **Dilation followed by Erosion is known as Closing:**

$$A \bullet B = (A \oplus B) \ominus B$$




### 6.4.1 Properties of Opening and Closing

The properties of opening and closing are the following:

- Opening and closing are idempotent: $(A \circ B) \circ B = A \circ B$ and $(A \bullet B) \bullet B = A \bullet B$.
- Opening and closing are not commutative: $A \circ B \neq B \circ A$ and $A \bullet B \neq B \bullet A$.
- Opening is anti-extensive, closing is extensive: $A \circ B \subseteq A$ and $A \bullet B \supseteq A$.
- Opening and closing are increasing transformations: $A \subseteq C \Rightarrow A \circ B \subseteq C \circ B, A \bullet B \subseteq C \bullet B$.
- The result of an opening operation can be expressed as the union of those elementary foreground parts that exactly match the structuring element: $A \circ B = (A \ominus B) \oplus B = \bigcup_{y \in A \ominus B} B_y = \bigcup_{B_y \subseteq A} B_y$.
- Duality between erosion and dilation implies duality between opening and closing:

$$(A \circ B)^c = [(A \ominus B) \oplus B]^c = (A \ominus B)^c \ominus \check{B} = (A^c \oplus \check{B}) \ominus \check{B} = A^c \bullet \check{B} \tag{6.7}$$

$$(A \bullet B)^c = [(A \oplus B) \ominus B]^c = (A \oplus B)^c \oplus \check{B} = (A^c \ominus \check{B}) \oplus \check{B} = A^c \circ \check{B} \tag{6.8}$$

If $B$ is symmetric ($B = \check{B}$): $(A \circ B)^c = A^c \bullet B$ and $(A \bullet B)^c = A^c \circ B$.

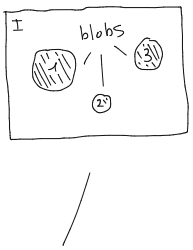
## 6.5 Hit-and-Miss Transform

This operator allows to detect specific patterns in binary images. The resulting image highlights the positions of the desired pattern. Given a structuring element, $B$, comprising both foreground, $B_1$, and background, $B_0$, points, the hit-and-miss transform is defined as follows:

$$A \otimes (B_1, B_0) = (A \ominus B_1) \cap (A^c \ominus B), \quad \text{with} \quad B_1 \cap B_0 = \emptyset \tag{6.9}$$

# Chapter 7

# Blob Analysis




Once foreground/background segmentation has been accomplished, in most applications the next task deals with analysis of the individual foreground objects to achieve some kind of high level knowledge on the observed scene. Individual objects are usually referred to as *blobs* (*binary large objects*) or *regions* need to be isolated within the overall foreground region, this process is known as *connected components labelling*. Then, individual objects can be processed to extract specific features. The process whereby features are extracted from blobs is often referred to as *blob analysis*.

## 7.1 Distances and Connectivity (how connectivity is defined)

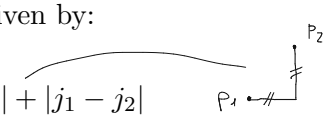
- Given $p_1(i_1, j_2), p_2(i_2, j_2), p_3(i_3, j_3) \in E^2$, $D$ is a distance if and only if:

properties of vector norms —

$$\begin{cases} D(p_1, p_2) \geq 0, D(p_1, p_2) = 0 \Leftrightarrow p_1 = p_2 & (7.1) \\ D(p_1, p_2) = D(p_2, p_1) & (7.2) \\ D(p_1, p_3) \leq D(p_1, p_2) + D(p_1, p_3) \text{ - triangle inequality} & (7.3) \end{cases}$$

- The city-block distance, $D_4$, between $p_1$ and $p_2$ is given by:
(Manhattan)

$$D_4(p_1, p_2) = |i_1 - i_2| + |j_1 - j_2| \qquad (7.4)$$




The set of points having distance $\leq r$ from a given one is a rhombus with diagonals of length $2r + 1$. For example, with $r = 2$:

$$\begin{array}{ccccc} & & 2 & & \\ & 2 & 1 & 2 & \\ 2 & 1 & 0 & 1 & 2 \\ & 2 & 1 & 2 & \\ & & 2 & & \end{array}$$

Defined as *neighbours of* $p$ the set of points having $D = 1$ from $p$, it follows that, using $D = D_4$, such a set is given by:

$$\begin{matrix} & n & \\ n & p & n \\ & n & \end{matrix}$$

This is usually called *4-neighbourhood of* $p$ ($n_4(p)$).

- The chessboard distance, $D_8$, between $p_1$ and $p_2$ is given by:

$$D_8(p_1, p_2) = \max(|i_1 - i_2|, |j_1 - j_2|) \tag{7.5}$$

The set of points having distance $\leq r$ from a given one is a square with side of length $2r + 1$. For example, with $r = 2$:

$$\begin{matrix} 2 & 2 & 2 & 2 & 2 \\ 2 & 1 & 1 & 1 & 2 \\ 2 & 1 & 0 & 1 & 2 \\ 2 & 1 & 1 & 1 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{matrix}$$

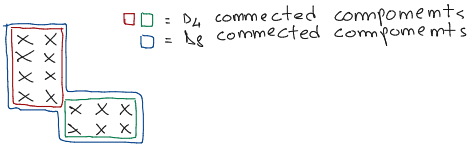
The set of neighbours of $p$ such that $D_8 = 1$ is called *8-neighbourhood of* $p$ ($n_8(p)$):

$$\begin{matrix} n & n & n \\ n & p & n \\ n & n & n \end{matrix}$$

In particular:

- A path of length $n$ from pixel $p$ to pixel $q$ is a sequence of pixels, $p = p_1, p_2, \ldots, p_n = q$, such that $p_i$ and $p_{i+1}$ are neighbours according to the chosen distance.
- A set of pixels, $R$, is said to be a *connected region* if for any two pixels $p, q$ in $R$ there exist a path contained in $R$ between $p$ and $q$.
- A set of pixels is said to be a *connected foreground (background) region* if it is a connected region and includes foreground (background) pixels only.
- A *connected component* of a binary image is a maximal connected foreground region.



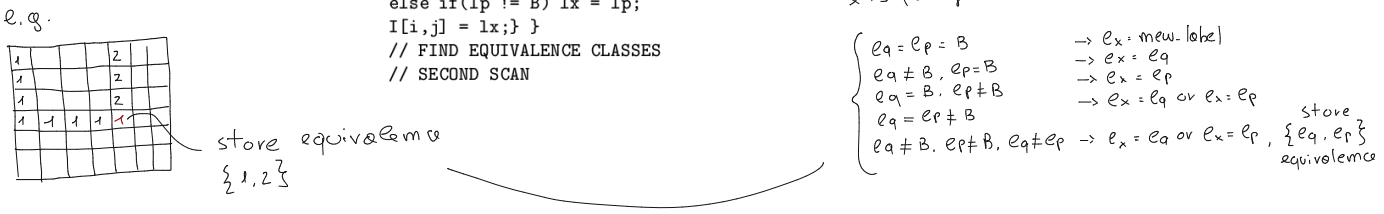


## 7.2 Connected Components Labelling

In order to label blobs, a two-scans algorithm is used:

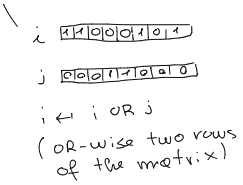
- By the first scan, foreground pixels take temporary labels based on those given to already visited neighbours, which depend on both the chosen distance and the scan order.
- Upon the first scan, different blobs have certainly been given different labels, though, depending on shape, this may be the case for connected parts of a single blob too.
- The second scan allows a unique final label to be assigned to those parts belonging to the same blob that had been given different temporary labels by the first scan.
- Equivalent temporary labels need to be found between the two scans, so as to assign a unique final label to each of the equivalence classes among temporary labels.

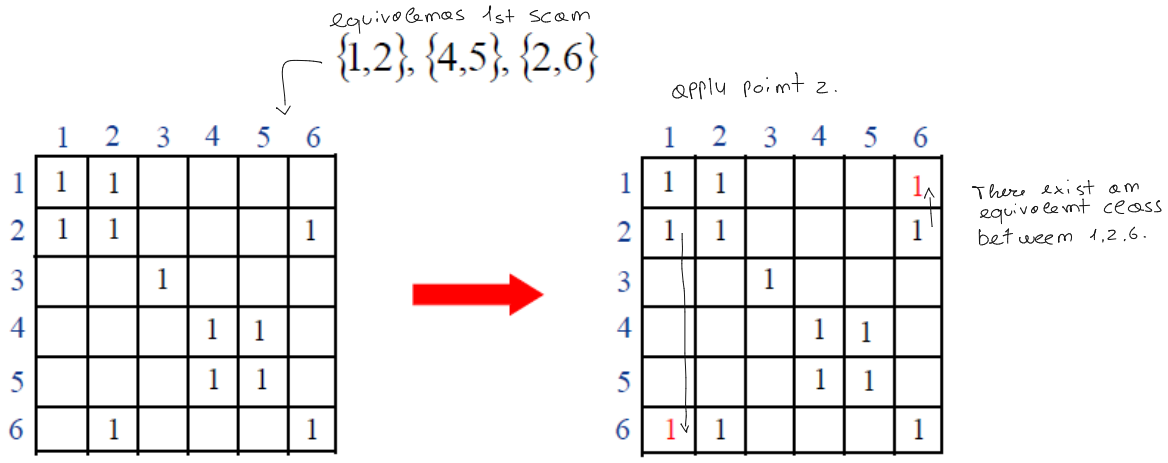
```
// lp,lq,lx: labels assigned to p,q,x
// B:background, F:foreground.
// FIRST SCAN:
for(i=1; i<NROWS-1; i++)
for(j=1; j<NCOLS-1; j++) {
if (I[i,j]==F) {
lp = I[i-1,j];
lq = I[i,j-1];
if(lp == B && lq == B) {
NewLabel++;
lx = NewLabel;}
else if((lp != lq)&&(lp != B)&&(lq != B)){
// REGISTER EQUIVALENCE (lp,lq)
lx = lq;}
else if(lq != B) lx = lq;
else if(lp != B) lx = lp;
I[i,j] = lx;} }
// FIND EQUIVALENCE CLASSES
// SECOND SCAN
```



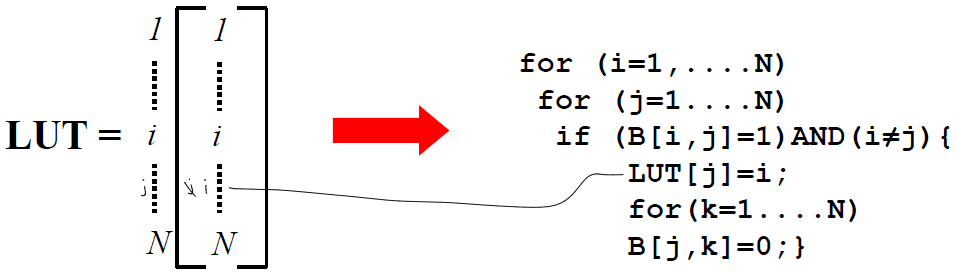

### 7.2.1 Handling Equivalences

1. During the first scan, the equivalences found between temporary label pairs are recorded into a $N \times N$ matrix, $B$, where $N$ is the number of labels. In particular, $B$ is a symmetric matrix and $B[i,j] = 1$ if $i = j$ and $B[i,j] = B[j,i] \forall i,j = 1,\ldots,N$. B[i,j] = 1, i ≠ j if there is an equivalence between i, j.
2. After the first scan, $B$ is processed so as to elicit the equivalences among temporary labels implied by virtue of the transitive property: $B[i,k] = B[i,k] \vee B[j,k] \forall k = 1,\ldots,N$. For example:






3. Having found equivalence classes, it is now necessary to assign a unique final label to each of them. A simple table initialized by the identity function may be conveniently deployed:




4. The second scan consists only in an image look-up operation using the table:

```
for (i=1....NROWS-1)
 for (j=1....NCOLS-1)
     I[i,j]=LUT[I[i,j]];
```

## 7.3 Blob Features

Once blobs have been found, it is possible to extract features from these objects.

### 7.3.1 Area and Barycentre

The area represents the amount of pixels belonging to the region and is computed as follows:

$$A = \sum_{p \in R} 1 \tag{7.6}$$

connected component

The barycentre represents the centre of mass of the region and is computed as follows:

$$P = \begin{bmatrix} i \\ j \end{bmatrix} \rightarrow \quad B = \begin{bmatrix} i_b \\ j_b \end{bmatrix}, \quad \text{with} \quad i_b = \frac{1}{A}\sum_{p \in R} i, \quad j_b = \frac{1}{A}\sum_{p \in R} j \tag{7.7}$$

### 7.3.2 Perimeter

The perimeter represents the length of the contour of the region. To measure such a length, it is necessary to define which pixels belong to the contour. A given pixel, $p$, is said to belong to the contour of the region if there exist at least one background pixel, $q$, between its neighbours:

4-neighbourhood — contour

$$\exists q \in n_4(p) \rightarrow p \in C_4 \tag{7.8}$$

$$\exists q \in n_8(p) \rightarrow p \in C_8 \tag{7.9}$$

$C_4$ is a 8-connected contour; $C_8$ is a 4-connected contour

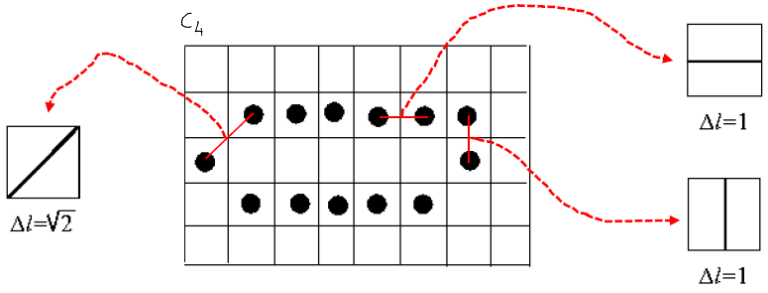
8-neighbourhood

The perimeter is then computed as follows:

$$P_8 = \sum_{p \in C_4} 1 \tag{7.10}$$

$$P_4 = \sum_{p \in C_8} 1 \tag{7.11}$$

Usually, $\tilde{P}$ is computed, which is the average between $P_4$ and $P_8$: $\tilde{P} = (P_4 + P_8)/2$.
Considering $C_4$, a more accurate approximation of the perimeter can be obtained by taking into account whether the ideal curve would better join two nearby pixels through a horizontal/vertical segment or a diagonal one:




Once $C_4$ has been computed, $C_4 = \{p_1, p_2, \ldots, p_m\}$, then the perimeter can be computed as follows:

$$\tilde{P}_8 = \sum_{p_k : p_{k+1} \in n_4(p_k)} 1 + \sum_{p_k : p_{k+1} \in (n_8(p_k) - n_4(p_k))} \sqrt{2} \tag{7.12}$$

### 7.3.3 Compactness (scale-invariant feature)

The compactness is a simple shape-feature calculated as follows:

$$C = \frac{P^2}{A} \qquad (7.13)$$

— minimum value in the continuous domain is $4\pi$ (C of a circle):

$$C_{circle} = \frac{(2\pi r)^2}{r^2\pi} = \frac{4r^2\pi^2}{r^2\pi} = 4\pi$$

This is measure is higher for shapes showing many concavities.

### 7.3.4 Haralick's Circularity

In the discrete domain, compactness takes its smallest value, not for a circle (no concavities), but for an octagon or a diamond. Therefore, Haralick has proposed the following circularity feature:

contour: $C = \{p_1, p_2, \ldots, p_m\}$, point of the contour: $p_k = \begin{bmatrix} i_k \\ j_k \end{bmatrix}$, barycentre: $B = \begin{bmatrix} i_b \\ j_b \end{bmatrix}$, $d_k = \sqrt{(i_k - i_b)^2 + (j_k - j_b)^2}$ (7.14)

$d_k$ = distance between point and barycentre

In particular, $d_k$ is the distance from a contour point to the barycentre. It is now possible to compute the mean distance, $\mu_R$, and the variance of such distances, $\sigma_R^2$:

$$\mu_R = \frac{1}{m}\sum_{k=1}^{m} d_k, \quad \sigma_R^2 = \frac{1}{m}\sum_{k=1}^{m}(d_k - \mu_R)^2 \qquad (7.15)$$

The Heralick's circularity is then computed as follows:

$$\tilde{C} = \frac{\mu_R}{\sigma_R}, \quad \sigma_R = \sqrt{\sigma_R^2} \qquad (7.16)$$

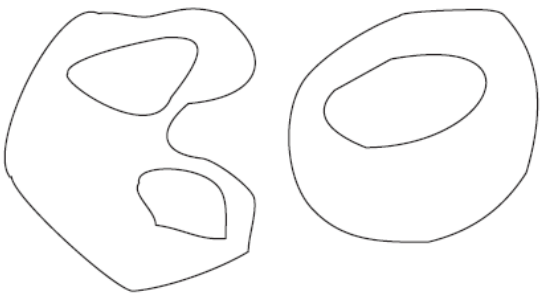
— if it was $\frac{1}{\sigma_R}$, $\tilde{C}$ would have not been scale-invariant, so $\mu_R$ is added.

### 7.3.5 Euler Number

The Euler number of a binary image is defined as:

$$E = C - H \qquad (7.17)$$

In particular, $C$ is the number of connected components and $H$ the number of holes:



C=2, H=3 → E = C - H = -1

### 7.3.6 Moments

The moment of order $(m, n)$ of a region is defined as:

$$M_{m,n} = \sum_{p \in R} i^m j^n, \quad m \geq 0, n \geq 0 \tag{7.18}$$

The area is the moment of order $(0, 0)$: $M_{0,0} = \sum_{p \in R} i^0 j^0 = \sum_{p \in R} 1 = A$. Other relevant ones are the second moments $(m + n = 2)$:

$$M_{0,2} = \sum_{p \in R} i^0, j^2 = \sum_{p \in R} j^2 \qquad \text{Moment of inertia with respect to the } i\text{-th axis}$$

$$M_{2,0} = \sum_{p \in R} i^2, j^0 = \sum_{p \in R} i^2 \qquad \text{Moment of inertia with respect to the } j\text{-th axis}$$

$$M_{1,1} = \sum_{p \in R} i^1, j^1 = \sum_{p \in R} ij \qquad \text{Deviation moment of inertia}$$

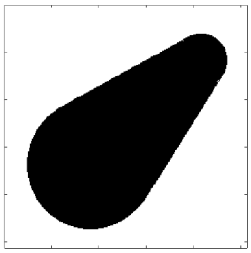
These moments change according to the position of the region in the image. Invariance to translation can be achieved by simply calculating the moments relative to the barycentre:

$$M'_{m,n} = \sum_{p \in R} (i - i_b)^m (j - j_b)^n \tag{7.19}$$

This is called *central moment*. To achieve, also, invariance to scaling, central moments need to be normalized:

$$V_{m,n} = \frac{M'_{m,n}}{A^\alpha}, \quad \alpha = \frac{m+n}{2} + 1 \tag{7.20}$$

For example:



256 x 256

128 x 128

64 x 64

$M'_{m,n}$

| | 256x256 | 128x128 | 64x64 |
|---|---|---|---|
| $M'_{20}$ | $6.1067 \cdot 10^7$ | $3.8184 \cdot 10^6$ | $2.3967 \cdot 10^5$ |
| $M'_{02}$ | $6.4680 \cdot 10^7$ | $4.0366 \cdot 10^6$ | $2.5315 \cdot 10^5$ |
| $M'_{11}$ | $-3.9126 \cdot 10^7$ | $-2.4467 \cdot 10^6$ | $-1.5327 \cdot 10^5$ |
| $M'_{30}$ | $-1.4325 \cdot 10^8$ | $-4.4165 \cdot 10^6$ | $-1.3700 \cdot 10^5$ |
| $M'_{03}$ | $3.0780 \cdot 10^8$ | $9.7300 \cdot 10^6$ | $3.1312 \cdot 10^5$ |
| $M'_{21}$ | $8.0244 \cdot 10^8$ | $2.4995 \cdot 10^7$ | $7.9233 \cdot 10^5$ |
| $M'_{12}$ | $-8.9164 \cdot 10^8$ | $-2.7851 \cdot 10^7$ | $-8.8653 \cdot 10^5$ |

$V_{m,n}$

| | 256x256 | 128x128 | 64x64 |
|---|---|---|---|
| $V_{20}$ | 0.1035 | 0.1037 | 0.1037 |
| $V_{02}$ | 0.1096 | 0.1096 | 0.1096 |
| $V_{11}$ | −0.0663 | −0.0664 | −0.0663 |
| $V_{30}$ | −0.0016 | −0.0015 | −0.0015 |
| $V_{03}$ | 0.0033 | 0.0034 | 0.0035 |
| $V_{21}$ | 0.0087 | 0.0087 | 0.0088 |
| $V_{12}$ | −0.0097 | −0.0097 | −0.0098 |

$* \quad M(\theta) = \sum_{p \in R} d_\theta^2(p) = \sum_{p \in R} (\alpha \bar{j} - \beta \bar{i})^2 = \alpha^2 \sum \bar{j}^2 - 2\alpha\beta \sum \bar{i}\bar{j} + \beta^2 \sum \bar{i}^2 = \alpha^2 \underbrace{\sum (j - j_B)^2}_{M'_{0,2}} - 2\alpha\beta \underbrace{\sum (i - i_B)(j - j_B)}_{M'_{1,1}} + \beta^2 \underbrace{\sum (i - i_B)^2}_{M'_{2,0}}$

where $\alpha = -\sin\theta$, $\beta = \cos\theta$
in order to find the angle between the major axis and the horizontal axis:

$\underset{\theta}{\operatorname{argmin}}\, M(\theta) \rightarrow \underset{\theta}{\operatorname{argmin}}\, \sin^2\theta\, M'_{0,2} + 2\sin\theta\cos\theta\, M'_{1,1} + \cos^2\theta\, M'_{2,0}$ (always minimized in an analytic way, i.e. finding $\frac{dM(\theta)}{d\theta}$ and $\frac{d^2M(\theta)}{d\theta^2}$)

### 7.3.7 Orientation

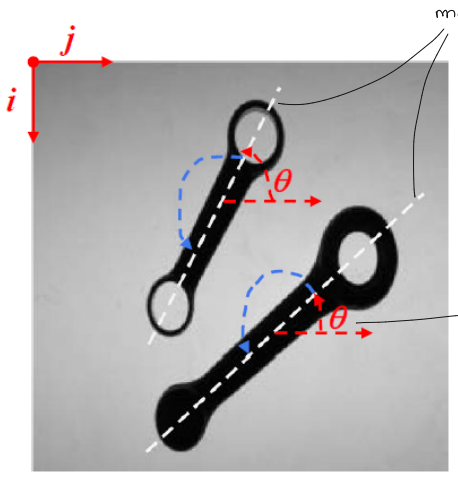
The orientation of elongated objects can be defined according to the direction of the *axis of least inertia*, that is the line through the barycentre of least moment of inertia (*major axis*):

$$\text{major axis} = \underset{l}{\operatorname{argmin}} \left( \sum_{p \in R} d_l^2(p) \right) \tag{7.21}$$

$l$: line parameter

turns out that, major axis: $-\sin\theta\, j + \cos\theta\, i + \cos\theta\, i_B + \sin\theta\, j_B = 0$

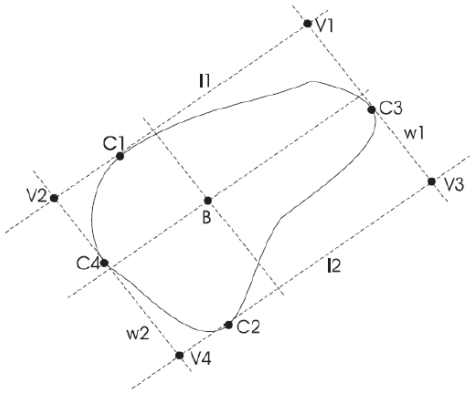
The orientation is typically determined as the angle, $\theta$, between the major axis and the horizontal axis: $*$




orientation. Defined up to $\pi$, i.e. $\theta$ and $\theta + \pi$ represent the same orientation.

### 7.3.8 Oriented Enclosing Rectangle

Given the major axis and the minor axis of a specific blob, one might want to draw a bounding box aligned to the object (*minimum enclosing rectangle*). The four points laying at maximum distance on opposite sides of the two axes need to be determined $(C_1, C_2, C_3, C_4)$:



Then, one can find the lines through $C_1$ and $C_2$, which is parallel to the major axis, and through $C_3$ and $C_4$, which is parallel to the minor axis:

$$\begin{aligned} l_1 : aj + bi + c_{l_1} &\Rightarrow c_{l_1} = -(aj_1 + bi_1) \\ l_2 : aj + bi + c_{l_2} &\Rightarrow c_{l_2} = -(aj_2 + bi_2) \\ w_1 : a'j + b'i + c_{w_1} &\Rightarrow c_{w_1} = -(a'j_3 + b'i_3) \\ w_2 : a'j + b'i + c_{w_2} &\Rightarrow c_{w_2} = -(a'j_4 + b'i_4) \end{aligned}$$

The vertices of the oriented rectangle are given by:

$$j_{V_1} = (bc_{w_1} - b'c_{l_1})/(ab' - ba'), \quad i_{V_1} = (a'c_{l_1} - ac_{w_1})/(ab' - ba') \tag{7.22}$$
$$j_{V_2} = (bc_{w_2} - b'c_{l_1})/(ab' - ba'), \quad i_{V_2} = (a'c_{l_1} - ac_{w_2})/(ab' - ba') \tag{7.23}$$
$$j_{V_3} = (bc_{w_1} - b'c_{l_2})/(ab' - ba'), \quad i_{V_3} = (a'c_{l_2} - ac_{w_1})/(ab' - ba') \tag{7.24}$$
$$j_{V_4} = (bc_{w_2} - b'c_{l_2})/(ab' - ba'), \quad i_{V_4} = (a'c_{l_2} - ac_{w_2})/(ab' - ba') \tag{7.25}$$

Once computed the MER, it is possible to find some useful features of the blob:

- Length ($L$) and width ($W$): ( invariant to rotation)

$$L = \overset{\text{distance}}{d_{V_1V_2}} = d_{V_3V_4} = \sqrt{(i_{V_1} - i_{V_2})^2 + (j_{V_1} - j_{V_2})^2} = \sqrt{(i_{V_3} - i_{V_4})^2 + (j_{V_3} - j_{V_4})^2} \tag{7.26}$$
$$W = d_{V_1V_3} = d_{V_2V_4} = \sqrt{(i_{V_1} - i_{V_3})^2 + (j_{V_1} - j_{V_3})^2} = \sqrt{(i_{V_2} - i_{V_4})^2 + (j_{V_2} - j_{V_4})^2} \tag{7.27}$$

- Elongatedness ($E$):

$$E = \frac{L}{W} \tag{7.28}$$

- Rectangularity ($R$):

$$R = \frac{A}{LW} \tag{7.29}$$

- Ellipticity ($E$):

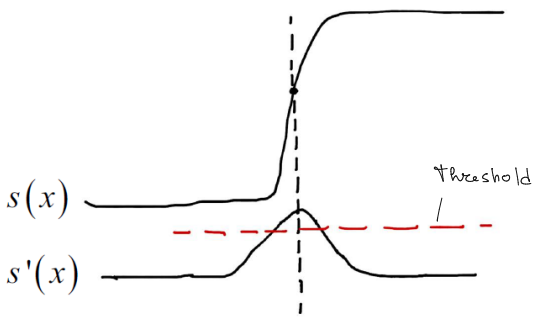
$$E = \frac{A}{A_{LW}}, \quad A_{LW} = \frac{\pi}{4}LW \tag{7.30}$$

# Chapter 8

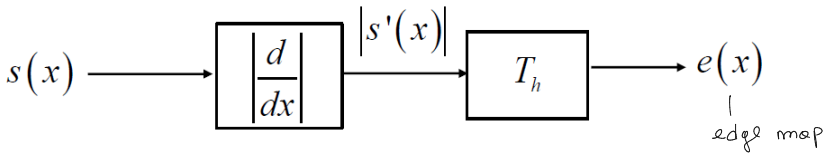
# Edge Detection

Edges (or contour points) are features that we can extract from an image (typically grey-scale), important because they contain relevant semantic information about the image. Edges are 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.

## 8.1 1D Step-Edge

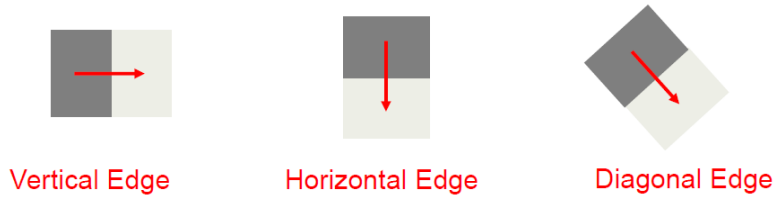



The simplest edge-detection operator relies on thresholding the absolute value of the derivative of the signal:



## 8.2 2D Step-Edge

A 2D step-edge is characterized not only by its strength but also its direction.

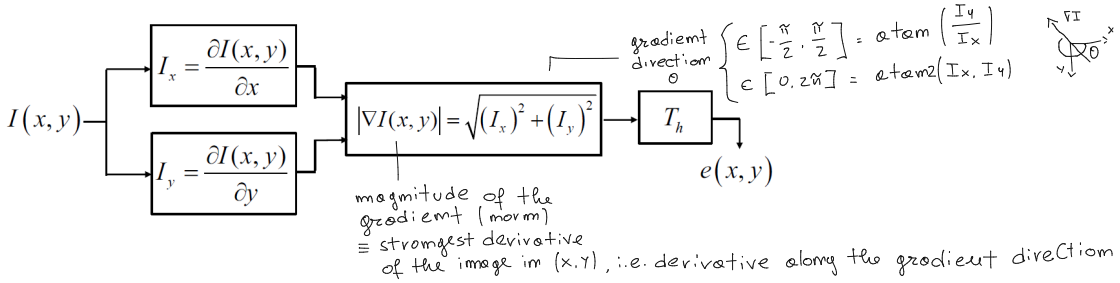



In particular, the gradient's direction gives the direction along which the function exhibits its maximum variation. A generic directional derivative can be computed by the dot product between the gradient and the unit vector along the direction:

the gradient points in the direction of steepest change of the function

$$\nabla I(x,y) = \frac{\partial I(x,y)}{\partial x}\mathbf{i} + \frac{\partial I(x,y)}{\partial j}\mathbf{j} \tag{8.1}$$

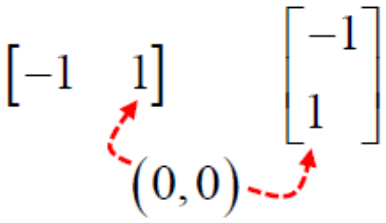
In this case, edge detection can be also done by gradient thresholding:

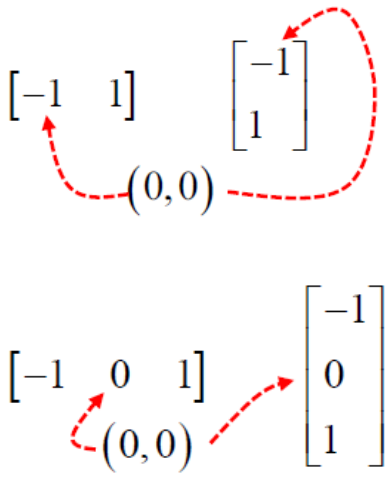



In particular, to approximate the gradient numerically, the backward, forward and central differences methods can be used:

$$I_x(i,j) = I(i,j) - I(i,j-1), \quad I_y(i,j) = I(i,j) - I(i-1,j) \qquad \text{(Backward differences)}$$
$$I_x(i,j) = I(i,j+1) - I(i,j), \quad I_y(i,j) = I(i+1,j) - I(i,j) \qquad \text{(Forward differences)}$$
$$I_x(i,j) = I(i,j+1) - I(i,j-1), \quad I_y(i,j) = I(i+1,j) - I(i-1,j) \qquad \text{(Central differences)}$$

The correlation kernels used to apply these methods are, respectively, the following:





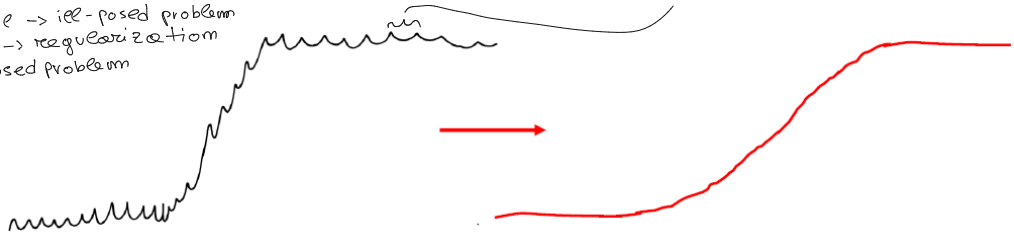

The estimation of the magnitude can be done in differenct ways:

faster and more invariant w.r.t. to edge direction

$$|\nabla I| = \sqrt{(I_x)^2 + (I_y)^2}, \quad |\nabla I|_+ = |I_x| + |I_y|, \quad |\nabla I| = \max(|I_x|, |I_y|) \tag{8.2}$$

The problem with this is approach is noise. In real images, the signal is full of noise. Therefore, it is necessary to de-noise the image (i.e. smooth the image): - otherwise noisy-steps can be seem as edges.

- noisy signal -> ill-posed problem
- smoothing -> regularization
  -> well-posed problem

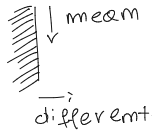


The smoothed signal, however, will also present smoothed edges. To avoid this, *smooth derivatives* can be applied.

### 8.2.1 Smooth Derivatives

Smoothing and differentiation can be carried out jointly within a single step. This is achieved by computing differences of averages. For example, if one wishes to smooth out noise by averaging over three pixels:

if one wants to differentiate along one axis, one takes the mean along the other axis (this is done in order to preserve edges)

e.g. 

3-pixel vertical mean

$$\begin{aligned}\tilde{I}_x(i,j) &= I_{3y}(i,j+1) - I_{3y}(i,j) \\ &= \frac{1}{3}[I(i-1,j+1) + I(i,j+1) + I(i+1,j+1) - I(i-1,j) - I(i,j) - I(i+1,j)] \\ &\Rightarrow \frac{1}{3}\begin{bmatrix} -1 & 1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix}\end{aligned} \tag{8.3}$$

forward differences

3-pixel horizontal mean

$$\begin{aligned}\tilde{I}_y(i,j) &= I_{3x}(i+1,j) - I_{3x}(i,j) \\ &= \frac{1}{3}[I(i+1,j-1) + I(i+1,j) + I(i+1,j+1) - I(i,j-1) - I(i,j) - I(i,j+1)] \\ &\Rightarrow \frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix}\end{aligned} \tag{8.4}$$

In particular:

these multiplicative factors can be omitted by simply increasing the threshold

$$I_{3x}(i,j) = \frac{1}{3}[I(i,j-1) + I(i,j) + I(i,j+1)] \tag{8.5}$$

$$I_{3y}(i,j) = \frac{1}{3}[I(i-1,j) + I(i,j) + I(i+1,j)] \tag{8.6}$$

$$\tag{8.7}$$

Given the same smoothing, central difference can be used:

$$\tilde{I}_x(i,j) = I_{3y}(i,j+1) - I_{3y}(i,j-1) \quad \Rightarrow \quad \frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{8.8}$$

$$\tilde{I}_y(i,j) = I_{3x}(i+1,j) - I_{3x}(i-1,j) \quad \Rightarrow \quad \frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{8.9}$$

This operator is called *Prewitt operator*. Moreover, the central pixel can be weighted more to further improve isotropy:

$$I_{4y}(i,j) = \frac{1}{4}[I(i-1,j) + 2I(i,j) + I(i+1,j)]$$

$$I_{4x}(i,j) = \frac{1}{4}[I(i,j-1) + 2I(i,j) + I(i,j+1)]$$

**Sobel operator**

$$\tilde{I}_x(i,j) = I_{4y}(i,j+1) - I_{4y}(i,j-1) \quad \Rightarrow \quad \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\tilde{I}_y(i,j) = I_{4x}(i+1,j) - I_{4y}(i-1,j) \quad \Rightarrow \quad \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\tilde{I}_x = \begin{bmatrix} -1 & 0 & 1 \\ -\sqrt{2} & 0 & \sqrt{2} \\ -1 & 0 & 1 \end{bmatrix} \qquad \tilde{I}_y = \begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix} \qquad |\nabla I| = \sqrt{(I_x)^2 + (I_y)^2}$$

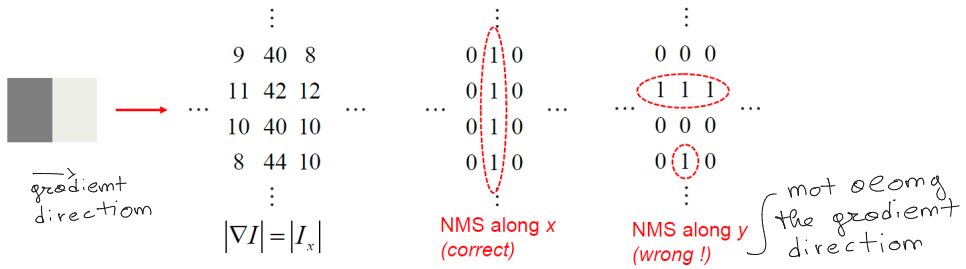
This operator is called *Sobel operator*. Lastly, full isotropy is provided by the *Frei-Chen operator*, which implies the following:

Detecting edges by gradient thresholding is inherently inaccurate because choosing the right threshold is difficult. A better approach to detect edges consists in finding the local maxima of the absolute value of the derivative of the signal.

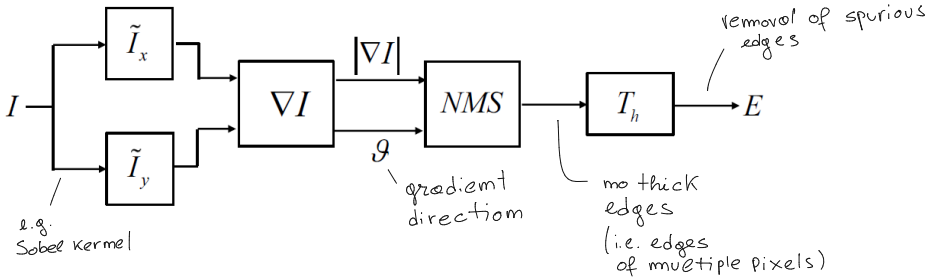
if one chooses a low threshold, one could obtain many pixels as edges, and viceversa. In order to avoid so, NMS is used

### 8.2.2 Non-Maxima Suppression (NMS)

This method allows to find the local maxima of the absolute value of the derivative (i.e. the gradient magnitude) along the gradient direction. The direction of the gradient, $\theta$, need to be estimated.



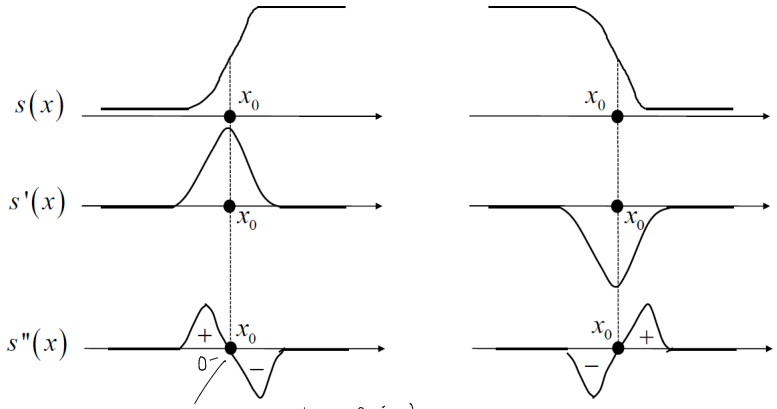

The overall flow-chart of an edge detector based on smooth derivatives and NMS is sketched below:




A final threshold helps pruning out unwanted edges due to noise or less important details.

### 8.2.3 Zero-Crossing of the Second Derivative (instead of NMS)

Edges may also be located by looking for zero-crossing of the second derivative of the signal.




In order to do so, it is possible to rely on the Laplacian as second order differential operator:

$$\nabla^2 I(x,y) = \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2} = I_{xx} + I_{yy} \tag{8.10}$$

continuous

Using forward and backward differences to approximate, respectively, first and second order derivatives:

discrete

$$\begin{cases} I_{xx} \cong I_x(i,j) - I_x(i,j-1) = I(i,j-1) - 2I(i,j) + I(i,j+1) & (8.11) \\ I_{yy} \cong I_y(i,j) - I_y(i-1,j) = I(i-1,j) - 2I(i,j) + I(i+1,j) & (8.12) \end{cases}$$

The corresponding convolution kernel is the following:

$$\nabla^2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{8.13}$$

As already mentioned, a robust edge detector should include a smoothing step to filter out noise. The *Laplacian of Gaussian (LOG)* method does de-noise the image and then applies the second order differentiation by the Laplacian:

convolution  Gaussian Kernel

- Gaussian smoothing: $\tilde{I}(x,y) = I(x,y) * G(x,y)$.
- Second order differentiation by the Laplacian: $\nabla^2 \tilde{I}(x,y)$.
- Extraction of the zero-crossing of $\nabla^2 \tilde{I}(x,y)$. — This is done by looking where there is a sign change in $\nabla^2 \tilde{I}(x,y)$, either horizontally or vertically.

zero must be contained between -15 and +3. (sub-pixel interpolation)

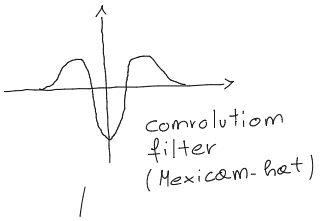
In this case, the $\sigma$ parameter of the Gaussian filter controls the smoothing of the image.
In particular:

computation of the LOG

$$\nabla^2 \tilde{I}(x,y) = \nabla^2 (I(x,y) * G(x,y)) = I(x,y) * \nabla^2 G(x,y) \tag{8.14}$$

property of convolution

Where:




$$\nabla^2 G(x,y) = \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2} = \frac{1}{2\pi\sigma^4}(\frac{r^2}{\sigma^2} - 2)e^{-\frac{r^2}{2\sigma^2}}, \quad r^2 = x^2 + y^2 \tag{8.15}$$

Thanks to the separability of the Gaussian, computing the LOG boils down to four 1D convolutions, which is substantially faster:

$$\begin{aligned} I(x,y) * \nabla^2 G(x,y) &= I(x,y) * (G''(x)G(y) + G''(y)G(x)) \\ &= I(x,y) * (G''(x)G(y)) + I(x,y) * (G''(y)G(x)) \\ &= (I(x,y) * G''(x)) * G(y) + (I(x,y) * G''(y)) * G(x) \end{aligned} \tag{8.16}$$

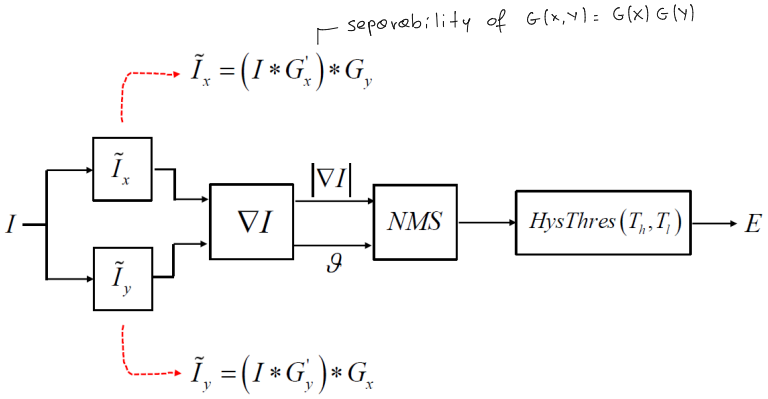
## 8.3 Canny's Edge Detector

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

- Good detection, i.e. the filter should correctly extract edges in noisy images.
- Good localization, i.e. the distance between the found edge and the true edge should be minimum.
- One response to one edge, i.e. the filter should detect one single edge pixel at each true edge.

He showed that the optimal edge detection operation consists in finding local extrema of the convolution of the directional derivative along the gradient. A Canny edge detector can be achieved by Gaussian smoothing followed by gradient computation and NMS along the gradient direction. ~~In the 1D case~~:

separability of $G(x,y) = G(x)\,G(y)$




In particular, $HysThres(T_h, T_l)$ is an hysteresis thresholding function which filters out unwanted edge detected due to noise.

A pixel is taken to be an edge if:

1. $|\nabla I| > T_h$ or $|\nabla I| > T_e$
2. The pixel is a neighbour of an already detected edge

# Chapter 9

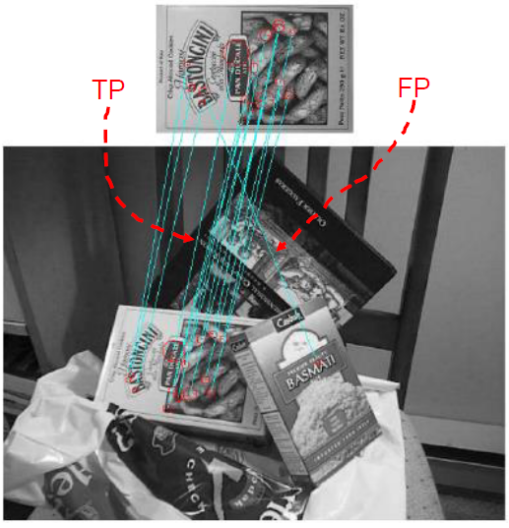
# Local Invariant Features

A great variety of computer vision problems can be dealt with by finding *corresponding points* (i.e. image points which are the projection of the same 3D point) between two or more images of a scene. In order to establish correspondences, it is necessary to follow three steps:

1. Detection of interest points. — saliemt poimts — A good detector should find the same saliemt poimts im differemt views
2. Description of each interest point by means of a suitable descriptor. — A good description should be robust w.r.t. light changes, noise, rotetiom, etc.
3. Matching descriptors between images.

The performance of the matching process can be measured using the *recall* and *precision* metrics:

$$\text{Recall} = \frac{TP}{TP + FN}, \qquad \text{Precision} = \frac{TP}{TP + FP} \tag{9.1}$$



## 9.1 Detection of Interest Points

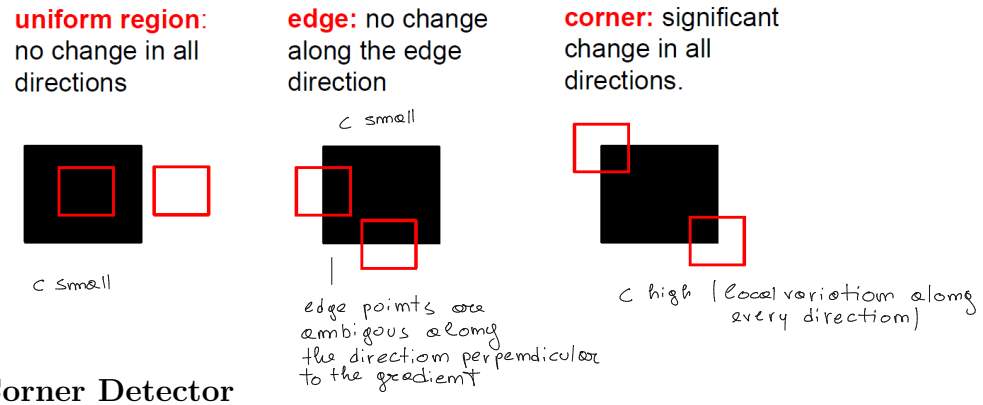
In general, pixels exhibiting a large variation along all directions are more amenable to establish reliable correspondences.

### 9.1.1 Moravec Interest Point Detector

Moravec proposed the following error function, which basically returns a big value is the considered point is a corner:

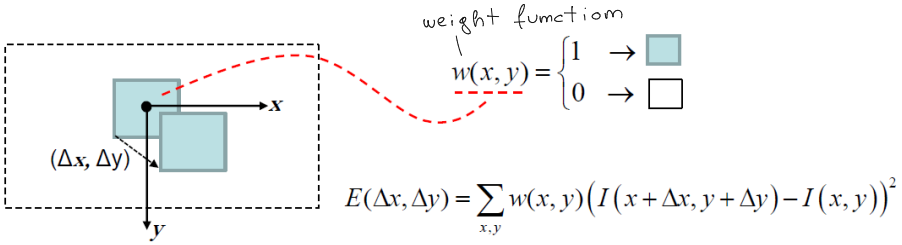
local variation along every direction

$$C = \min \left\{ \sum_{i,j \in w} (I(i+m, j+n) - I(i,j))^2 \right\}, \quad m, n \in \{-1, 0, 1\}, \neq (0,0) \tag{9.2}$$




### 9.1.2 Harris Corner Detector

Harris and Stephens proposed to rely on a continuous formulation of the Moravec's error function. Detoned as $(\Delta x, \Delta y)$ a generic infinitesimal shift, such a function can be written as:




Due to the shift being infinitesimal, the Taylor's expansion of the intensity function $(x, y)$ can be expanded:

$$I(x+\Delta x, y+\Delta y) - I(x,y) \cong \frac{\partial I(x,y)}{\partial x}\Delta x + \frac{\partial I(x,y)}{\partial y}\Delta y = I_x(x,y)\Delta x + I_y(x,y)\Delta y \tag{9.3}$$

Therefore:

$$E(\Delta x, \Delta y) = \sum_{x,y} w(x,y)\overbrace{(I_x(x,y)\Delta x + I_y(x,y)\Delta y)}^{I(x+\Delta x, y+\Delta y) - I(x,y)}{}^2$$
$$= \dots \text{ expansion of } (\dots)^2$$
$$= \begin{bmatrix}\Delta x & \Delta y\end{bmatrix} M \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix} \tag{9.4}$$

Where $M$ encodes the local image structure around the considered pixel:

structure matrix (contains all the local structure of the image)

$$M = \begin{pmatrix} \sum_{x,y} w(x,y) I_x(x,y)^2 & \sum_{x,y} w(x,y) I_x(x,y) I_y(x,y) \\ \sum_{x,y} w(x,y) I_x(x,y) I_y(x,y) & \sum_{x,y} w(x,y) I_y(x,y)^2 \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

$\Rightarrow E(\Delta x, \Delta y) = \lambda_1 \Delta x^2 + \lambda_2 \Delta y^2$

assuming $M$ is diagonal: the main diagonal contains the eigenvalues of $M$

$M$ can be diagonalized since it is a $2 \times 2$ symmetric matrix

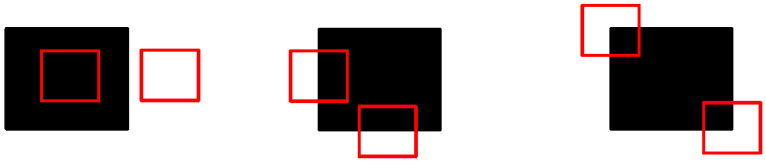
$M = R \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R^T$

rotation containing the eigenvectors of $M$

In particular:

($\lambda_1$ is always greater than $\lambda_2$, but could be not that greater)

$\lambda_1, \lambda_2 \cong 0$ : **Flat** $\quad \lambda_1 \gg \lambda_2$: **Edge** $\quad \lambda_1, \lambda_2 \uparrow$ : **Corner**

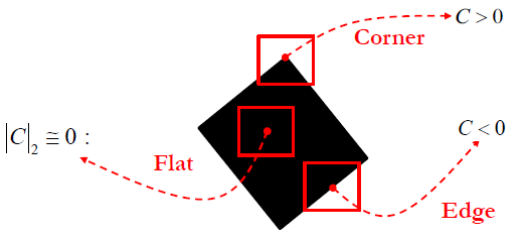


However, computing the eigenvalues, $\lambda_1, \lambda_2$ of $M$ may be slow. Thus, Harris and Stephens proposed to compute a more efficient function:

parameter

$$C = \text{determinant}(M) - k \cdot \text{trace}(M)^2 = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{9.5}$$

Lastly, it turns out that:




The Harris corner detector has the following invariance properties:

- It is invariant to rotation, since the eigenvalues of $M$ are invariant to a rotation of the images axes.

- It is invariant to intensity changes only in case of ad additive bias ($I = I + b$) since the derivative of $I$ does not consider $b$. If the intensity of the image is multiplicated by a gain factor ($I = a \cdot I$) then the intensity invariance does not hold since the derivative of $I$ gets multiplicated by $a$.
- It is not invariant to scaling, since different window sizes could produce different results. For example, given a too small windows size, a corner can be detected as an edge. As a matter of fact, in practice, an image contains features at different scales.

change of scale

detection window

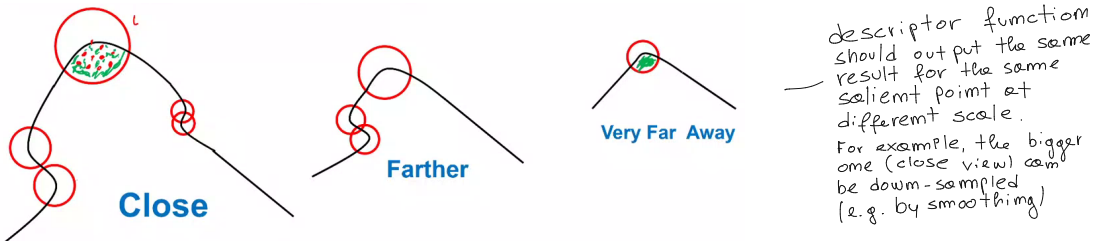
edges

corner

### 9.1.3 Scale-Space

one wants to probe every possible scale in order to find the maximum number of interest points (i.e. the maximum number of correspondence point candidates)

As already mentioned, scale invariance is an issue.




descriptor function should output the same result for the same salient point at different scale. For example, the bigger one (close view) can be down-sampled (e.g. by smoothing)

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 operations. A Gaussian scale-space is obtained by repeatedly applying larger and larger Gaussian kernels:


rather than considering different windows, one sub-samples the image (while blurring it) maintaining the same window.

increasing value of $\sigma$

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{9.6}$$

Where the one parameter is $\sigma$ which controls the scale. However, this representation does not include any criterion to detect features nor to select their characteristic scale (i.e. establishing at which scale a feature turns out maximally interesting and should therefore be described). The fundamental research work on multi-scale feature detection and automatic scale selection is due to Lindberg, who proposed to compute suitable combinations of scale normalized derivatives of the Gaussian scale space (normalized Gaussian derivatives) and find their extrema. Below is the proposed scale-normalized LOG:

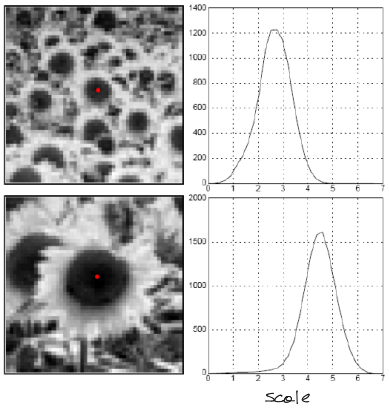
$$F(x, y, \sigma) = \sigma^2 \nabla^2 L(x, y, \sigma) = \sigma^2 (\nabla^2 G(x, y, \sigma) * I(x, y)) \tag{9.7}$$

Where, $\sigma^2$ is the normalization factor.

boosted Laplacian of the Gaussian scale-space ($\sigma$)

derivative are boosted by a factor when blurring the image

In order to find feature points one has to find maxima of $F(x,y,\sigma)$.




Considering the centers (red points) of the two dark blobs, it can be observed that the extremum of the scale-normalized LOG is found at a larger scale for the larger blob.

The ratio between the two characteristic scales is roughly the same as the ratio between the sizes (diameters) of the two blobs.

ratio of scales ≈ ratio of diameters

Lowe then proposes to detect interest points by finding the extrema of the *Difference of Gaussian (DoG)* function across the (x,y,$\sigma$) domain:

diff. between two levels of the Gaussian scale-space

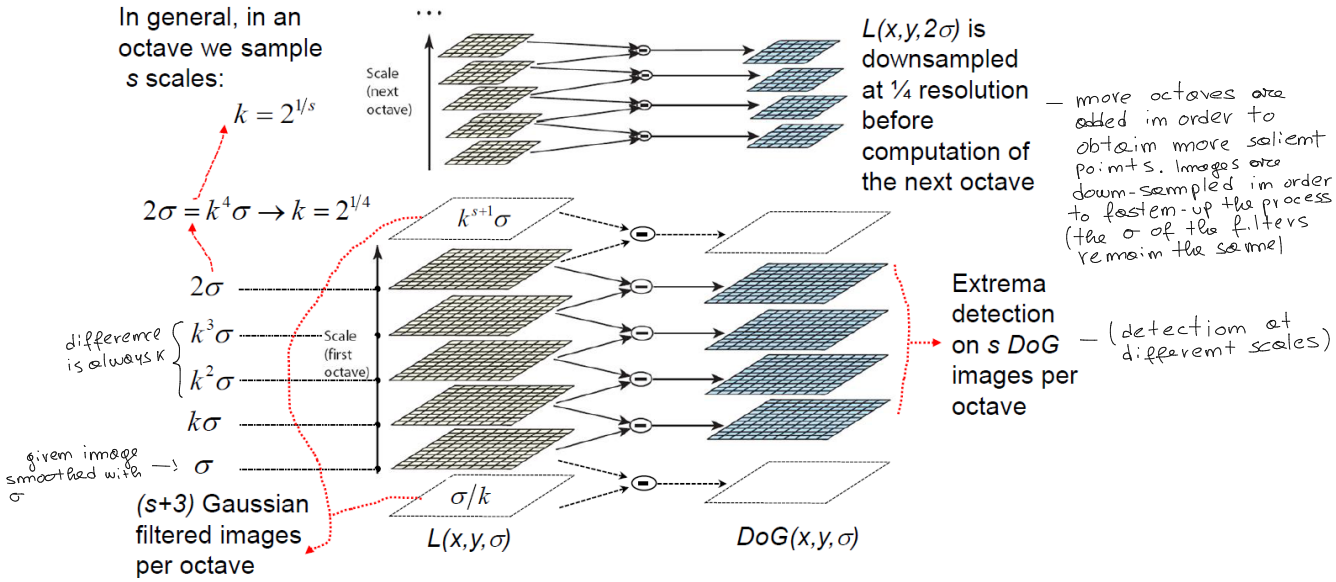
$$\mathrm{DoG}(x,y,\sigma) = (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) = L(x,y,k\sigma) - L(x,y,\sigma) \quad (9.8)$$

Lowe proved that, this approach provides a computationally efficient approximation of Lindberg's scale-normalized LOG:

$G(x,y,k\sigma)$

$$\cancel{G(x,y,\sigma)} - G(x,y,\sigma) \approx (k-1)\sigma^2\nabla^2 G(x,y,\sigma) \quad (9.9)$$

Since $(k-1)$ is a constant, it does not influence the extrema location. Below is the computation of DoG:

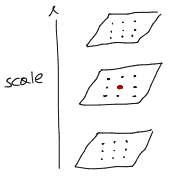



more octaves are added in order to obtain more salient points. Images are down-sampled in order to fasten-up the process (the σ of the filters remain the same)

(detection at different scales)

A point $(x,y,\sigma)$ is detected as salient point iff its DoG is higher (lower) than that of the 26 neighbours (8 at some scale and 9+9 at the two nearby scales) in the $(x,y,\sigma)$ space.

After this process, spurious keypoints are pruned (especially edge points, since these are hard to match)

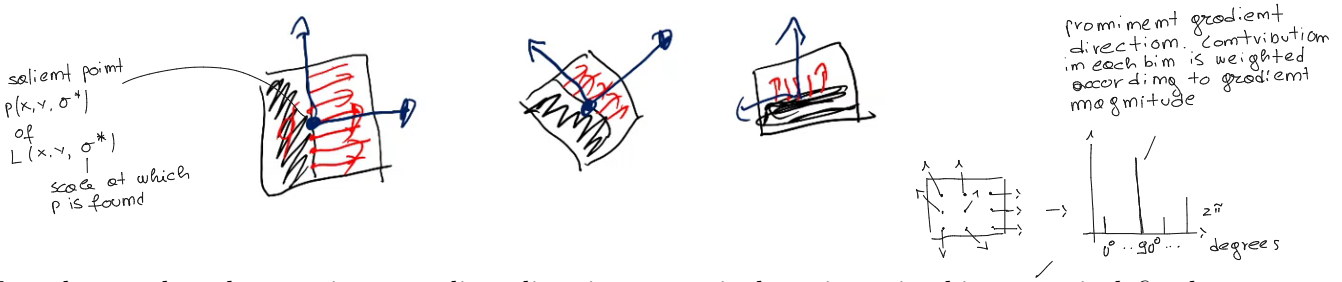


## 9.2 Definition of Descriptors

Once key points are found thanks to the DoG operator, it is possible to define descriptors of the given points. A desirable descriptor should be scale and rotation invariant:

description has to be computed using the corresponding $L(x,y,\sigma)$ image to the DoG image in which saliency is found

- To attain scale invariance, the patch is taken from the stack image ($L(x, y, \sigma)$ in Lowe's) which corresponds to the characteristic scale.
- To attain rotation invariance, a canonical patch orientation is computed, so that the descriptor can then be computed on a canonically-oriented patch (i.e. is it necessary to normalize, with respect to rotation, the considered patch. This is achieved by an algorithm which finds a reference system inherent to the patch). The gradient of the given patch allows all this. In particular, to define the reference system, two directions are taken: the prominent gradient direction and the directional normal to the first one:

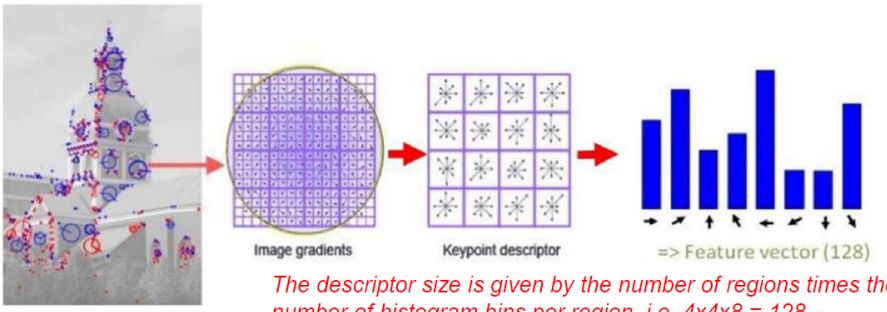



In order to select the prominent gradient direction, a particular orientation histogram is defined, where the $x$-axis is quantized into bins of 10°.

### 9.2.1 Scale Invariant Feature Transform Descriptor (SIFT)

This descriptor is computed as follows:

1. A $16 \times 16$ oriented pixel grid around each key point is considered. This is further divided into $4 \times 4$ regions (each of size $4 \times 4$ pixels).
2. A gradient orientation histogram is created for each region. Each histogram has 8 bins (i.e. the bin size is equal to 45°). Gradients are rotated according to the canonical orientation of the key point.
3. Each pixel in the region contributes to its designated bin according to gradient magnitude as well as to a Gaussian weighting function centred at the key point.




*The descriptor size is given by the number of regions times the number of histogram bins per region, i.e. 4x4x8 = 128.*

array of 128 numbers

## 9.3 Matching of Descriptors

Descriptors are then compared across diverse views of a scene to find corresponding interest points. This represents a classical nearest neighbours problem. In particular, given a target image, $T$, and a reference image, $R$:

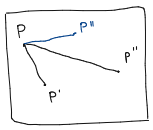
- The features in $T$ represent query points, $q$.
- The features in $R$ provide a set of points $S$ in a specific metric space $M$.
- The metric space, $M$, is the descriptor space endowed by a distance function. When matching SIFT descriptors, $M = R^{128}$, the distance function is typically the Euclidean distance.

In general, given a set $S$ of points in a metric space $M$ and a query point $q \in M$, it is necessary to find the closest point, in $S$, to $q$. In this case, given a feature of $T$, one aims at finding the most similar feature in $R$.
However, some features in $T$ may not have correspondence in $R$. It is then necessary to establish some criteria to accept or reject a ~~math~~ match found by the search process. For example:

$$d_{NN} \leq T_{hreshold} \qquad \text{(Nearest neighbours distance)}$$

$$\frac{d_{NN}}{d_{2-NN}} \leq T_{hreshold} \qquad \text{(Ratio of distances)}$$

second best distance (the lower, the better)




The black point is better than the blue one ($d_{2\text{-}NN}$ is larger)

# Chapter 10

# Object Detection

The object detection problem consists in determining if the object in a given model image is present in a given target image and, in case of detection, estimating the pose. Depending on the application the pose may consists in:

- A translation.
- A roto-translation.
- A similarity (i.e. roto-translation and scaling)

## 10.1 Template Matching

The model image is slid across the target image to be compared at each position to an equally sized window. The difference between the model image and the window is computed by means of a similarity function.


Template matching is, however, exceedingly slow.

This process can be made faster by decreasing the size of the image and of the template.

### 10.1.1 Similarity Functions

The sum of squared differences (SSD) between two given images is computed as follows:

$$\mathrm{SSD}(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(I(i+m,j+n) - T(m,n))^2 \tag{10.1}$$

The sum of absolute differences (SAD) between two given images is computed as follows:

$$\mathrm{SAD}(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}|I(i+m,j+n) - T(m,n)| \tag{10.2}$$

The normalized cross-correlation (NCC) between two given images is computed as follows:

$$\mathrm{NCC}(i,j) = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m,j+n)T(m,n)}{\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m,j+n)^2}\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} T(m,n)^2}} \tag{10.3}$$

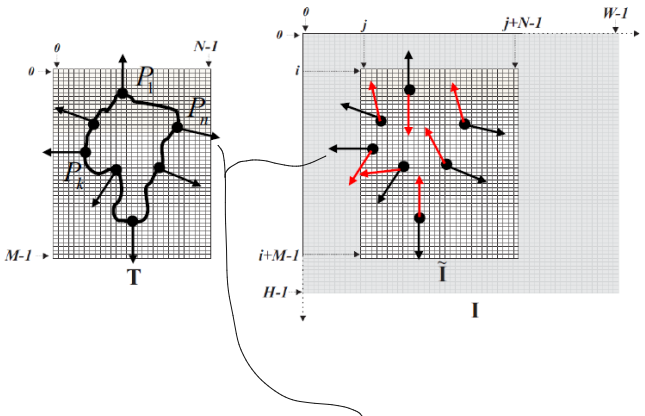
The zero-mean normalized cross-correlation (ZNCC) between two given images is computed as follows:

$$\mathrm{ZNCC}(i,j) = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(I(i+m,j+n) - \mu(\tilde{I}))(T(m,n) - \mu(T))}{\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(I(i+m,j+n) - \mu(\tilde{I}))^2}\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(T(m,n) - \mu(T))^2}} \tag{10.4}$$

In particular, NCC is invariant to linear intensity changes ($\tilde{I}(i,j) = \alpha T$), while ZNCC is invariant to affine intensity changes ($\tilde{I} = \alpha T + \beta$).

## 10.2 Shape-Based Matching

Shape-based matching can be thought of as an edge-based template matching approach:




1. A set of control points, $P_k$, is extracted from the model image by and edge detection operation and the gradient direction at each point is recorded.

2. At each position $(i, j)$ of the target image, the recorded gradient directions associated with the control points are compared to those at their corresponding image points, $P_k(i, j)$, in order to compute a similarity function.

unit vector (gradient normalized with its magnitude)

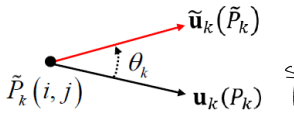
$$\mathbf{G}_k(P_k) = \begin{bmatrix} I_x(P_k) \\ I_y(P_k) \end{bmatrix}, \mathbf{u}_k(P_k) = \frac{1}{\|\mathbf{G}_k(P_k)\|} \begin{bmatrix} I_x(P_k) \\ I_y(P_k) \end{bmatrix}, k = 1..n$$

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

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

number of edges

the more the gradient are aligned, the higher is the similarity score




cosine similarity (invariance to intensity)

("hat")

## 10.3 Hough Transform

– usually applied after edge detection (input data consists of the edge pixels)

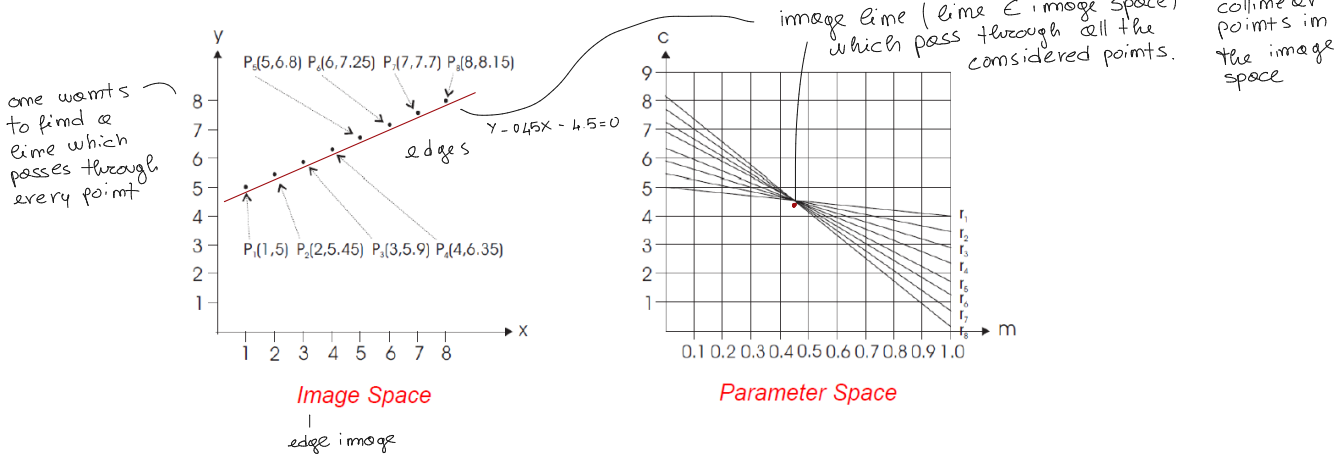
The Hough Transform enables to detect objects having a known shape based on projection of the input data into a suitable space referred to as *parameter* or *Hough space*. This method was first invented to detect lines and then extended by including other analytical shapes.

example of a line

Given the equation of a generic line, $y - mx - c = 0$, one can fix $y$ and $x$, instead of $m$ and $c$, and find all possible lines through the specific point at $(x, y)$. If one considers $n$ collinear points (i.e. points of the same line), their corresponding lines, in the parameter space, will intersect in a single point, which represents the image line passing through all the $n$ considered points:

collinear points in the image space

image line (line ∈ image space) which pass through all the considered points.



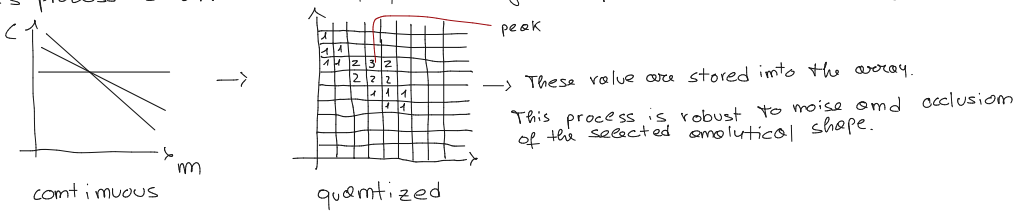

edge image

The Hough Transform consists of mapping image points (usually edge points) to their corresponding curves into the parameter space and then finding intersections between these curves. The more the curves intersect, the higher the evidence of that line in the considered image.

example of a circle

The same process is repeated for circle detection. In particular, the equation of circular objects, $(x-a)^2 + (y-b)^2 = r^2$ is interpreted as a mapping from the image space $(x, y)$ to the parameter space

* This process is carried out by quantizing the parameter space, using an accumulator array:

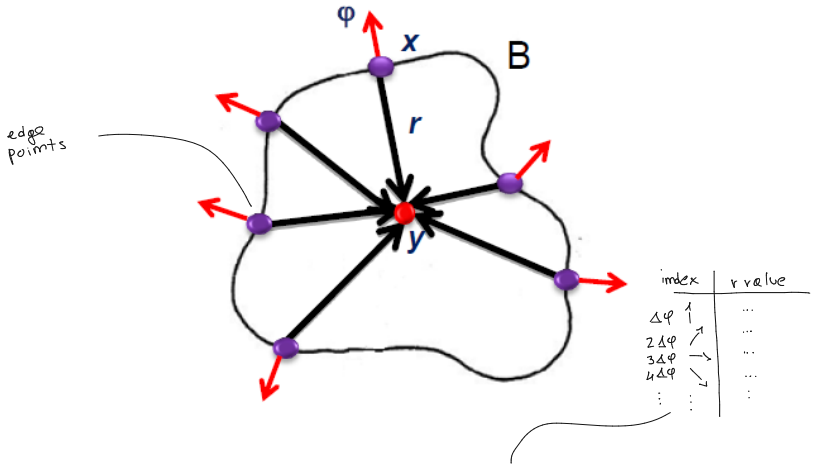



$(a, b, r)$. By fixing $x$ and $y$, one can obtain all the parameters triplets representing circles through image point $(x, y)$. One can them find intersections, finding that the particular circle is present in the input image.
*

### 10.3.1 Generalized Hough Transform

The Hough Transform is then been extended to detect arbitrary (i.e. non analytical) shapes. Given a non-analytical shape, this method operates in two different phases:

- The off-line phase, used to build the object's model:
  1. A reference point $y$ is chosen (e.g. barycentre).
  2. The gradient direction is quantized according to a chosen step $\Delta\varphi$.
  3. For each point $x$ belonging to the object's border $B$:
     (a) The gradient direction, $\varphi(x)$, is computed.
     (b) The $r$ vector, from $y$ to $x$ (i.e. $r = y - x$) is computed.




e.g.  no more directions are allowed

  4. The $r$ vector is stored in a specific table (R-table) as a function of $\Delta\varphi$.

| i | $\phi_i$ | $R_{\phi_i}$ |
|---|---|---|
| 0 | 0 | $\{r \mid y - r = x, x \in B, \phi(x) = 0\}$ |
| 1 | $\Delta\phi$ | $\{r \mid y - r = x, x \in B, \phi(x) = \Delta\phi\}$ |
| 2 | $2\Delta\phi$ | $\{r \mid y - r = x, x \in B, \phi(x) = 2\Delta\phi\}$ |
| ... | ... | ... |

*An entry in the R-Table can contain several r vectors.*

- The on-line phase, used to detect the specific object:
  1. An image, $A[y]$, is initialized as accumulator array.
  2. For each pixel $x$ of the input image:
     (a) The gradient direction, $\varphi(x)$, is computed.

this process allows one to find the most suitable barycentre, and thus to detect the sought target in the image.

(b) The gradient direction is quantized to index the computed R-table.

(c) For each $r_i$ vector stored into the accessed row:

i. The position $y = x + r_i$ is computed. — y is the reference point (i.e. the barycentre)

ii. A vote is casted, $A[y] + +$, into the accumulator array.

3. Instances of the sought object are detected by finding peaks of the accumulator array.

In order to handle rotation and scaling, at point 2.(c).i, $y$ is computed as: $y = x + s \cdot r_\theta$. In this case, $A$ is a three-dimensional array and, at point 2.(c).ii the vote is casted as follows: $A[y, \theta, s] + +$.

one has to consider all the possible rotations

one has to consider all the possible scaling

usually infeasible