ECN – **Co**mputer **Vis**ion – Lecture Notes

Davide Lanza

2018-2019

Contents

Chapter 1

Introduction to CoVis

1.1 What is computer vision

Computer vision deals with the automatic extraction of "meaningful" information from images and videos, as shown in Figure 1.1. In this course we will deal only with **geometric information**.

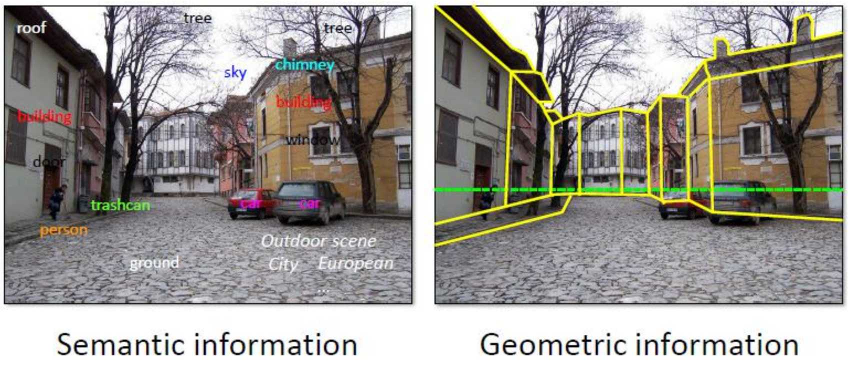


Figure 1.1: Two types of information extraction

But why study computer vision? CoVis is useful for many goals, like relieving humans of boring and easy tasks; enhance human abilities (human-computer interaction, visualization, augmented reality); organize and give access to visual content; and (for us most important) perception for **autonomous robots**. In fact in an Artificial system we have three phases: (...→) **Perception** → Learning → Action (→ Perception →...) and CoVis deals with perception problems.

1.2 Vision in humans

Vision is our most powerful sense. Half of primate cerebral cortex is devoted to visual processing. Retina is ~1,000 mm^2. It contains 130 million photo-receptors (120 mil. rods (low light vision) and 10 mil. cones for color sampling) and it provides enormous amount of information: data-rate of ~3GBytes/s.

To match the eye resolution we would need a 500 Megapixel camera. But in practice the acuity of an eye is **8 Megapixels** within a **18-degree field of view** (5.5 mm diameter) region called **fovea**.

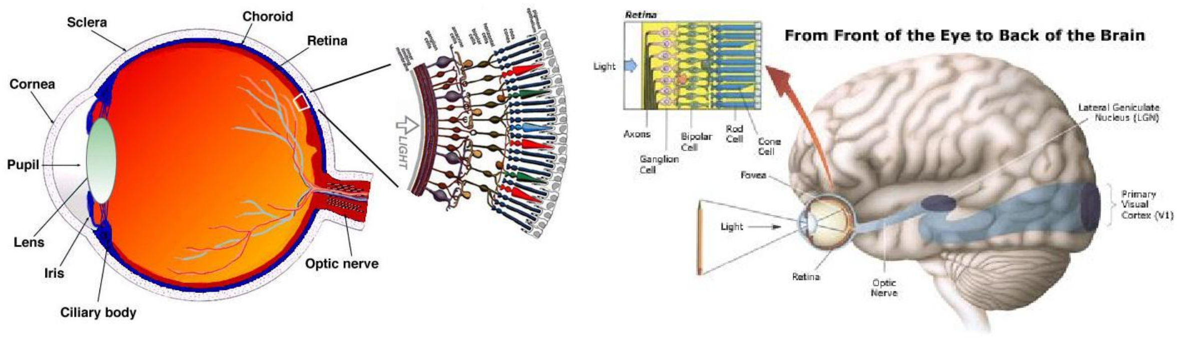


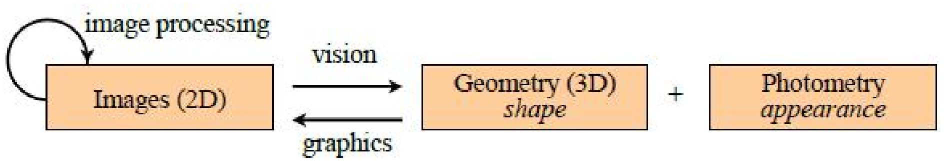
Figure 1.2: The human vision system

CoVis is hard because it deals with the question "How do we go from an array of number (digital image) to recognizing an object?".

CoVis research was born in the 60s, and one of the earliest articles was by L. G. Roberts (*Machine Perception of Three Dimensional Solids* packet.cc/files/mach-per-3D-solids.html), a Ph.D. thesis for the MIT Department of Electrical Engineering, published in 1963. Another important publication was by the Artificial Intelligence group of MIT: in 1966 Semyour Papert published *The summer vision project*.

1.3 Computer Vision vs Computer Graphics

What is the difference between Computer Vision and Computer Graphics? CoVis deals on how to describe the world that we see in one or more images and to reconstruct its properties, such as shape, illumination, and color distributions, so it is structured to solve an **inverse problem**: recover some unknowns given insufficient information to fully specify the solution:



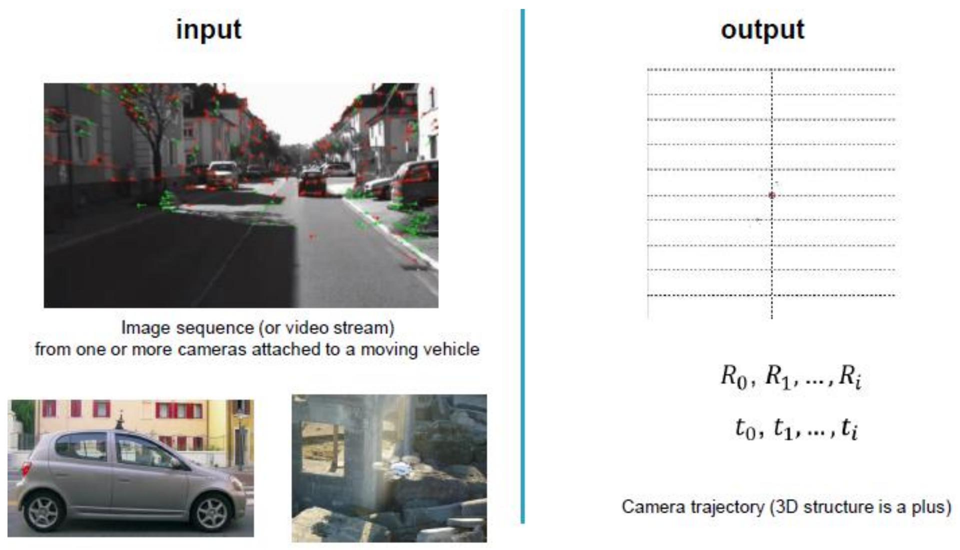
Computer Vision = Physics (radiometry, optics, sensor design) + Computer Graphics (3D modeling, rendering, animation)

Chapter 2

Visual Odometry

Intelligent systems require **robust vision**, characterized by feature invariance, good prior, tractable representations, efficient learning and inference and model uncertainty.

Visual Odomentry (VO) is the process of incrementally estimating the pose of the vehicle by examining the changes that motion induces on the images of its onboard cameras. It is based on the idea that humans do not process all the information from the eye, but with them they normally detect only the variations in the vision environment:


input
Image sequence (or video stream)
from one or more cameras attached to a moving vehicle
output
$R_0, R_1, \dots, R_i$
$t_0, t_1, \dots, t_i$
Camera trajectory (3D structure is a plus)

But why VO? Contrary to wheel odometry (see MobRo course), VO is not affected by **wheel slippage** on uneven terrain or other adverse conditions. It also leads to **more accurate trajectory** estimates compared to wheel odometry (relative position error 0.1%-2%). VO can be used also as a **complement to wheel encoders** (wheel odometry), GPS, inertial measurement units (IMUs), laser odometry and it is crucial for flying, walking, and underwater robots

For VO we need 4 assumptions: **sufficient illumination** in the environment, **dominance of static scene** over moving objects, **enough texture** to allow apparent motion to be extracted and sufficient **scene overlap** between consecutive frames.

2.1 A brief history of VO

1980: First known VO real-time implementation on a robot by Hans Moraveck PhD thesis (NASA/JPL) for Mars rovers using one sliding camera (sliding stereo).

1980: First known VO real-time implementation on a robot by Hans Moraveck PhD thesis (NASA/JPL) for Mars rovers using one sliding camera (sliding stereo).

1980 to 2000: The VO research was dominated by NASA/JPL in preparation of the 2004 mission to Mars.

2004: VO was used on a robot on another planet: Mars rovers Spirit and Opportunity (see seminal paper from NASA/JPL, 2007).

2004: VO was revived in the academic environment by David Nister's "Visual Odometry" paper. The term VO became popular.

To a full introduction see:

- Scaramuzza, D., Fraundorfer, F., **Visual Odometry: Part I** - The First 30 Years and Fundamentals, *IEEE Robotics and Automation Magazine*, Volume 18, issue 4, 2011.
- Fraundorfer, F., Scaramuzza, D., **Visual Odometry: Part II** - Matching, Robustness, and Applications, *IEEE Robotics and Automation Magazine*, Volume 19, issue 1, 2012.
- C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I.D. Reid, J.J. Leonard, **Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age**, *IEEE Transactions on Robotics*, Vol. 32, Issue 6, 2016.

2.2 From SFM to VO

Structure for Motion (SFM) is more general than VO and tackles the problem of 3D reconstruction and 6DOF pose estimation from unordered image sets. We can see an example in Figure 2.1.

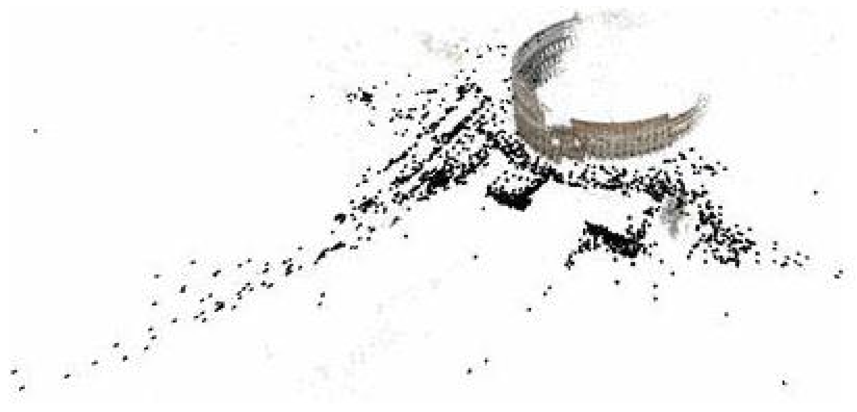
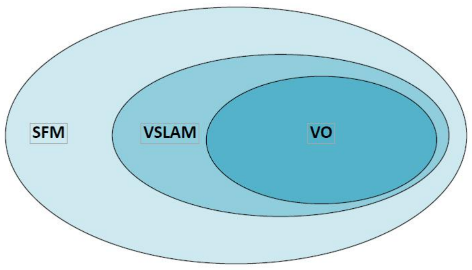


Figure 2.1: Reconstruction from 3 million images from *Flickr.com.* Cluster of 250 computers, 24 hours of computation. Paper: *Building Rome in a Day*, ICCV'09

VO is a particular case of SFM, that focuses on estimating the 6DoF motion of the camera **sequentially** (as a new frame arrives) and in real time. Be careful though because terminology: sometimes SFM is used as a synonym of VO!



While VO focuses on incremental estimation/**local consistency**, another technique called **Visual Simultaneous Localization And Mapping** (V-SLAM) focuses on globally consistent estimation.

$$\text{V-SLAM} = \text{VO} + \text{loop detection} + \text{loop closure}$$

The choice between VO and V-SLAM depends on the tradeoff between performance and consistency, and simplicity of implementation. VO trades off consistency for real-time performance, without the need to keep track of all the previous history of the camera (see Figure 2.2).

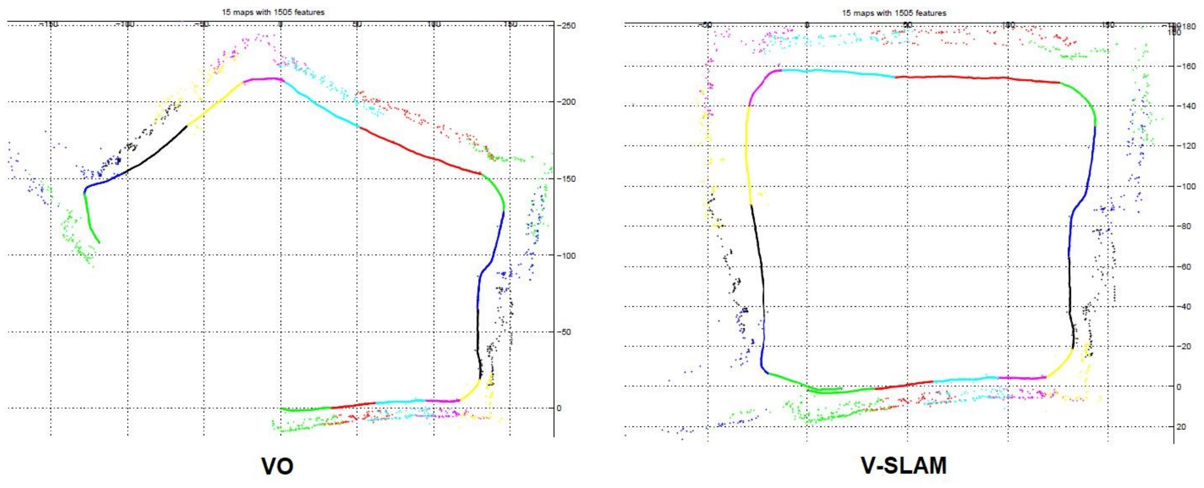


Figure 2.2: VO vs V-SLAM

2.3 VO working principle

In this section we will illustrate the steps of VO computation:

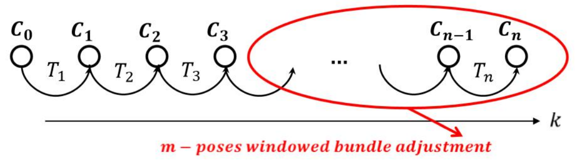
1. Compute the relative motion T_k from images I_{k-1} to image I_k:

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix}$$

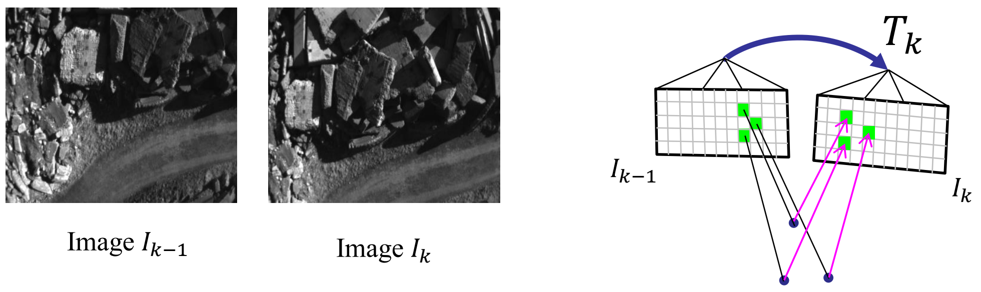
2. Concatenate them to recover the full trajectory of 6DoF poses C_n:

$$C_n = C_{n-1}T_n$$

3. An optimization over the last m poses can be done to refine locally the trajectory (Pose-Graph or Bundle Adjustment):



But how do we estimate the relative motion T_k (step 1)? We do it in this way:

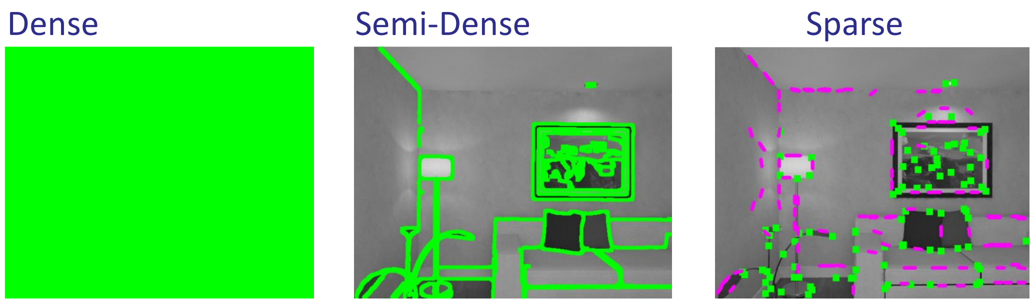


$$T_k = \arg\min_T \int\int_{\bar{\mathcal{R}}} \rho\left[I_k\left(\pi\left(\boldsymbol{T}\cdot\pi^{-1}\left(\boldsymbol{u}, d_{\boldsymbol{u}}\right)\right)\right) - I_{k-1}(\boldsymbol{u})\right] d\boldsymbol{u}$$

We have some direct methods for dense geometric reconstruction (see D. Cremers, **Direct methods for 3D reconstruction and visual SLAM**, *International Conference on Machine Vision Applications*, 2017) to simplify the the **image alignment** process. The **direct image alignment** minimizes the sum of per-pixel intensity difference:

$$T_{k,k-1} = \arg\min_T \sum_i \|I_k(\boldsymbol{u'}_i) - I_{k-1}(\boldsymbol{u}_i)\|_\sigma^2$$

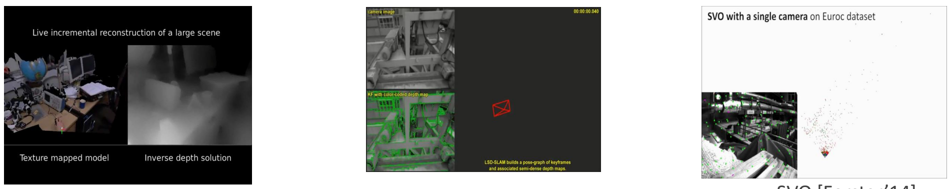
($\boldsymbol{u}'$ and $\boldsymbol{u}_i$ are corresponding pixels in images I_k and I_{k-1}, respectively)



DTAM [Newcombe et al. '11]
300'000+ pixels

LSD [Engel et al. 2014]
~10'000 pixels

SVO [Forster et al. 2014]
100 features × 4 × 4 patches
~ 2,000 pixels


Live incremental reconstruction of a large scene
Texture mapped model
Inverse depth solution
SVO with a single camera on Euroc dataset

DTAM [Newcombe '11] REMODE [Pizzoli'14]
300'000+ pixels

LSD-SLAM [Engel'14]
~10,000 pixels

SVO [Forster'14]
100-200 x 4x4 patches ≅ 2,000 pixels

VO computes the camera path incrementally (pose after pose) as shown in the flowchart in Figure 2.3.

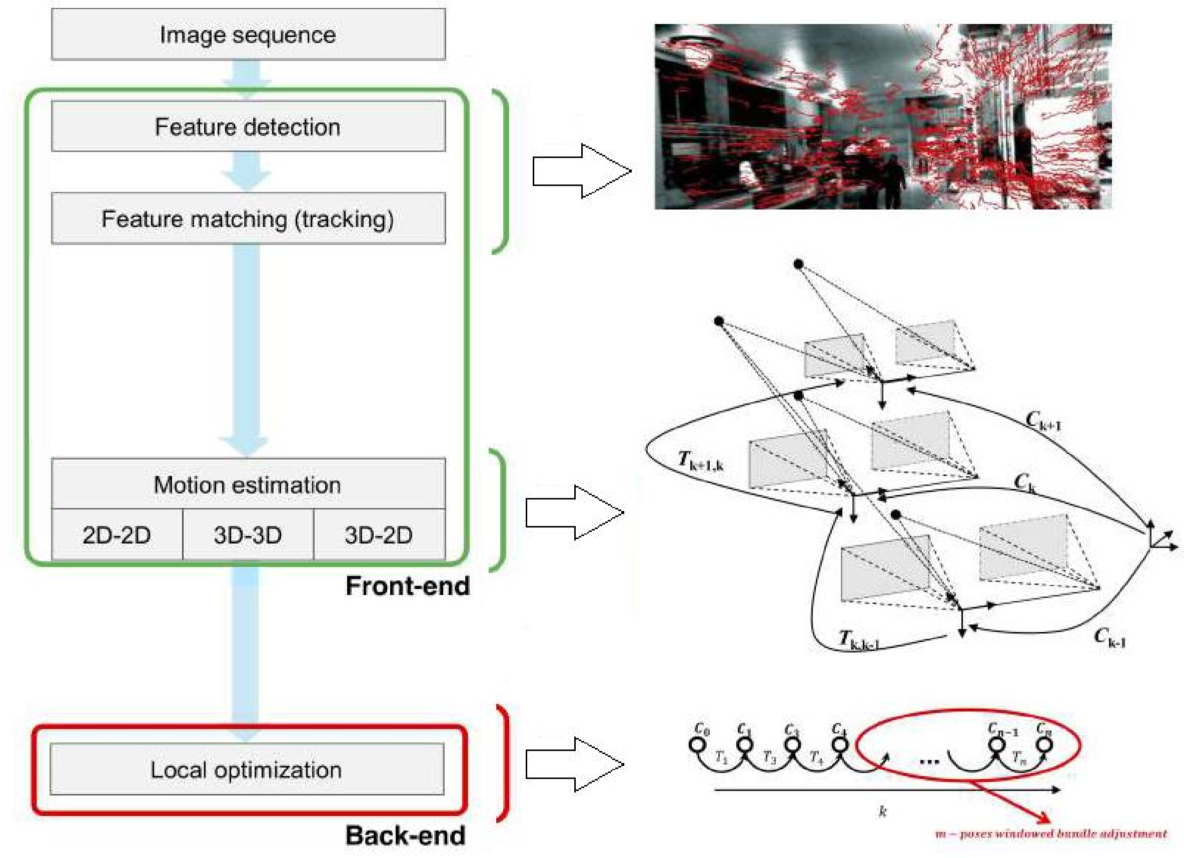


Figure 2.3: VO Flowchart

Chapter 3

Image Formation

3.1 Overview on optics

Historical context:
– **Pinhole model**: Mozi (470-390 BCE), Aristotle (384322 BCE)
– Principles of optics (including lenses): Alhacen (965-1039)
– **Camera obscura**: Leonardo da Vinci (1452-1519), Johann Zahn (1631-1707)
– **First photo**: Joseph Nicephore Niepce (1822)
– Daguerrotypes (1839)
– Photographic film (Eastman, 1888, founder of Kodak)
– Cinema (Lumiere Brothers, 1895)
– Color Photography (Lumiere Brothers, 1908)
– Television (Baird, Farnsworth, Zworykin, 1920s)
– First consumer camera with CCD: Sony Mavica (1981)
– **First fully digital camera**: Kodak DCS100 (1990)

The question behind the image formation study is “how are objects in the world captured in an image?”.

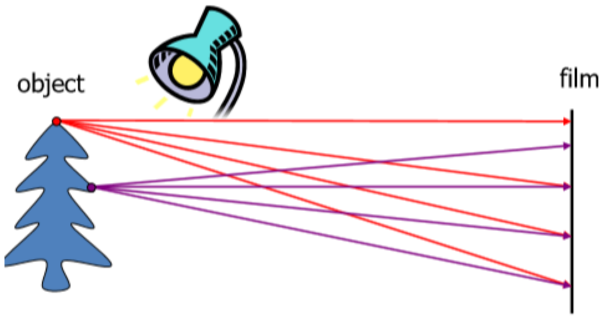
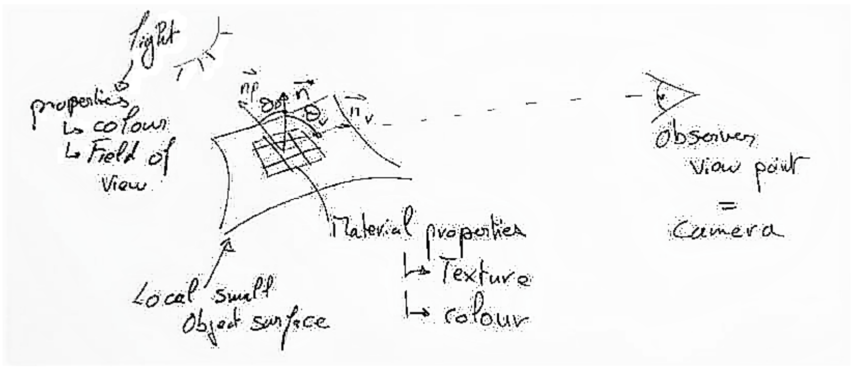


Figure 3.1: *Naive* approach

If we place a piece of film in front of an object, as shown in Figure 3.1, we don’t get a reasonable image.

In computer graphics we have **light-material interaction models** (e.g. Shading model): to have more photo-realistic view in computer graphics you need to compute those models, but you need a powerful GPU for that, because you have to compute the reflection, refraction, diffusion... of the light while interacting with the surface in w.r.t. the observer view point. The 3D rendering of the image is hard to compute because there are many factor to take in consideration:



Based on this model, the **pinhole camera** has been created. How we can project a 3D image on that surface. The idea will be to "catch" only a small part of the light. In fact, in a pinhole camera we add a barrier to block off most of the rays to reduce the blurring. Here the opening is known as the **aperture**. If i want to increase the size of my viewpoint i have to increase my hole i have to increase the aperture, but I will notice a worse blurring effect. In fact in an ideal pinhole, only one ray of light reaches each point on the film, so the image can be very dim, but making the aperture bigger makes the image blurry (see Figure 3.3).

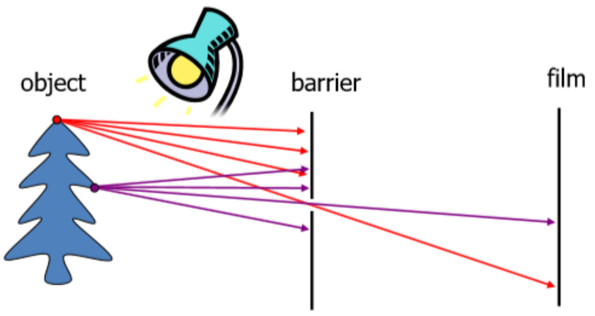


Figure 3.2: Pinhole camera

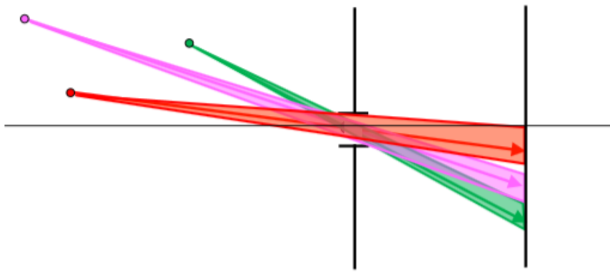


Figure 3.3: Blurring problem

So the main solution, later on, was to catch all the needed lights with a **lens** (Figure 3.4). In here a lens focuses light onto the film. Notice that the rays passing through the **Optical Center** are not deviated. One of the parameter that we gave to take into consideration while processing the image is the focal length f (Figure 3.5), the distance of the point where all the rays converge.

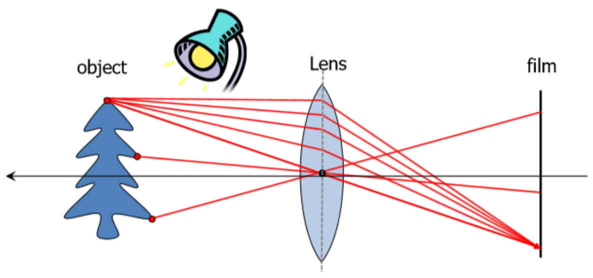


Figure 3.4: Image formation using a converging lens

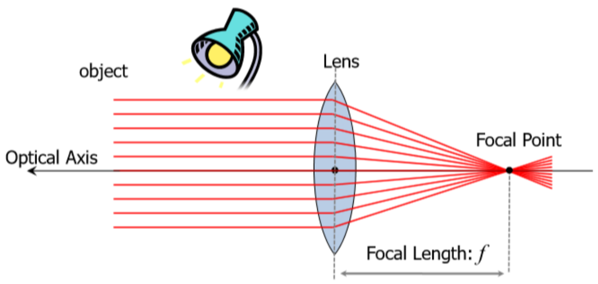


Figure 3.5: Converging lens model

So, let's analyze a thin lens equation. We have the situation illustrated by Figure 3.6 and we want to find a relationship between the **focal length** f, the **distance object-lens** z and the **distance film-lens** e.

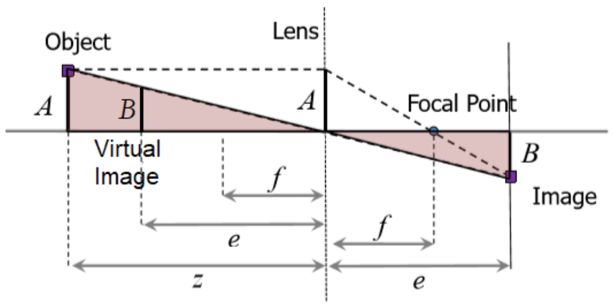


Figure 3.6: Lens geometrical model

We have then the following equation, using similar triangles properties:

“Thin lens equation”

$$\left.\begin{aligned} \frac{B}{A} &= \frac{e}{z} \\ \frac{B}{A} &= \frac{e-f}{f} = \frac{e}{f} - 1 \end{aligned}\right\} \frac{e}{f} - 1 = \frac{e}{z} \Rightarrow \boxed{\frac{1}{f} = \frac{1}{z} + \frac{1}{e}}$$

Any object point satisfying this equation is in focus. So, can we use this to measure distances?

For a **fixed** film distance from the lens e_0, there is a specific distance between the object and the lens, at which the object appears **in focus** in the image. The other points project to a “blur circle” in the image, as seen in Figure 3.7.

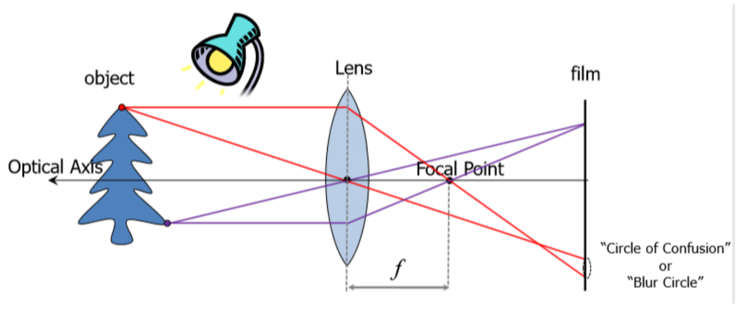


Figure 3.7: Lens focus

For an object out of focus the **blur circle** has radius:

$$R = \frac{L\delta}{2e}$$

So a small L (pinhole) gives a small R (Blur Circle). To capture a "good" image we have to adjust camera settings, such that R remains smaller than the image resolution. But what happens if $z >> f$ and $z >> L$? We have what we call the **Pin-hole approximation** (Figure 3.8).

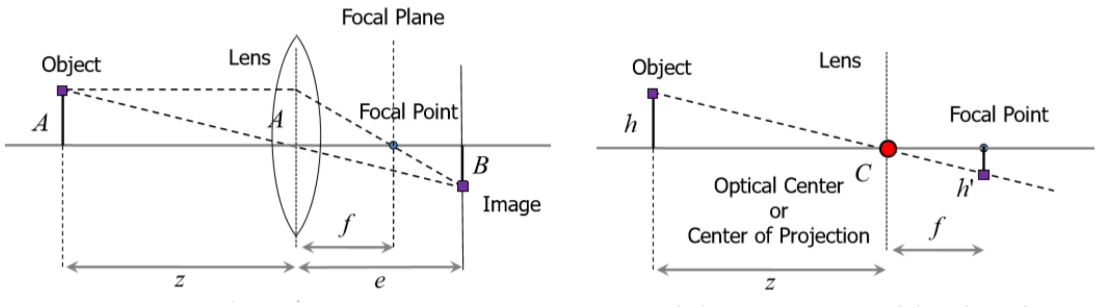


Figure 3.8: The Pin-hole approximation

So, we need to adjust the image plane such that objects at infinity are in focus. As the object gets far, the image plan gets closer to the focal plane:

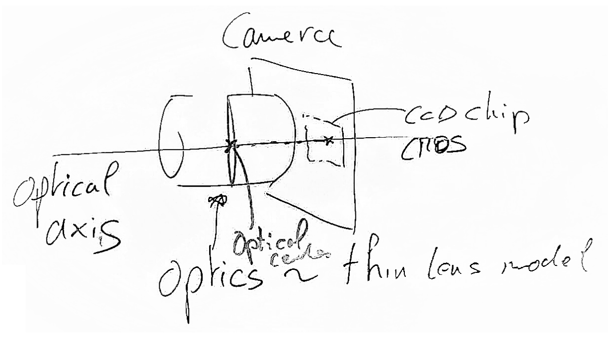
$$\frac{1}{f} = \underbrace{\frac{1}{z}}_{\simeq 0} + \frac{1}{e} \rightarrow \frac{1}{f} \simeq +\frac{1}{e} \rightarrow f \simeq e$$

This is known as Pinhole Approximation and the relation between the image and object becomes:

$$\frac{h'}{h} = \frac{f}{z} = \!\!\rightarrow h' = \frac{f}{z}h$$

The dependence of the image of an object on its depth (i.e. distance from the camera) is known as **perspective**. f/z is called **scale ambiguity factor** and determines the relation between the 3D and the 2D projected image. To process the images and get this information (recovering h from h') we need, for example, two images in *parallax* ($\rightarrow$ **visual odometry**).

In electronic cameras the CCO chip/CMOS sensor distance from the lens is important in order to maintain the focus on the object, so the glue that links it to the camera body has to be of good quality and assure no movements:



3.2 Perspective geometry

In image processing for controlling a robot, modelling explicitly the perspective effect is a big problem (Figure 3.9: how can I detect the same object if way smaller to another because farther or how can I know that the lines in a landscape does not converge in the *vanishing point* on the horizon line?)

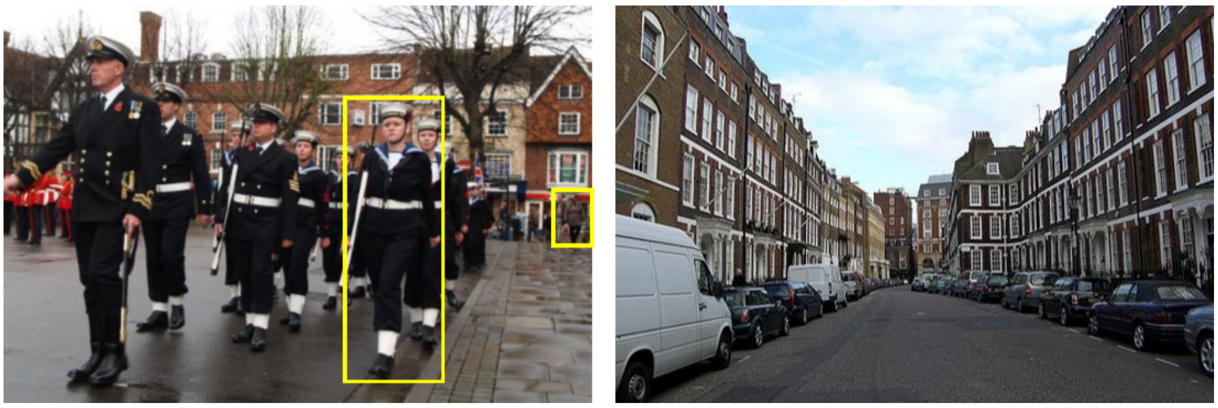


Figure 3.9: Perspective problems

But in the perspective projection, what is preserved? The straight lines are still straight. But what is lost? Lengths and angles (see Figure 3.10).

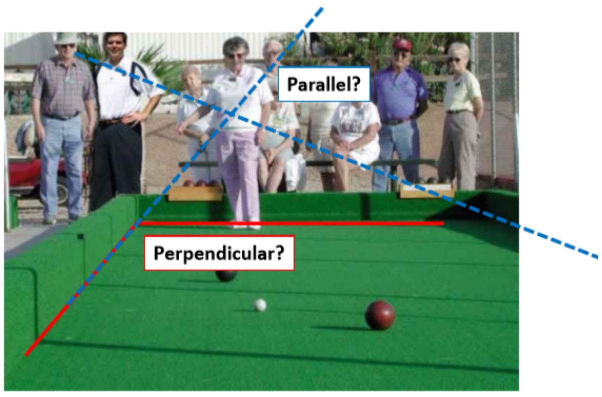


Figure 3.10: Perspective projection lines

Parallel lines in the world intersect in the image at a "**vanishing point**" and parallel planes in the world intersect in the image at a "vanishing line" (Figure 3.11). To process those "infinite" points represented in the perspective figures we cannot use Euclidean geometry.

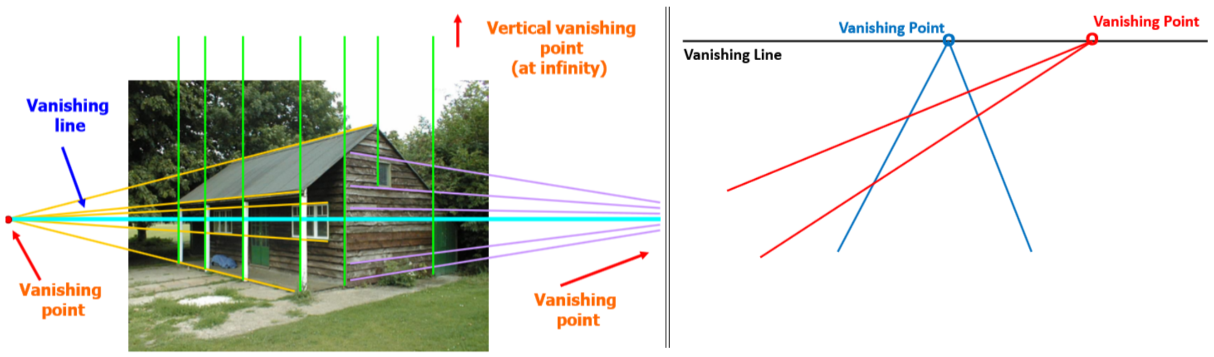


Figure 3.11: Vanishing points and lines

Let's consider a straight line $\boldsymbol{l}$ in an image and a point $\boldsymbol{x}$ on it.

In 2D $\boldsymbol{x} = [x \quad y]^T$ and $\boldsymbol{l} = [a \quad b \quad c]^T$. We know that $\boldsymbol{x} \in \boldsymbol{l} \iff ax + by + c = 0$. So, the 2D algebraic line eq. is equal to his Cartesian representation, so we can write:

$$ax + by + c = [a \quad b \quad c] \cdot [x \quad y \quad 1]^T = \boldsymbol{l}^T \cdot [x \quad y \quad 1]^T = \boldsymbol{l}^T \cdot [\boldsymbol{x} \quad 1]^T = 0$$

We know that $[\boldsymbol{x} \quad 1]^T$ is the **homogeneous vector** of the point $\boldsymbol{x}$. So:

$$l^T \cdot [\boldsymbol{x} \quad 1]^T = 0 = \lambda \boldsymbol{l}^T \cdot [\boldsymbol{x} \quad 1]^T \quad \forall \lambda \in \mathbb{R} \qquad \rightarrow \qquad \boldsymbol{l}^T \cdot [\lambda \boldsymbol{x} \quad \lambda]^T = 0$$

So, as we can see, we are able to treat the scale factor ambiguity thanks to the homogeneous vector: we just need to divide the first two rows (vector $\lambda \boldsymbol{x}$) for the third row (λ) and we obtain the original point. We define then the 2D homogeneous points **projective space** as $\mathbb{P}^2 = \mathbb{R} - (0,0,0)^T$. So we will have:

2D points (pxl. coords in an image)	$\boldsymbol{x} = (x \quad y)^T$	where $(x, y) \in \mathbb{R}^2$
2D homogeneous points	$\tilde{\boldsymbol{x}} = (\tilde{x} \quad \tilde{y} \quad \tilde{w})^T$	where $(\tilde{x}, \tilde{y}, \tilde{w}) \in \mathbb{P}^2$

$$\begin{cases} x = \dfrac{\tilde{x}}{\tilde{w}} \\ y = \dfrac{\tilde{y}}{\tilde{w}} \end{cases} \rightarrow \text{ perspective propjection } \rightarrow h' = \frac{f}{z} h$$

So we can rewrite the previous equation like this:

$$\tilde{\boldsymbol{x}} = [\tilde{x} \quad \tilde{y} \quad \tilde{w}]^T \qquad \boldsymbol{l} = [a \quad b \quad c]^T \qquad \tilde{\boldsymbol{x}} \in \boldsymbol{l} \iff \boldsymbol{l}^T \tilde{\boldsymbol{x}} = \tilde{\boldsymbol{x}}^T \boldsymbol{l} = 0$$

We can then express a normal vector with different notations:

$$\tilde{\boldsymbol{l}} = (\hat{n}_x; \hat{n}_y; d)^T = (\hat{\boldsymbol{n}}, d) \text{ with } ||\hat{\boldsymbol{n}}|| = 1.$$

or, setting $\hat{\boldsymbol{n}} = (\cos\theta; \sin\theta)^T$, we can express it like:

$$(\hat{\boldsymbol{n}}, d)^T \qquad \text{Polar coordinates}$$

The **line at infinity** is the one that contains all the (ideal) points at infinity and it **cannot** be normalized:

$$\tilde{\boldsymbol{m}} = (0, 0, 1)^T$$

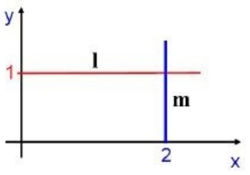
– **Intersection and join operators:**

We said that the point $\boldsymbol{x}$ lies on the line $\boldsymbol{l}$ if and only if $\boldsymbol{l}^T\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{x}}^T\boldsymbol{l} = 0$.

We define also an **intersection** operator that allows us to find the intersection of two lines $\boldsymbol{l}$ and $\boldsymbol{l}'$ is $\tilde{\boldsymbol{x}} = \boldsymbol{l} \times \boldsymbol{l}'$:

$$\boldsymbol{l} \times \boldsymbol{l}' = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ l_1 & l_2 & l_3 \\ l_1' & l_2' & l_3' \end{vmatrix} = \begin{bmatrix} +(l_2l_3' - l_2'l_3) \\ -(l_1l_3' - l_1'l_3) \\ +(l_2l_1' - l_1'l_2) \end{bmatrix}$$

Let's see an example : compute the point of intersection of the two lines $\boldsymbol{l}$ and $\boldsymbol{m}$ in the figure below:



$$\mathrm{l} = \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix} \quad \mathrm{m} = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} \quad \mathrm{x} = \mathrm{l} \times \mathrm{m} = \begin{pmatrix} -2 \\ -1 \\ -1 \end{pmatrix}$$

is the point $(2, 1)$

We can also "define" a **join** operator that returns the line joining two points $\boldsymbol{x}$ and $\boldsymbol{x}'$:

$$\boldsymbol{l} = \tilde{\boldsymbol{x}} \times \tilde{\boldsymbol{x}}'$$

That's the same computation but with points instead of lines!

Let's notice that in the case of an **intersection of parallel lines** if $\boldsymbol{l} = (a, b, c)^T$ and $\boldsymbol{l}' = (a, b, c')^T$ then

$$\boldsymbol{l} \times \boldsymbol{l}' = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ a & b & c \\ a & b & c' \end{vmatrix} = \begin{bmatrix} bc' - bc \\ -(ac' - ac) \\ ab - ab \end{bmatrix} = \begin{bmatrix} b(c' - c) \\ -a(c' - c) \\ 0 \end{bmatrix}$$

So we have a point $\boldsymbol{x}$ in the form $\boldsymbol{l} \times \boldsymbol{l}' = (x; y; 0)^T \rightarrow$ we have a **point at infinite** $(\in \boldsymbol{l}_\infty)$. .

We can also say that our peojective space is $\mathbb{P}^2 = \mathbb{R}^2 \cup \boldsymbol{l}_\infty \rightarrow$ in $\mathbb{P}^2$ there is no distinction between ideal points and other.

So, if we want to compute the join of 2 points (the line passing through the two points) we just do:

$$l \underset{\substack{\text{equal up to a} \\ \text{scale factor}}}{\sim} \tilde{\boldsymbol{x}}_1 \times \tilde{\boldsymbol{x}}_2 = [x_1 \quad y_1 \quad 1]^T \times [x_2 \quad y_2 \quad 1]^T = \begin{bmatrix} y_2 - y_1 \\ x_2 - x_1 \\ x_1x_2 - x_2y_1 \end{bmatrix} = [a \quad b \quad c]^T$$

Let's consider the case of a point at infinite ($\tilde{w} = 0$):

$$\tilde{\boldsymbol{x}} = [\tilde{x} \quad \tilde{y} \quad \tilde{w} = 0]^T \rightarrow \tilde{\boldsymbol{x}} = [\tilde{x}/\tilde{w} \quad \tilde{y}/\tilde{w}]^T = [\infty \quad \infty]^T$$

We can notice that, taking a random point at infinite:

$$\tilde{\boldsymbol{x}}_\infty = [-b \quad a \quad 0]^T \qquad \tilde{\boldsymbol{x}}_\infty^T \boldsymbol{l}_\infty = [-b \quad a \quad 0][0 \quad 0 \quad 1]^T = 0$$

We can have a **model for the projective plane** (Figure 3.12) in which we can notice exactly one line through two points and exactly one point at intersection of two lines.

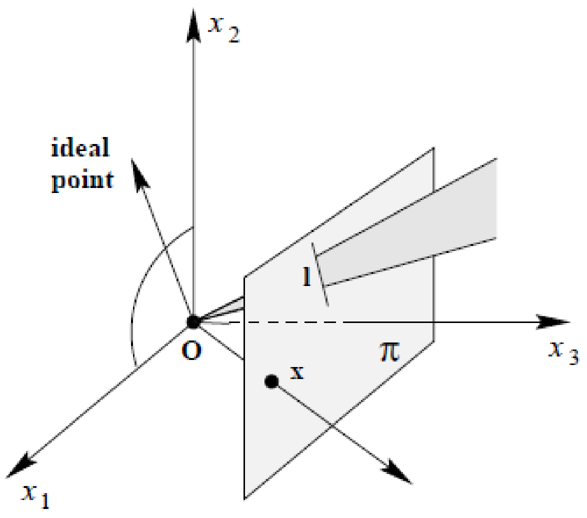


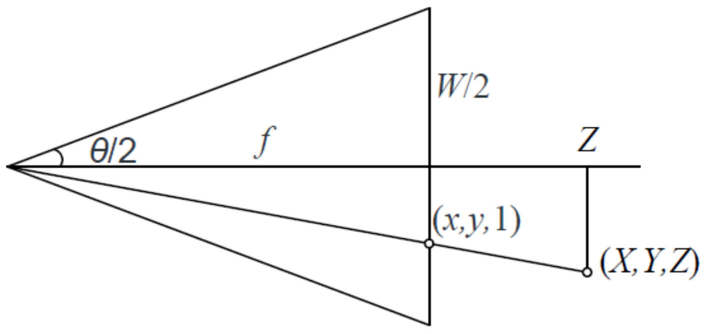
Figure 3.12: Model for the projective plane

Let's notice that there is a relation about points, lines and homogeneous coordinates:

$$\begin{array}{rcl} \text{point} & \Longleftrightarrow & \text{line} \\ \mathbf{x} & \Longleftrightarrow & \mathbf{l} \\ \mathbf{x}^T\mathbf{l} = 0 & \Longleftrightarrow & \mathbf{l}^T\mathbf{x} = 0 \\ \mathbf{x} = \mathbf{l} \times \mathbf{l}' & \Longleftrightarrow & \mathbf{l} = \mathbf{x} \times \mathbf{x}' \end{array}$$

Duality principle: To any theorem of 2-dimensional projective geometry there corresponds a dual theorem, which may be derived by interchanging the role of points and lines in the original theorem.

There is a **link** between the **field of view** and the **focal length**, as shown in Figure 3.13.



$$\tan\frac{\theta}{2} = \frac{W}{2f} \text{ or } f = \frac{W}{2}[\tan\frac{\theta}{2}]^{-1}$$

Figure 3.13: Field of view (θ), diameter of lens (W), the distance object-lens (Z) and the focal length (f)

3.3 Digital cameras

In digital cameras the film is substituted by a sensor array, that is often an array of charge coupled devices in which each CCD/CMOS is light sensitive diode that converts photons (light energy) to electrons (see Figure 3.14).

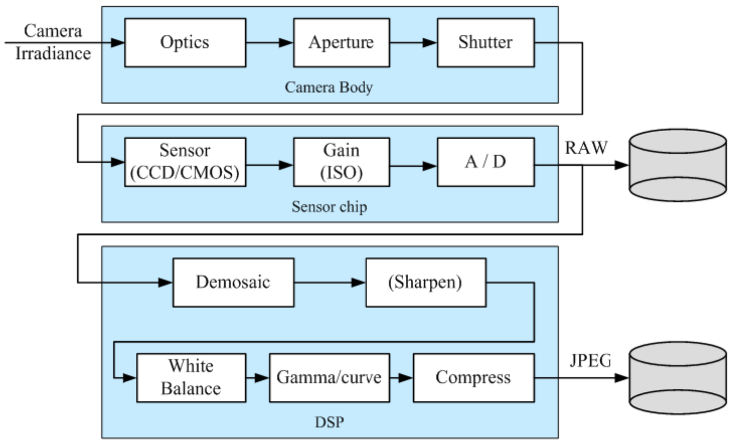


Figure 3.14: Image sensing pipeline, showing the various sources of noise as well as typical digital post-processing steps.

We will consider images in a matrix form like shown in Figure 3.15.

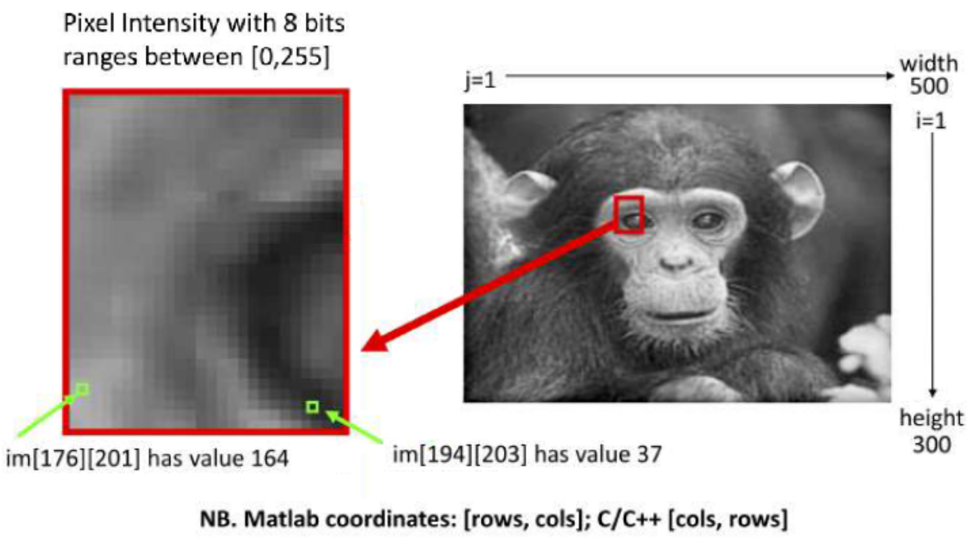

Pixel Intensity with 8 bits ranges between [0,255]
j=1
width 500
i=1
height 300
im[176][201] has value 164
im[194][203] has value 37
NB. Matlab coordinates: [rows, cols]; C/C++ [cols, rows]

Figure 3.15: Digital image characteristics

Regarding the color sensing in digital cameras, we have to know that the color sensor in the camera is built respecting the **Bayer pattern** (invented by Bayer in 1976, who worked at Kodak), a pattern that places green filters over half of the sensors (in a checkerboard pattern), and red and blue filters over the remaining ones (See Figure 3.16).

This is because the luminance signal is mostly determined by green values and the human visual system is much more sensitive to high frequency detail in luminance than in chrominance.

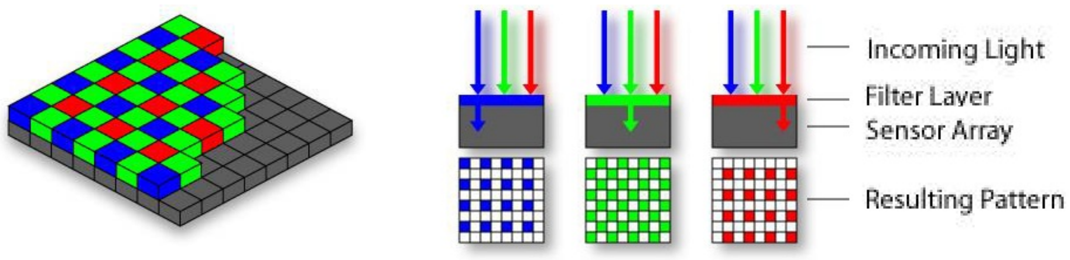
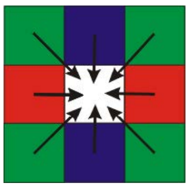


Figure 3.16: Bayern pattern sensor disposition and functioning

For each pixel, the sensor estimate the missing color components from neighboring values (**demosaicing**):



There is also another chip design: the **Foveon** chip design (http://www.foveon.com), that stacks the red, green, and blue sensors beneath each other but has not gained widespread adoption.

So, in the end, we have an image that is created from three monochromatic images, as seein in Figure 3.17.

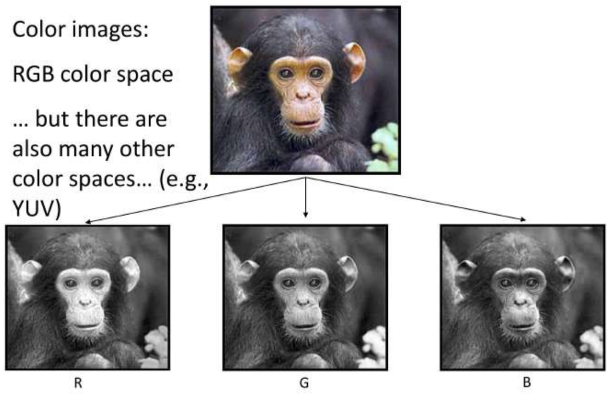

Color images:
RGB color space
... but there are also many other color spaces... (e.g., YUV)
R
G
B

Figure 3.17: RGB color space

3.4 Perspective camera model

We want to use a **camera as a sensor**. For convenience, the image plane is usually represented in front of C (see Figure 3.18) such that the image preserves the same orientation (i.e. not flipped). Note: a **camera** does not **measure** distances but **angles**! $\rightarrow$ a camera is a "**bearing sensor**". In fact you have two angles: *azimuth* (φ) and elevation. To have a **depth sensor** we need a **second camera**!

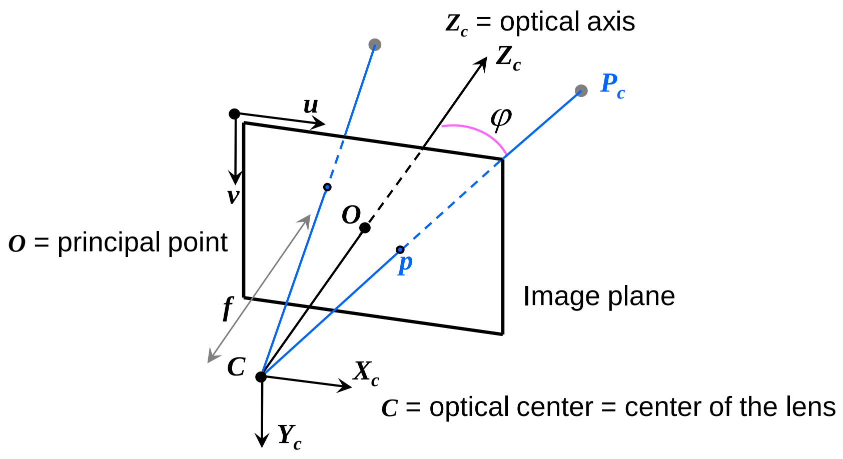


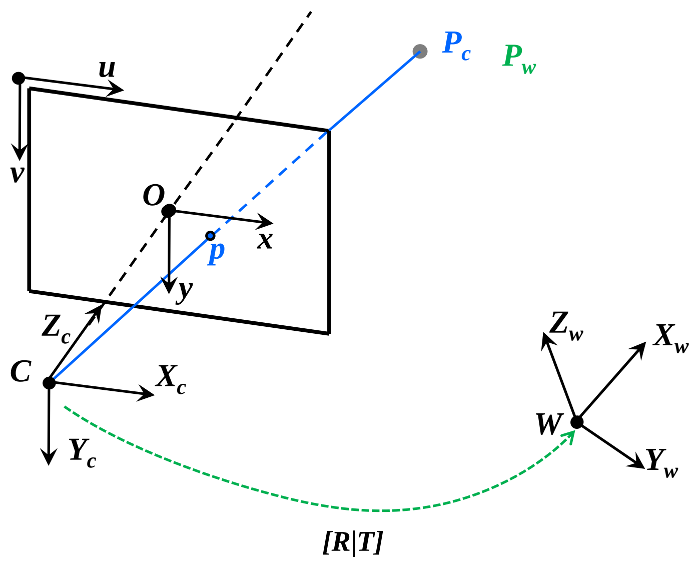
Figure 3.18: Perspective camera

But how to pass from world to pixels coordinates? The goal is to find the pixel coordinates (u, v) of point P_w in the world frame. We need 3 steps:

0. Convert world point P_w to camera point P_c through rigid body transform $[R|T]$

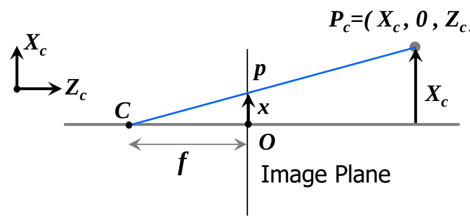
1. Convert P_c to image-plane coordinates (x, y)

2 Convert (x, y) to (discretized) pixel coordinates (u, v)



Step 1. Convert $\mathcal{F}_c$ to $\mathcal{F}_o$

First we compute from the camera frame $\mathcal{F}_c$ to the image plane $\mathcal{F}_o$. The camera point $P_c = (X_c, 0, Z_c)^T$ projects to $p = (x, y)^T$ onto the image plan. Then, from similar triangles:



$$\frac{x}{f} = \frac{X_c}{Z_c} \to x = f\frac{X_c}{Z_c}$$

Similarly, in the general case:

$$\frac{y}{f} = \frac{Y_c}{Z_c} \to y = f\frac{Y_c}{Z_c}$$

Step 2. Convert $\mathcal{F}_c$ to $\mathcal{F}_o$

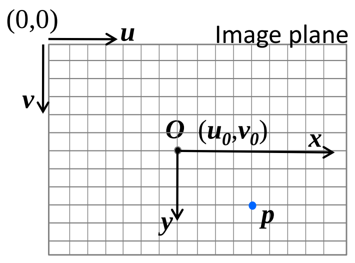
Then we pass from the camera frame $\mathcal{F}_c$ to pixel coordinates $\mathcal{F}_o$. To convert p from the local image plane coords $(x, y)^T$ to the pixel coordinates $(u, v)^T$, we need to account for:

- The pixel coordinates of the camera optical center $O = (u_0, v_0)^T$
- Scale factors k_u, k_v for the pixel-size in both dimensions:

$$u = u_0 + k_u x \to u = u_0 + k_u f\frac{X_c}{Z_c}$$

$$v = v_0 + k_v y \to v = v_0 + k_v f\frac{Y_c}{Z_c}$$

Then we use homogeneous coordinates for linear mapping from 3D to 2D, by introducing an extra element (scale):

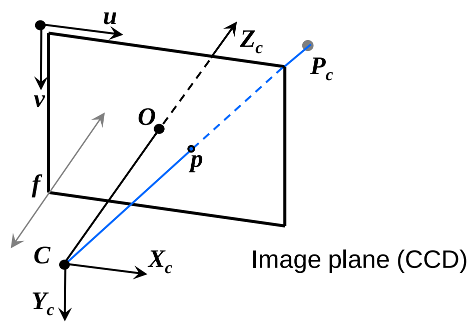


$$\boldsymbol{p} = \begin{pmatrix} u \\ v \end{pmatrix} \to \tilde{\boldsymbol{p}} = \begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

So we have:

$$u = u_0 + k_u f\frac{X_c}{Z_c}$$

$$v = v_0 + k_v f\frac{Y_c}{Z_c}$$



Expressed in matrix form and homogeneous coordinates:

$$\begin{pmatrix} \lambda u \\ \lambda v \\ \lambda \end{pmatrix} = \begin{bmatrix} k_u f & 0 & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$$

Or alternatively:

$$\begin{pmatrix} \lambda u \\ \lambda v \\ \lambda \end{pmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = K \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$$

K is called the "**Calibration Matrix**" or the "Matrix of Intrinsic Parameters". u and v are the focal length in pixels.

Sometimes, it is common to assume a skew factor ($K_{12} \neq 0$) to account for possible misalignments between CCD and lens. However, the camera manufacturing process today is so good that we can safely assume $K_{12} = 0$ and $\alpha_u = \alpha_v$.

Step 0. Convert $\mathcal{F}_w$ to $\mathcal{F}_c$

Now we can finally add the conversion from the world frame to the camera frame:

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \rightarrow$$

$$\rightarrow \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & |t_x \\ r_{21} & r_{22} & r_{23} & |t_y \\ r_{31} & r_{32} & r_{33} & |t_z \end{bmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = [R|T] \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

So we can have the so-called **Perspective Projection Equation**:

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \quad \rightarrow \quad \underline{\underline{\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K[R|T] \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}}}$$

3.5 Lens distortion

Lens causes distortions on images. **Radial distortions** (Figure 3.19) in images shoot with lenses are very common and have to be taken in consideration.

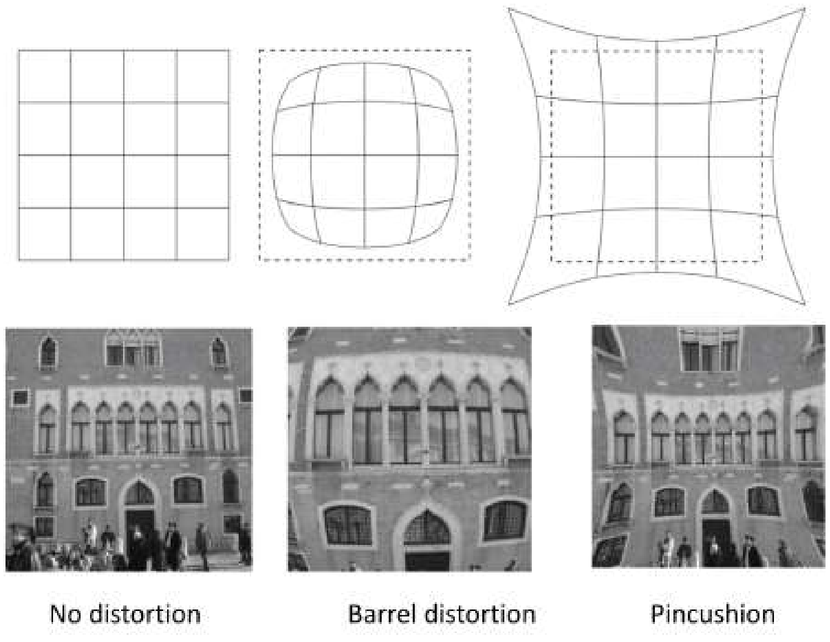


Figure 3.19: Radial distortions

The standard model of radial distortion is a transformation from the ideal coordinates $(u, v)^T$ (i.e., undistorted) to the real observable coordinates (distorted) $(u_d, v_d)^T$.

The amount of distortion of the coordinates of the observed image is a nonlinear function of their radial distance. For most lenses, a simple **quadratic model** of distortion produces good results:

$$\begin{pmatrix} u_d \\ v_d \end{pmatrix} = (1 + k_1 r^2) \begin{pmatrix} u - u_0 \\ v - v_0 \end{pmatrix} + \begin{pmatrix} u_0 \\ v_0 \end{pmatrix}$$

where $r^2 = (u - u_0)^2 + (v - v_0)^2$

Depending on the amount of distortion (an thus on the camera field of view), **higher order terms** can be introduced:

$$\begin{pmatrix} u_d \\ v_d \end{pmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \begin{pmatrix} u - u_0 \\ v - v_0 \end{pmatrix} + \begin{pmatrix} u_0 \\ v_0 \end{pmatrix}$$

Summary: Perspective projection equations

- To recap, a 3D world point $\mathbf{P} = (X_w, Y_w, Z_w)^T$ projects into the image point $\mathbf{p} = (u, v)^T$

$$\tilde{p} = \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathtt{K}[\mathtt{R}|\mathbf{T}] \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \quad \text{where } \mathtt{K} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

  and λ is the depth ($\lambda = Z_c$) of the scene point.

- If we want to take into account the radial distortion, then the distorted coordinates $(u_d, v_d)^T$(in pixels) can be obtained as

$$\begin{pmatrix} u_d \\ v_d \end{pmatrix} = (1 + k_1 r^2) \begin{pmatrix} u - u_0 \\ v - v_0 \end{pmatrix} + \begin{pmatrix} u_0 \\ v_0 \end{pmatrix}$$

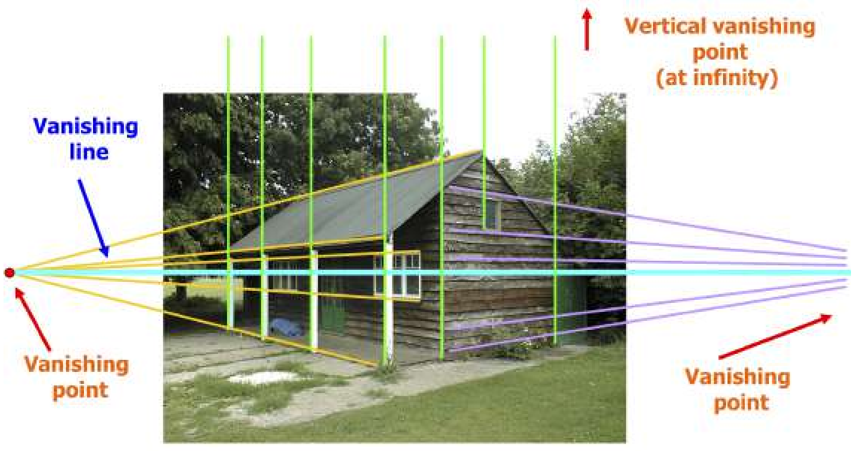
  where

$$r^2 = (u - u_0)^2 + (v - v_0)^2$$

See also the OpenCV documentation: docs.opencv.org/2.4.13.3/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

Exercise 1 Determine the Intrinsic Parameter Matrix (K) for a digital camera with image size 640x480 pixels and horizontal field of view equal to 90deg. Assume the principal point in the center of the image and squared pixels. What is the vertical field of view?

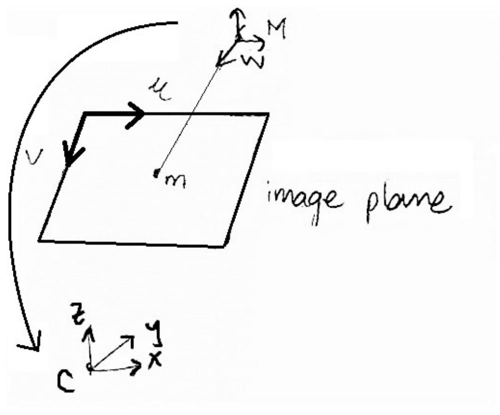
Exercise 2 Prove that world's parallel lines intersect at a vanishing point in the camera image:



Chapter 4

Image Formation II: Calibration

We have the perspective camera model (see Chapter 3) and we can also generalize it to **generic camera model** or **unit sphere camera model**, but still single view camera (a single center of projection).



$m \sim PM$ ← always in homogeneous coordinate

finite camera

$$\underset{3\times4}{P} = \underset{3\times3}{K}\,[\underset{3\times3}{R} \mid \underset{3\times1}{t}]$$

R: 3DOF

t: 3DOF

$$\text{Intrinsic parameters: } \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_x & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

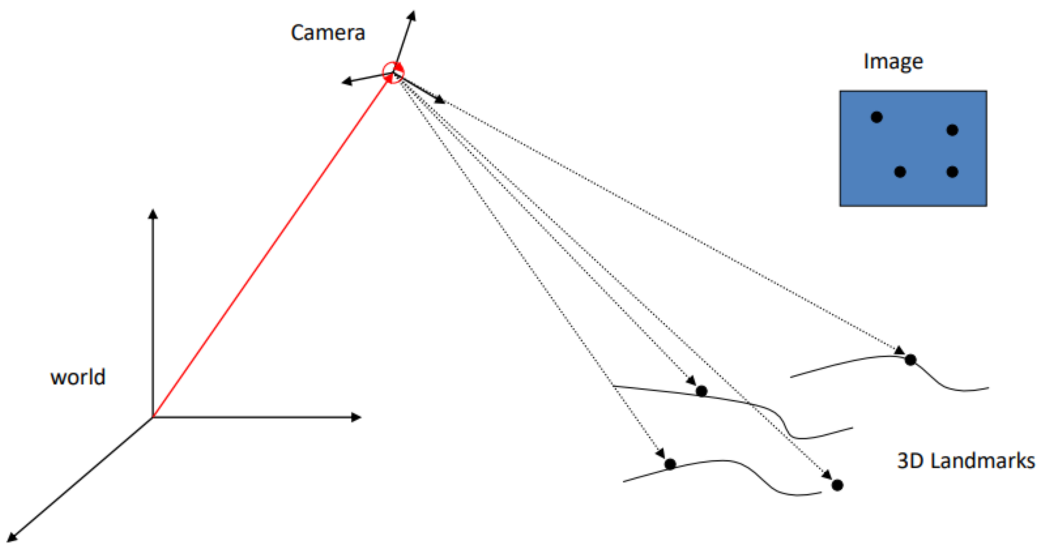
If we don't know the intrinsic parameters we call the camera **uncalibrated** (we will see calibration also for uncalibrated cameras)

4.1 Non-linear algorithms: P3P and PnP for calibrated cameras

4.1.1 DLT from general 3D objects

So, let's see how the pose determination depends by the number of n points of projection (**PnP Problem**).

Given known 3D landmarks in the world frame and given their image correspondences in the camera frame, determine the 6DOF pose of the camera in the world frame (including the intrinsinc parameters if uncalibrated):

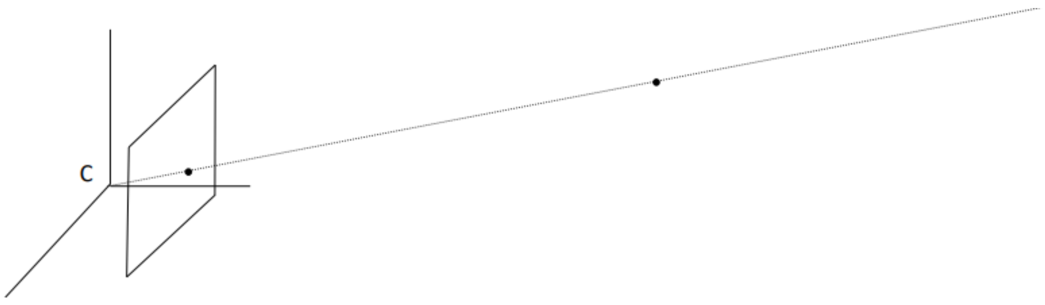


So, we said that with our previous model we have only onepoint of projection. We can ask ourself then: **how many points are enough**?

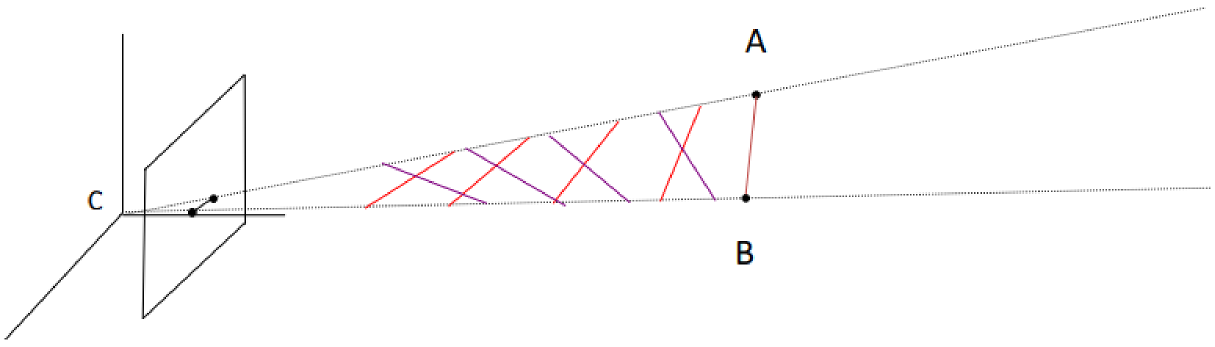
- 1 Point: infinitely many solutions.
- 2 Points: infinitely many solutions, but bounded.
- 3 Points: (no 3 collinear) finitely many solutions (up to 4) → **P3P problem**.
- 4 Points: Unique solution

Let's see:

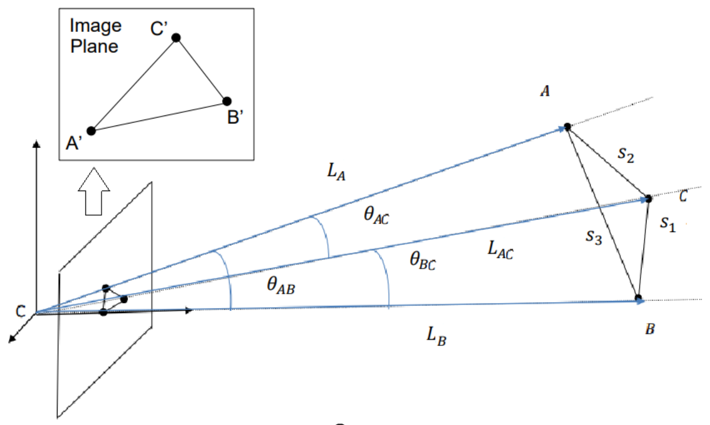
– <u>1 Point</u>: infinitely many solutions (classical projection) all along the line:

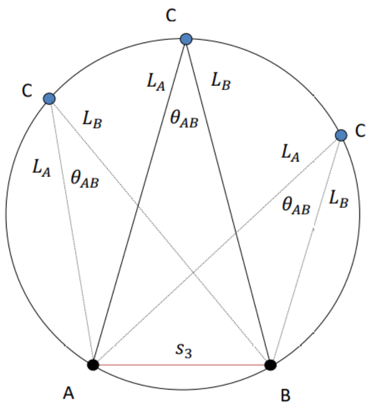


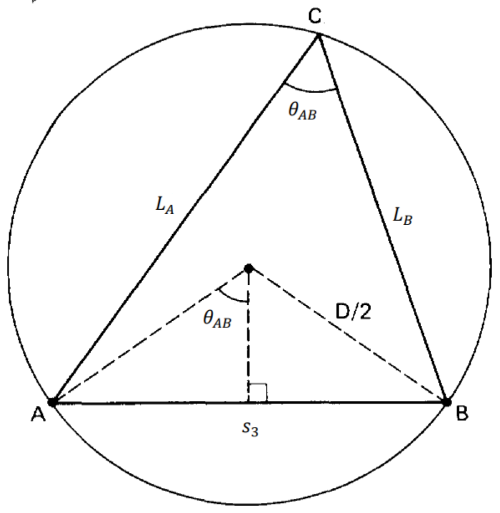
– <u>2 Points</u>: infinitely many solutions, but bounded by the segments shown here below:



– 3 Points: (no 3 collinear) finitely many solutions (up to 4) → when you move to three points you fall in what is called **P3P problem**:







In Euclidean geometry, Carnot's theorem states that the sum of the signed distances from the circumcenter D to the sides of an arbitrary triangle ABC is $DF + DG + DH = R + r$, where r is the inradius and R is the circumradius of the triangle

From Carnot's Theorem:

$$s_3^2 = L_A^2 + L_B^2 - 2L_AL_B\cos\theta_{AB}$$

Similarly:

$$s_1^2 = L_B^2 + L_C^2 - 2L_BL_C\cos\theta_{BC}$$

$$s_2^2 = L_A^2 + L_C^2 - 2L_AL_C\cos\theta_{AC}$$

So these 3 equations show a relationship that links the distances from the three point in the world frame to the center of the camera $L_{A,B,C}$ with the angles $\theta_{AB,BC,AC}$ and the distance between the points $s_{1,2,3}$ (the points in the world, not the one projected in the image plane!). So, from the equations appears that the angles θ between the points can be computed knowing the distances s between the image points. In fact, if you know the focal length, you can use the model that we saw in Chapter 3.

Now we will show the **algebraic approach** (Fischler and Bolles, 1981) used to reduce those three equations to a 4th order equation:

- It is known that n independent polynomial equations, in n unknowns, can have no more solutions than the product of their respective degrees. Thus, the system can have a maximum of 8 solutions.
- However, because every term in the system is either a constant or of second degree, for every real positive solution there is a negative solution.
- Thus, with 3 points, there are at most 4 valid (positive) solutions.
- A **4th point** can be used to disambiguate the solutions.

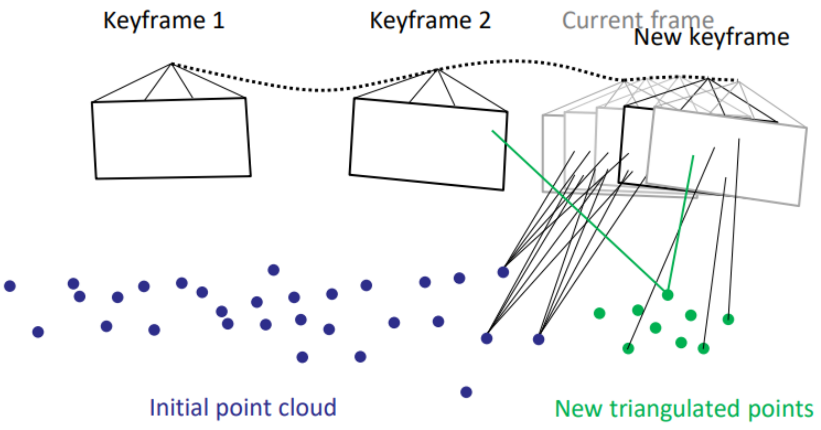
By defining $x = b/a$, it can be shown that the system can be reduced to a 4th order equation:

$$G_0 + G_1x + G_2x^2 + G_3x^3 + G_4x^4 = 0$$

So we can conclude with what we already anticipated:
– 4 Points: unique solution

It is important to note that using **more points** ($n > 4$) is faster.

If we see an application to Monocular Visual Odometry, we can notice how we can implement the camera pose estimation from known 3D-2D correspondences:



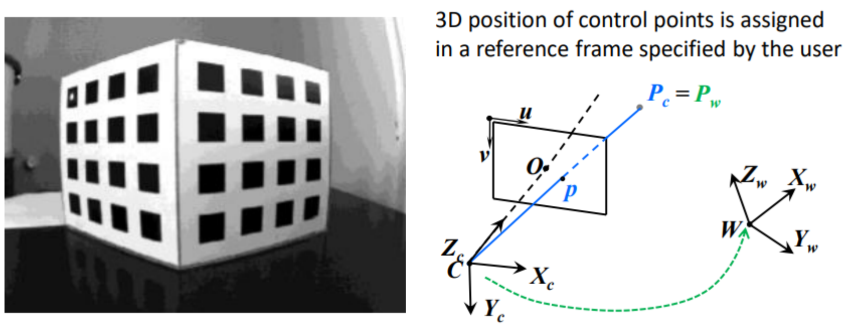
4.2 Linear algorithms (DLT) for uncalibrated cameras

4.2.1 DLT from 3D objects

If the camera is **uncalibrated** you will need to perform a camera calibration. The calibration is the process to **determine the intrinsic and extrinsic** parameters of the camera model.

A method proposed in 1987 by Tsai consists of measuring the 3D position of $n \geq 6$ control points on a three-dimensional calibration target and the 2D coordinates of their projection in the image. This problem is also called "Resection", or "Perspective from n Points" (before was "Pose from n points"), or "Camera pose from 3D-to-2D correspondences", and is one of the most widely used algorithms in Computer Vision and Robotics.

What is the solution to this problem? The intrinsic and extrinsic parameters are **computed directly from the perspective projection equation**. Let's see how:


3D position of control points is assigned in a reference frame specified by the user
$P_c = P_w$
u
v
O
p
Z_c
C
X_c
Y_c
Z_w
X_w
W
Y_w

Our goal is to compute K, R, and T that satisfy the perspective projection equation (we neglect the radial distortion):

Don't understand this part: R has 8 DoF, we put some constraints: $\det(R) = +1$, the column vector $||r_{i1}||^2 = 1$, the row vector $||r_{1i}||^2 = 1$, $R^{-1} = R^T$

So, we start with the Perspective Projection Equation:

$$\tilde{p} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \overset{\text{Perspective Projection Equation}}{\boxed{\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\left[R \mid T\right] \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}}} \Rightarrow$$

$$\Rightarrow \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} \alpha_u r_{11} + u_0 r_{31} & \alpha_u r_{12} + u_0 r_{32} & \alpha_u r_{13} + u_0 r_{33} & \alpha_u t_1 + u_0 t_3 \\ \alpha_v r_{21} + v_0 r_{31} & \alpha_v r_{22} + v_0 r_{32} & \alpha_v r_{23} + v_0 r_{33} & \alpha_v t_2 + v_0 t_3 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = M \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} m_1^T \\ m_2^T \\ m_3^T \end{bmatrix} \cdot \boxed{\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}} \rightarrow P$$

In those equations the notation m_i^T is used to indicate the i-th row of M. Then, a conversion from homogeneous coordinates back to pixel coordinates leads to:

$$\begin{aligned} u &= \frac{\tilde{u}}{\tilde{w}} = \frac{m_1^T \cdot P}{m_3^T \cdot P} \\ v &= \frac{\tilde{v}}{\tilde{w}} = \frac{m_2^T \cdot P}{m_3^T \cdot P} \end{aligned} \Rightarrow \begin{aligned} (m_1^T - u_i m_3^T) \cdot P_i = 0 \\ (m_2^T - v_i m_3^T) \cdot P_i = 0 \end{aligned}$$

By re-arranging the terms, we obtain:

$$\begin{aligned} (m_1^T - u_i m_3^T) \cdot P_i = 0 \\ (m_2^T - v_i m_3^T) \cdot P_i = 0 \end{aligned} \Rightarrow \begin{pmatrix} P_1^T & 0^T & -u_1 P_1^T \\ 0^T & P_1^T & -v_1 P_1^T \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

For n points, we can stack all these equations into a big matrix:

$$\begin{bmatrix} \mathbf{P}_1^T & \mathbf{0}^T & -u_1 \mathbf{P}_1^T \\ \mathbf{0}^T & \mathbf{P}_1^T & -v_1 \mathbf{P}_1^T \\ \cdots & \cdots & \cdots \\ \mathbf{P}_n^T & \mathbf{0}^T & -u_n \mathbf{P}_n^T \\ \mathbf{0}^T & \mathbf{P}_n^T & -v_n \mathbf{P}_n^T \end{bmatrix} \begin{pmatrix} \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \Rightarrow \mathbf{A.M} = \mathbf{0}$$

$$\underbrace{\left(\begin{array}{cccc|cccc|cccc} X_w^1 & Y_w^1 & Z_w^1 & 1 & 0 & 0 & 0 & 0 & -u_1X_w^1 & -u_1Y_w^1 & -u_1Z_w^1 & -u_1 \\ 0 & 0 & 0 & 0 & X_w^1 & Y_w^1 & Z_w^1 & 1 & -v_1X_w^1 & -v_1Y_w^1 & -v_1Z_w^1 & -v_1 \\ \cdots & \cdots & \cdots & & \cdots & \cdots & \cdots & & \cdots & \cdots & \cdots & \\ X_w^n & Y_w^n & Z_w^n & 1 & 0 & 0 & 0 & 0 & -u_nX_w^n & -u_nY_w^n & -u_nZ_w^n & -u_n \\ 0 & 0 & 0 & 0 & X_w^n & Y_w^n & Z_w^n & 1 & -v_nX_w^n & -v_nY_w^n & -v_nZ_w^n & -v_n \end{array}\right)}_{\substack{A \\ \text{(matrix } \textbf{known}\text{)}}} \underbrace{\left(\begin{array}{c} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ \hline m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ \hline m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{array}\right)}_{\text{M (matrix } \textbf{unknown}\text{)}} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \Rightarrow \mathrm{A.M} = 0$$

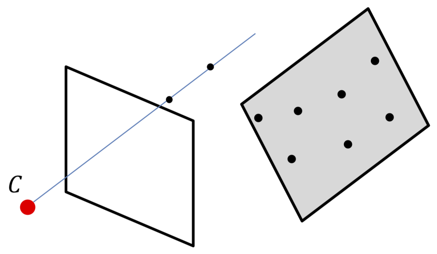
So we have a **homogeneous linear system**:

$$A \cdot M = 0$$

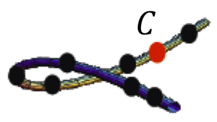
Minimal solution:
– $A(2n \times 12)$ should have $rank = 11$ to have a unique (up to a scale) non-trivial solution M
– Each 3D-to-2D point correspondence provides 2 independent equations
– Thus, $5 + \frac{1}{2}$ point correspondences are needed (in practice **6 point** correspondences!)

Over-determined solution:
– $n \geq 6$ points – A solution is to minimize $||AM||^2$ subject to the constraint $||M||^2 = 1$. It can be solved through **Singular Value Decomposition** (SVD). The solution is the eigenvector corresponding to the smallest eigenvalue of the matrix A^TQ (because it is the unit vector x that minimizes $||Ax||^2 = x^TA^TAx$.[1]
– Matlab instructions: [U,S,V] = svd(A); M = V(:,12);

Degenerated configurations:
– Points lying on a plane and/or along a single line passing through the projection center:



– Camera and points on a twisted cubic (i.e., smooth curve in 3D space of degree 3)



[1]See in Appendix - **Handritten Notes** the SVD methodology

Solving for K, R, T:
Once we have the M matrix, we can recover the intrinsic and extrinsic parameters by remembering that:

$$M = K[R|T]$$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} =$$

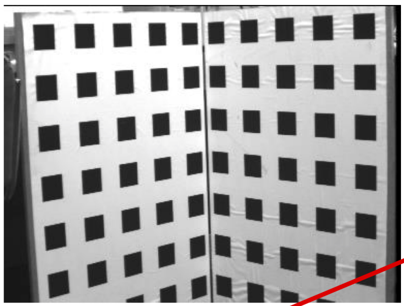
$$\begin{bmatrix} \alpha_u r_{11} + u_0 r_{31} & \alpha_u r_{12} + u_0 r_{32} & \alpha_u r_{13} + u_0 r_{33} & \alpha_u t_x + u_0 t_z \\ \alpha_v r_{21} + v_0 r_{31} & \alpha_v r_{22} + v_0 r_{32} & \alpha_v r_{23} + v_0 r_{33} & \alpha_v t_y + v_0 t_z \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}$$

However, notice that we are not enforcing the constraint that R is orthogonal (i.e. $R^T R = I$). To do this, we can use the so-called QR factorization of M, which decomposes M into a R (orthogonal), T, and an upper triangular matrix (i.e. K).

4.2.2 DLT from planar grids

A calibration method proposed by Tsai in 1987 is based on **non co-planar grids** (instead of 3D objects):

1. Edge detection
2. Straight line fitting to the detected edges
3. Intersecting the lines to obtain the image corners (corner accuracy < 0.1 pixels)
4. **Use more than 6 points** (ideally more than 20) **and not all lying on a plane**



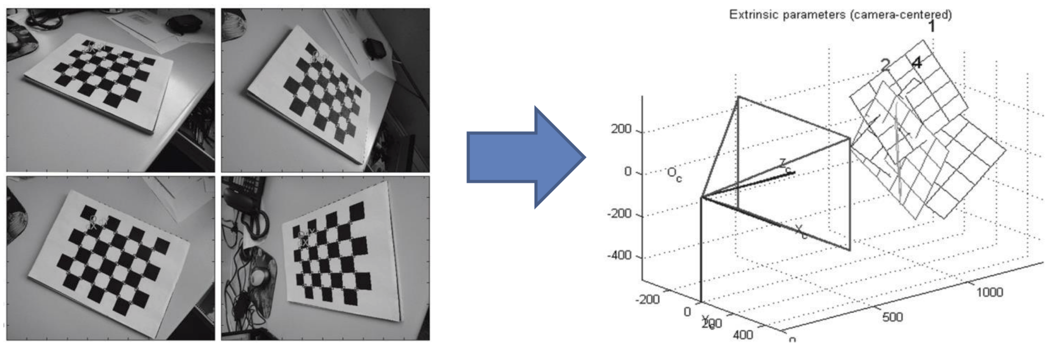
Why is this ratio not 1?

What are the «skew» and «residuals»?

f_y	f_x/f_y	skew	x_0	y_0	residual
1673.3	1.0063	1.39	379.96	305.78	0.365

Tsai calibration is based on DLT algorithm, which requires points not to lie on the same plane. That is not the best choice.

An **alternative method** (today's standard camera calibration method) consists of using a **planar grid** (e.g., a chessboard) and a few images of it shown at different orientations.This method was invented by Zhang (1999) in Microsoft:



In a camera calibration from planar grids we need to find the **homographies**. Our goal is to compute K, R, and T that satisfy the perspective projection equation (we neglect the radial distortion). Remember that since the points lie on a plane (advantage of using a planar grid), we have $Z_w = 0$:

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathtt{K}[\mathtt{R}|\mathbf{T}] \begin{pmatrix} X_w \\ Y_w \\ 0 \\ 1 \end{pmatrix} \Rightarrow$$

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} \alpha_u r_{11} + u_0 r_{31} & \alpha_u r_{12} + u_0 r_{32} & \alpha_u t_x + u_0 t_z \\ \alpha_v r_{21} + v_0 r_{31} & \alpha_v r_{22} + v_0 r_{32} & \alpha_v t_y + v_0 t_z \\ r_{31} & r_{32} & t_z \end{bmatrix} \begin{pmatrix} X_w \\ Y_w \\ 1 \end{pmatrix} \Rightarrow$$

$$\Rightarrow \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} X_w \\ Y_w \\ 1 \end{pmatrix} \Rightarrow \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathtt{H} \begin{pmatrix} X_w \\ Y_w \\ 1 \end{pmatrix}$$

$$\Rightarrow \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} \mathbf{h}_1^T \\ \mathbf{h}_2^T \\ \mathbf{h}_3^T \end{bmatrix} \begin{pmatrix} X_w \\ Y_w \\ 1 \end{pmatrix}$$

So we found the matrix H called **Homography**, where $\boldsymbol{h}_i^T$ is the i-th row of H.

By re-arranging the terms, we obtain:

$$\begin{aligned} (\mathbf{h}_1^T - u_i \mathbf{h}_3^T).\mathbf{P}_i &= 0 \\ (\mathbf{h}_2^T - v_i \mathbf{h}_3^T).\mathbf{P}_i &= 0 \end{aligned} \Rightarrow \begin{bmatrix} \mathbf{P}_i^T & \mathbf{0}^T & -u_i \mathbf{P}_i^T \\ \mathbf{0}^T & \mathbf{P}_i^T & -v_i \mathbf{P}_i^T \end{bmatrix} \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

For n points, we can stack all these equations into a big matrix:

$$\begin{bmatrix} \mathbf{P}_1^T & \mathbf{0}^T & -u_1 \mathbf{P}_1^T \\ \mathbf{0}^T & \mathbf{P}_1^T & -v_1 \mathbf{P}_1^T \\ \cdots & \cdots & \cdots \\ \mathbf{P}_n^T & \mathbf{0}^T & -u_n \mathbf{P}_n^T \\ \mathbf{0}^T & \mathbf{P}_n^T & -v_n \mathbf{P}_n^T \end{bmatrix} \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \Rightarrow \mathtt{Q}.\mathbf{H} = \mathbf{0}$$

So we have a **homogeneous linear system**:

$$Q \cdot H = 0$$

Minimal solution:
– $Q(2n \times 9)$ should have $rank = 11$ to have a unique (up to a scale) non-trivial solution H
– Each point correspondence provides 2 independent equations
– Thus, a minimum of **4 non-collinear points** are needed.

Over-determined solution:
– $n \geq 4$ points – It can be solved through **Singular Value Decomposition** ((same considerations as the case before apply).
– Matlab instructions: [U,S,V] = svd(Q); H = V(:,9);

Solving for K, R, T:
H can be decomposed by recalling that:

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix}$$

It is possible to find a demo of Camera Calibration Toolbox for Matlab (world's standard toolbox for calibrating perspective cameras) at www.vision.caltech.edu/bouguetj/calib_doc/

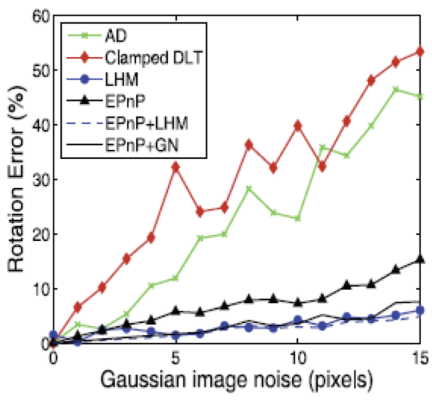
There are a lot of application of calibration from planar grids today, for example in Augmented reality and in Robotics (beacon-based localization).

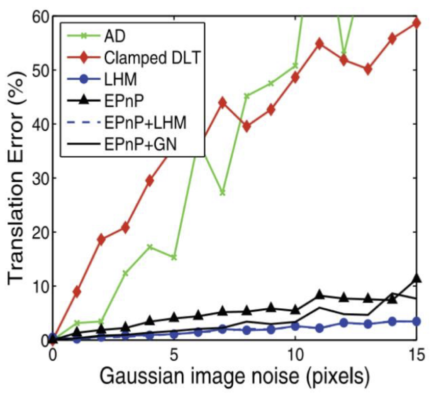
4.3 DLT vs PnP

If the camera is calibrated, only R andT need to be determined. In this case, should we use DLT (linear system of equations) or PnP (nonlinear)?

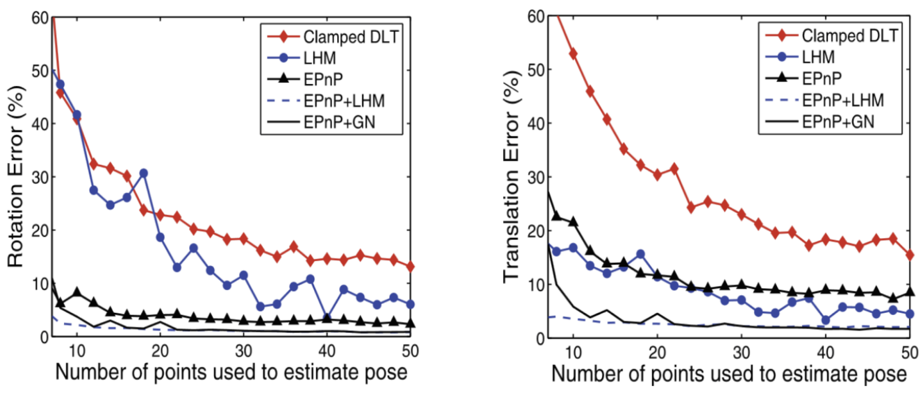
– **BibRef**: Lepetit, Moreno Noguer, Fua, EPnP: An Accurate O(n) Solution to the PnP Problem, IJCV'09

Accuracy vs noise:

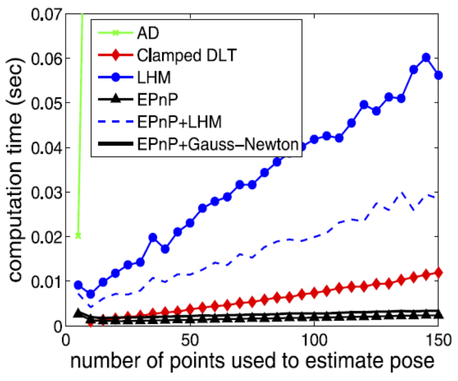




Accuracy vs number of points:



Timing:



4.4 Non-Linear Estimation

Consider a matrix X of data values and an objective function $f(X;\boldsymbol{a})$ where $\boldsymbol{a}$ is the set of parameters to be estimated.

When f is a quadratic function, which means ∇f (with respect to the parameters) is linear, minimizing this is straight-forward. Here, we are interested in the case that f is non-quadratic, which means that ∇f is non-linear.

Two examples :

Ex. 1) Each row of X contains the vector x_i^T, y_i. The parameter vector is $\boldsymbol{a}, \sigma$ it includes the set of line/plane parameters plus σ. The goal is to minimize:

$$f(\mathtt{X};\mathbf{a},\sigma) = \left\{ \sum_i \rho\left(\left(y_i - \mathbf{x}_i^T \mathbf{a} \right) / \sigma \right) \right\} + N \log \sigma$$

This is a robust estimation problem, but we have added σ (the robust scale value).

Ex. 2) X contains the set of corresponding image points $x_i \leftrightarrow x_i'$. The goal is to minimize:

$$f(\mathtt{X};\mathbf{h},) = \sum_i d\left(\mathbf{x}_i', \mathtt{H}\mathbf{x}_i\right)^2 = \sum_i \left(u_i' - \frac{\mathbf{x}_i^T \mathbf{h}_1}{\mathbf{x}_i^T \mathbf{h}_3} \right)^2 + \left(v_i' - \frac{\mathbf{x}_i^T \mathbf{h}_2}{\mathbf{x}_i^T \mathbf{h}_3} \right)^2$$

In other words, this is the minimization of the geometric error in a single image.

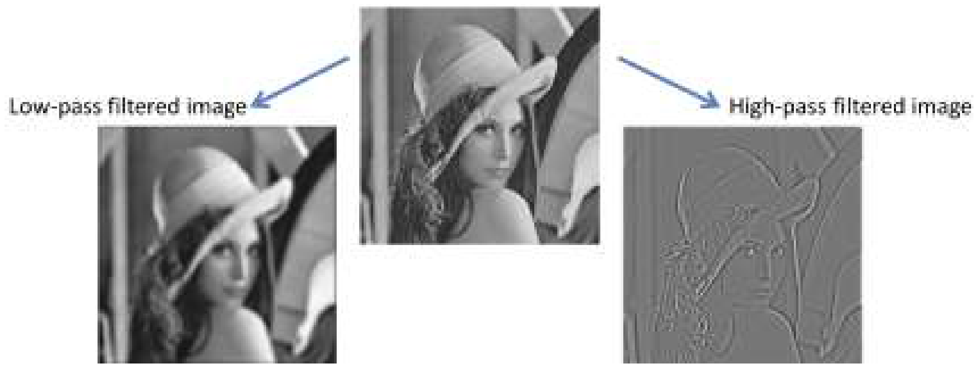
The solution techniques used here are all iterative. The iterations of the parameter vector estimates will be denoted $\boldsymbol{a}^t$. We reserve subscripting, such as in a_k, a_l, to denote components of vectors or matrices.

Chapter 5

Filtering and Edge detection

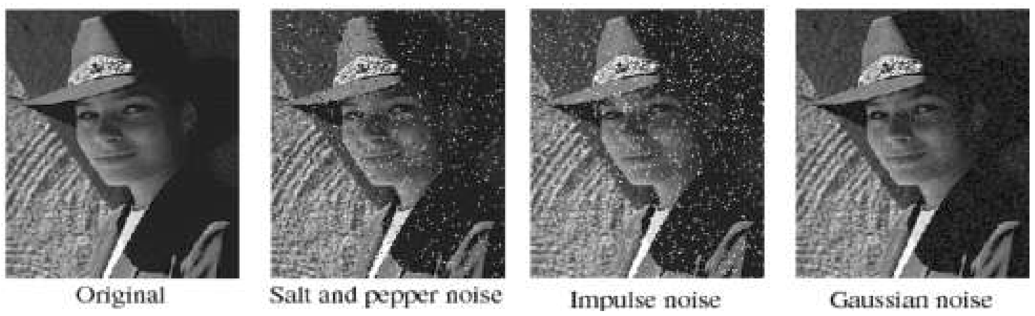
The word filter comes from frequency-domain processing, where "filtering" refers to the process of accepting or rejecting certain frequency components

We distinguish between low-pass and high-pass filtering:
– A **low-pass filter** smooths an image (retains low-frequency components)
– A **high-pass filter** retains the contours (also called edges) of an image (high frequency)

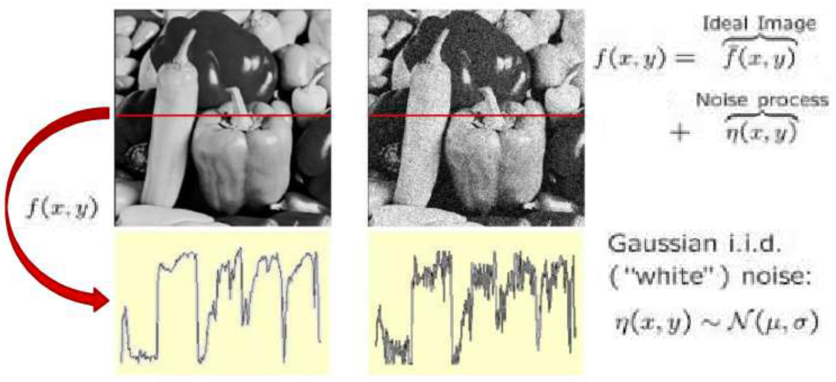


5.1 Noise reduction

We can have in an image different kinds of noises:
– **Salt and pepper noise**: random occurrences of black and white pixels
– **Impulse noise**: random occurrences of white pixels
– **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution



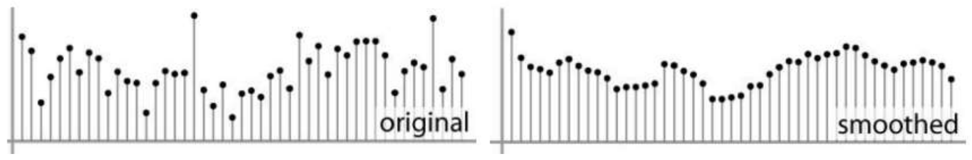
5.1.1 1D Filtering for Gaussian Noise


$f(x, y) = \overbrace{\hat{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$
$f(x, y)$
Gaussian i.i.d. ("white") noise:
$\eta(x, y) \sim \mathcal{N}(\mu, \sigma)$

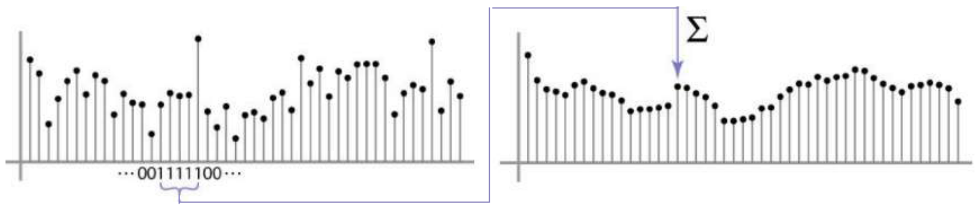
How could we reduce the noise to try to recover the "real image"?

Moving Average (Smoothing) This filter replaces each pixel with an average of all the values in its neighborhood. The assumptions to make are:
– Expect pixels to be like their neighbors
– Expect noise process to be independent from pixel to pixel.

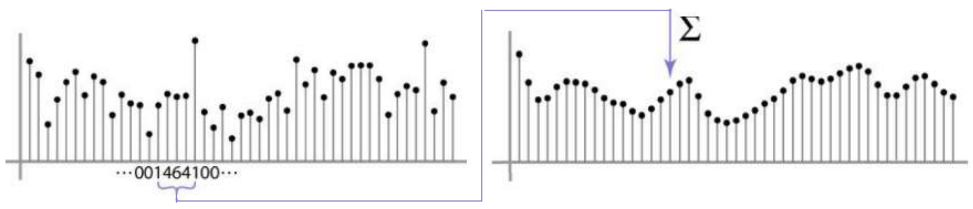
A moving average in 1D is the following:



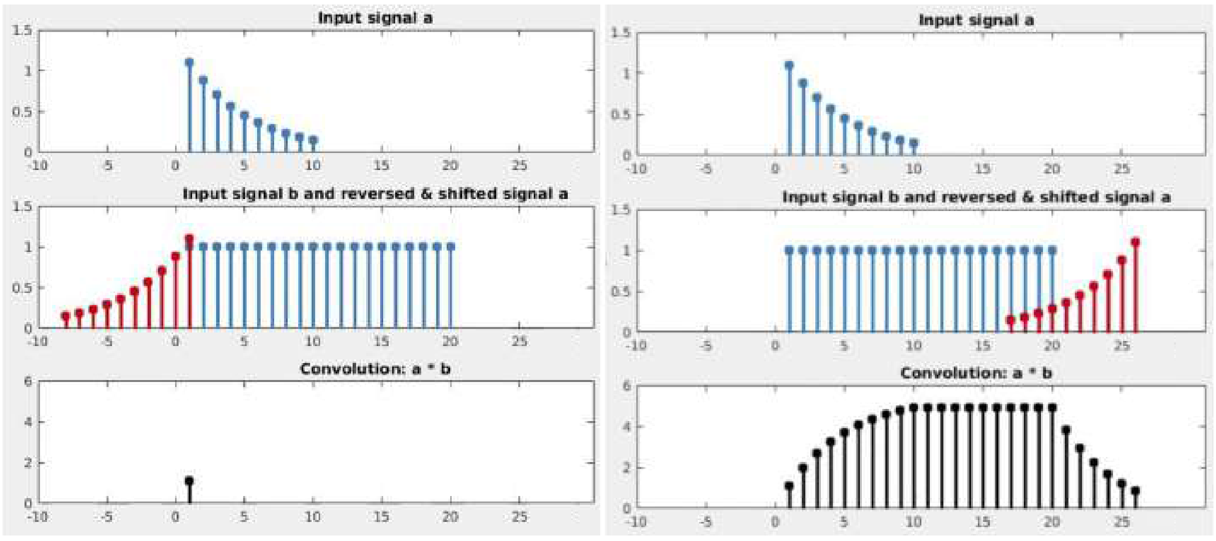
We can implement also a **weighted** moving average filter, adding weights to the moving average. For example, the previous filter with weights $[1, 1, 1, 1, 1]/5$ will become:



We can also have non-uniform weights, like $[1, 4, 6, 4, 1]/16$:



The operation performed by the filter is called **convolution**. Let's see an: example of convolution of two sequences (or "signals"):



$\rightarrow$ One of the sequences is flipped (right to left) before sliding over the other

The notation used is : $a \star b$. This operation has some nice properties like **linearity, associativity, commutativity**, etc...

5.1.2 2D Filtering - Correlation and convolution

Let's introduce another operation comparing it to the convolution:

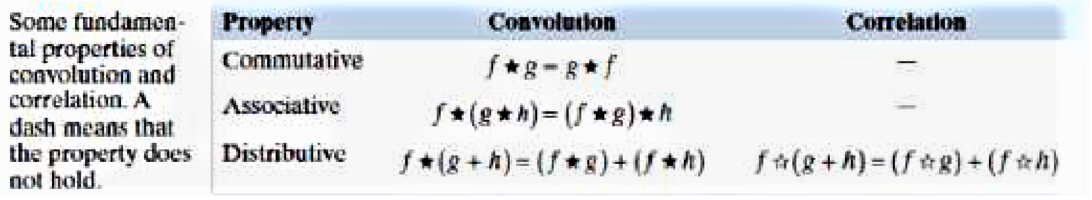
– **Correlation**:

$$(w \hat{\star} f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s; y+t)$$

– **Convolution:**

$$(w \star f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s; y-t)$$

For a kernel of size $m \times n$, we assume that $m = 2a+1$ and $n = 2b+1$, where a and b are nonnegative integers. This means that our focus is on kernels of odd size in both coordinate directions.

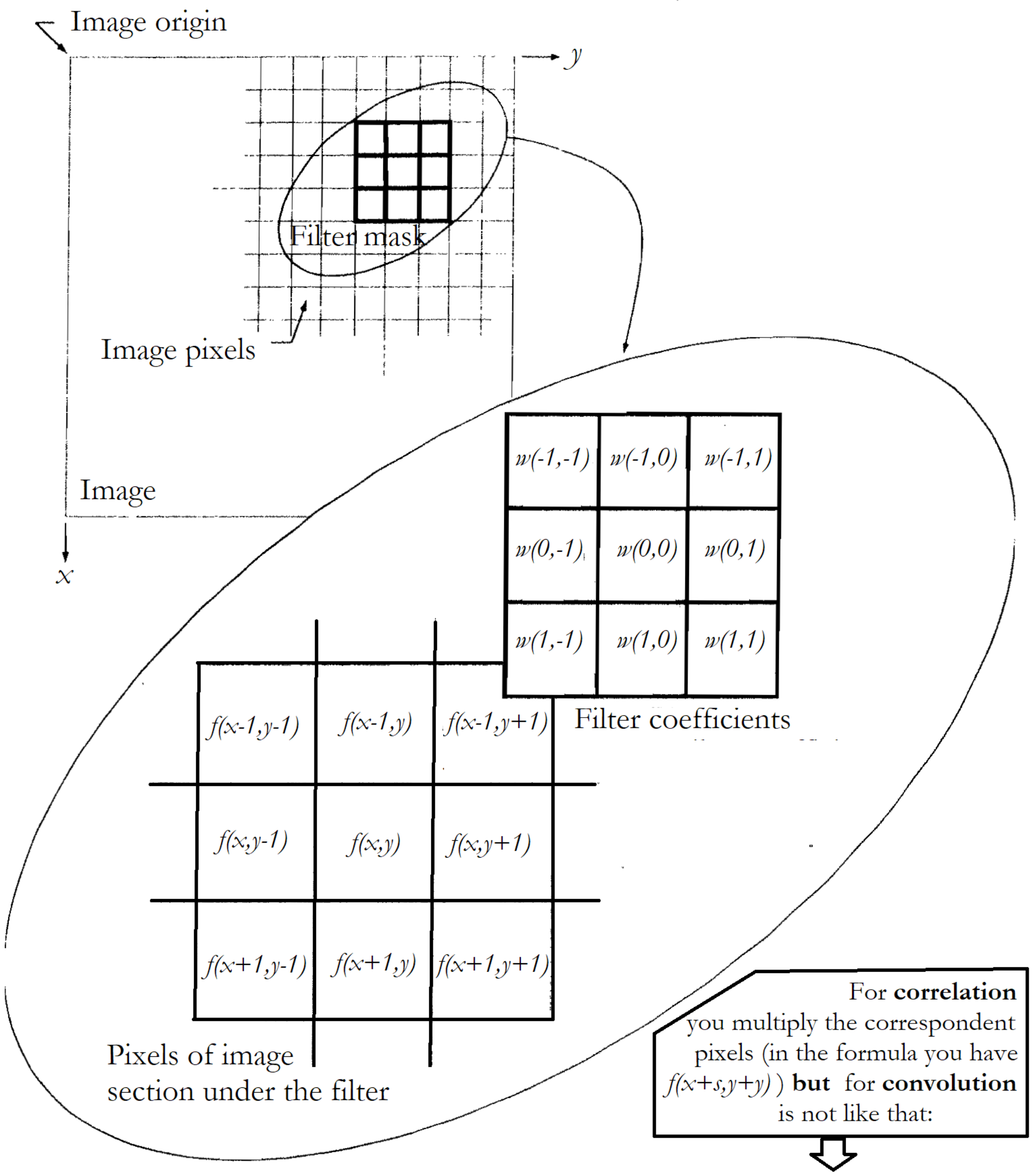

Some fundamental properties of convolution and correlation. A dash means that the property does not hold.

Property	Convolution	Correlation
Commutative	$f \star g = g \star f$	—
Associative	$f \star (g \star h) = (f \star g) \star h$	—
Distributive	$f \star (g + h) = (f \star g) + (f \star h)$	$f \,☆ (g + h) = (f \,☆\, g) + (f \,☆\, h)$

So, seeing what a convolution is, we can say that **filtering an image** is replace each pixel with a linear combination of its neighbors.

The filter w is also called "kernel" or "mask". It allows to have different weights depending on neighboring pixel's relative position. In the following image is shown the filtering process via the kernel coefficients:

The mechanics of linear spatial filtering using a 3×3 filter mask. The form chosen to denote the coordinates of the filter mask coefficients simplifies writing expressions for linear filtering.

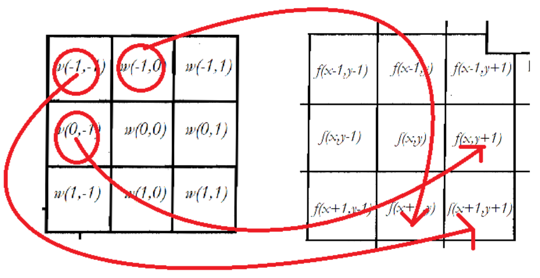


For **correlation** you multiply the correspondent pixels (in the formula you have $f(x+s,y+t)$) **but** for **convolution** is not like that:

Be careful!!! For the formula, the coefficient are **crossing**, they are not multiplied with the correspondent:

$$(w \star f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s; y-t)$$

$w(-1,-1)\ f(x+1,y+1)$
$w(0,-1)\ f(x,y+1)$
$w(-1,0)\ f(x+1,y)$



Sometimes, part of w lies outside f, so the summation is undefined in that area. A solution is the **zero padding**: to use a pad function f with enough zeros on either side. In general, if the kernel is of size $1 \times m$, we need $(m-1) = 2$ zeros on either side of f. Let's see an example in 1D:

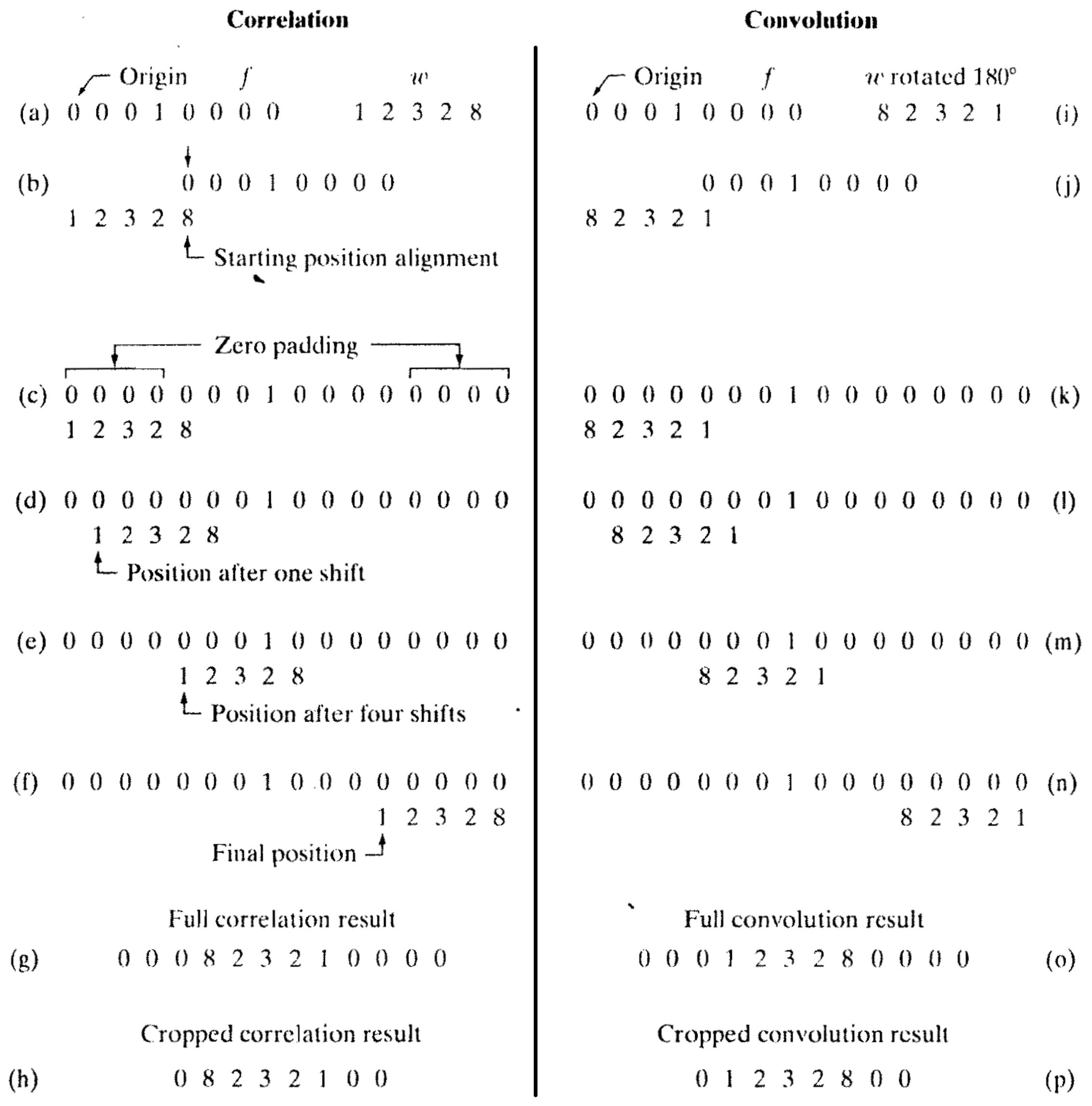
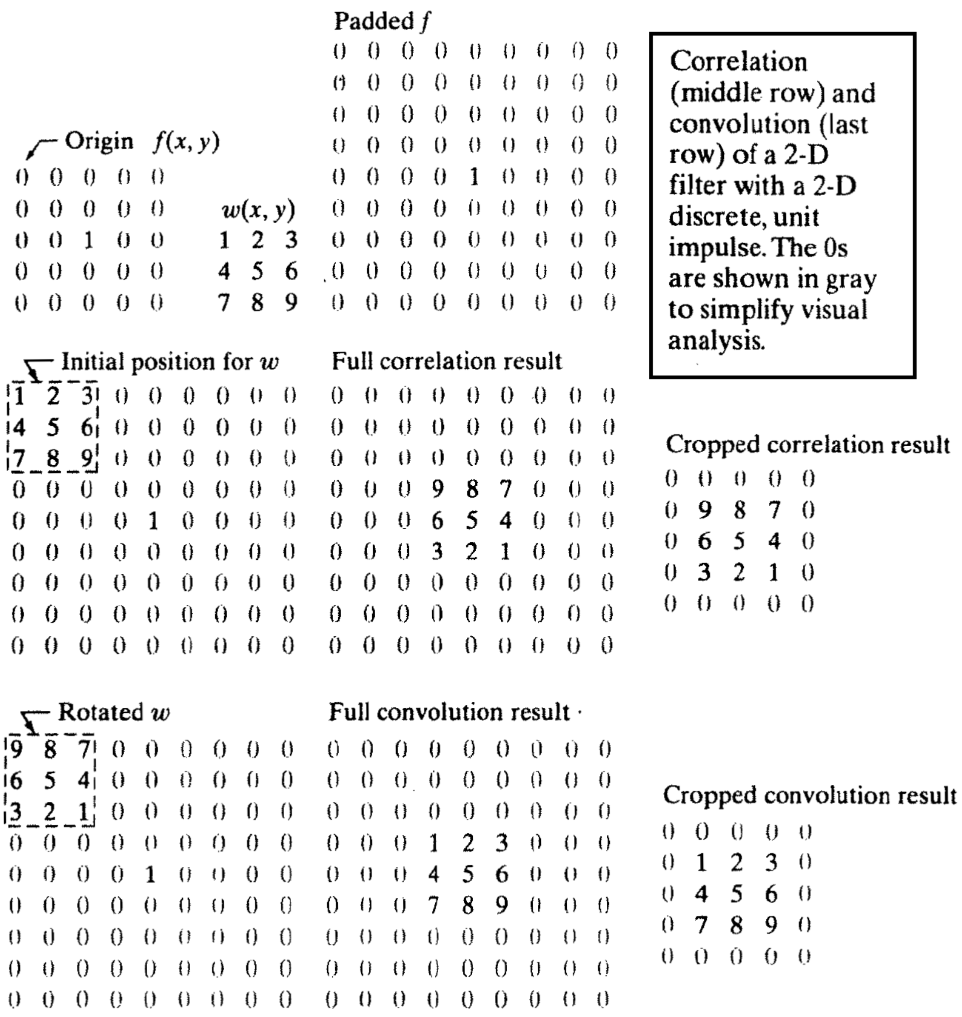

Correlation
Convolution
Origin f w
(a) 0 0 0 1 0 0 0 0 1 2 3 2 8
Origin f w rotated 180°
0 0 0 1 0 0 0 0 8 2 3 2 1 (i)
(b) 0 0 0 1 0 0 0 0
1 2 3 2 8
Starting position alignment
0 0 0 1 0 0 0 0 (j)
8 2 3 2 1
Zero padding
(c) 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 2 3 2 8
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 (k)
8 2 3 2 1
(d) 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 2 3 2 8
Position after one shift
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 (l)
8 2 3 2 1
(e) 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 2 3 2 8
Position after four shifts
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 (m)
8 2 3 2 1
(f) 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 2 3 2 8
Final position
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 (n)
8 2 3 2 1
Full correlation result
(g) 0 0 0 8 2 3 2 1 0 0 0 0
Full convolution result
0 0 0 1 2 3 2 8 0 0 0 0 (o)
Cropped correlation result
(h) 0 8 2 3 2 1 0 0
Cropped convolution result
0 1 2 3 2 8 0 0 (p)

Illustration of 1-D correlation and convolution of a filter with a discrete unit impulse. Note that correlation and convolution are functions of *displacement*.

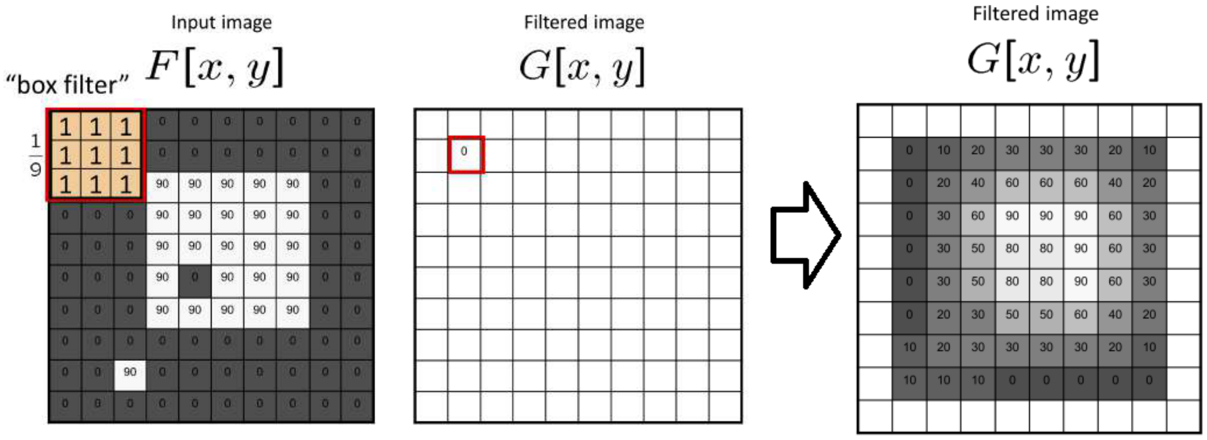
As we saw before, in the convolution (not in the correlation, we don't multiply the correspondent coefficients. To visualize it we can say that **in convolution** we **pre-rotate the kernel** and then repeat the sliding sum of products. So, the convolution of a function with an impulse copies the function to the location of the impulse. It is notable to see that **correlation and convolution** yield the **same** result **if** the **kernel** values are **symmetric** about the center.

In the following image we can compare correlation and convolution processes:

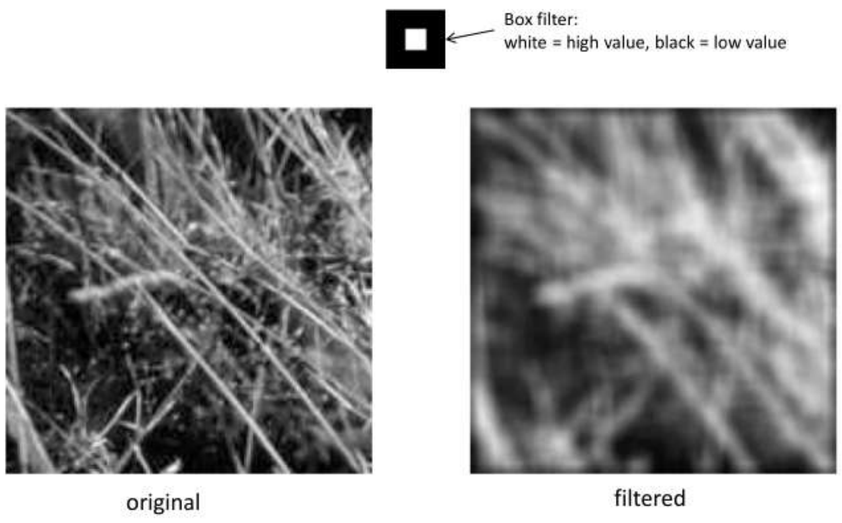


Correlation (middle row) and convolution (last row) of a 2-D filter with a 2-D discrete, unit impulse. The 0s are shown in gray to simplify visual analysis.

Moving Average in 2D We can now see how the moving average filtering works in 2D:



With this kind of filter we can **smooth** images:



Exercise You are given the following kernel and image:

$$w = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad f = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

1. Encircle the area when the kernel is centered at point $(2, 3)$ (2nd row, 3rd col) of the image shown above. Show specific values of w and f.
2. Compute the convolution $w \star f$ using the minimum zero padding needed. Show the detail of your computations when the kernel is centered on point $(2, 3)$ of f; and the show the final full convolution result.
3. Repeat 2, but for correlation $w \hat{\star} f$

5.1.3 2D Filtering - Separable Filter Kernels

A 2D function $G(x, y)$ is said to be **separable** if it can be written as the product of two 1D functions, $G_1(x)$ and $G_2(y)$, that is:

$$G(x, y) = G_1(x)G_2(y)$$

A spatial filter kernel is a matrix, so a separable kernel of size $m \times n$ can be expressed as the outer product of two vectors w_1 and w_2 (vectors of size respectively $m \times 1$ and $n \times 1$:

$$\textbf{Separable filter kernel: } w = w_1 \cdot w_2^T \rightarrow (rank\ 1)$$

It is useful to note how **convolution** behaves with separable kernels ($w = w_1 w_2$):

$$w \star f = (w_1 \star w_2) \star f = (w_1 \star f) \star w_2$$

But why separable filters are **convenient**? For an image of size $M \times N$ and a kernel size $m \times n$ the filtering process requires $M \cdot N \cdot m \cdot n$ multiplications and additions. If for the kernel holds separability, the filtering process requires only $M \cdot N \cdot (m+n)$ multiplications and additions, so we have a **reduction factor** of $\frac{m \cdot n}{m+n}$.[1]

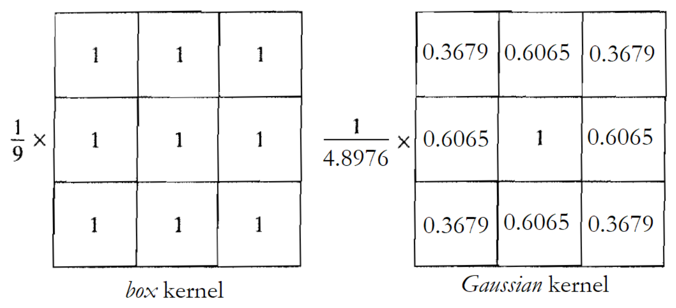
But **how to find** the separable filters?

1) Find any nonzero element in the kernel and let E denote its value.

2) Form vectors c and r equal, respectively, to the column and row in the kernel containing the element found in step 1.

3) Let $w_1 = c$ and $w_2^T = \frac{r}{E}$.

5.1.4 Lowpass Filter Kernels



$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \frac{1}{4.8976} \times \begin{bmatrix} 0.3679 & 0.6065 & 0.3679 \\ 0.6065 & 1 & 0.6065 \\ 0.3679 & 0.6065 & 0.3679 \end{bmatrix}$$

box kernel — *Gaussian* kernel

Two 3 × 3 smoothing (averaging) filter masks. The constant multiplier in front of each mask is equal to 1 divided by the sum of the values of its coefficients, as is required to compute an average.

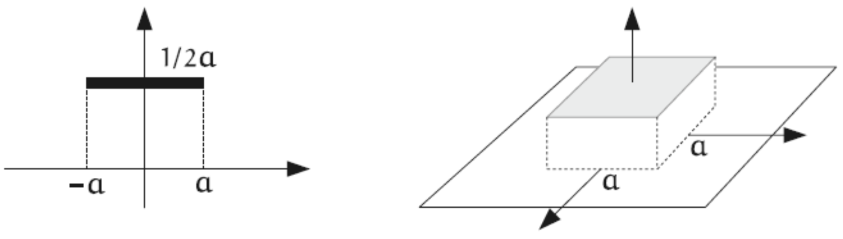
Box Filter In one dimension, the box filter has kernel:

$$box_{1D}(x) = \begin{cases} 1/(2a) & \text{if } -a \le x \le a \\ 0 & \text{if } |x| > a. \end{cases}$$

The definition just given can easily be extended to the two-dimensional case imposing a separability condition:

$$box_{2D}(x, y) = box_{1D}(x)\; box_{1D}(y)$$

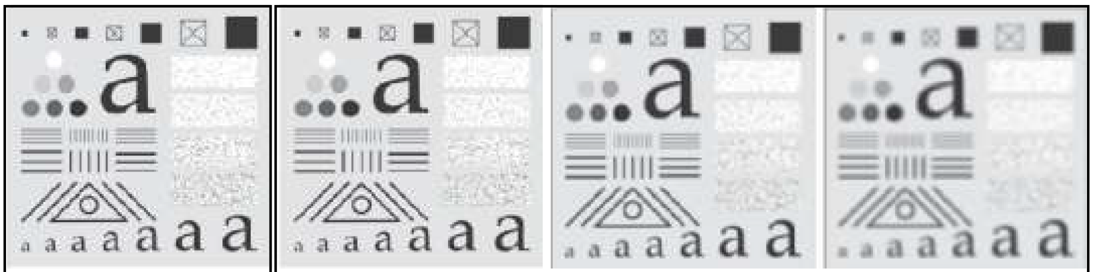
The graph of those kernel is shown in the following figure:



Kernel of the box filter in one and two dimensions.

The box filter is a **lowpass filter**. This is easy to verify by analyzing the filter in the frequency domain. In fact, the transfer function of the filter box_{2D} is the two-dimensional *sinc* function. Thus, in the frequency domain, filtering (in the spatial domain) with a box filter corresponds to multiplying the Fourier transform by the *sinc* function; this clearly dampens high frequencies.

[1]It's a lot, if we think about an 11×11 kernel the factor is of 5.2!!!



Test pattern of size 1024x1024 pixels

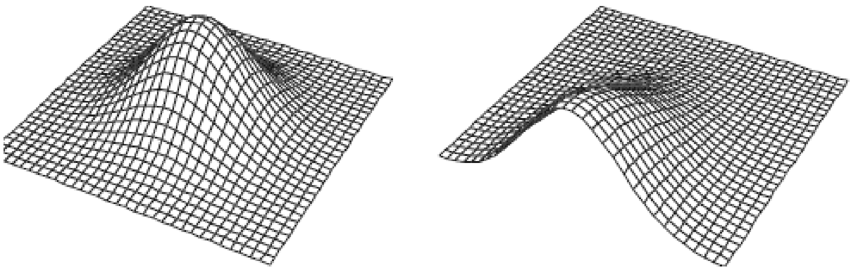
Result of lowpass filtering with box kernels of size 3x3 , 11x11 and 21x21

Gaussian Filter In one dimension the kernel $w(x) = G_\sigma(x)$ of the continuous-domain gaussian filter is given by the gaussian function

$$G_\sigma(x) = K\, e^{-\dfrac{x^2}{2\sigma^2}}$$

where σ is a constant, called the **variance** of the function. In two dimensions the kernel is defined by (note how the definition makes this filter separable too):

$$G_\sigma(x, y) = K\, e^{-\dfrac{x^2 + y^2}{2\sigma^2}}$$

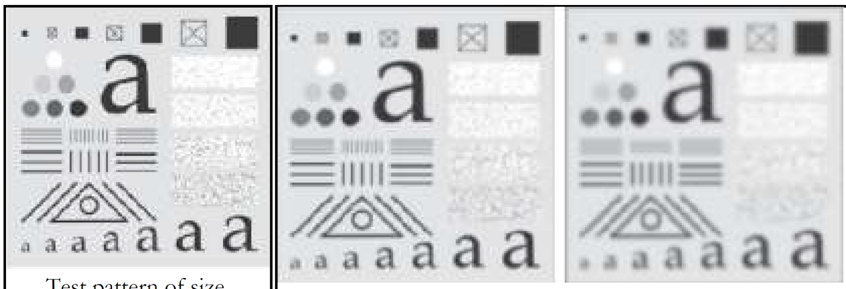


Gaussian distribution function with mean 0 and variance 2

So:

$$w(s, t) = K\, e^{-\dfrac{s^2 + t^2}{2\sigma^2}}$$

A quick analysis in the continuous domain shows that the gaussian filter is a **lowpass filter**. We just observe that the Fourier transform of a gaussian distribution is also a gaussian. In other words, the transfer function is gaussian, so that high frequencies in the filtered signal are damped by a factor that grows exponentially with the frequency. The transfer function assumes only nonnegative values, and the rotational symmetry of the gaussian shows that the filter is **isotropic**.



Test pattern of size 1024x1024 pixels

1)

2)

1) Result of lowpass filtering with a Gaussian kernel of size 21x21 and Sigma = 3.5

2) Result of lowpass filtering with a Gaussian kernel of size 43x43 and Sigma = 7

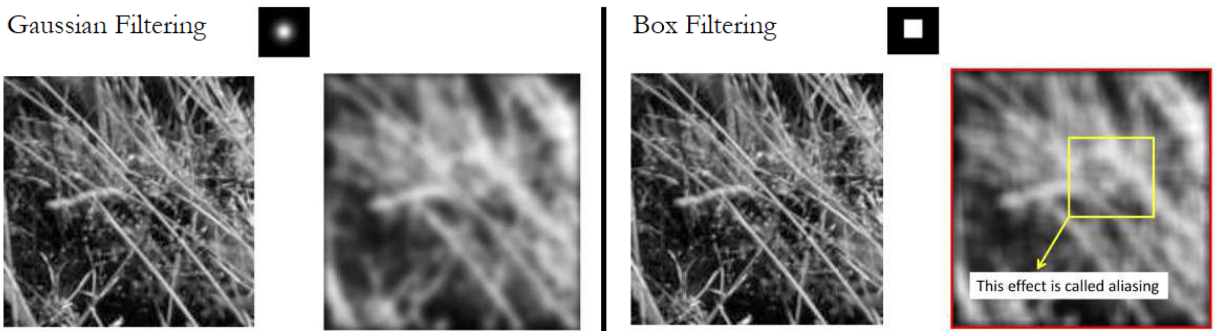
Exercise Solve the following questions:

1) Show that the Gaussian kernel, $G(s;t)$ is separable.

2) Because G is separable and circularly symmetric, it can be expressed in the form $G = w_1 w_1^T$. Assume that the following kernel form is used and that the function is sampled to yield an $m \times m$ kernel. What is w_1 in this case?

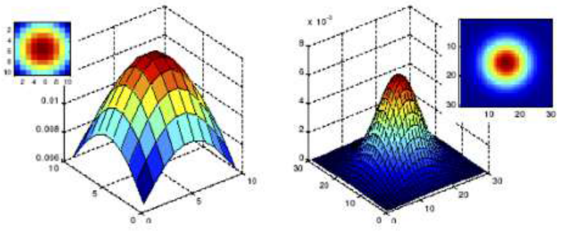
$$G(r) = K\, e^{-\dfrac{r^2}{2\sigma^2}}$$

Here there is a comparison between the two lowpass filter shown:

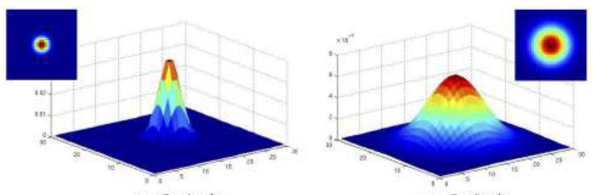


So, Gaussian kernels are separable. But they have also another interesting property: the **product** and **convolution** of two Gaussian functions are **Gaussian functions too**.

What **parameters** matter in a **Gaussian filter**?
– The **size of the kernel** (NB: a Gaussian function has infinite support, but discrete filters use finite kernels)


σ = 5 pixels with 10 x 10 pixel kernel   σ = 5 pixels with 30 x 30 pixel kernel

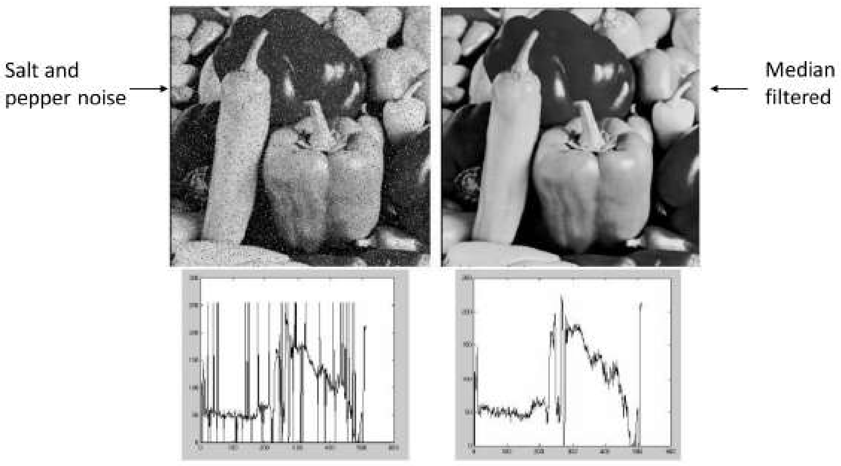
- The **variance** (determines extent of smoothing)


σ = 2 pixels with 30 x 30 pixel kernel   σ = 5 pixels with 30 x 30 pixel kernel

5.1.5 Non-linear filtering

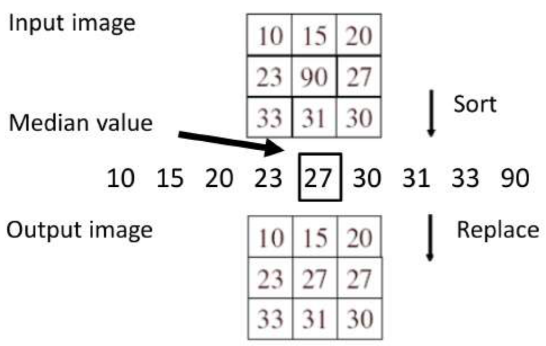
There is a problem with linear smoothing filters: they **do not alleviate salt and pepper noise**! How we can do it?

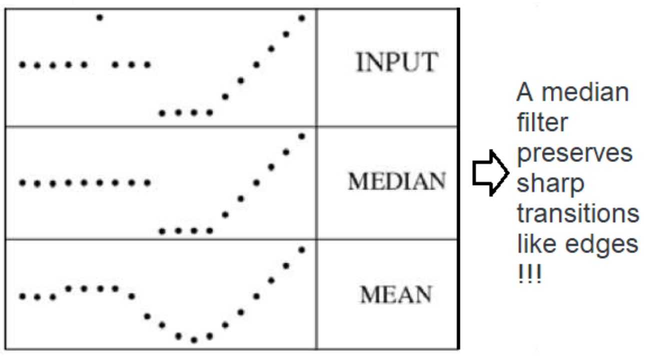
Median filter It is a non-linear filter that **removes spikes**: is good for impulse and salt & pepper noise:



Plots of a row of the image

Here is presented the kernel of the median filter:





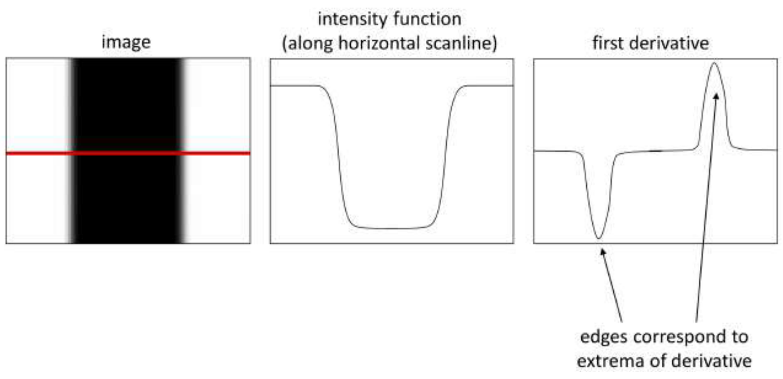
The advantage of the Median filter is that preserves sharp transitions, but the drawback is that it **removes small brightness variations**.

5.2 Edge detection

5.2.1 1D Sharpening (Highpass) Spatial Filters

Highpass filters are used to perform **Edge Detection** on the image processed. The ultimate goal of the edge detection is to have an "idealized line drawing". Edge detection is fundamental in computer vision because edge contours in the image correspond to important **scene contours**.

An edge is a place of rapid change in the image intensity function:



The **derivatives** of a digital function are defined in terms of differences.

First derivative:
– Must be zero in areas of constant intensity.
– Must be nonzero at the onset of an intensity step or ramp.
– Must be nonzero along intensity ramps.

Second derivative:
– Must be zero in areas of constant intensity.
– Must be nonzero at the onset and end of an intensity step or ramp.
– Must be zero along intensity ramps.

For 2D function, $f(x,y)$, the partial derivative is:

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon, y) - f(x,y)}{\epsilon}$$

For discrete data, we can approximate using finite differences:

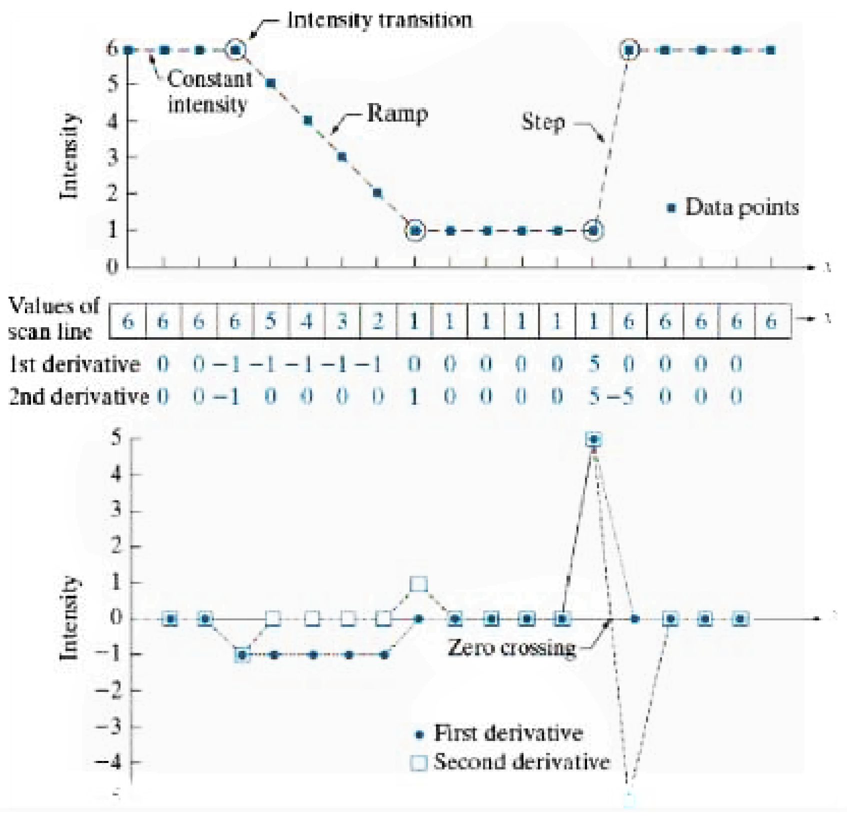
$$\frac{\partial f(x,y)}{\partial x} \approx f(x+1,y) - f(x,y)$$

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

Second-order derivative:

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

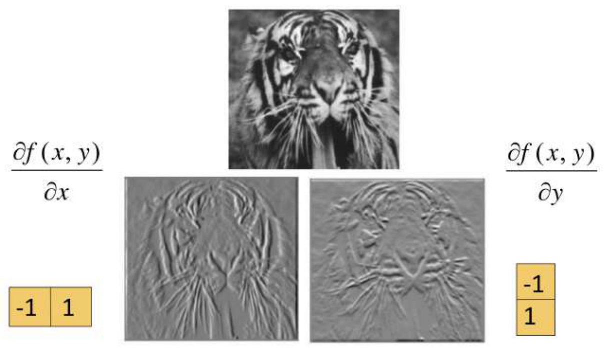
$$\frac{\partial^2 f(x,y)}{\partial y^2} \approx f(x,y+1) + f(x,y-1) - 2f(x,y)$$



(a) A section of a horizontal scan line from an image, showing ramp and step edges, as well as constant segments.
(b) Values of the scan line and its derivatives.
(c) Plot of the derivatives, showing a zero crossing. In (a) and (c) points were joined by dashed lines as a visual aid.

To implement the above as a convolution, what would be the associated filter?

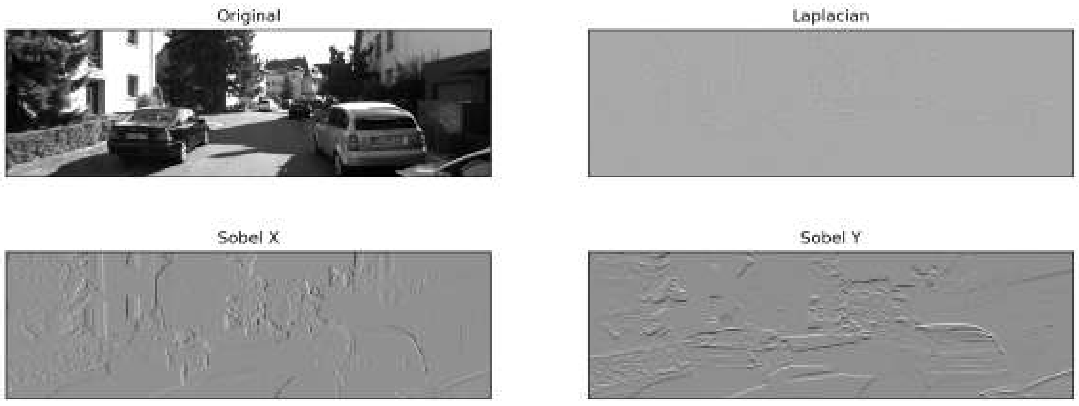
Partial derivatives filter This filter performs the derivate along one axis:


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

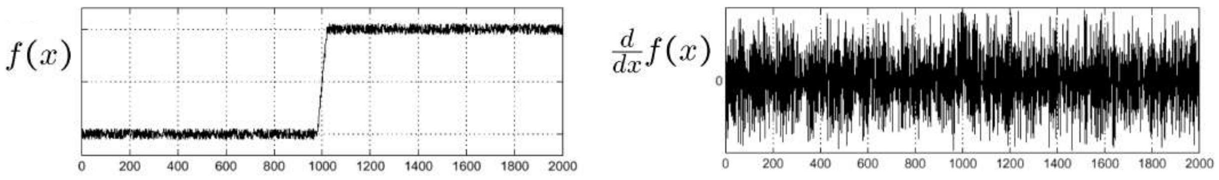
Alternative Finite-difference filters We can have different kernels that performs this kind of operation:

Prewitt filter $\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}$ and $\mathbf{G_y} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}$

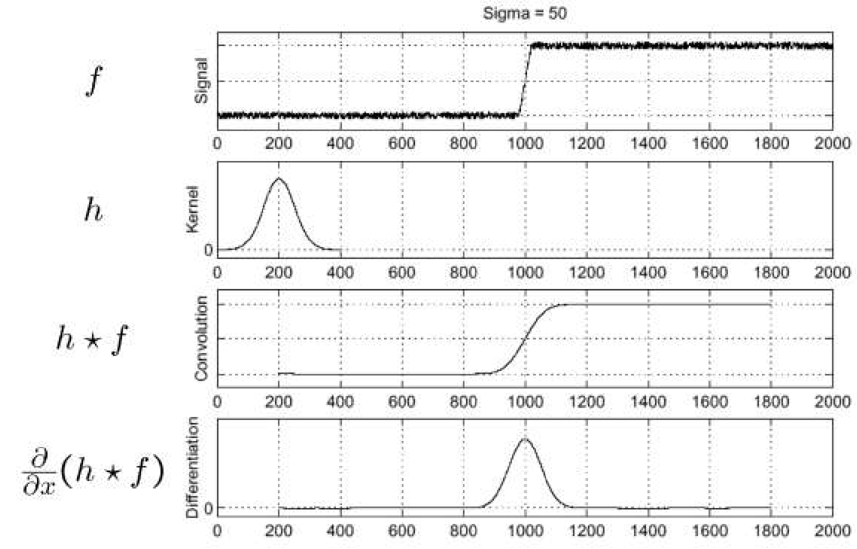
Sobel filter $\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$ and $\mathbf{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$



Noise Smoothing Let's consider a single row or column of an image. We know that there will be a certain amount of noise, that we can visualize by plotting intensity as a function of position:



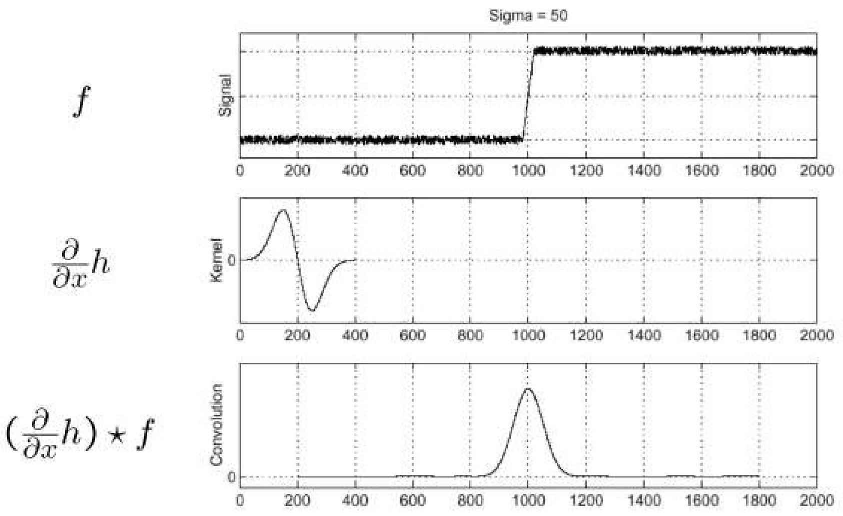
As we can notice, from that kind of derivate one cannot understand where is the edge. The solution is to **smooth** the image first and then perform the derivation (in the following image the single line of the image f is convoluted by a 1D Gaussian filter h):



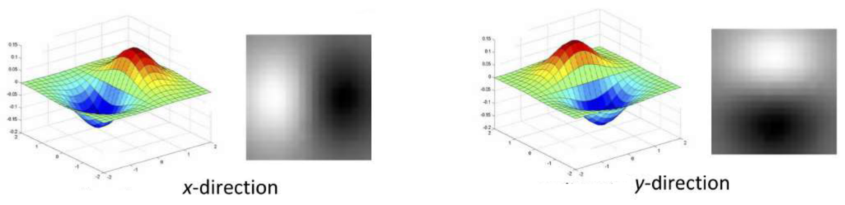
An alternative is to combine derivative and smoothing filter in a unique filter, using the differentiation property of the convolution:

$$\frac{\partial}{\partial x}(h \star f) = \left(\frac{\partial}{\partial x}h\right) \star f$$

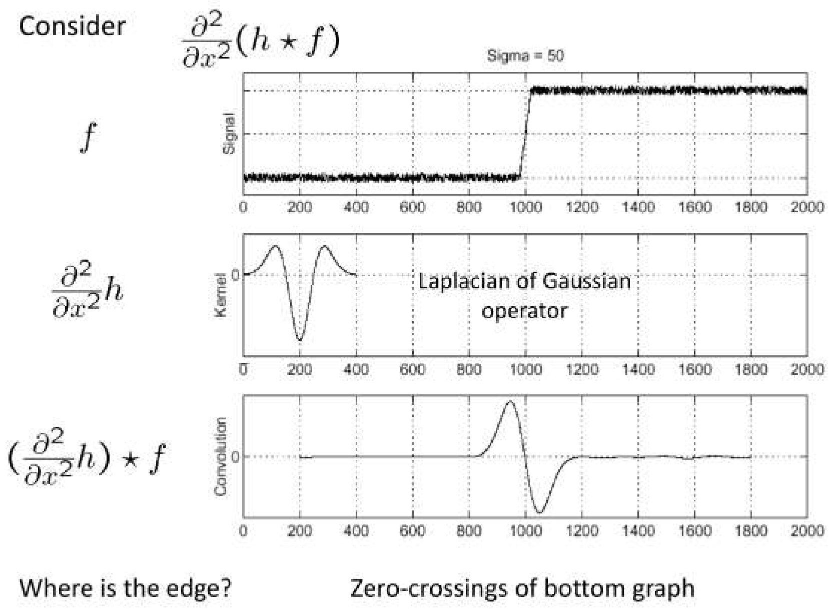
The result is the following:



Derivative of Gaussian filter We can then introduce this effective derivative filters (one for axis direction):



Laplacian of Gaussian We can go farther and use the second derivative of Gaussian filter:



Remind: the Laplacian operator is $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

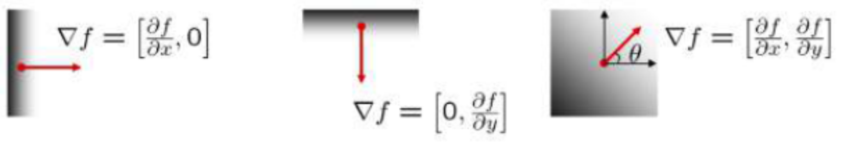
5.2.2 2D Sharpening Spatial Filters

Image gradient To perform derivation in 2D **image gradient**-based filters could be useful:

The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient points in the direction of fastest intensity change


$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$
$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$
θ
$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

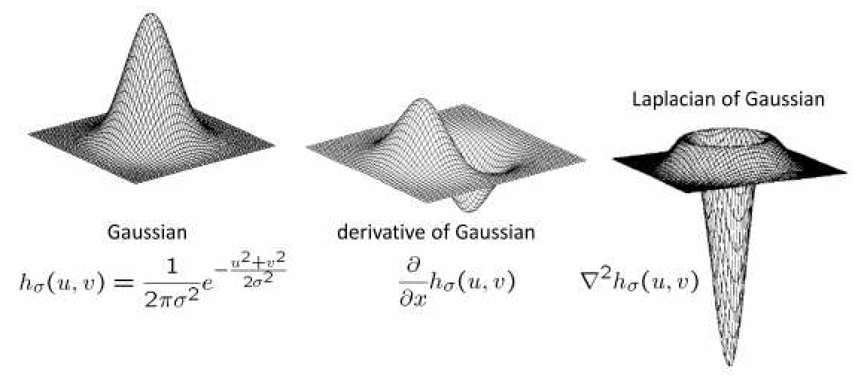
The gradient direction (orientation of edge normal) is given by:

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

The *edge strength* is given by the gradient magnitude

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

Since we defined the previous filters in 1D, we can now easily extend them in 2D:

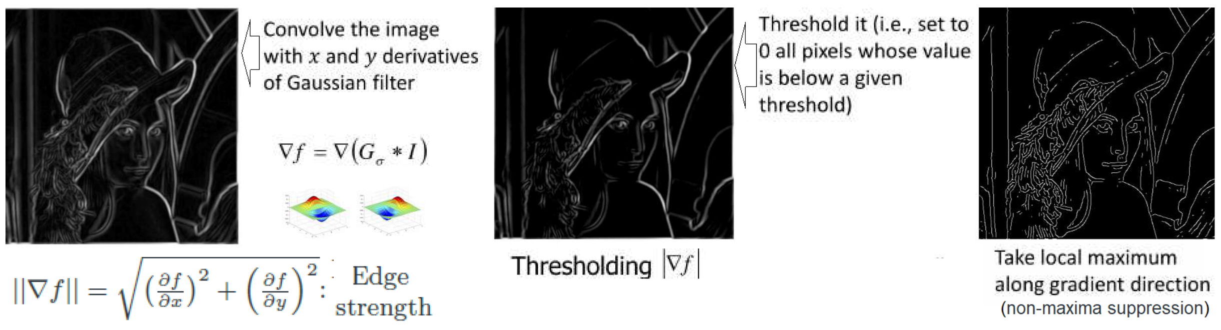


Summary on (linear) filters

- Smoothing filter:
    - has positive values
    - sums to 1 $\rightarrow$ preserve brightness of constant regions
    - removes "high-frequency" components: "low-pass" filter
- Derivative filter:
    - has opposite signs used to get high response in regions of high contrast
    - sums to 0 $\rightarrow$ no response in constant regions
    - highlights "high-frequency" components: "high-pass" filter

5.3 Canny edge-detection algorithm

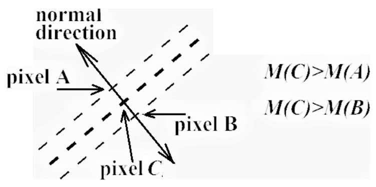
The Canny edge-detection algorithm (1986) computes the gradient of a smoothed image in both directions. It discards pixels whose gradient magnitude is below a certain threshold and implements a **non-maximal suppression**: it identifies local maxima along gradient direction:


Convolve the image with x and y derivatives of Gaussian filter
$\nabla f = \nabla(G_\sigma * I)$
$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$: Edge strength
Threshold it (i.e., set to 0 all pixels whose value is below a given threshold)
Thresholding $|\nabla f|$
Take local maximum along gradient direction
(non-maxima suppression)

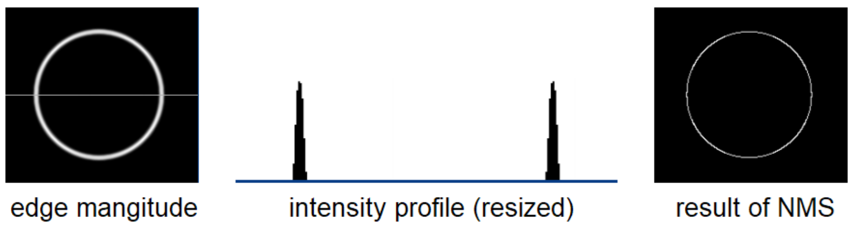
What is the "Non-maxima suppression"? Due to the multiple response, edge magnitude $M(x, y)$ may contain **wide ridges** around the local maxima. Non-maxima suppression removes the non-maxima pixels **preserving the connectivity** of the contours.

Algorithm 1: Non-maxima suppression:
1. From each position (x, y), step in the two directions perpendicular to edge orientation $\Theta(x, y)$.
2. Denote the inital pixel (x, y) by C, the two neighbouring pixels in the perpendicular directions by A and B.
3. If the $M(A) > M(C)$ or $M(B) > M(C)$, discard the pixel (x, y) by setting $M(x, y) = 0$.

In the following image is illustrated the non-maxima suppression. Pixels A and B are deleted because $M(C) > M(A)$ and $M(C) > M(B)$. Pixel C is not deleted:



In the following image is illustrated the process of thinning wide contours in edge magnitude images by non-maxima suppression. The intensity profile along the indicated line is shown resized for better visibility:



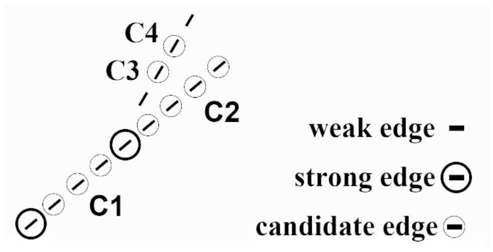
Hysteresis thresholding The output of the non-maxima suppression still contains noisy local maxima. The contrast (edge strength) may be different in different points of the contour. So, a careful thresholding of $M(x, y)$ is needed to remove these weak edges while preserving the connectivity of the contours.

Hysteresis thresholding receives the output of the non-maxima suppression, $M_{NMS}(x, y)$. The algorithm uses 2 thresholds, T_{high} and T_{low}:
– A pixel (x, y) is called **strong** if $M_{NMS}(x, y) > T_{high}$.
– A pixel (x, y) is called **weak** if $M_{NMS}(x, y) \leq T_{low}$.
– All other pixels are called **candidate** pixels.

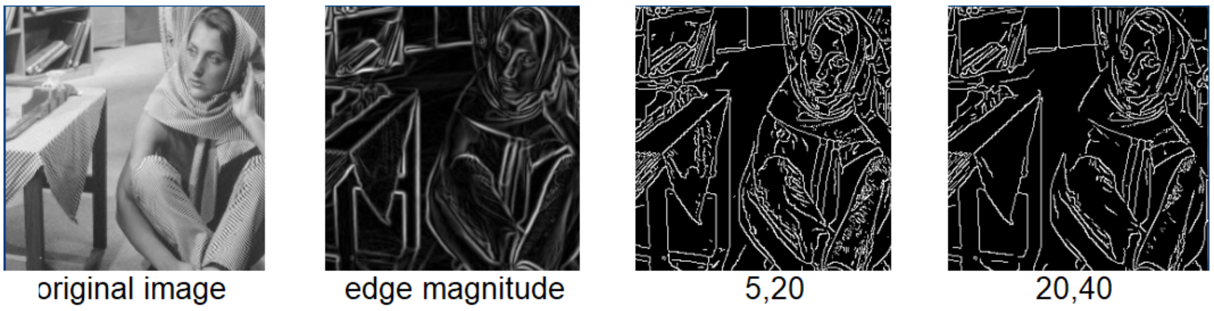
Algorithm 2: Hysteresis thresholding
1. In each position of (x, y), discard the pixel (x, y) if it is weak; output the pixel if it is strong.

2. If the pixel is a candidate, follow the chain of connected local maxima in both directions along the edge, as long as $M_{NMS} > T_{low}$.
3. If the starting candidate pixel (x, y) is connected to a strong pixel, output this candidate pixel; otherwise, do not output the candidate pixel.

In the following image is illustrated the hysteresis thresholding. The candidate edges C1 and C2 are output, the candidate edges C3 and C4 are not:



We can see here some examples of edge localisation with different hysteresis thresholds:



Chapter 6

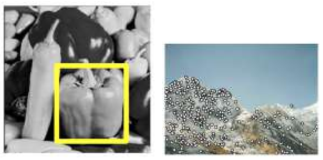
Feature Point Detection

6.1 Filters for Feature detection and Point-feature extraction

Filters for Feature Detection

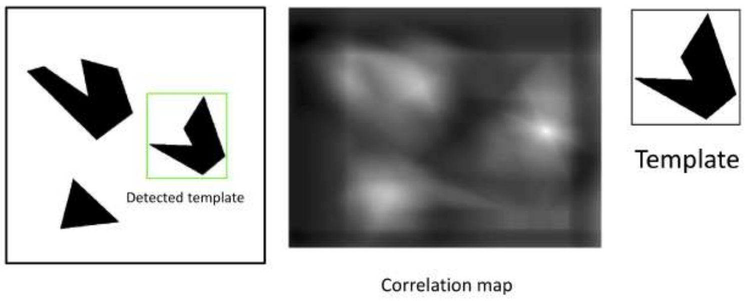
- In the last lecture, we used filters to reduce noise or enhance contours

- However, filters can also be used to detect "**features**"
  - Goal: reduce amount of data to process in later stages, discard redundancy to preserve only what is useful (leads to **lower bandwidth and memory storage**)
    - **Edge detection** (we have seen this already; edges can enable line or shape detection)
    - **Template matching**
    - **Keypoint detection**



Filters for Template Matching

- Find locations in an image that are similar to a **template**
- If we look at filters as templates, we can use **correlation** (like convolution but without flipping the filter) to detect these locations

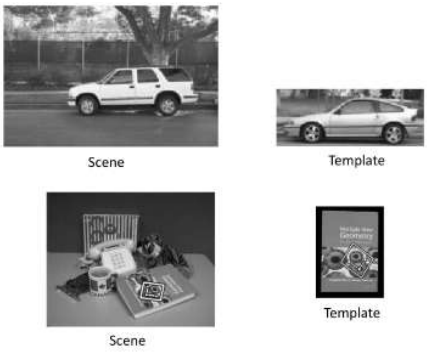


Summary of Filters

- Smoothing filter:
  - has positive values
  - sums to 1 $\rightarrow$ preserve brightness of constant regions
  - removes "high-frequency" components: "low-pass" filter
- Derivative filter:
  - has opposite signs used to get high response in regions of high contrast
  - sums to 0 $\rightarrow$ no response in constant regions
  - highlights "high-frequency" components: "high-pass" filter
- Filters as templates:
  - Highest response for regions that "look similar to the filter"

Template Matching

- What if the template is not identical to the object we want to detect?
- Template Matching will only work if scale, orientation, illumination, and, in general, the appearance of the template and the object to detect are very similar. What about the pixels in template background (object-background problem)?



Correlation as Scalar Product

- Consider images H and F as vectors, their correlation is:

$$\langle H, F \rangle = \|H\|.\|F\|.\cos\theta$$

- In Normalized Cross Correlation (NCC), we consider the unit vectors of H and F, hence we measure their similarity based on the angle θ. If H and F are identical, then NCC $= 1$.

$$\cos\theta = \frac{\langle H, F \rangle}{\|H\|.\|F\|}$$

$$= \frac{\sum_{u=-k}^{k}\sum_{v=-k}^{k} H(u,v)F(u,v)}{\sqrt{\sum_{u=-k}^{k}\sum_{v=-k}^{k} H(u,v)^2}\sqrt{\sum_{u=-k}^{k}\sum_{v=-k}^{k} F(u,v)^2}}$$

Other Similarity Measures

- Sum of Absolute Differences (SAD) (used in optical mice)

$$SAD = \sum_{u=-k}^{k} \sum_{v=-k}^{k} |H(u,v) - F(u,v)|$$

- Sum of Squared Differences (SSD)

$$SSD = \sum_{u=-k}^{k} \sum_{v=-k}^{k} (H(u,v) - F(u,v))^2$$

- Normalized Cross Correlation (NCC): takes values between -1 and +1 (+1 = identical)

$$NCC = \frac{\sum_{u=-k}^{k} \sum_{v=-k}^{k} H(u,v)F(u,v)}{\sqrt{\sum_{u=-k}^{k} \sum_{v=-k}^{k} H(u,v)^2}\sqrt{\sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)^2}}$$

Zero-mean SAD, SSD, NCC

To account for the difference in mean of the two images (typically caused by illumination changes), we subtract the mean value of each image:

- Zero-mean Sum of Absolute Differences (ZSAD)

  $ZSAD = \sum_{u=-k}^{k} \sum_{v=-k}^{k} |(H(u,v) - \mu_H) - (F(u,v) - \mu_F)|$

- Zero-mean Sum of Squared Differences (ZSSD)

  $SSD = \sum_{u=-k}^{k} \sum_{v=-k}^{k} ((H(u,v) - \mu_H) - (F(u,v) - \mu_F))^2$

- Zero-mean Normalized Cross Correlation (ZNCC)

  $ZNCC = \frac{\sum_{u=-k}^{k} \sum_{v=-k}^{k} (H(u,v)-\mu_H)(F(u,v)-\mu_F)}{\sqrt{\sum_{u=-k}^{k} \sum_{v=-k}^{k} (H(u,v)-\mu_H)^2}\sqrt{\sum_{u=-k}^{k} \sum_{v=-k}^{k} (F(u,v)-\mu_F)^2}}$

ZNCC is invariant to affine intensity changes!

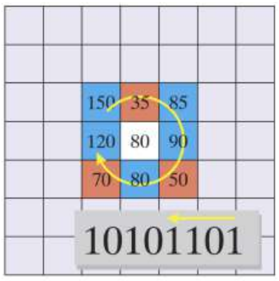
$$\mu_H = \frac{\sum_{u=-k}^{k} \sum_{v=-k}^{k} H(u,v)}{(2N+1)^2}, \mu_F = \frac{\sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)}{(2N+1)^2}$$

Census Transform

- Maps an image patch to a bit string:
  - if a pixel is greater than the center pixel its corresponding bit is set to 1, else to 0
  - For a $w \times w$ window the string will be $w^2 - 1$ bits long
- The two bit strings are compared using the Hamming distance, which is the number of bits that are different. This can be computed by counting the number of 1s in the Exclusive-OR (XOR) of the two bit strings

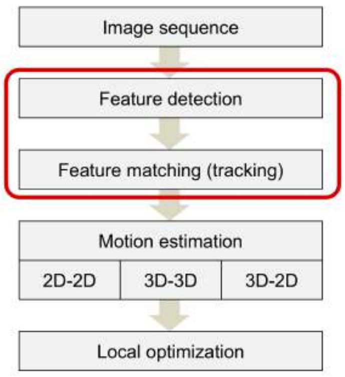
Advantages

- More robust to object-background problem
- No square roots or divisions are required, thus very efficient to implement, especially on FPGA
- Intensities are considered relative to the center pixel of the patch making it invariant to monotonic intensity changes



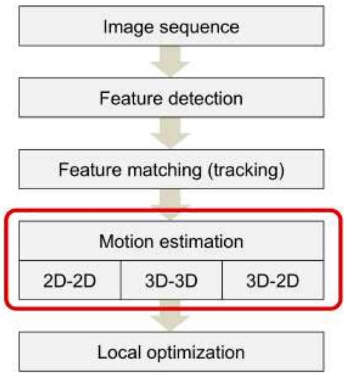
What do we need point features for?

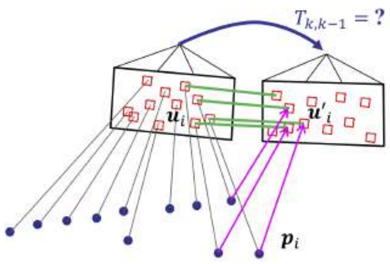
Recall the Visual-Odometry flow chart:



Example of features tracks

Keypoint extraction is the key ingredient of motion estimation!





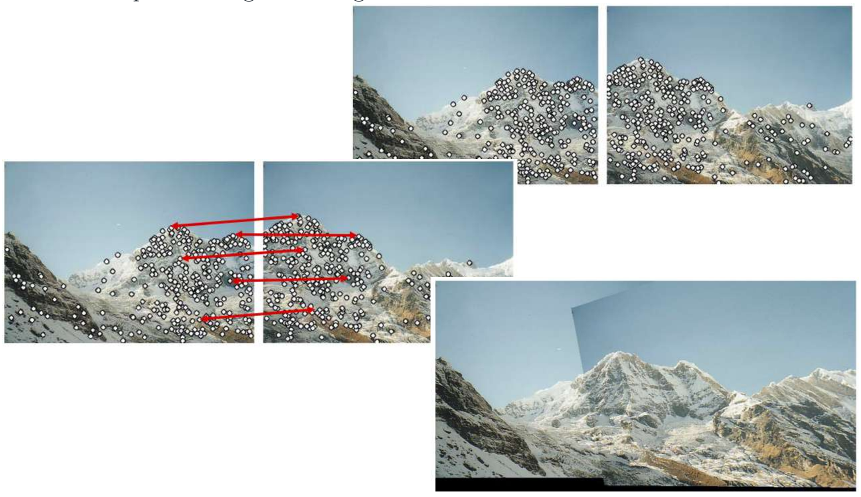
Point Features are also used for:

- Panorama stitching
- Object recognition
- 3D reconstruction
- Place recognition
- Indexing and database retrieval (e.g., Google Images or http://tineye.com)

Example: panorama stitching Local features and alignment

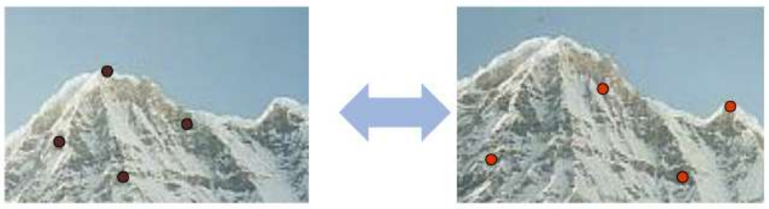
- We need to align images
- How would you do it?

- Detect point features in both images
- Find corresponding pairs
- Use these pairs to align the images



Matching with Features

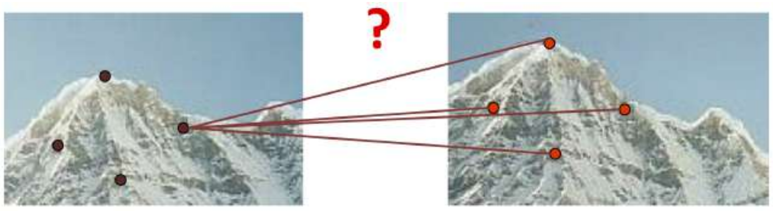
- Problem 1:
  - Detect the same points independently in both images



No chance to match!

We need a repeatable feature detector

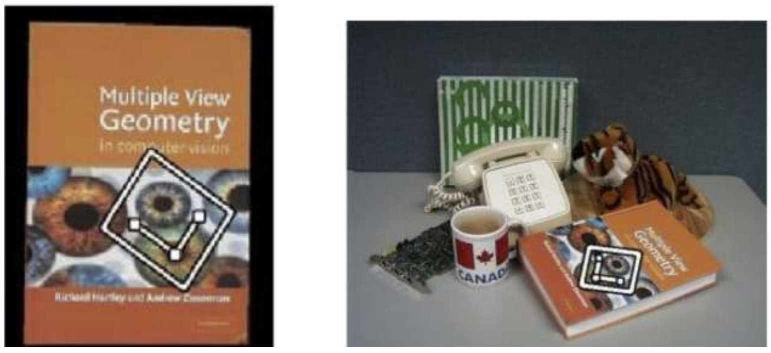
- Problem 2:
  - For each point, identify its correct correspondence in the other image(s)



We need a reliable and distinctive feature descriptor
that is robust to geometric and illumination changes

Geometric changes

- Rotation
- Scale (i.e., zoom)
- View point (i.e, perspective changes)



Illumination changes

Typically, small illumination changes are modeled with an affine transformation (so called affine illumination changes):

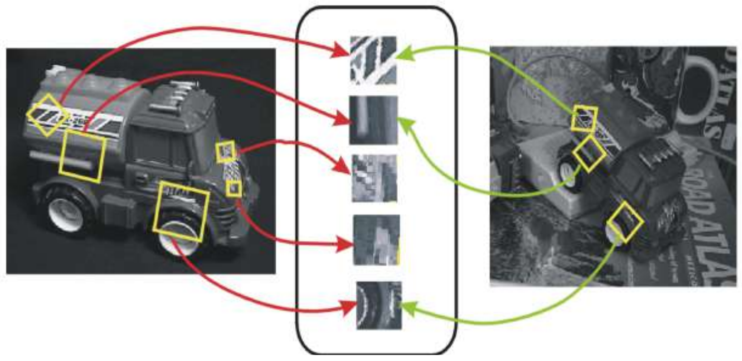
$$I'(x, y) = \alpha I(x, y) + \beta$$

Invariant local features

Subset of local feature types designed to be invariant to common geometric and photometric transformations.

Basic steps:

- Detect distinctive interest points
- Extract invariant descriptors

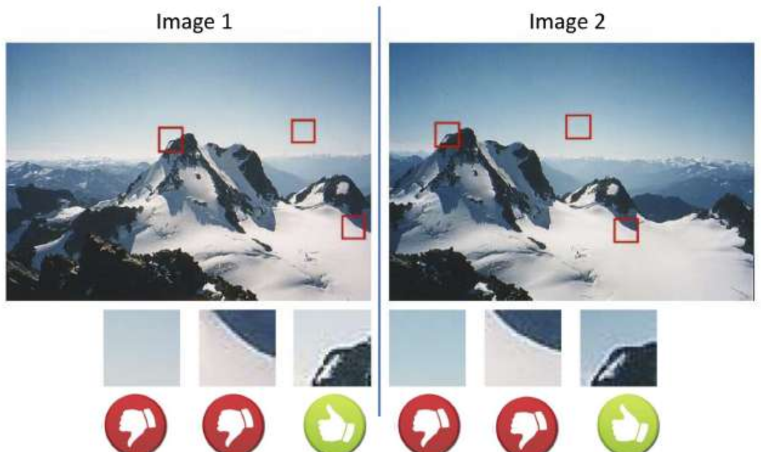


Main questions

- What points are distinctive (i.e., features, keypoints, salient points), such that they are repeatable? (i.e., can be re-detected from other views)
- How to describe a local region?
- How to establish correspondences, i.e., compute matches?

What is a distinctive feature?

- Consider the image pair below with extracted patches
- Notice how some patches can be localized or matched with higher accuracy than others



Feature Points: Corners vs Blob Detectors

- A corner is defined as the intersection of one or more edges
  - A corner has high localization accuracy
  - It’s less distinctive than a blob
  - E.g., Harris, Shi-Tomasi, SUSAN, FAST

- A blob is any other image pattern, which is not a corner, that differs significantly from its neighbors in intensity and texture (e.g., a connected region of pixels with similar color, a circle, etc.)
  - Has less localization accuracy than a corner
  - Blob detectors are better for place recognition
  - It’s more distinctive than a corner
  - E.g., MSER, LOG, DOG (SIFT), SURF, CenSurE

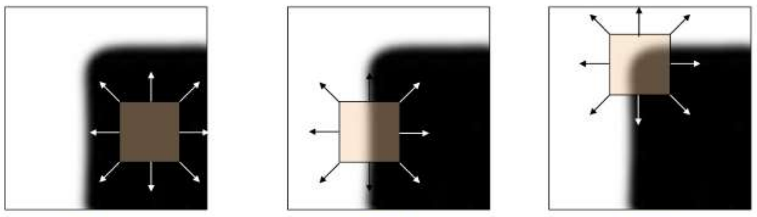
Corner detection

- Key observation: in the region around a corner, image gradient has two or more dominant directions
- Corners are repeatable and distinctive

C.Harris and M.Stephens. "A Combined Corner and Edge Detector." , 1988 Proceedings of the 4th Alvey Vision Conference: pages 147–151.

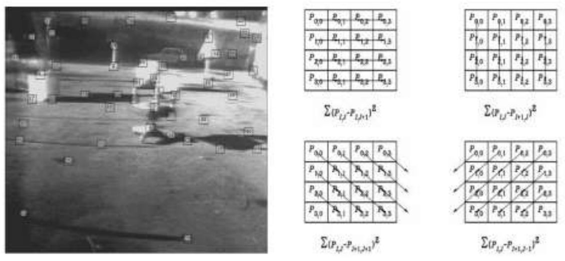
The Moravec Corner detector (1980)

- How do we identify corners?
- We can easily recognize the point by looking through a small window
- Shifting a window in any direction should give a large change in intensity (e.g., in SSD) in at least 2 directions


“flat” region:
no intensity change
(i.e., SSD ≈ 0 in all directions)
“edge”:
no change along the edge direction
(i.e., SSD ≈ 0 along edge but ≫ 0 in other directions)
“corner”:
significant change in at least 2 directions
(i.e., SSD ≫ 0 in all directions)

H. Moravec, Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover, PhD thesis, Chapter 5, Stanford University, Computer Science Department, 1980.

"Sums of squares of differences of pixels adjacent in each of four directions (horizontal, vertical and two diagonals) over each window are calculated, and the window's interest measure is the minimum of these four sums." [Moravec'80, Ch. 5]



H. Moravec, Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover, PhD thesis, Chapter 5, Stanford University, Computer Science Department, 1980.

The Harris Corner detector (1988)

- It implements the Moravec corner detector without having to physically shift the window but rather by just looking at the patch itself, by using differential calculus.

C.Harris and M.Stephens. "A Combined Corner and Edge Detector.? , 1988 Proceedings of the 4th Alvey Vision Conference: pages 147-151.

How do we implement this?

- Consider the reference patch centered at (x, y) and the shifted window centered at $(x + \Delta x, y + \Delta y)$. The patch has size P.
- The Sum of Squared Differences between them is:

$$SSD(\Delta x, \Delta y) = \sum_{x,y \in P} (I(x, y) - I(x + \Delta x, y + \Delta y))^2$$

- Let $I_x = \frac{\partial I(x,y)}{\partial x}$ and $I_y = \frac{\partial I(x,y)}{\partial y}$. Approximating with a 1st order Taylor expansion:

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

- This produces the approximation

$$SSD(\Delta x, \Delta y) \approx \sum_{x,y \in P} (I_x(x, y)\Delta x + I_y(x, y)\Delta y)^2$$

This is a simple quadratic function in two variables $(\Delta x, \Delta y)$

$$SSD(\Delta x, \Delta y) = \sum_{x,y \in P} (I(x,y) - I(x+\Delta x, y+\Delta y))^2$$

- This can be written in a matrix form as

$$SSD(\Delta x, \Delta y) \approx \sum \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

$$\Rightarrow SSD(\Delta x, \Delta y) \approx \sum \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \mathtt{M} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

With

$$\mathtt{M} = \sum_{x,y \in P} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

Pixel-wise products in the autocorrelation matrix

What does this matrix reveal?

- First, consider an edge or a flat region.

Edge

$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

Flat region

$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

- We can conclude that if either λ is close to 0, then this is not a corner.
- Now, let's consider an axis-aligned corner:

Corner

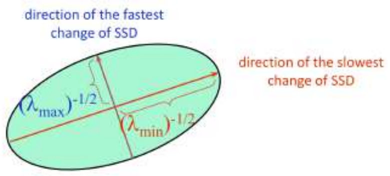
$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \cos\frac{\pi}{4} & \sin\frac{\pi}{4} \\ -\sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{bmatrix}$$

- This means dominant gradient directions are at 45 degrees with x and y axes
- What if we have a corner that is **not aligned** with the image axes?

General Case

Since $\mathtt{M}$ is symmetric, it can always be decomposed into $\mathtt{M} = \mathtt{R}^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \mathtt{R}$

- We can visualize $\begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \mathtt{M} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = const$ as an ellipse with axis lengths determined by the eigenvalues and the two axes' orientations determined by $\mathtt{R}$ (i.e., the eigenvectors of $\mathtt{M}$)
- The two eigenvectors identify the directions of largest and smallest changes of SSD


direction of the fastest change of SSD
direction of the slowest change of SSD
$(\lambda_{max})^{-1/2}$
$(\lambda_{min})^{-1/2}$

How to compute λ_1, λ_2, $\mathtt{R}$ from $\mathtt{M}$

Eigenvalue/eigenvector review

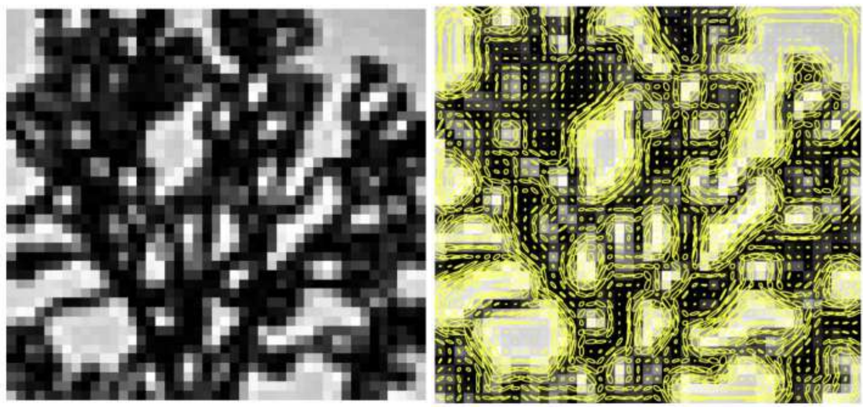
- You can easily proof that λ_1, λ_2 are the eigenvalues of $\mathtt{M}$
- The eigenvectors and eigenvalues of a matrix $\mathtt{A}$ are the vectors $\mathbf{x}$ and scalars λ that satisfy:

$$\mathtt{A}\mathbf{x} = \lambda \mathbf{x}$$

- The scalar λ is the eigenvalue corresponding to $\mathbf{x}$
  - The eigenvalues are found by solving $\det(\mathtt{A} - \lambda \mathtt{I})$
  - In our case, $\mathtt{A} = \mathtt{M}$ is a 2×2 matrix, so we have $\det \begin{bmatrix} m_{11} - \lambda & m_{12} \\ m_{21} & m_{22} - \lambda \end{bmatrix} = 0$
  - The solution is: $\lambda_{1,2} = \frac{1}{2}\left[(m_{11} + m_{22}) \pm \sqrt{4m_{12}m_{21} + (m_{11} - m_{22})^2}\right] = 0$
  - Once you know λ, you find the two eigenvectors $\mathbf{x}$ (i.e. the two columns of $\mathtt{R}$) by solving:

$$\begin{bmatrix} m_{11} - \lambda & m_{12} \\ m_{21} & m_{22} - \lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

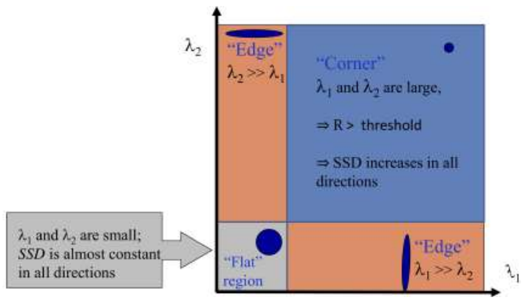
Visualization of 2^{nd} moment matrices



NB: the ellipses here are plot proportionally to the eigenvalues and not as iso-SSD ellipses as explained before. So small ellipses here denote a flat region, and big ones a corner.

Interpreting the eigenvalues

- Classification of image points using eigenvalues of $\mathtt{M}$
- A corner can then be identified by checking whether the minimum of the two eigenvalues of $\mathtt{M}$ is larger than a certain user-defined threshold $\Rightarrow R = \min(\lambda_1, \lambda_2) > \epsilon$
- R is called "cornerness function"
- The corner detector using this criterion is called "Shi-Tomasi" detector

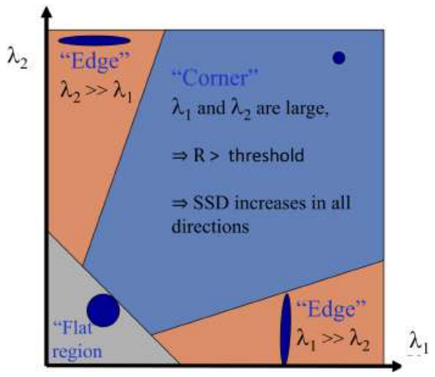

λ_2
"Edge"
$\lambda_2 >> \lambda_1$
"Corner"
λ_1 and λ_2 are large,
⇒ R > threshold
⇒ SSD increases in all directions
λ_1 and λ_2 are small;
SSD is almost constant in all directions
"Flat" region
"Edge"
$\lambda_1 >> \lambda_2$
λ_1

J. Shi and C. Tomasi (June 1994). "Good Features to Track,". 9th IEEE Conference on Computer Vision and Pattern Recognition

- Computation of λ_1 and λ_2 is expensive $\Rightarrow$ Harris & Stephens suggested using a different cornerness function:

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 = \det(\texttt{M}) - k.\text{trace}^2(\texttt{M})$$

- k is a magic number in the range (0.04 to 0.15)



Harris Corner Detector

❶ Compute derivatives in x and y directions (I_x, I_y) e.g. with Sobel filter

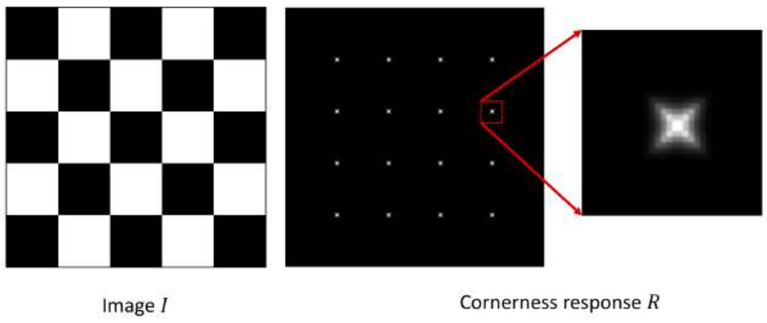
❷ Compute I_x^2, I_y^2, I_xI_y

❸ Convolve I_x^2, I_y^2, I_xI_y with a box filter to get $\sum I_x^2$, $\sum I_y^2$, $\sum I_xI_y$, which are the entries of the matrix M (optionally use a Gaussian filter instead of a box filter to avoid aliasing and give more "weight" to the central pixels)

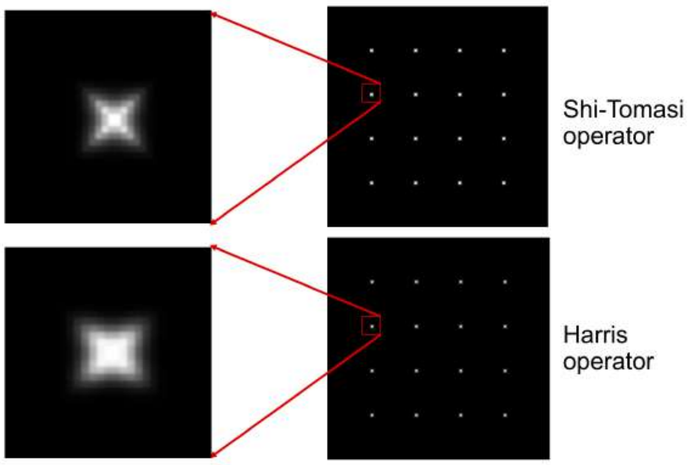
❹ Compute Harris Corner Measure R (according to Shi-Tomasi or Harris)

❺ Find points with large corner response $(R > \epsilon)$

❻ Take the points of local maxima of R

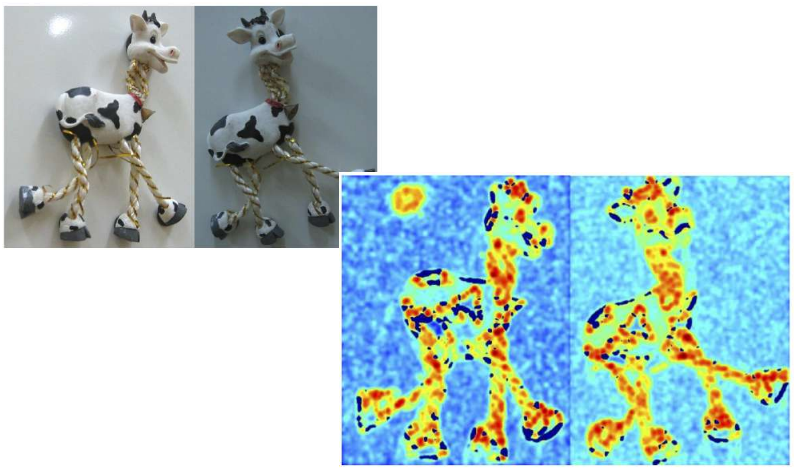


Harris vs Shi-Tomasi

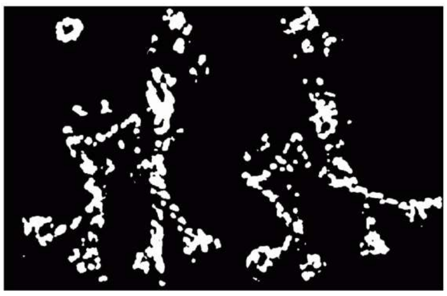


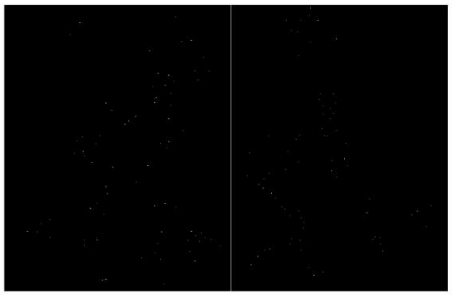
Harris Detector: Workflow

Compute Corner Measure R



Find points with large corner response: $R > \epsilon$





Take only the points of local maxima of thresholded R

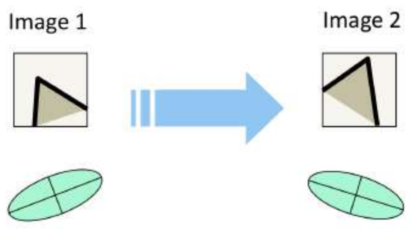
Harris Detector: Some Properties

How does the size of the Harris detector affect the performance?

Repeatability:

- How does the Harris detector behave to common image transformations?
- Can it re-detect the same image patches (Harris corners) when the image exhibits changes in
  - Rotation,
  - View-point,
  - Scale (zoom),
  - Illumination ?
- Solution: Identify properties of detector & adapt accordingly

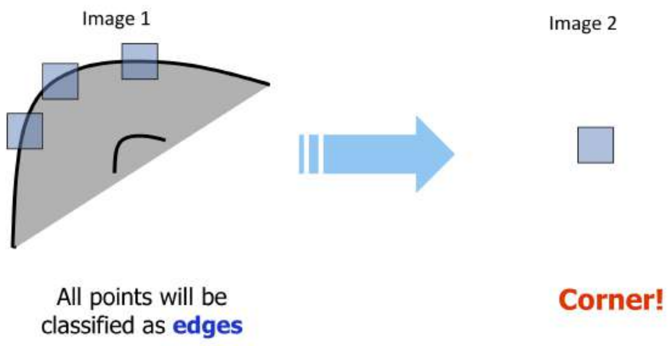
Rotation Invariance:



Ellipse rotates but its shape (i.e., eigenvalues) remains the same

$$\boxed{\text{Corner response } R \text{ is invariant to image rotation}}$$

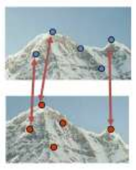
But: non-invariant to image scale!

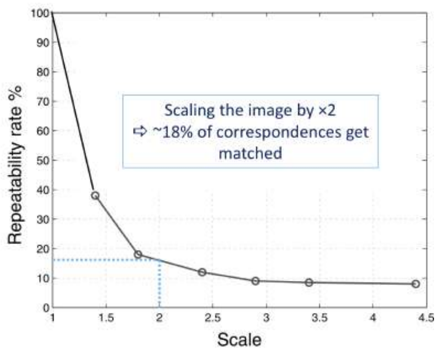


Quality of Harris detector for different scale changes

Repeatability=

$$\frac{\text{\# correspondences detected}}{\text{\# correspondences present}}$$





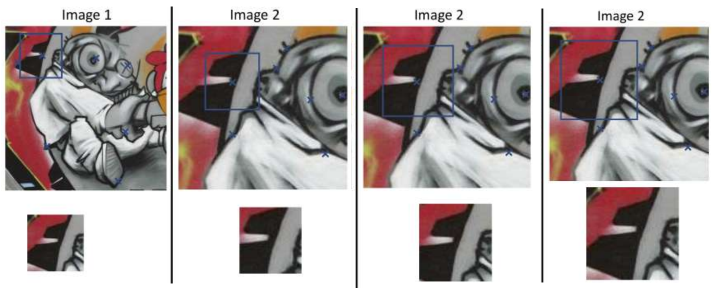
Summary (things to remember)

- Filters as templates
- Correlation as a scalar product
- Similarity metrics: NCC (ZNCC), SSD (ZSSD), SAD (ZSAD), Census Transform
- Point feature detection
  - Properties and invariance to transformations
    - Challenges: rotation, scale, view-point, and illumination changes
  - Extraction
    - Moravec
    - Harris and Shi-Tomasi: Rotation Invariance

6.2 Automatic Scale Selection, Detectors and Descriptors

Scale changes

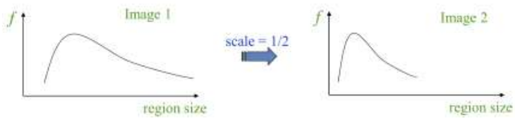
- How can we match image patches corresponding to the same feature but belonging to images taken at different scales?
  - Possible solution: rescale the patch!



- Scale search is time consuming (needs to be done individually for all patches in one image)
  - Complexity would be $(NM)^2$ (assuming that we have N features per image and M scale levels for each image)
- Possible solution: assign each feature its own "scale" (i.e., size).
  - What's the optimal scale (i.e., size) of the patch?

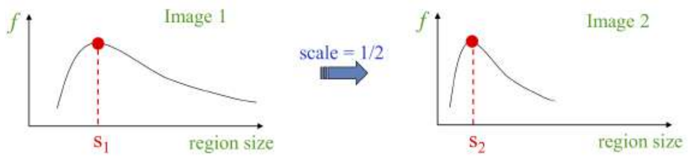
Automatic Scale Selection

- Solution:
  - Design a function on the image patch, which is "scale invariant" (i.e., which has the same value for corresponding regions, even if they are at different scales)
  - For a point in one image, we can consider it as a function of region size (patch width)

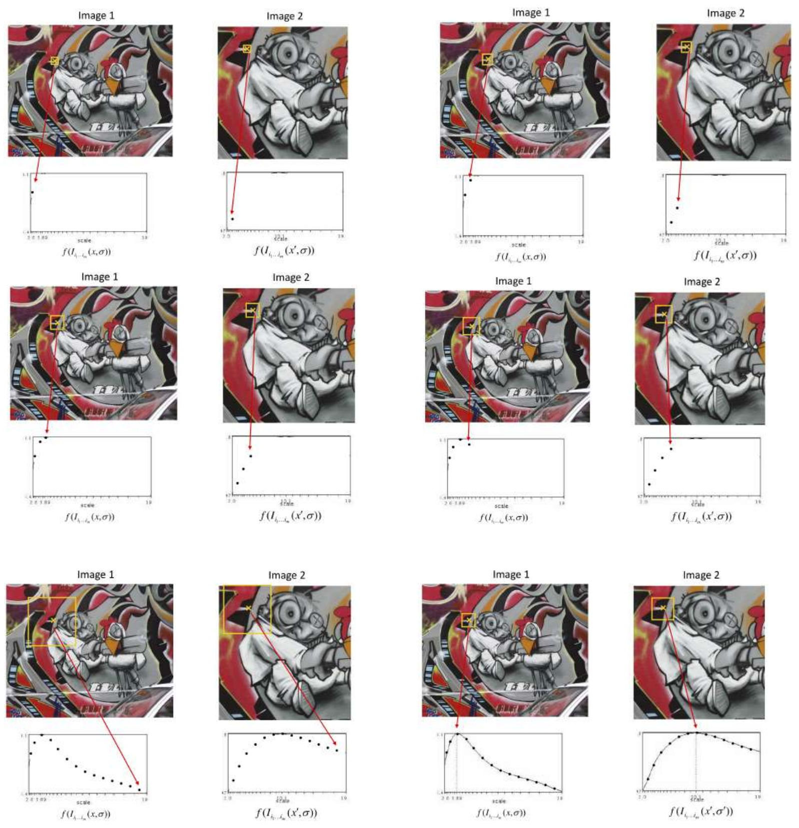


- Common approach:
  - Take a local maximum or minima of this function
  - Observation: region size, for which the maximum or minima is achieved, should be *invariant* to image scale.

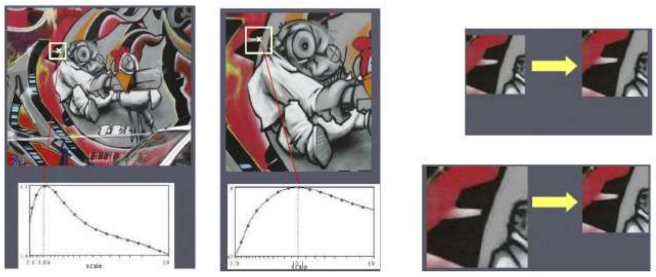
Important: this scale invariant region size is found in each image independently!



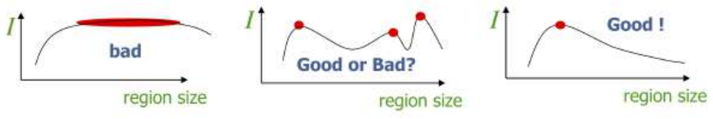
- Function responses for increasing scale (scale signature)



  - When the right scale is found, the patch must be normalized

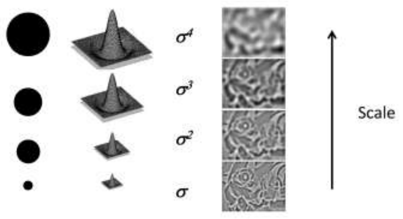


- A "good" function for scale detection should have a single & sharp peak



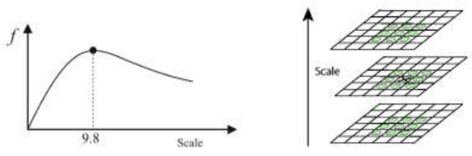
- What if there are multiple peaks?
- **Sharp, local intensity changes** are good regions to monitor in order to identify the scale
  $\Rightarrow$ **Blobs and corners** are the ideal locations!

- Function for determining scale: convolve image with kernel to identify sharp intensity discontinuities
$$f = \text{Kernel} \star \text{Image}$$
- It has been shown that the Laplacian of Gaussian kernel is optimal under certain assumptions [Lindeberg?94]:
$$LoG = \nabla^2 G(x,y) = \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2}$$
- Correct scale is found as local maxima or minima across consecutive smoothed images



Lindeberg, Scale-space theory: A basic tool for analysing structures at different scales, Journal of Applied Statistics, 1994

- Function for determining scale: convolve image with kernel to identify sharp intensity discontinuities
$$f = \text{Kernel} \star \text{Image}$$
- It has been shown that the Laplacian of Gaussian kernel is optimal under certain assumptions [Lindeberg?94]:
$$LoG = \nabla^2 G(x,y) = \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2}$$
- Correct scale is found as local maxima or minima across consecutive smoothed images



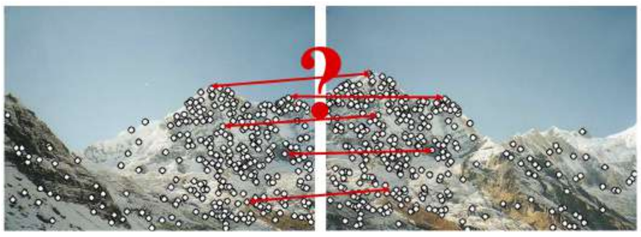
Lindeberg, Scale-space theory: A basic tool for analysing structures at different scales, Journal of Applied Statistics, 1994

Main questions

- What points are distinctive (i.e., features, keypoints, salient points), such that they are repeatable? (i.e., can be re-detected from other views)
- How to *describe* a local region?
- How to establish correspondences, i.e., compute matches?

Feature descriptors

- We know how to detect points
- Next question:
  How to describe them for matching?



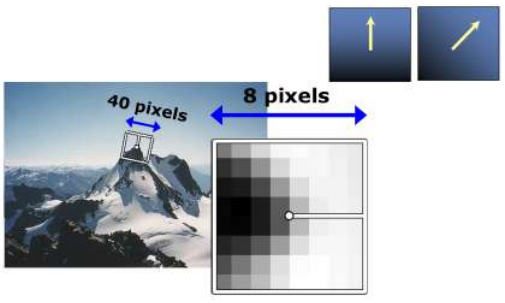
- Simplest descriptor: intensity values within a squared patch
- Alternative: Census Transform or Histograms of Oriented Gradients (like in SIFT, see later)
- Then, descriptor matching can be done using Hamming Distance (Census) or (Z)SSD, (Z)SAD, or (Z)NCC

- We'd like to find the same features regardless of the transformation (rotation, scale, view point, and illumination)
  - Most feature methods are designed to be invariant to
    - 2D translation,
    - 2D rotation,
    - Scale
  - Some of them can also handle
    - Small view-point invariance (e.g., SIFT works up to about 60 degrees)
    - Linear illumination changes

How to achieve invariance

Step 1: Re-scaling and De-rotation

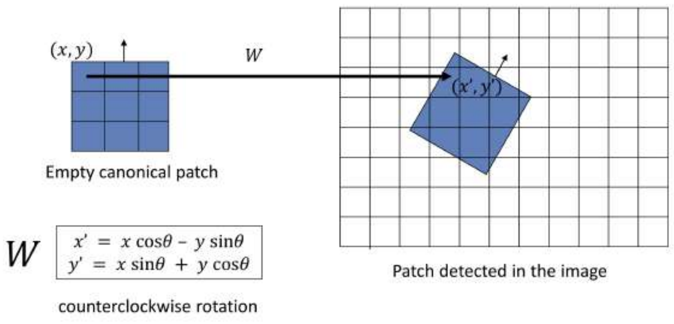
- Find correct scale using LoG operator
- Rescale the patch to a default size (e.g., 8 × 8 pixels)
- Find local orientation
  - Dominant direction of gradient for the image patch (e.g., Harris eigenvectors)



How to warp a patch?

- Start with an "empty" canonical patch (all pixels set to 0)
- For each pixel (x, y) in the empty patch, apply the warping function $\mathtt{W}(x, y)$ to compute the corresponding position in the detected image. It will be in floating point and will fall between the image pixels.
- Interpolate the intensity values of the 4 closest pixels in the detected image:
    - use nearest neighbor
    - or bilinear interpolation

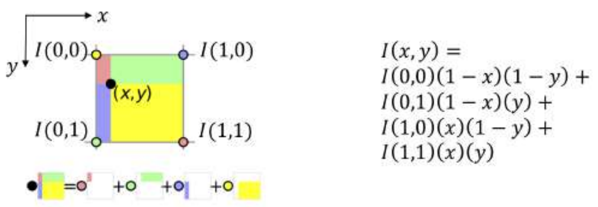
Example 1: Rotational warping



(x, y)
W
(x', y')
Empty canonical patch
W
$x' = x\cos\theta - y\sin\theta$
$y' = x\sin\theta + y\cos\theta$
counterclockwise rotation
Patch detected in the image

Bilinear Interpolation

- It is an **extension of linear interpolation** for interpolating functions of two variables (e.g., x and y) on a rectilinear 2D grid.
- The key idea is to perform linear interpolation first in one direction, and then again in the other direction. Although each step is linear in the sampled values and in the position, the interpolation as a whole is not linear but rather quadratic in the sample location.

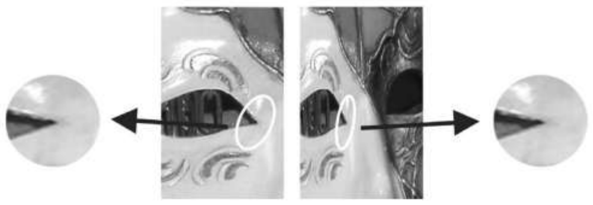


In this geometric visualization, the value at the black spot is the sum of the value at each colored spot multiplied by the area of the rectangle of the same color.

Example 2: Affine Warping

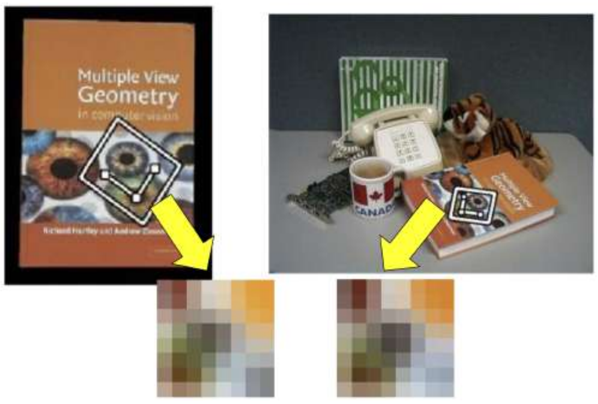
Affine warping (to achieve slight view-point invariance

- The second moment matrix `M` can be used to identify the two directions of fastest and slowest change of intensity around the feature.
- Out of these two directions, an elliptic patch is extracted at the scale computed by with the LoG operator.
- The region inside the ellipse is normalized to a circular one



How to achieve invariance

Example: de-rotation, re-scaling, and affine un-warping

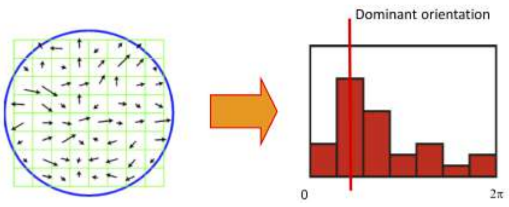


Feature descriptors

- Disadvantage of patches as descriptors:
  - If not warped, very small errors in rotation, scale, and view-point will affect matching score significantly
  - Computationally expensive (need to unwarp every patch)
- Better solution nowadays: build descriptors from Histograms of Oriented Gradients (HOGs)

HOG descriptor (Histogram of Oriented Gradients)

- Compute a histogram of orientations of intensity gradients
- Peaks in histogram: dominant orientations
- **Keypoint orientation = histogram peak**
  - If there are multiple candidate peaks, **construct a different keypoint for each such orientation**
- **Rotate patch according to this angle**
  - This puts the patches into a canonical orientation

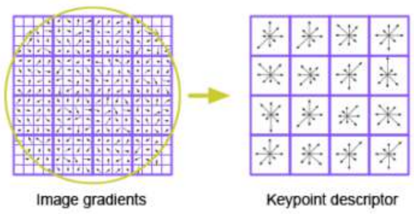


Rotation and Scale Normalization

- Rotate the window to standard orientation
- Scale the window size based on the scale at which the point was found

SIFT descriptor

- **S**cale **I**nvariant **F**eature **T**ransform
- Invented by David Lowe [IJCV, 2004] (now at Google)
- Descriptor computation:
  - Divide patch into 4×4 sub-patches = 16 cells
  - Compute HOG (8 bins, i.e., 8 directions) for all pixels inside each sub-patch
  - Concatenates all HOGs into a single 1D vector:
    - Resulting SIFT descriptor: $4 \times 4 \times 8 = 128$ values
  - Descriptor Matching: SSD (i.e., Euclidean-distance)



Intensity Normalization

- The descriptor vector v is then normalized such that its l_2 norm is 1:
$$\bar{v} = \frac{v}{\sqrt{\sum_i^n v_i^2}}$$
- This guarantees that the descriptor is invariant to linear illumination changes (the descriptor is already invariant to additive illumination because it is based on gradients).

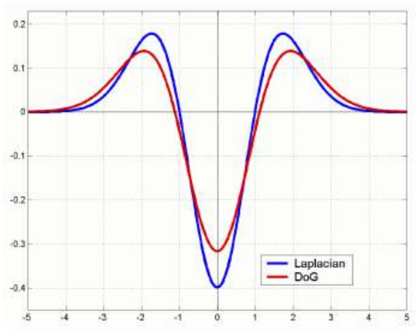
SIFT matching robustness

- Can handle changes in viewpoint (up to 60 degree out-of-plane rotation)
- Can handle significant changes in illumination (low to bright scenes)
- Expensive: 10 fps
- Original SIFT code (binary files): http://people.cs.ubc.ca/ lowe/keypoints

Scale Invariant Feature Detection

Difference of Gaussian (DoG) kernel instead of Laplacian of Gaussian (computationally cheaper)

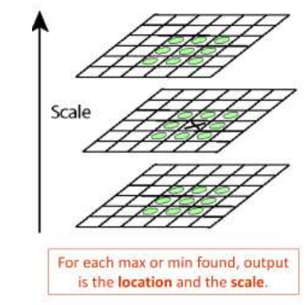
$$LoG \approx DoG = G_{k\sigma}(x,y) - G_{\sigma}(x,y)$$



SIFT detector (location + scale)

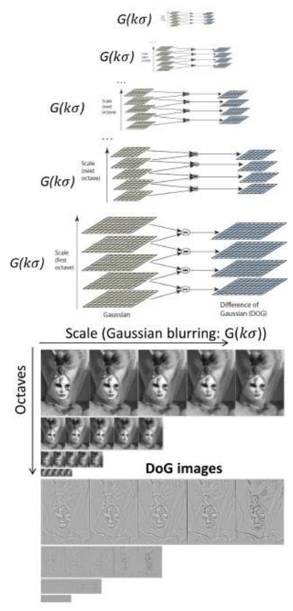
SIFT keypoints: local extrema (i.e., maxima and minima) in **both space and scale** of the DoG images

- Detect maxima and minima of difference-of-Gaussian in scale space
- Each point is compared to its 8 neighbors in the current image and 9 neighbors each in the scales above and below


Scale
For each max or min found, output is the location and the scale.

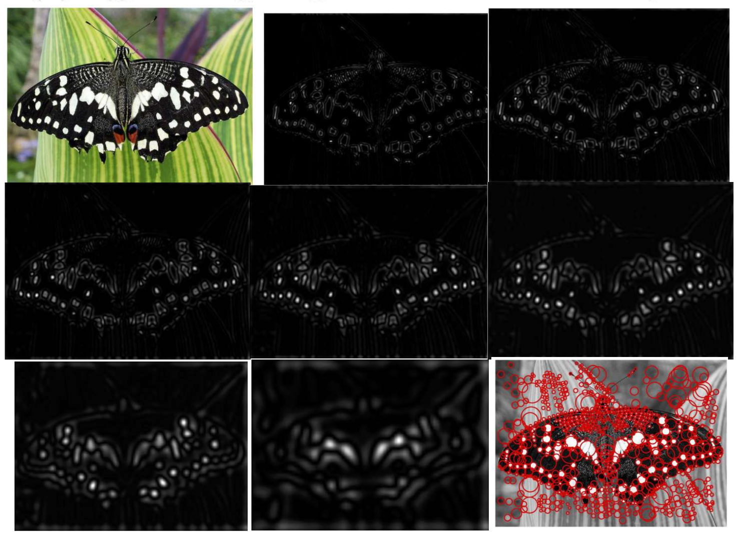
How it is implemented in practice

1. The initial image is **incrementally convolved with Gaussians** $G(k\sigma)$ to produce images separated by a constant factor k in scale space, shown stacked in the left column
   1. The initial Gaussian $G(k\sigma)$ has $\sigma = 1.6$
   2. k is chosen such that $k = 2^{1/s}$, where s is an integer (typically $s = 3$)
   3. For efficiency reasons, when k reaches 2, the image is downsampled by a factor of 2 and then the procedure is repeated again up to 4 or 6 octaves (pyramid levels)
2. Adjacent image scales are then subtracted to produce the Difference-of-Gaussian (DoG) images


G(kσ)
G(kσ)
G(kσ)
G(kσ)
G(kσ)
Gaussian
Difference of Gaussian (DOG)
Scale (Gaussian blurring: G(kσ))
Octaves
DoG images

Scale-space detection: Example

$G(k\sigma) - G(\sigma)$ with increasing σ starting from $\sigma = 1.6$ and $s = 6$ number of scales per octave

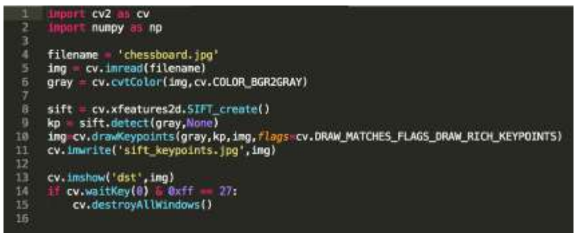


SIFT Features: Summary

- SIFT: Scale Invariant Feature Transform [Lowe, IJCV 2004]
- An approach to detect and describe regions of interest in an image.
  - NB: SIFT detector = DoG detector
- SIFT features are reasonably invariant to changes in rotation, scaling, and changes in viewpoint (up to 60deg) and illumination
- Real-time but still slow (10 Hz on an i7 laptop)
  - Expensive steps are the scale detection and descriptor extraction

SIFT Demo

Use OpenCV lib in Python

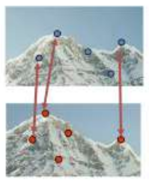


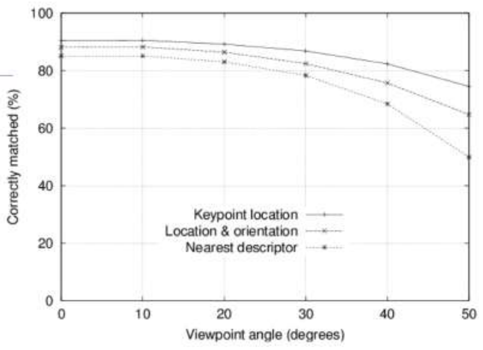
```python
import cv2 as cv
import numpy as np

filename = 'chessboard.jpg'
img = cv.imread(filename)
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)

sift = cv.xfeatures2d.SIFT_create()
kp = sift.detect(gray,None)
img=cv.drawKeypoints(gray,kp,img,flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv.imwrite('sift_keypoints.jpg',img)

cv.imshow('dst',img)
if cv.waitKey(0) & 0xff == 27:
    cv.destroyAllWindows()
```

SIFT repeatability vs. viewpoint angle

Repeatability=

$$\frac{\text{\# correspondences detected}}{\text{\# correspondences present}}$$



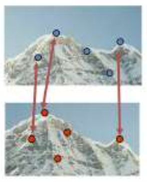


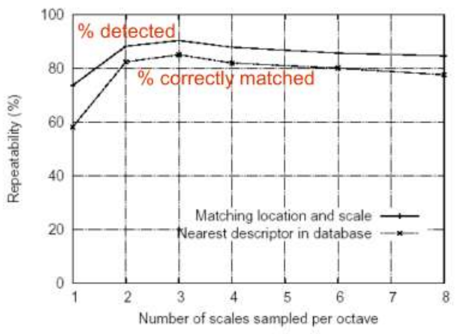
SIFT repeatability vs. Scale

The highest repeatability is obtained when sampling 3 scales per octave

Repeatability=

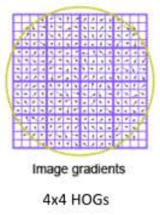
$$\frac{\text{\# correspondences detected}}{\text{\# correspondences present}}$$




% detected
% correctly matched
Repeatability (%)
Number of scales sampled per octave
Matching location and scale
Nearest descriptor in database
0
20
40
60
80
100
1
2
3
4
5
6
7
8

Influence of Number of Orientations and Number of Sub-patches

The graph shows that a single orientation histogram ($n = 1$) is very poor at discriminating, but the results continue to improve up to a 4x4 array of histograms with 8 orientations. After that, adding more orientations or a larger descriptor can actually hurt matching by making the descriptor more sensitive to distortion.



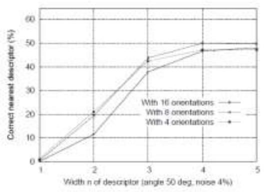
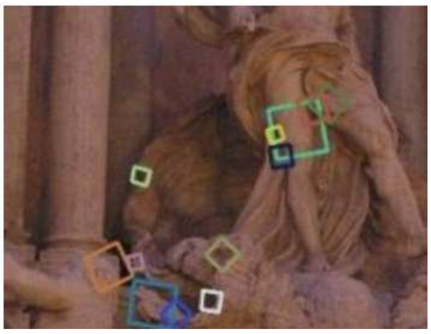


Figure 8: This graph shows the percent of keypoints giving the correct match to a database of 40,000 keypoints as a function of width of the $n \times n$ keypoint descriptor and the number of orientations in each histogram. The graph is computed for images with affine viewpoint change of 50 degrees and addition of 4% noise.

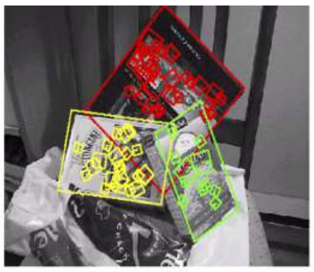
How many parameters are used to define a SIFT feature‘

- **Descriptor**: $4 \times 4 \times 8 = 128$-element 1D vector
- **Location** (pixel coordinates of the center of the patch): 2D vector
- **Scale** (i.e., size) of the patch: 1 scalar value
- **Orientation** (i.e., angle of the patch): 1 scalar value



SIFT for Object recognition

- Can be simply implemented by returning as best object match the one with the largest number of correspondences with the template (object to detect)
- 4 or 5 point RANSAC can be used to remove outliers (see next lectures)



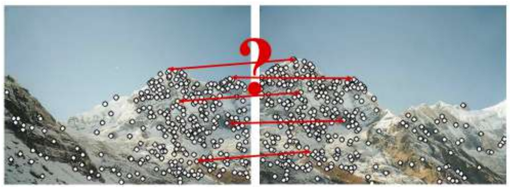
SIFT for Panorama Stitching

- AutoStitch: http://matthewalunbrown.com/autostitch/autostitch.html
- M. Brown and D. G. Lowe. Recognising Panoramas. ICCV 2003

Main questions

- What points are distinctive (i.e., *features, keypoints, salient points*), such that they are *repeatable*? (i.e., can be re-detected from other views)
- How to *describe* a local region?
- How to establish *correspondences*, i.e., compute matches?

Feature matching



- Given a feature in I_1, how to find the best match in I_2?
  1. Define distance function that compares two descriptors ((Z)SSD, SAD, NCC or Hamming distance for binary descriptors (e.g., Census, BRIEF, BRISK)
  2. **Brute-force matching:**
     1. Test all the features in I_2
     2. Take the one at min distance
- **Issues with closest descriptor**: can give good scores to very ambiguous (bad) matches (curse of dimensionality)
- **Better approach**: compute ratio of distances to 1st to 2nd closest match

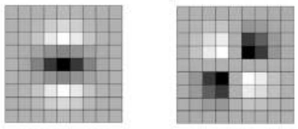
$$d(f_1)/d(f_2) < Thres \text{ usually } 0.8$$

  - $d(f_1)$ is the distance of the closest neighbor
  - $d(f_2)$ is the distance of the 2nd closest neighbor

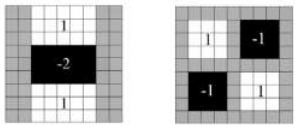
SURF [Bay et al., ECCV 2006]

- ***S**peeded **U**p **R**obust **F**eatures*
- Based on ideas similar to **SIFT**
- Approximated computation for detection and descriptor
- Results comparable with SIFT, plus:
  - Faster computation
  - Generally shorter descriptors

Gaussian second order partial derivatives



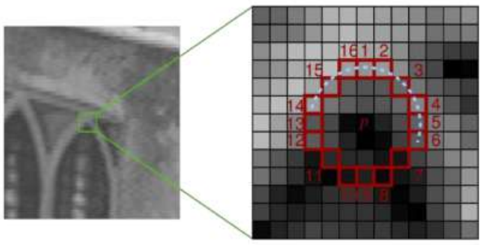
Approximation using **box filter**



Bay, Tuytelaars, Van Gool, " Speeded Up Robust Features ", European Conference on Computer Vision (ECCV) 2006

FAST detector [Rosten et al., ICCV'05]

- **FAST**: **F**eatures from **A**ccelerated **S**egment **T**est
- Studies intensity of pixels on circle around candidate pixel ***C***
- ***C*** is a FAST corner **if** a set of ***N*** contiguous pixels on circle are:
  - all brighter than $\boldsymbol{intensity\_of(C) + theshold}$, or
  - all darker than $\boldsymbol{intensity\_of(C) + theshold}$

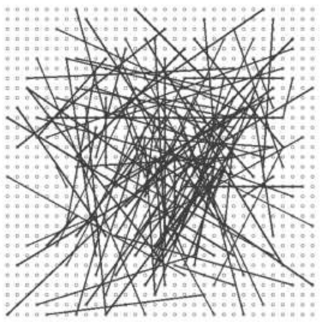


- Typically tests for **9** contiguous pixels in a **16**-pixel circumference
- **Very fast detector** - in the order of 100 Mega-pixel/second

Rosten, Drummond, Fusing points and lines for high performance tracking, International Conference on Computer Vision (ICCV), 2005

BRIEF descriptor [Calonder et. al, ECCV 2010]

- **B**inary **R**obust **I**ndependent **E**lementary **F**eatures
- Goal: high speed (in description and matching)
- **Binary** descriptor formation:
  - Smooth image
  - **for each** detected keypoint (e.g. FAST),
  - ***sample*** 256 intensity pairs $(p_1{}^i, p_2{}^i)$ $(i = 1, \dots, 256)$ within a squared patch around the keypoint
  - Create an empty 256-element descriptor
  - **for each** i^{th} **pair**
    - **if** $I_{p_1{}^i} < I_{p_2{}^i}$ **then *set*** i^{th} bit of descriptor to **1**
    - **else** to **0**
- The **pattern is generated randomly** (or by **machine learning**) only once; then, the same pattern is used for all patches
- Pros: **Binary descriptor**: allows **very fast** Hamming distance matching: count the number of bits that are different in the descriptors matched
- Cons: **Not scale/rotation invariant**

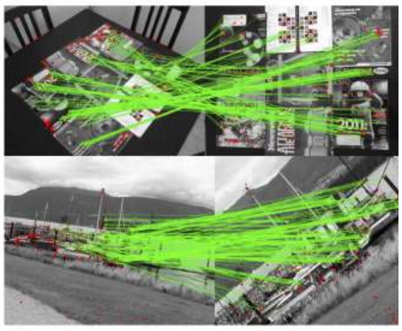


Pattern for intensity pair samples – generated randomly

Calonder, Lepetit, Strecha, Fua, BRIEF: Binary Robust Independent Elementary Features, ECCV'10]

ORB descriptor [Rublee et al., ICCV 2011]

- ***O**riented FAST and **R**otated **B**RIEF*
- Keypoint detector based on **FAST**
- **BRIEF** descriptors are *steered* according to keypoint orientation (to provide rotation invariance)
- Good Binary features are learned by minimizing the correlation on a set of training patches.

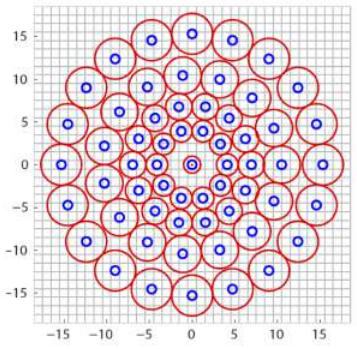


Rublee,Rabaud, Konolige, Bradski, (2011). "ORB: an efficient alternative to SIFT or SURF" (PDF). IEEE International Conference on Computer Vision (ICCV).

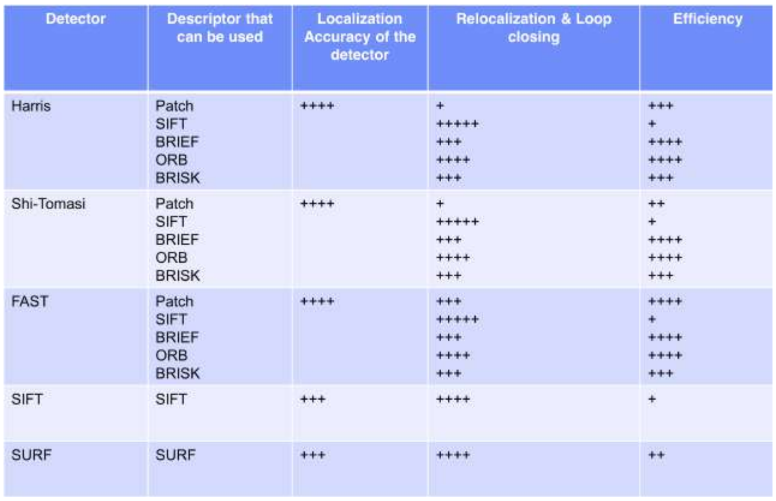
BRISK descriptor [Leutenegger, Chli, Siegwart, ICCV 2011]

- **B**inary **R**obust **I**nvariant **S**calable **K**eypoints
- Detect corners in scale-space using FAST
- Rotation and scale invariant

- **Binary**, formed by pairwise intensity comparisons (like BRIEF)
- **Pattern** defines intensity comparisons in the keypoint neighborhood
- **Red circles**: size of the smoothing kernel applied
- **Blue circles**: smoothed pixel value used
- Compare short- and long-distance pairs for orientation assignment & descriptor formation
- Detection and descriptor speed: ~10 times faster than SURF
- Slower than BRIEF, but scale- and rotation- invariant



Recap Table



Detector	Descriptor that can be used	Localization Accuracy of the detector	Relocalization & Loop closing	Efficiency
Harris	Patch SIFT BRIEF ORB BRISK	++++	+ +++++ +++ ++++ +++	+++ + ++++ ++++ +++
Shi-Tomasi	Patch SIFT BRIEF ORB BRISK	++++	+ +++++ +++ ++++ +++	++ + ++++ ++++ +++
FAST	Patch SIFT BRIEF ORB BRISK	++++	+++ +++++ +++ ++++ +++	++++ + ++++ ++++ +++
SIFT	SIFT	+++	++++	+
SURF	SURF	+++	++++	++

Summary (things to remember)

- Similarity metrics: NCC (ZNCC), SSD (ZSSD), SAD (ZSAD), Census Transform
- Point feature detection
  - Properties and invariance to transformations
    - Challenges: rotation, scale, view-point, and illumination changes
  - Extraction
    - Moravec
    - Harris and Shi-Tomasi: Rotation invariance
  - Automatic Scale selection
  - Descriptor
    - Intensity patches: Canonical representation: how to make them invariant to transformations: rotation, scale, illumination, and view-point (affine)
    - Better solution: Histogram of oriented gradients: SIFT descriptor
  - Matching
    - (Z)SSD, SAD, NCC, Hamming distance (last one only for binary descriptors) ratio 1st /2nd closest descriptor
  - Depending on the task, you may want to trade off repeatability and robustness for speed: approximated solutions, combinations of efficient detectors and descriptors.
    - Fast corner detector: FAST;
    - Keypoint descriptors faster than SIFT: SURF, BRIEF, ORB, BRISK

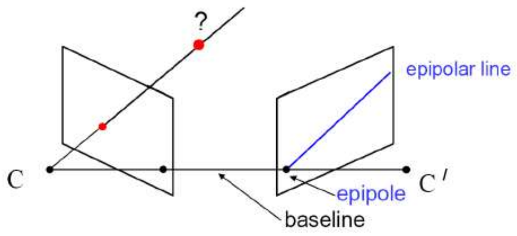
Chapter 7

Multiple-view Geometry

7.1 Epipolar Geometry

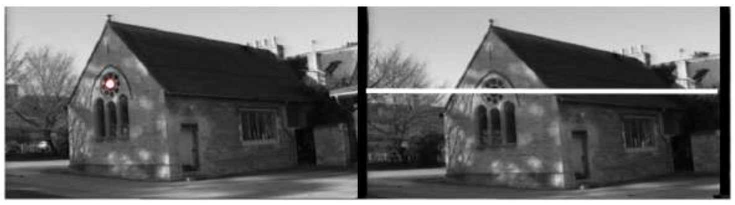
Epipolar Geometry

Given an image point in one view, where is the corresponding point in the other view?



- A point in one view "generates" an epipolar line in the other view
- The corresponding point lies on this line

Epipolar line

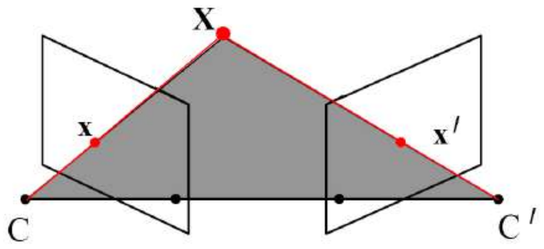


Epipolar Constraint

- Reduces correspondence problem to 1D search along an epipolar line

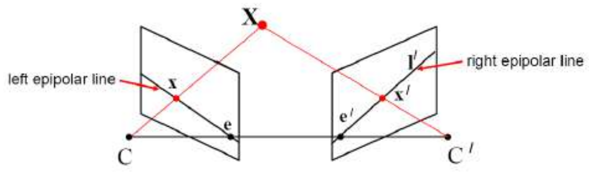
Epipolar geometry continued

Epipolar geometry is a consequence of the coplanarity of the camera centres and scene point



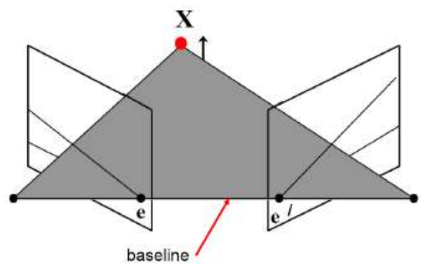
The camera centres, corresponding points and scene point lie in a single plane, known as the epipolar plane

Nomenclature



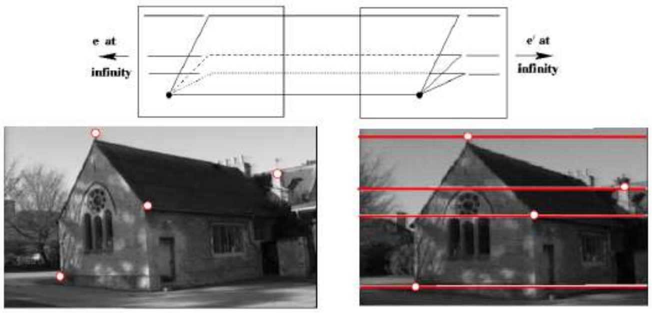
- The epipolar line $\mathbf{l}'$ is the image of the ray through $\mathbf{x}$
- The epipole $\mathbf{e}$ is the point of intersection of the line joining the camera centres with the image plane
  - this line is the baseline for a stereo rig, and
  - the translation vector for a moving camera
- The epipole is the image of the centre of the other camera: $\mathbf{e} = \mathtt{P}\mathbf{C}'$, $\mathbf{e}' = \mathtt{P}'\mathbf{C}$

The Epipolar Pencil



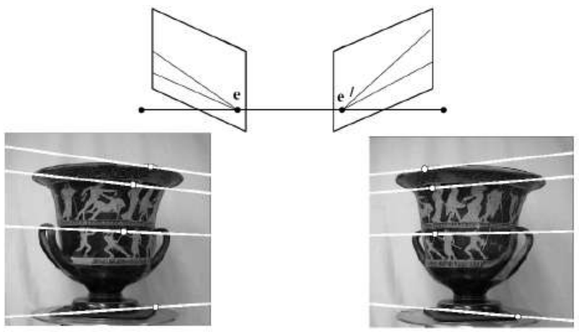
As the position of the 3D point X varies, the epipolar planes "rotate" about the baseline. This family of planes is known as an epipolar pencil. All epipolar lines intersect at the epipole.
(a pencil is a one parameter family)

Epipolar Geometry for Parallel Cameras



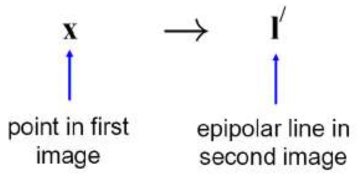
Epipolar geometry depends only on the relative pose (position and orientation) and internal parameters of the two cameras, i.e. the position of the camera centres and image planes. It does not depend on the scene structure (3D points external to the camera).

Epipolar Geometry for Converging Cameras



Note : epipolar lines are in general not parallel

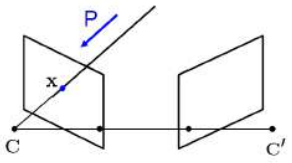
Algebraic representation of the epipolar geometry



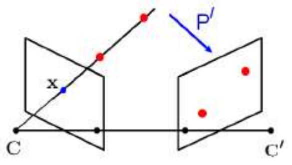
- the map only depends on the cameras $\mathtt{P}$, $\mathtt{P}'$ (not on structure)
- it will be shown that the map is linear and can be written as $\mathbf{l}' = \mathtt{F}\mathbf{x}$, where $\mathtt{F}$ is a 3×3 matrix called the fundamental matrix

Derivation of the algebraic expression $\mathbf{l}' = \mathtt{F}\mathbf{x}$

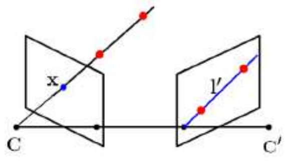
Step 1: for a point x in the first image back project a ray with camera $\mathtt{P}$



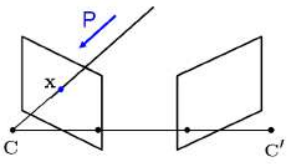
Step 2: choose two points $\mathbf{p}$ and $\mathbf{q}$ on the ray and project into the second image with camera $\mathtt{P}'$



Step 3: compute the line through the two image points using the relation $\mathbf{l}' = \mathbf{p} \times \mathbf{q}$



Step 1 : for a point x in the first image back project a ray with camera $\mathtt{P} = \mathtt{K}[\mathtt{I}|\mathbf{0}]$



A point $\mathbf{x}$ back projects to a ray $\mathbf{x}(\mathrm{z})$ that satisfies

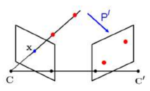
$$\mathtt{P}\mathbf{x}(\mathrm{z}) = \mathtt{K}[\mathtt{I}|\mathbf{0}]\mathbf{x}(\mathrm{z}) = \mathbf{x}$$

where z is the point's depth, since

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \mathtt{K}[\mathtt{I}|\mathbf{0}] \begin{pmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \\ 1 \end{pmatrix} = \mathtt{K} \begin{pmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \end{pmatrix}$$

$$\mathbf{x}(\mathrm{z}) = \begin{pmatrix} \mathrm{z}\mathtt{K}^{-1}\mathbf{x} \\ 1 \end{pmatrix}$$

Step 2 : choose two points $\mathbf{p}$ and $\mathbf{q}$ on the ray and project into the second image with camera $\mathtt{P}'$



Consider two points on the ray $\mathbf{x}(\mathrm{z}) = \begin{pmatrix} \mathrm{z}\mathtt{K}^{-1}\mathbf{x} \\ 1 \end{pmatrix}$

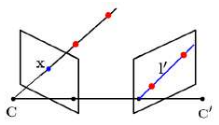
- $\mathrm{z} = 0$ is the camera centre $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- $\mathrm{z} = \infty$ is the point at infinity $\begin{pmatrix} \mathtt{K}^{-1}\mathbf{x} \\ 0 \end{pmatrix}$

Project these two points into the second view

$$\mathtt{P}' \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \mathtt{K}'[\mathtt{R}|\mathbf{t}] \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \mathtt{K}'\mathbf{t}$$

$$\mathtt{P}' \begin{pmatrix} \mathtt{K}^{-1}\mathbf{x} \\ 0 \end{pmatrix} = \mathtt{K}'[\mathtt{R}|\mathbf{t}] \begin{pmatrix} \mathtt{K}^{-1}\mathbf{x} \\ 0 \end{pmatrix} = \mathtt{K}'\mathtt{R}\mathtt{K}^{-1}\mathbf{x}$$

Step 3 : compute the line through the two image points using the relation $\mathbf{l}' = \mathbf{p} \times \mathbf{q}$



Compute the line through the points $\mathbf{l}' = (\mathtt{K}'\mathbf{t}) \times (\mathtt{K}'\mathtt{R}\mathtt{K}^{-1}\mathbf{x})$

Using the identity $(\mathtt{M}\mathbf{a}) \times (\mathtt{M}\mathbf{b}) = \mathtt{M}^{-\mathsf{T}}(\mathbf{a} \times \mathbf{b})$ where $\mathtt{M}^{-\mathsf{T}} = (\mathtt{M}^{-1})^{-\mathsf{T}} = (\mathtt{M}^{\mathsf{T}})^{-1}$

$$\mathbf{l}' = \mathtt{K}'^{-\mathsf{T}}(\mathbf{t} \times (\mathtt{R}\mathtt{K}^{-1}\mathbf{x})) = \mathtt{K}'^{-\mathsf{T}}[\mathbf{t}]_\times \mathtt{R}\mathtt{K}^{-1}\mathbf{x} \text{ with } [\mathbf{t}]_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

Important Result

$\mathbf{l}' = \mathtt{F}\mathbf{x}$ $\qquad$ $\mathtt{F} = \mathtt{K}'^{-\mathsf{T}}[\mathbf{t}]_\times \mathtt{R}\mathtt{K}^{-1}$ $\qquad$ $\mathtt{F}$ = fundamental matrix

Points $\mathbf{x}$ and $\mathbf{x}'$ correspond $(\mathbf{x} \leftrightarrow \mathbf{x}')$ then $\mathbf{x}'^{\mathsf{T}}\mathbf{l}' = 0$

Constraint

$$\mathbf{x}'^{\mathsf{T}}\mathtt{F}\mathbf{x} = 0$$

Example I : compute the fundamental matrix for a parallel camera stereo rig

$\mathtt{P} = \mathtt{K}[\ \mathtt{I}\ |\ \mathbf{0}\ ]$ $\qquad$ $\mathtt{P}' = \mathtt{K}'[\ \mathtt{R}\ |\ \mathbf{t}\ ]$

$$\mathtt{K} = \mathtt{K}' = \begin{bmatrix} f & & \\ & f & \\ & & 1 \end{bmatrix} \qquad \mathtt{R} = \mathtt{I} \qquad \mathbf{t} = \begin{pmatrix} t_x \\ 0 \\ 0 \end{pmatrix}$$

$$\mathtt{F} = \mathtt{K}'^{-\mathsf{T}}[\mathbf{t}]_\times \mathtt{R}\mathtt{K}^{-1}$$

$$= \begin{bmatrix} 1/f & & \\ & 1/f & \\ & & 1 \end{bmatrix} \begin{bmatrix} 0 & & \\ & 0 & -t_x \\ & t_x & 0 \end{bmatrix} \begin{bmatrix} 1/f & & \\ & 1/f & \\ & & 1 \end{bmatrix} = \begin{bmatrix} 0 & & \\ & 0 & -1 \\ & 1 & 0 \end{bmatrix}$$

$$\mathbf{x}'^{\mathsf{T}}\mathtt{F}\mathbf{x} = \begin{pmatrix} x' & y' & 1 \end{pmatrix} \begin{bmatrix} 0 & & \\ & 0 & -1 \\ & 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = 0$$

- reduces to $y = y'$, i.e. raster correspondence (horizontal scan-lines)

Important Result

$\mathtt{F}$ is a rank 2 matrix

The epipole $\mathbf{e}$ is the null-space vector of $\mathtt{F}$, i.e. $\mathtt{F}\mathbf{e} = \mathbf{0}$
In the case of parallel cameras

$$\begin{bmatrix} 0 & & \\ & 0 & -1 \\ & 1 & 0 \end{bmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

so that

$$\mathbf{e} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Geometric Interpretation ?

Summary : properties of the fundamental matrix

- $\mathtt{F}$ is a rank-2 homogeneous matrix with 7 DOF.
- Point correspondence :
  if $\mathbf{x}$ and $\mathbf{x}'$ are corresponding image points, then $\mathbf{x}'^{\mathsf{T}}\mathtt{F}\mathbf{x} = 0$.
- Epipolar lines :
  $\mathbf{l}' = \mathtt{F}\mathbf{x}$ is the epipolar line corresponding to $\mathbf{x}$
  $\mathbf{l} = \mathtt{F}^{\mathsf{T}}\mathbf{x}'$ is the epipolar line corresponding to $\mathbf{x}'$
- Epipoles :
  $\mathtt{F}\mathbf{e} = \mathbf{0}$
  $\mathtt{F}^{\mathsf{T}}\mathbf{e}' = \mathbf{0}$
- Computation from camera matrices $\mathtt{P}$ and $\mathtt{P}'$:
  $\mathtt{P} = \mathtt{K}\left[\ \mathtt{I} \mid \mathbf{0}\ \right]$, $\mathtt{P}' = \mathtt{K}'\left[\ \mathtt{R} \mid \mathbf{t}\ \right]$, $\mathtt{F} = \mathtt{K}'^{-\mathsf{T}}[\mathbf{t}]_\times \mathtt{R}\mathtt{K}^{-1}$
  $\mathtt{E} = [\mathbf{t}]_\times \mathtt{R}$ is called *the essential matrix* is the calibrated case.

How to compute the Essential Matrix?

Image 1

Image 2

- If we don't know $\mathtt{R}$ and $\mathbf{t}$, can we estimate $\mathtt{E}$ from two images?
- Yes, given at least 5 correspondences

The 8-point algorithm

- The Essential matrix $\mathtt{E}$ is defined by

$$\bar{\mathbf{p}}_2^T \mathbf{E} \bar{\mathbf{p}}_1 = 0$$

- Each pair of point correspondences $\bar{\mathbf{p}}_1 = (\bar{u}_1, \bar{v}_1, 1)^T$,
  $\bar{\mathbf{p}}_2 = (\bar{u}_2, \bar{v}_2, 1)^T$ provides a linear equation:

$$\bar{u}_2\bar{u}_1 e_{11} + \bar{u}_2\bar{v}_1 e_{12} + \bar{u}_2 e_{13} + \bar{v}_2\bar{u}_1 e_{21} + \bar{v}_2\bar{v}_1 e_{22} + \bar{v}_2 e_{23} + \bar{u}_1 e_{31} + \bar{v}_1 e_{32} + e_{33} = 0$$

with

$$\mathtt{E} = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}$$

- For n points, we can write

$$\underbrace{\begin{bmatrix} \bar{u}_{2,1}\bar{u}_{1,1} & \bar{u}_{2,1}\bar{v}_{1,1} & \bar{u}_{2,1} & \bar{v}_{2,1}\bar{u}_{1,1} & \bar{v}_{2,1}\bar{v}_{1,1} & \bar{v}_{2,1} & \bar{u}_{1,1} & \bar{v}_{1,1} & 1 \\ \bar{u}_{2,2}\bar{u}_{1,2} & \bar{u}_{2,2}\bar{v}_{1,2} & \bar{u}_{2,2} & \bar{v}_{2,2}\bar{u}_{1,2} & \bar{v}_{2,2}\bar{v}_{1,2} & \bar{v}_{2,2} & \bar{u}_{1,2} & \bar{v}_{1,2} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \bar{u}_{2,n}\bar{u}_{1,n} & \bar{u}_{2,n}\bar{v}_{1,n} & \bar{u}_{2,n} & \bar{v}_{2,n}\bar{u}_{1,n} & \bar{v}_{2,n}\bar{v}_{1,n} & \bar{v}_{2,n} & \bar{u}_{1,n} & \bar{v}_{1,n} & 1 \end{bmatrix}}_{\mathtt{Q}\ \textbf{(known)}} \underbrace{\begin{pmatrix} e_{11} \\ e_{12} \\ e_{13} \\ e_{21} \\ e_{22} \\ e_{23} \\ e_{31} \\ e_{32} \\ e_{33} \end{pmatrix}}_{\bar{\mathtt{E}}\ \textbf{(unknown)}} = \mathbf{0}$$

- Solve the overdetermined $\mathtt{Q}.\bar{\mathtt{E}} = \mathbf{0}$ using SVD factorization $[\mathtt{U}, \mathtt{S}, \mathtt{V}] = \mathrm{SVD}(\mathtt{Q})$.
- The solution $\bar{\mathtt{E}}_{est}$ is the singular vector of $\mathtt{V}$ associated with the smallest singular value of $\mathtt{S}$

Extract $\mathtt{R}$ and $\mathbf{t}$ from $\mathtt{E}$

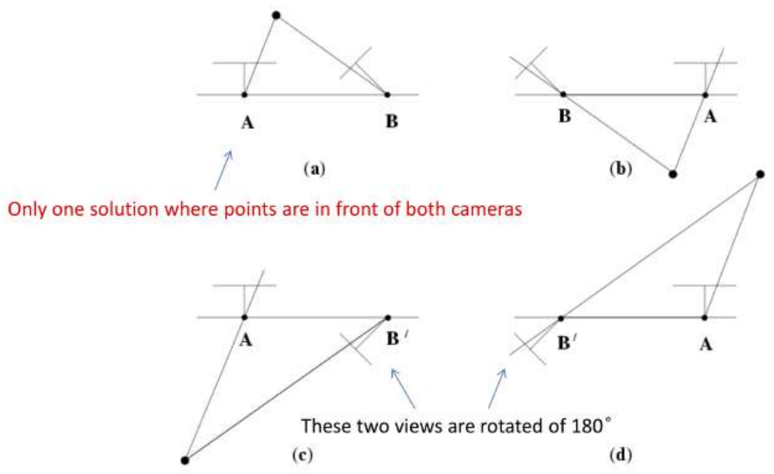
- Singular Value Decomposition: $\mathtt{E} = \mathtt{USV}^T$
- Enforcing rank-2 constraint: set smallest singular value of $\mathtt{S}$ to 0:

$$\mathtt{S} = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \cancel{\sigma_3} \end{bmatrix} = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$[\hat{\mathtt{t}}]_\times = \mathtt{U} \begin{bmatrix} 0 & \mp 1 & 0 \\ \pm 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathtt{SV}^T \text{ and } \hat{\mathtt{R}} = \mathtt{U} \begin{bmatrix} 0 & \mp 1 & 0 \\ \pm 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathtt{V}^T$$

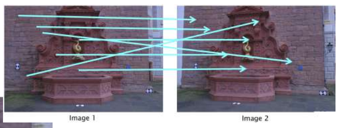
$$\boxed{\mathbf{t} = \mathtt{K}\hat{\mathtt{t}} \text{ and } \mathtt{R} = \mathtt{K}\hat{\mathtt{R}}\mathtt{K}^{-1}}$$

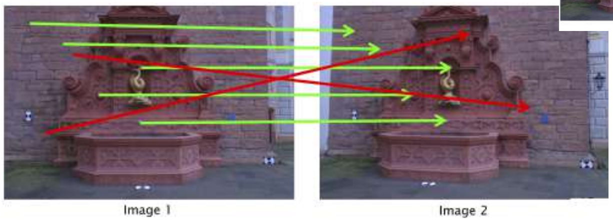
4 possible solutions for $\mathtt{R}$ and $\mathbf{t}$


A
B
(a)
B
A
(b)
Only one solution where points are in front of both cameras
A
B'
B'
A
These two views are rotated of 180°
(c)
(d)

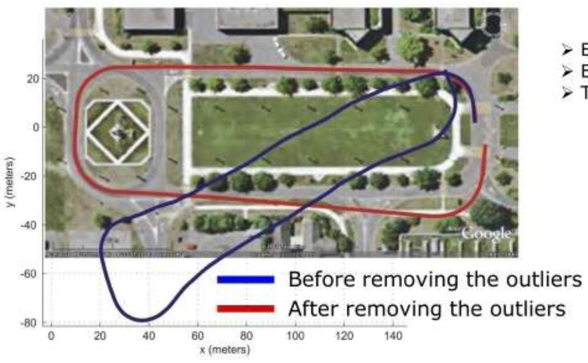
Robust Estimation

- Matched points are usually contaminated by outliers (i.e., wrong image matches)
- Causes of outliers are:
  - changes in view point (including scale) and illumination
  - image noise
  - occlusions
  - blur
- For the camera motion to be estimated accurately, outliers must be removed
- This is the task of **Robust Estimation**





Influence of Outliers on Motion Estimation



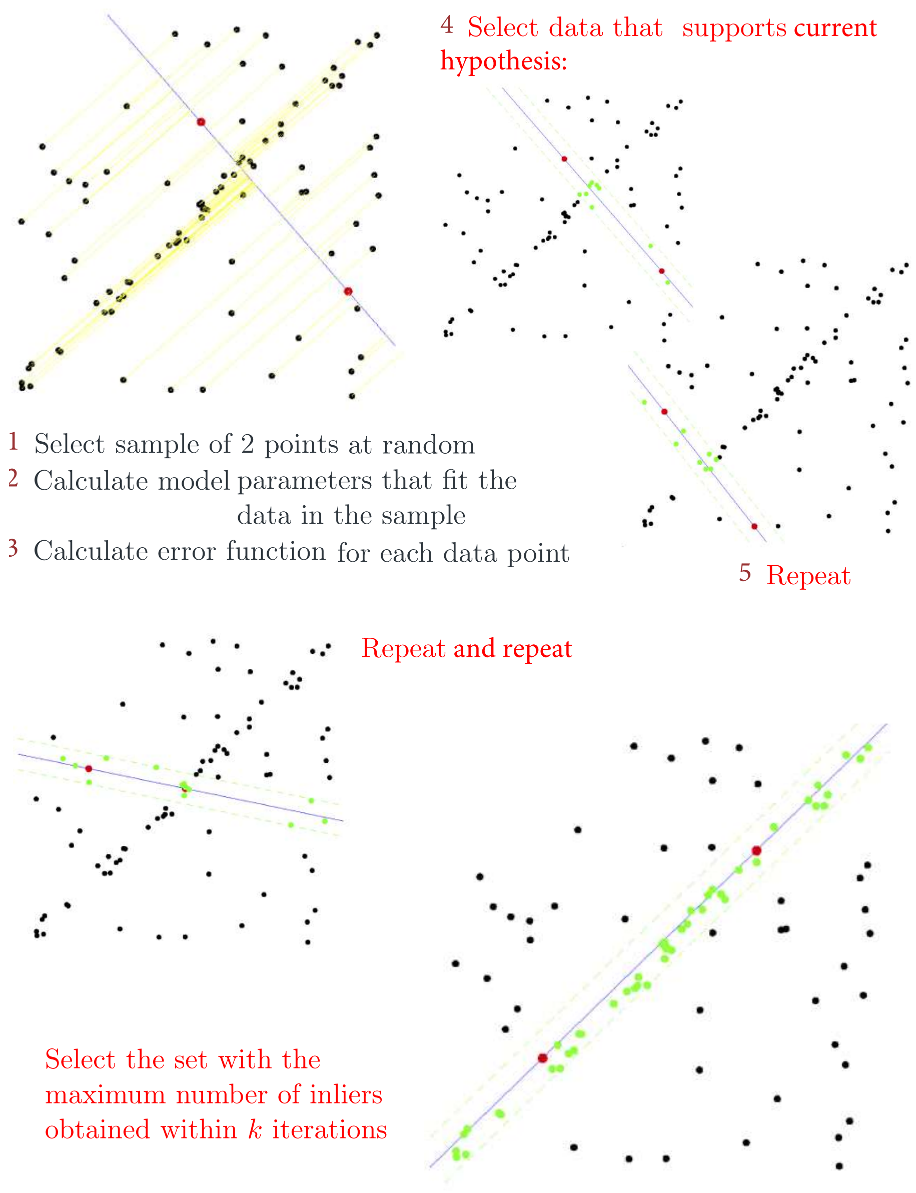
➢ Error at the loop closure:	6.5 m
➢ Error in orientation:	5 deg
➢ Trajectory length:	400 m

Outliers can be removed using RANSAC [Fishler & Bolles, 1981]

RANSAC (RAndom SAmple Consensus)

- RANSAC is the standard method for model fitting in the presence of outliers (very noisy points or wrong data)
- It can be applied to all sorts of problems where the goal is to estimate the parameters of a model from the data (e.g., camera calibration, Structure from Motion, DLT, PnP, P3P, Homography, etc.)
- Let's review RANSAC for line fitting and see how we can use it to do Structure from Motion

M. A.Fischler and R. C.Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Graphics and Image Processing, 1981.


4 Select data that supports current hypothesis:
1 Select sample of 2 points at random
2 Calculate model parameters that fit the data in the sample
3 Calculate error function for each data point
5 Repeat
Repeat and repeat
Select the set with the maximum number of inliers obtained within k iterations

How many iterations does RANSAC need?

- Ideally: check all possible combinations of 2 points in a dataset of N points.
- Number of all pairwise combinations: $N(N-1)/2$
  - computationally unfeasible if N is too large.
  - example: 1000 points $\Rightarrow$ need to check all 1000*999/2 $\approx$ 500 000 possibilities!
- Do we really need to check all possibilities or can we stop RANSAC after some iterations?
  - Checking a subset of combinations is enough if we have a rough estimate of the percentage of inliers in our dataset
- This can be done in a probabilistic way

- w: number of inliers$/N$
  N: total number of data points
  $\Rightarrow w$: fraction of inliers in the dataset $\Rightarrow w =$P(selecting an inlier-point out of the dataset)

- Assumption: the 2 points necessary to estimate a line are selected independently
  $\Rightarrow w^2$: P(both selected points are inliers)
  $\Rightarrow 1 - w^2$: P(at least one of these two points is an outlier)
- Let k: nb. of RANSAC iterations executed so far
  $\Rightarrow (1 - w^2)^k$: P(RANSAC never selected two points that are both inliers)
- Let p: P(probability of success)
  $\Rightarrow 1 - p(1 - w^2)^k$ and therefore:
- The number of iteration k is: $k = \dfrac{\log(1-p)}{\log 1 - w^2}$

- Knowing the fraction of inliers w, after k RANSAC iterations we will have a probability p of finding a set of points free of outliers
- Example: if we want a probability of success p=99% and we know that w=50% $\Rightarrow$ k=16 iterations - these are dramatically fewer than the number of all possible combinations! As you can see, the number of points does not influence the estimated number of iterations, only w does!
- In practice we only need a rough estimate of w. More advanced variants of RANSAC estimate the fraction of inliers and adaptively update it at every iteration

RANSAC applied to Line Fitting

```
❶ Initial: let $A$ be a set of $N$ points
❷ Repeat
❸     Randomly select a sample of 2 points from $A$
❹     Fit a line through the 2 points
❺     Compute the distances of all other points to this line
❻     Construct the inlier set (i.e. count the number of points whose distance $< d$)
❼     Store these inliers
❽ Until maximum number of iterations $k$ reached
❾ The set with the maximum number of inliers is chosen as a solution to the problem
```

RANSAC applied to general model fitting

```
❶ Initial: let $A$ be a set of $N$ points
❷ Repeat
❸     Randomly select a sample of $s$ points from $A$
❹     Fit a model through the $s$ points
❺     Compute the distances of all other points to this model
❻     Construct the inlier set (i.e. count the number of points whose distance $< d$)
❼     Store these inliers
❽ Until maximum number of iterations $k$ reached
❾ The set with the maximum number of inliers is chosen as a solution to the problem
```

$$\boxed{k = \frac{\log(1-p)}{\log 1 - w^s}}$$

The Three Key Ingredients of RANSAC

In order to implement RANSAC for Structure From Motion (SFM), we need three key ingredients:

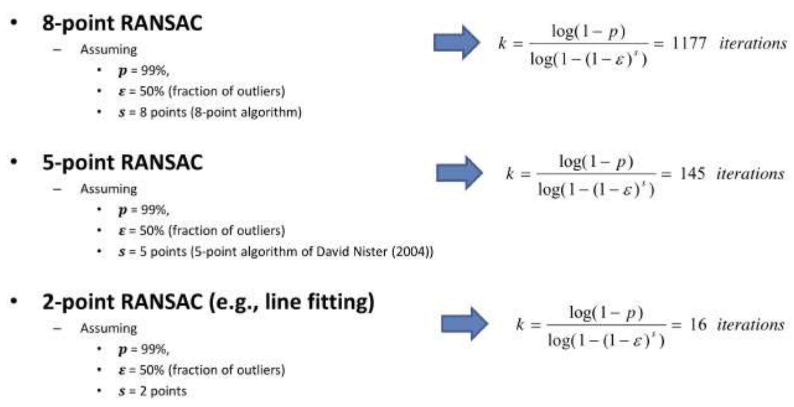
1. What's the **model** in SFM?
2. What's the **minimum number of points** to estimate the model?
3. How do we compute the distance of a point from the model? In other words, can we define a **distance metric** that measures how well a point fits the model?

Answers

1. What's the **model** in SFM?
   - The Essential Matrix (for calibrated cameras) or the Fundamental Matrix (for uncalibrated cameras)
   - Alternatively, $\mathtt{R}$ and $\mathbf{t}$
2. What's the **minimum number of points** to estimate the model?
   - We know that 5 points is the theoretical minimum number of points
   - However, if we use the 8-point algorithm, then 8 is the minimum
3. How do we compute the distance of a point from the model? In other words, can we define a **distance metric** that measures how well a point fits the model?
   - We can use the epipolar constraint ($\bar{\mathbf{p}}_2^T \mathtt{E} \bar{\mathbf{p}}_1 = 0$ or $\mathbf{p}_2^T \mathtt{F} \mathbf{p}_1 = 0$) to measure how well a point correspondence verifies the model $\mathtt{E}$ or $\mathtt{F}$, respectively. However, the Directional error, the Epipolar line distance, or the Reprojection error (even better) are used

RANSAC iterations k vs. s

k is exponential in the number of points s necessary to estimate the model:



- **8-point RANSAC**
  - Assuming
    - p = 99%,
    - ε = 50% (fraction of outliers)
    - s = 8 points (8-point algorithm)

$$k = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^s)} = 1177 \ \ iterations$$

- **5-point RANSAC**
  - Assuming
    - p = 99%,
    - ε = 50% (fraction of outliers)
    - s = 5 points (5-point algorithm of David Nister (2004))

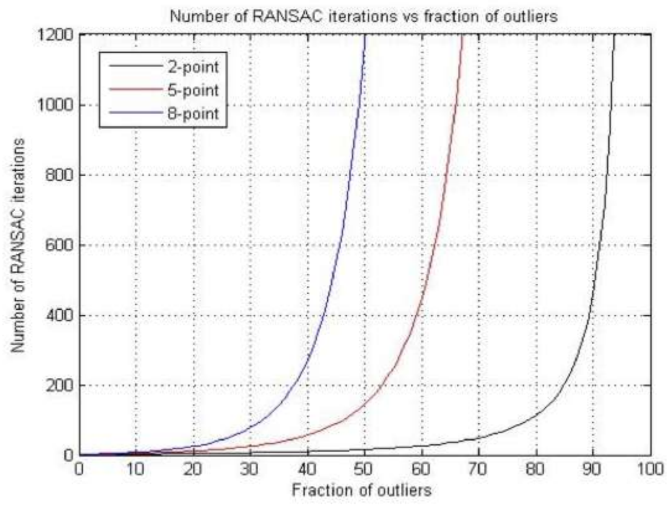
$$k = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^s)} = 145 \ \ iterations$$

- **2-point RANSAC (e.g., line fitting)**
  - Assuming
    - p = 99%,
    - ε = 50% (fraction of outliers)
    - s = 2 points

$$k = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^s)} = 16 \ \ iterations$$

RANSAC iterations k vs. ε

k is exponential with the fraction of outliers ε



RANSAC iterations

- As observed, k is exponential in the number of points s necessary to estimate the model
- The **8-point algorithm** is extremely simple and was very successful; however, it requires more than **1177 iterations**
- Because of this, there has been a large interest by the research community in **using smaller motion parameterizations** (i.e., smaller s)
- The first efficient solution to the minimal-case solution (5-point algorithm) took almost a century (Kruppa 1913 ? Nister 2004)
- The **5-point RANSAC** (Nister 2004) only requires **145 iterations**; however:
  - The **5-point algorithm** can return **up to 10 solutions of E (worst case scenario)**
  - The **8-point algorithm** only returns a **unique solution of E**

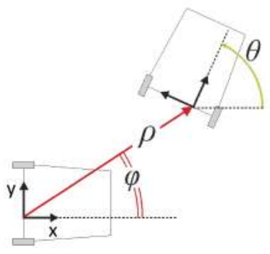
Can we use less than 5 points?
Yes, if you use motion constraints!

Planar Motion

- Planar motion is described by three parameters: θ, ϕ, ρ

$$\mathtt{R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{t} = \begin{pmatrix} \rho\cos\phi \\ \rho\sin\phi \\ 0 \end{pmatrix}$$

- Let's compute the Epipolar Geometry
  - $\mathtt{E} = [\mathbf{t}]_\times \mathtt{R}$, the Essential Matrix
  - $\bar{\mathbf{p}}_2^T \mathbf{E} \bar{\mathbf{p}}_1 = 0$, the Epipolar Constraint



- Planar motion is described by three parameters: θ, ϕ, ρ

$$\mathtt{R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{t} = \begin{pmatrix} \rho\cos\phi \\ \rho\sin\phi \\ 0 \end{pmatrix}$$

- Let's compute the Epipolar Geometry

$$\mathtt{E} = [\mathbf{t}]_\times \mathtt{R} = \begin{bmatrix} 0 & 0 & \rho\sin\phi \\ 0 & 0 & -\rho\cos\phi \\ -\rho\sin\phi & \rho\cos\phi & 0 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

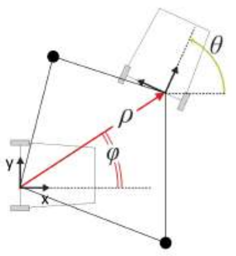
$$\Rightarrow \mathtt{E} = \begin{bmatrix} 0 & 0 & \rho\sin\phi \\ 0 & 0 & -\rho\cos\phi \\ -\rho\sin(\phi-\theta) & \rho\cos(\phi-\theta) & 0 \end{bmatrix}$$

Planar motion is described by three parameters: θ, ϕ, ρ

$$\mathtt{R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{t} = \begin{pmatrix} \rho\cos\phi \\ \rho\sin\phi \\ 0 \end{pmatrix}$$

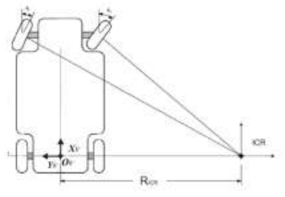
Observe that $\mathtt{E}$ has 2DoF (θ, ϕ because ρ is the scale factor); thus, 2 correspondences are sufficient to estimate θ and ϕ ["2-Point RANSAC", Ortin, 2001]

$$\mathtt{E} = [\mathbf{t}]_\times \mathtt{R} = \begin{bmatrix} 0 & 0 & \rho\sin\phi \\ 0 & 0 & -\rho\cos\phi \\ -\rho\sin(\phi-\theta) & \rho\cos(\phi-\theta) & 0 \end{bmatrix}$$

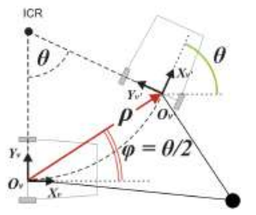


Planar & Circular Motion (e.g., cars)

Wheeled vehicles, like cars, follow locally-planar circular motion about the Instantaneous Center of Rotation (ICR)



Example of Ackerman steering principle



Locally-planar circular motion

$\varphi = \theta/2$ => only 1 DoF (θ);

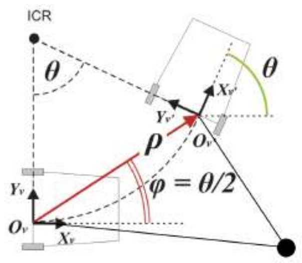
thus, only 1 point correspondence is needed

This is the smallest parameterization possible and results in the most efficient algorithm for removing outliers

Scaramuzza, **1-Point-RANSAC Structure from Motion for Vehicle-Mounted Cameras by Exploiting Non-holonomic Constraints**, International Journal of Computer Vision, 2011

Planar motion is described by three parameters: θ, ϕ, ρ

$$\mathtt{R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{t} = \begin{pmatrix} \rho\cos\phi \\ \rho\sin\phi \\ 0 \end{pmatrix}$$



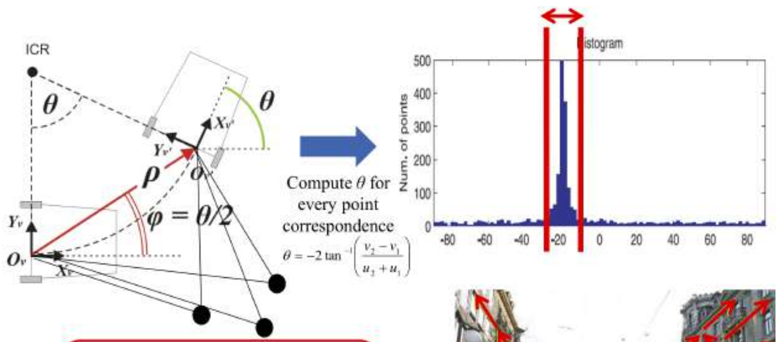
Let's compute the Epipolar Geometry

$$\mathtt{E} = [\mathbf{t}]_\times \mathtt{R} = \rho \begin{bmatrix} 0 & 0 & \sin\frac{\theta}{2} \\ 0 & 0 & -\cos\frac{\theta}{2} \\ \sin\frac{\theta}{2} & -\cos\frac{\theta}{2} & 0 \end{bmatrix}$$

$$\bar{\mathbf{p}}_2^T \mathtt{E} \bar{\mathbf{p}}_1 = 0 \Rightarrow \sin\tfrac{\theta}{2}(\bar{u}_2 + \bar{u}_1) + \cos\tfrac{\theta}{2}(\bar{v}_2 + \bar{v}_1) = 0$$

$$\boxed{\theta = -2\tan^{-1}\left(\frac{\bar{v}_2 + \bar{v}_1}{\bar{u}_2 + \bar{u}_1}\right)}$$

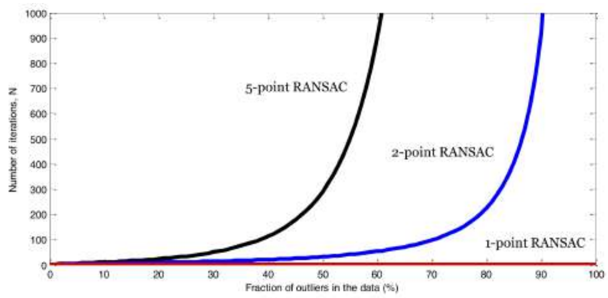
1-Point RANSAC algorithm



Only 1 iteration!
The most efficient algorithm for removing outliers, up to 1000 Hz

1-Point RANSAC is ONLY used to find the inliers.
Motion is then estimated from them in 6DOF

Comparison of RANSAC algorithms



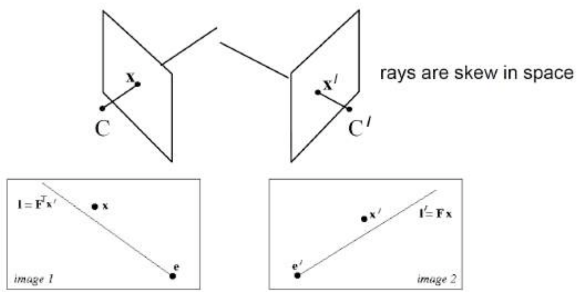
$$N = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^s)} \quad \text{where we typically use } p = 99\%$$

	8-Point RANSAC	5-Point RANSAC [Nister'03]	2-Point RANSAC [Ortin'01]	1-Point RANSAC [Scaramuzza, IJCV'10]
Numb. of iterations	**> 1177**	**>145**	**>16**	**=1**

7.2 Triangulation

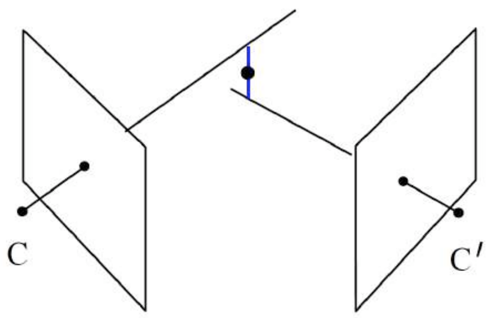
Problem statement

Given: corresponding measured (i.e. noisy) points $\mathbf{x}$ and $\mathbf{x}'$, and cameras (exact) $\mathtt{P}$ and $\mathtt{P}'$, compute the 3D point $\mathbf{X}$
Problem: in the presence of noise, back projected rays do not intersect



Measured points do not lie on corresponding epipolar lines

1. Vector solution



Compute the mid-point of the shortest line between the two rays

2. Linear triangulation (algebraic solution)

Use the equations $\mathbf{x} \sim \mathtt{P}\mathbf{X}$ and $\mathbf{x}' \sim \mathtt{P}'\mathbf{X}$ to solve for $\mathbf{X}$
For the first camera :

$$\mathtt{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} = \begin{pmatrix} \mathbf{p}^{1\mathsf{T}} \\ \mathbf{p}^{2\mathsf{T}} \\ \mathbf{p}^{3\mathsf{T}} \end{pmatrix}$$

Eliminate unknown scale in $\lambda\mathbf{x} = \mathtt{P}\mathbf{X}$ by forming a cross-product
$\mathbf{x} \times \mathtt{P}\mathbf{X} = \mathbf{0}$

$$\begin{aligned} x(\mathbf{p}^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}^{1\mathsf{T}}\mathbf{X}) &= 0 \\ y(\mathbf{p}^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}^{2\mathsf{T}}\mathbf{X}) &= 0 \\ x(\mathbf{p}^{2\mathsf{T}}\mathbf{X}) - y(\mathbf{p}^{1\mathsf{T}}\mathbf{X}) &= 0 \end{aligned}$$

Rearrange as

$$\begin{bmatrix} x\mathbf{p}^{3\mathsf{T}} - \mathbf{p}^{1\mathsf{T}} \\ y\mathbf{p}^{3\mathsf{T}} - \mathbf{p}^{2\mathsf{T}} \end{bmatrix} \mathbf{X} = \mathbf{0}$$

Similarly for the second camera :

$$\begin{bmatrix} x'\mathbf{p}'^{3\top} - \mathbf{p}'^{1\top} \\ y'\mathbf{p}'^{3\top} - \mathbf{p}'^{2\top} \end{bmatrix} \mathbf{X} = \mathbf{0}$$

Collecting together gives

$$\mathtt{A}\mathbf{X} = \mathbf{0}$$

where $\mathtt{A}$ is a 4×4 matrix

$$\mathtt{A} = \begin{bmatrix} x\mathbf{p}^{3\top} - \mathbf{p}^{1\top} \\ y\mathbf{p}^{3\top} - \mathbf{p}^{2\top} \\ x'\mathbf{p}'^{3\top} - \mathbf{p}'^{1\top} \\ y'\mathbf{p}'^{3\top} - \mathbf{p}'^{2\top} \end{bmatrix}$$

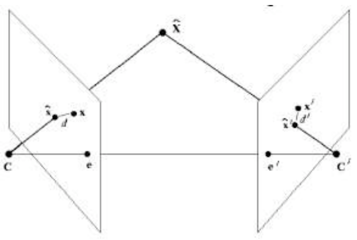
from which $\mathbf{X}$ can be solved up to scale.
Problem: does not minimize anything meaningful
Advantage: extends to more than two views

3. Minimizing a geometric/statistical error

The idea is to estimate a 3D point $\hat{\mathbf{X}}$ which exactly satisfies the supplied camera geometry, so it projects as

$$\hat{\mathbf{x}} \sim \mathtt{P}\hat{\mathbf{X}} \text{ and } \hat{\mathbf{x}}' \sim \mathtt{P}'\hat{\mathbf{X}}$$

and the aim is to estimate $\hat{\mathbf{X}}$ from the image measurements $\mathbf{x}$ and $\mathbf{x}'$



$$\min_{\hat{\mathbf{X}}} \mathcal{C}(\mathbf{x}, \mathbf{x}') = d(\mathbf{x}, \hat{\mathbf{x}})^2 + d(\mathbf{x}', \hat{\mathbf{x}}')^2$$

Stereo Correspondence Algorithms

Problem Statement Given: two images and their associated cameras compute corresponding image points.

Algorithms may be classified into two types:

1. Dense: compute a correspondence at every pixel
2. Sparse: compute correspondences only for features

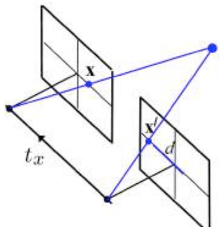
Depth and disparity for a parallel camera stereo rig

$$\mathtt{K} = \mathtt{K}' = \begin{bmatrix} f & & \\ & f & \\ & & 1 \end{bmatrix} \qquad \mathtt{R} = \mathtt{I} \qquad \mathbf{t} = \begin{pmatrix} t_x \\ 0 \\ 0 \end{pmatrix}$$

Then $y' = y$, and the disparity $d = x' - x = \frac{f t_x}{Z}$

Derivation : $\frac{x}{f} = \frac{X}{Z}, \frac{x'}{f} = \frac{X+t_x}{Z}$

$\frac{x'}{f} = \frac{x}{f} + \frac{t_x}{Z}$

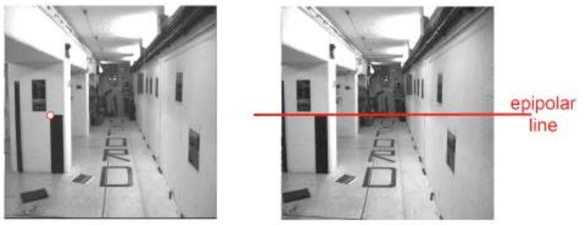


Note :

- image movement (disparity) is inversely proportional to depth Z
- depth is inversely proportional to disparity

Dense Correspondence Algorithm

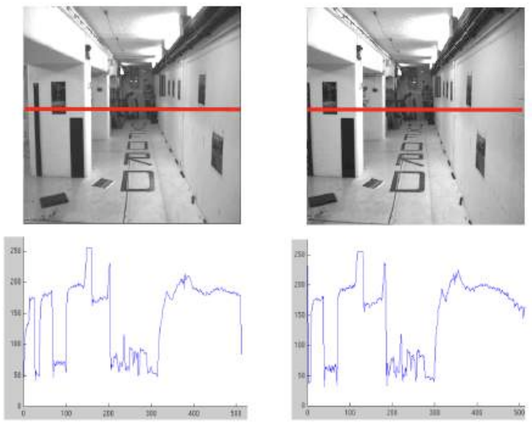
Parallel cameras: epipolar lines are corresponding rasters



Search problem (geometric constraint): for each point in the left image, the corresponding point in the right image lies on the epipolar line (1D ambiguity)
Disambiguating assumption (photometric constraint): the intensity neighbourhood of corresponding points are similar across images
Measure similarity of neighbourhood intensity by cross-correlation

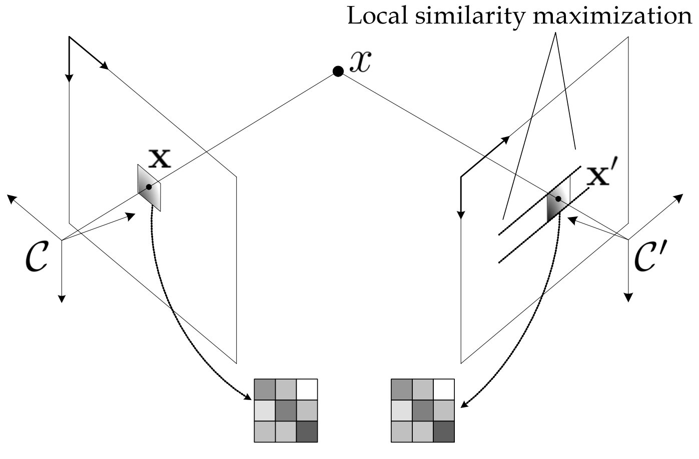
Dense Correspondence Algorithm

Intensity profiles



Clear correspondence between intensities, but also noise and ambiguity

Similarity Measure, principle



Block matching by correlation

Normalized cross-correlation

subtract mean: $A \leftarrow A - <A>, B \leftarrow B - <B>$

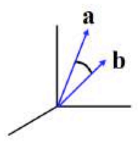
$$\text{NCC} = \frac{\sum_i \sum_j A(i,j)B(i,j)}{\sqrt{\sum_i \sum_j A(i,j)^2}\sqrt{\sum_i \sum_j B(i,j)^2}}$$

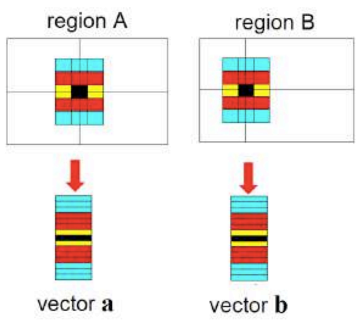
Write regions as vectors

$\mathtt{A} \to \mathbf{a}, \;\; \mathtt{B} \to \mathbf{b}$

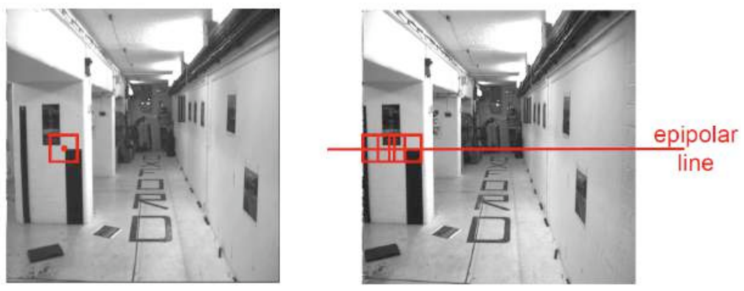
$$\text{NCC} = \frac{\mathbf{a}.\mathbf{b}}{|\mathbf{a}||\mathbf{b}|}$$

$-1 \leq \text{NCC} \leq 1$

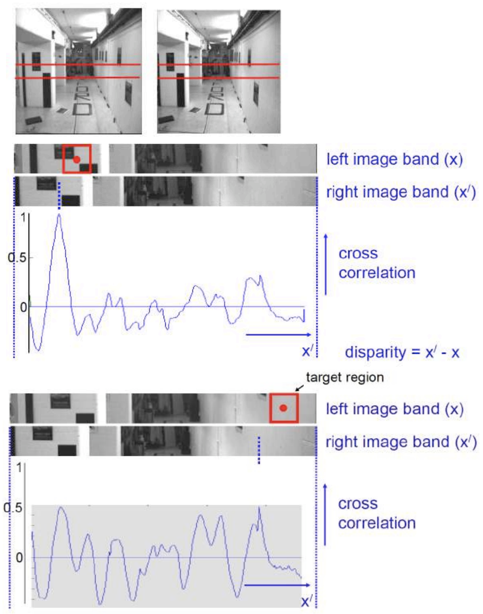




Cross-correlation of neighborhood regions



Invariant to $I \to \alpha I + \beta$



Sketch of a dense correspondence algorithm

For each pixel in the left image

- compute the neighborhood cross correlation along the corresponding epipolar line in the right image
- the corresponding pixel is the one with the highest cross-correlation

Parameters

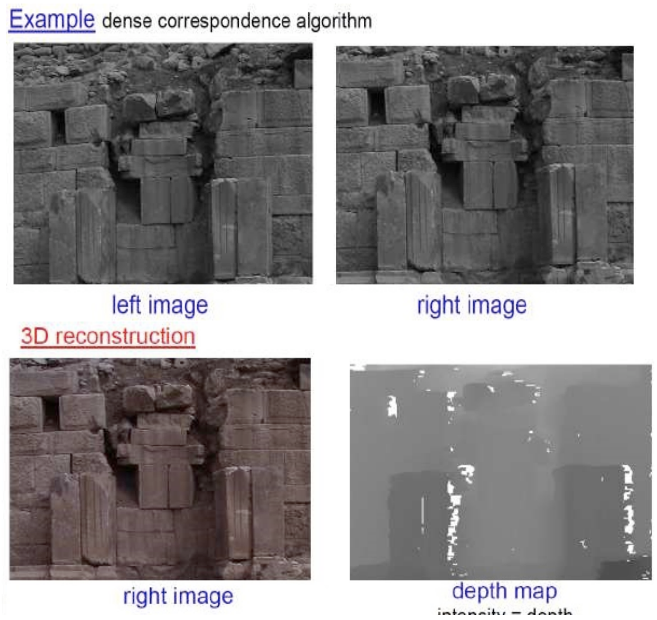
- size (scale) of neighborhood
- search disparity

Other constraints

- uniqueness
- ordering
- smoothness of disparity field

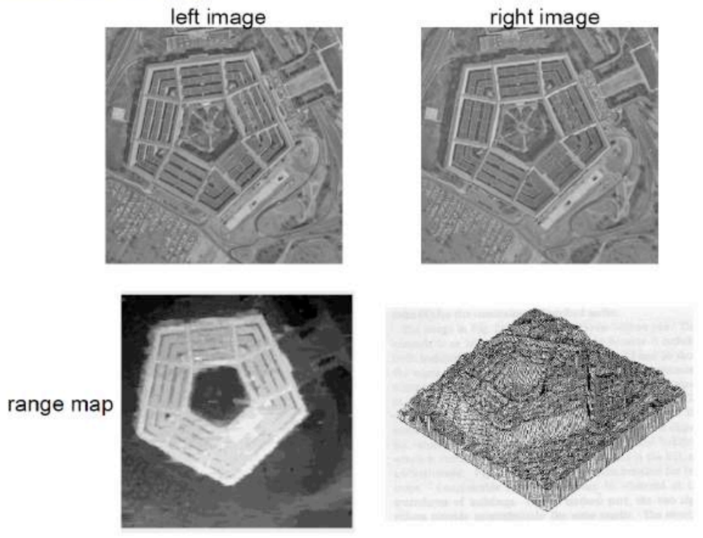
Applicability

- textured scene, largely fronto-parallel



Views of a texture mapped 3D triangulation

Pentagon example

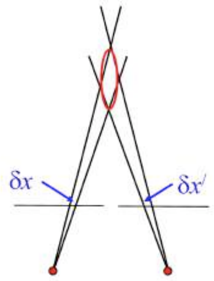


Error Analysis

$d = x' - x = \frac{ft_x}{Z}$, $Z = \frac{ft_x}{d}$, $\frac{\delta Z}{\delta d} = -\frac{ft_x}{d^2} = -\frac{Z^2}{ft_x}$

$$\delta Z = -\frac{Z^2}{ft_x}\delta d$$

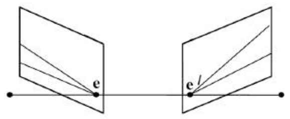
Depth error proportional to depth squared

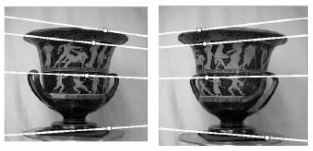


Rectification

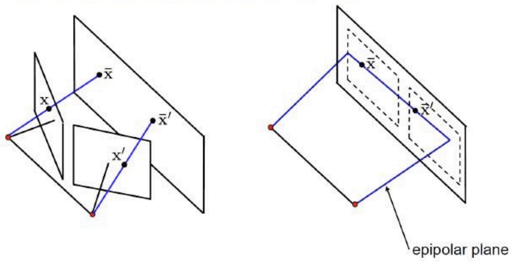
For converging cameras

- epipolar lines are not parallel





Project images onto plane parallel to baseline



Convert converging cameras to parallel camera geometry by an image mapping

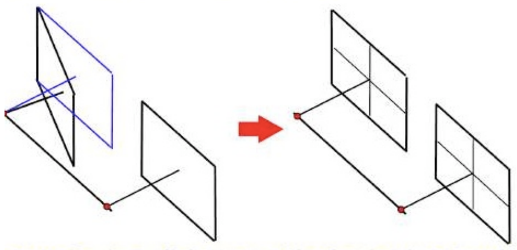
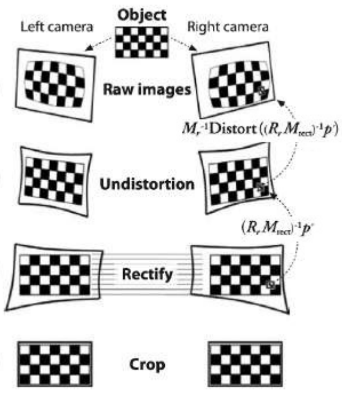
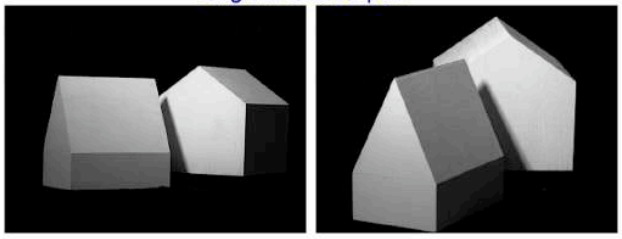


Image mapping is a 2D homography (projective transformation)

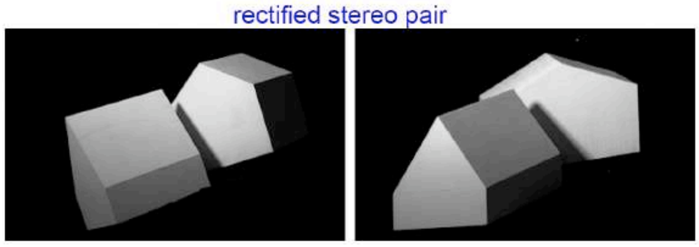


Example

original stereo pair



rectified stereo pair

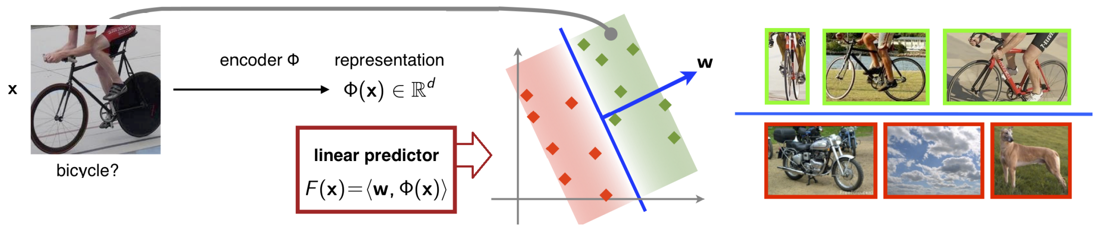


Chapter 8

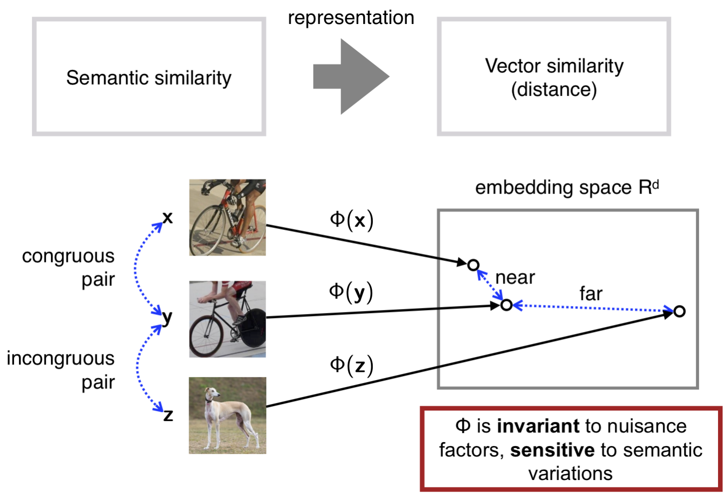
Deep Learning and Semantic Segmentation

8.1 Introduction

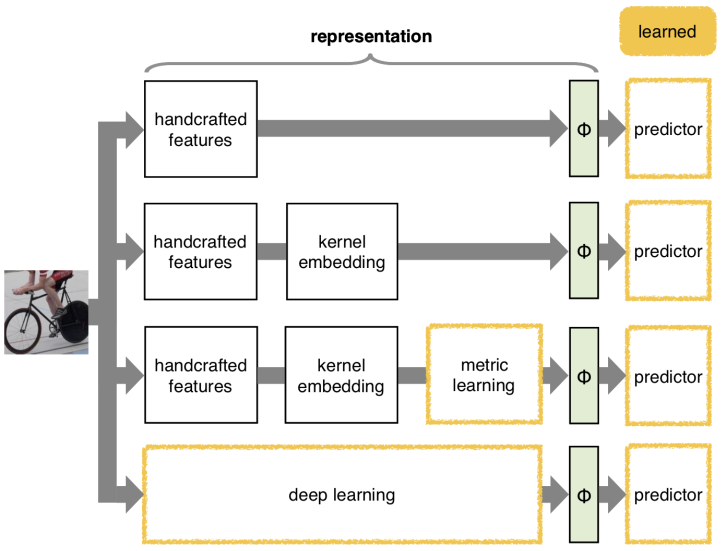
A **linear predictor** can be used to classify vector data. But how such a predictor can be applied to images, text, videos, or sounds? This is possible thanks to an **encoder**, which maps the data to a vectorial representation:



A meaningful representation ϕ has to be sensitive to semantic variations, and reflect in the embedding space the semantic distance between two images $\boldsymbol{x}, \boldsymbol{y}$:

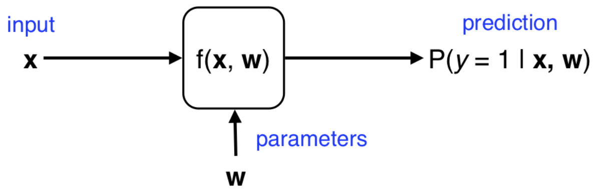


The difference between Deep Learning methods and other representation method is that in this case all the process is hidden **inside** the network:

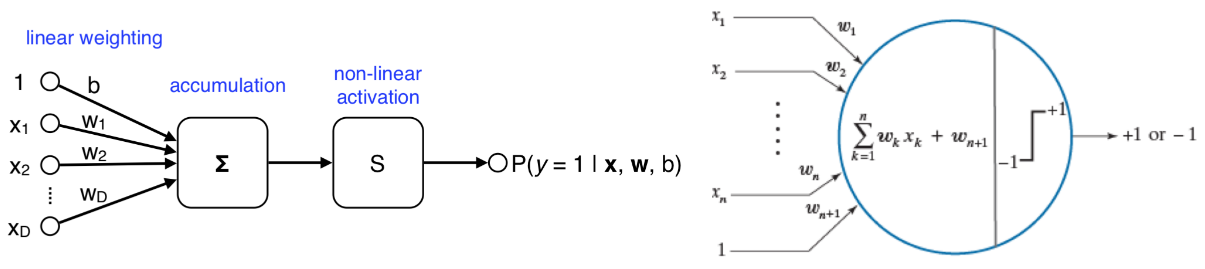


8.2 The perceptron

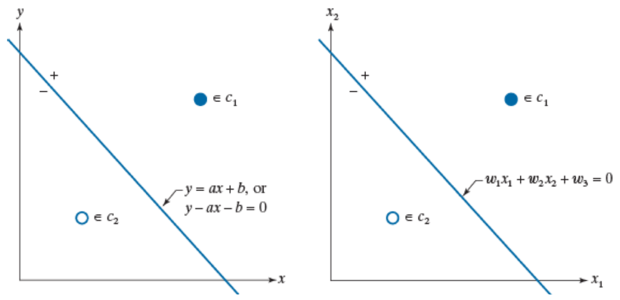
The perceptron is one of the earliest neural network (by Rosenblatt, 1957). It maps a data vector $\boldsymbol{x}$ to a posterior probability value y (for example the probability that $\boldsymbol{x}$ is an image of a bicycle as opposed to something else):

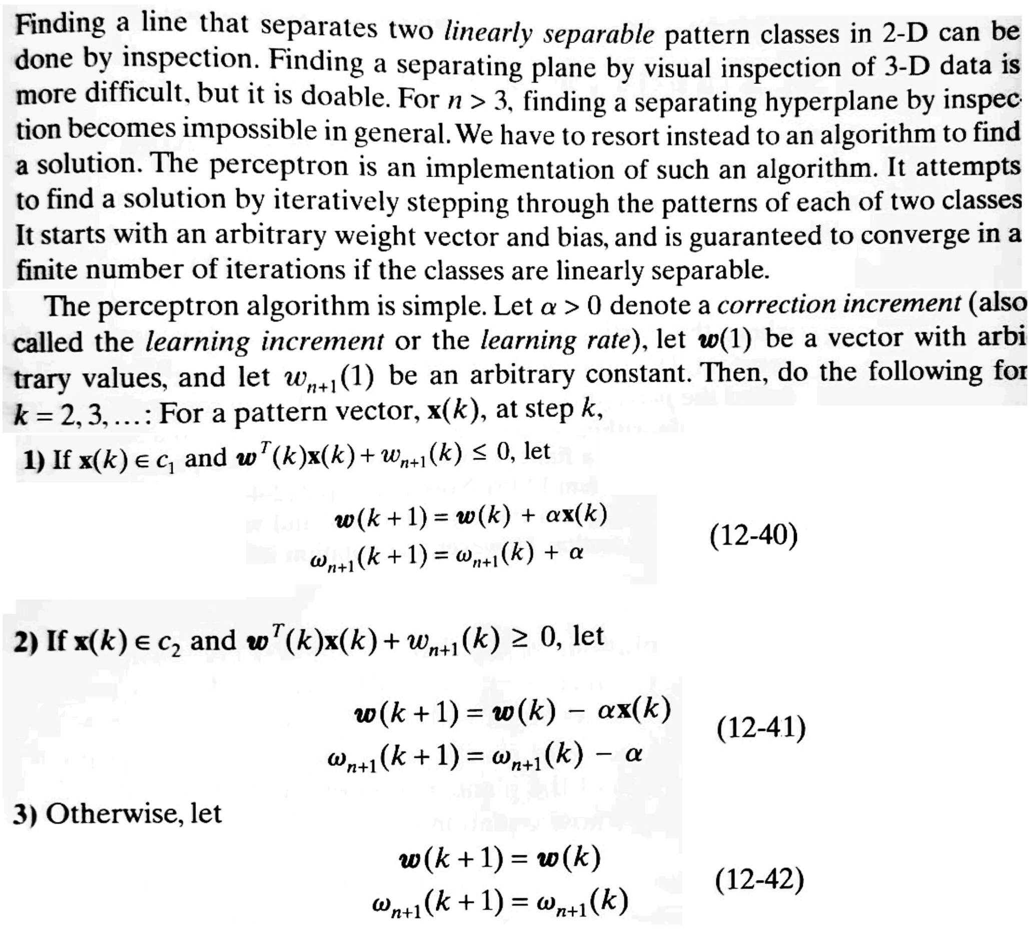


The perceptron computes this probability by weighing the vector elements, summing them, and then applying a non-linear activation function:



What the network learn is to classify elements w.r.t a **decision boundary**:




Finding a line that separates two *linearly separable* pattern classes in 2-D can be done by inspection. Finding a separating plane by visual inspection of 3-D data is more difficult, but it is doable. For $n > 3$, finding a separating hyperplane by inspection becomes impossible in general. We have to resort instead to an algorithm to find a solution. The perceptron is an implementation of such an algorithm. It attempts to find a solution by iteratively stepping through the patterns of each of two classes It starts with an arbitrary weight vector and bias, and is guaranteed to converge in a finite number of iterations if the classes are linearly separable.

The perceptron algorithm is simple. Let $\alpha > 0$ denote a *correction increment* (also called the *learning increment* or the *learning rate*), let $\mathbf{w}(1)$ be a vector with arbitrary values, and let $w_{n+1}(1)$ be an arbitrary constant. Then, do the following for $k = 2, 3, \ldots$: For a pattern vector, $\mathbf{x}(k)$, at step k,

1) If $\mathbf{x}(k) \in c_1$ and $\mathbf{w}^T(k)\mathbf{x}(k) + w_{n+1}(k) \leq 0$, let

$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) + \alpha\mathbf{x}(k) \\ \omega_{n+1}(k+1) &= \omega_{n+1}(k) + \alpha \end{aligned} \qquad (12\text{-}40)$$

2) If $\mathbf{x}(k) \in c_2$ and $\mathbf{w}^T(k)\mathbf{x}(k) + w_{n+1}(k) \geq 0$, let

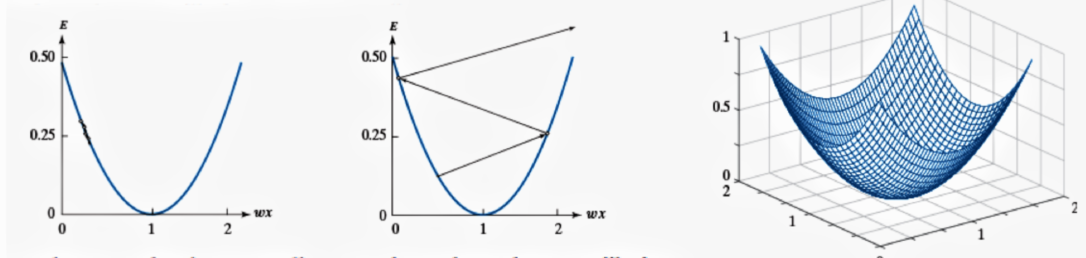
$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) - \alpha\mathbf{x}(k) \\ \omega_{n+1}(k+1) &= \omega_{n+1}(k) - \alpha \end{aligned} \qquad (12\text{-}41)$$

3) Otherwise, let

$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) \\ \omega_{n+1}(k+1) &= \omega_{n+1}(k) \end{aligned} \qquad (12\text{-}42)$$

What is the effects of the learning rate α in terms of classification error E?

Plots of E as a function of wx for $r = 1$.



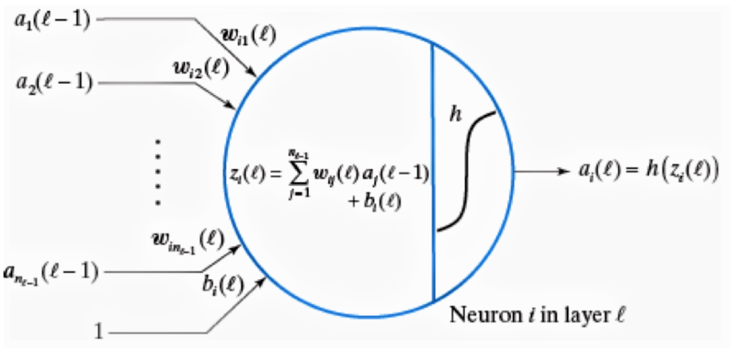
A value of α that is too small can slow down convergence.

If α is too large, large oscillations or divergence may occur.

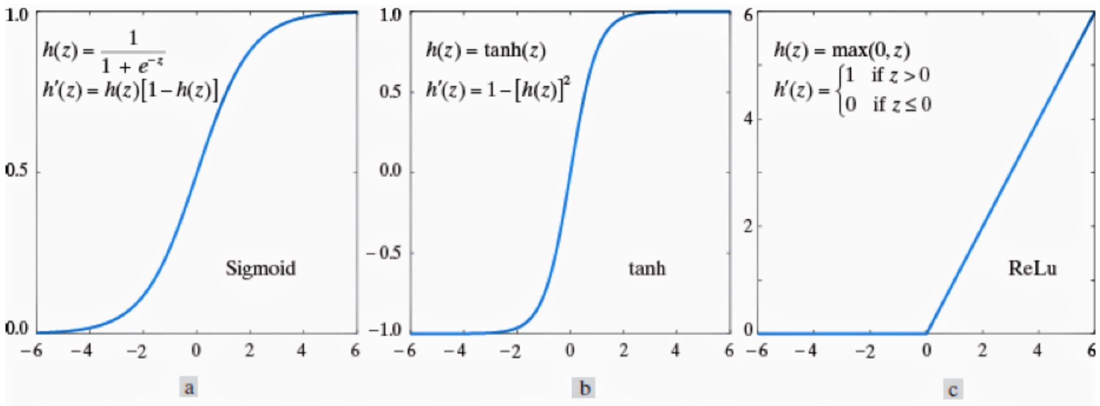
Shape of the error function in 2-D.

8.3 Multilayer feedforward neural networks

In this case the model of an artificial neuron is:

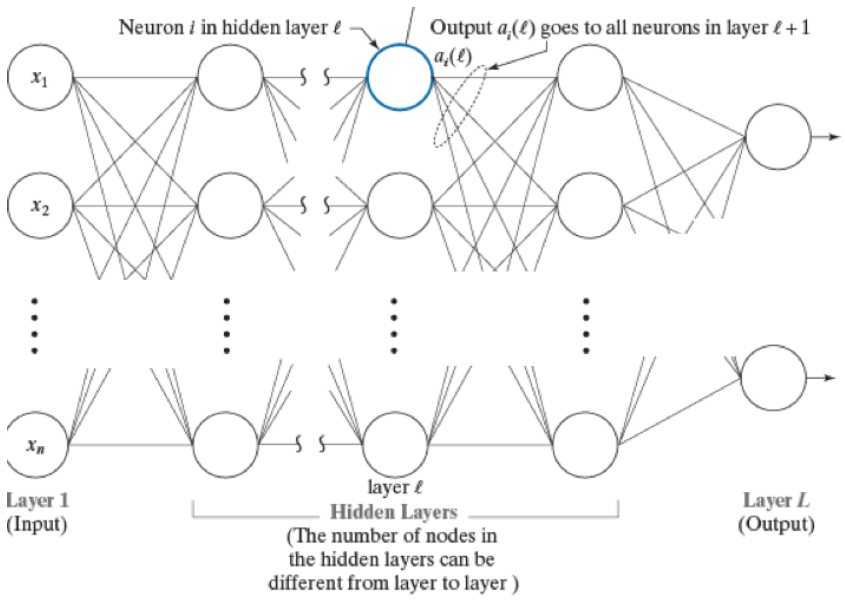


The activation functions can vary:



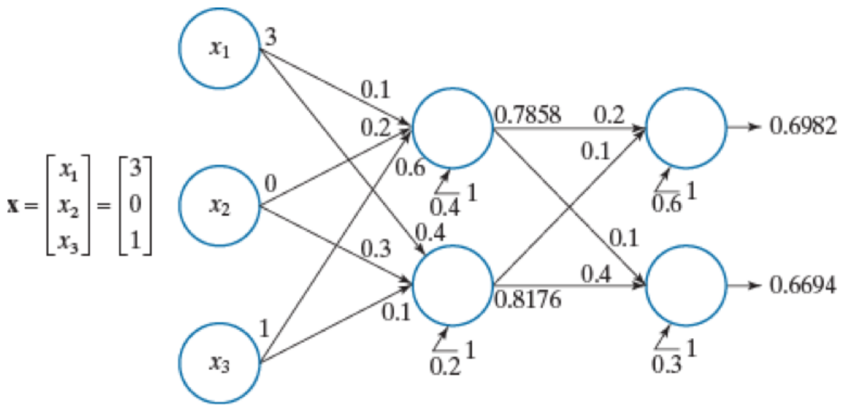
Various activation functions. (a) Sigmoid. (b) Hyperbolic tangent (also has a sigmoid shape, but it is centered about 0 in both dimensions). (c) Rectifier linear unit (ReLU).

So the complete architecture is:

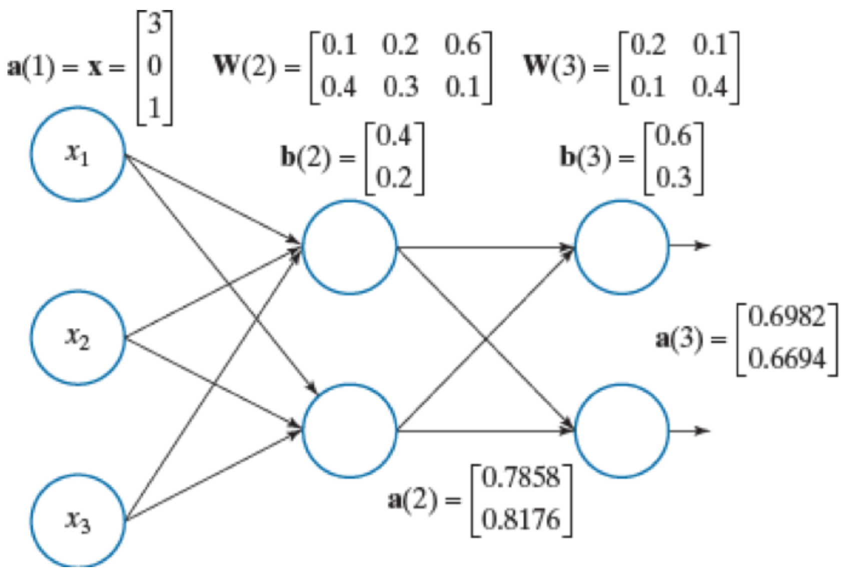


Neuron i in hidden layer ℓ
Output $a_i(\ell)$ goes to all neurons in layer $\ell+1$
$a_i(\ell)$
x_1
x_2
x_n
Layer 1 (Input)
layer ℓ
Hidden Layers
(The number of nodes in the hidden layers can be different from layer to layer)
Layer L (Output)

Example:


$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix}$
x_1
x_2
x_3
3
0
1
0.1
0.2
0.6
0.3
0.4
0.1
0.4
1
0.2
1
0.7858
0.8176
0.2
0.1
0.1
0.4
0.6
1
0.3
1
0.6982
0.6694

In a matrix form is:

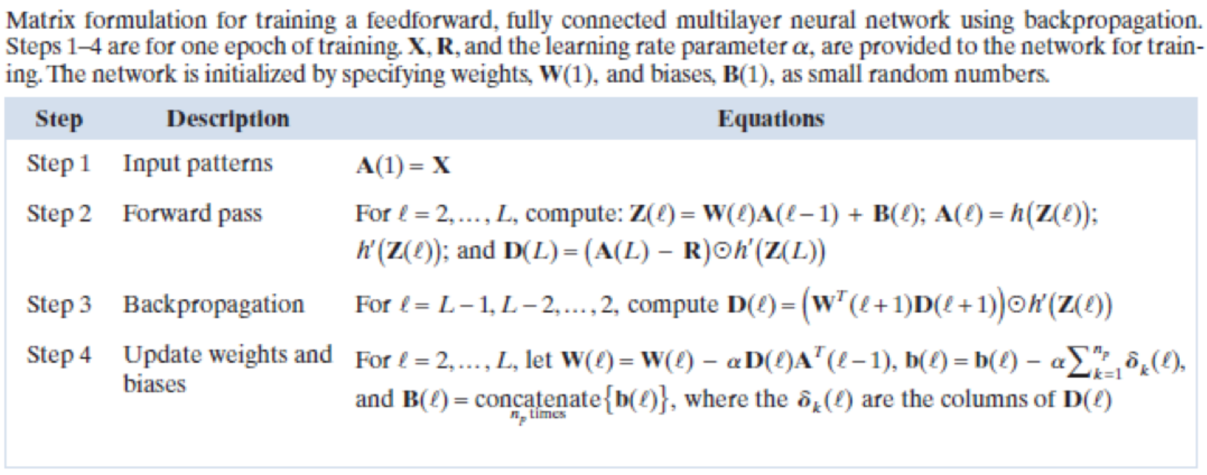

$\mathbf{a}(1) = \mathbf{x} = \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix}$
$\mathbf{W}(2) = \begin{bmatrix} 0.1 & 0.2 & 0.6 \\ 0.4 & 0.3 & 0.1 \end{bmatrix}$
$\mathbf{W}(3) = \begin{bmatrix} 0.2 & 0.1 \\ 0.1 & 0.4 \end{bmatrix}$
$\mathbf{b}(2) = \begin{bmatrix} 0.4 \\ 0.2 \end{bmatrix}$
$\mathbf{b}(3) = \begin{bmatrix} 0.6 \\ 0.3 \end{bmatrix}$
x_1
x_2
x_3
$\mathbf{a}(3) = \begin{bmatrix} 0.6982 \\ 0.6694 \end{bmatrix}$
$\mathbf{a}(2) = \begin{bmatrix} 0.7858 \\ 0.8176 \end{bmatrix}$

Steps in the matrix computation of a forward pass through a fully connected, feedforward multilayer neural net.

Step	Description	Equations
Step 1	Input patterns	$\mathbf{A}(1) = \mathbf{X}$
Step 2	Feedforward	For $\ell = 2, \dots, L$, compute $\mathbf{Z}(\ell) = \mathbf{W}(\ell)\mathbf{A}(\ell - 1) + \mathbf{B}(\ell)$ and $\mathbf{A}(\ell) = h(\mathbf{Z}(\ell))$
Step 3	Output	$\mathbf{A}(L) = h(\mathbf{Z}(L))$

8.4 Convolutional neural networks

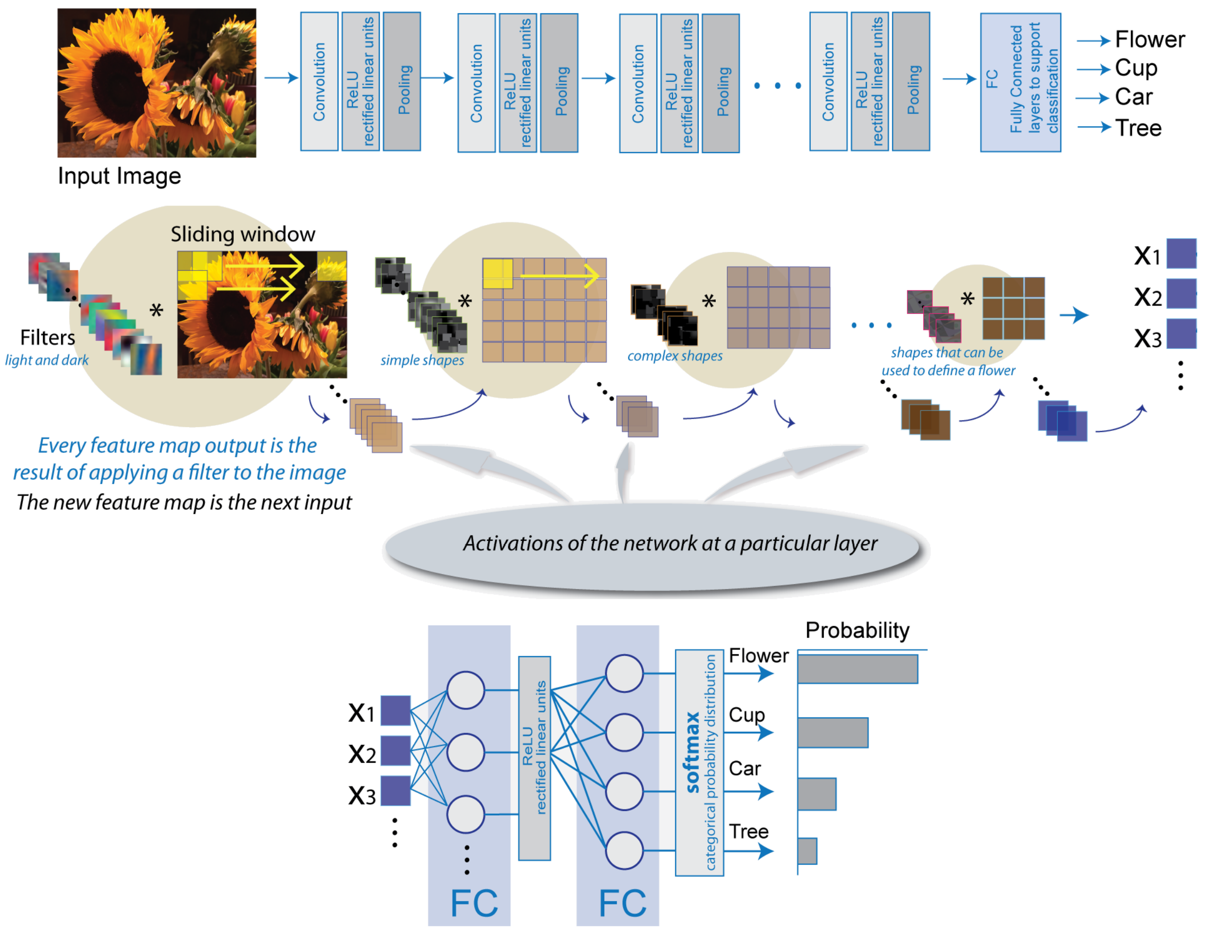
8.4.1 Backpropagation for FCN training



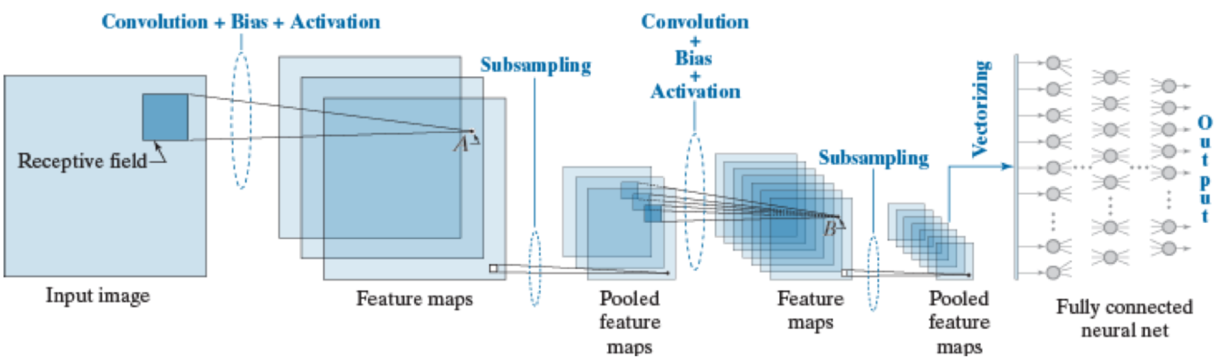
Matrix formulation for training a feedforward, fully connected multilayer neural network using backpropagation. Steps 1–4 are for one epoch of training. $\mathbf{X}$, $\mathbf{R}$, and the learning rate parameter α, are provided to the network for training. The network is initialized by specifying weights, $\mathbf{W}(1)$, and biases, $\mathbf{B}(1)$, as small random numbers.

Step	Description	Equations
Step 1	Input patterns	$\mathbf{A}(1)=\mathbf{X}$
Step 2	Forward pass	For $\ell=2,\dots,L$, compute: $\mathbf{Z}(\ell)=\mathbf{W}(\ell)\mathbf{A}(\ell-1)+\mathbf{B}(\ell)$; $\mathbf{A}(\ell)=h(\mathbf{Z}(\ell))$; $h'(\mathbf{Z}(\ell))$; and $\mathbf{D}(L)=(\mathbf{A}(L)-\mathbf{R})\odot h'(\mathbf{Z}(L))$
Step 3	Backpropagation	For $\ell=L-1, L-2,\dots,2$, compute $\mathbf{D}(\ell)=\left(\mathbf{W}^T(\ell+1)\mathbf{D}(\ell+1)\right)\odot h'(\mathbf{Z}(\ell))$
Step 4	Update weights and biases	For $\ell=2,\dots,L$, let $\mathbf{W}(\ell)=\mathbf{W}(\ell)-\alpha\mathbf{D}(\ell)\mathbf{A}^T(\ell-1)$, $\mathbf{b}(\ell)=\mathbf{b}(\ell)-\alpha\sum_{k=1}^{n_p}\boldsymbol{\delta}_k(\ell)$, and $\mathbf{B}(\ell)=\underset{n_p\text{ times}}{\text{concatenate}}\{\mathbf{b}(\ell)\}$, where the $\boldsymbol{\delta}_k(\ell)$ are the columns of $\mathbf{D}(\ell)$

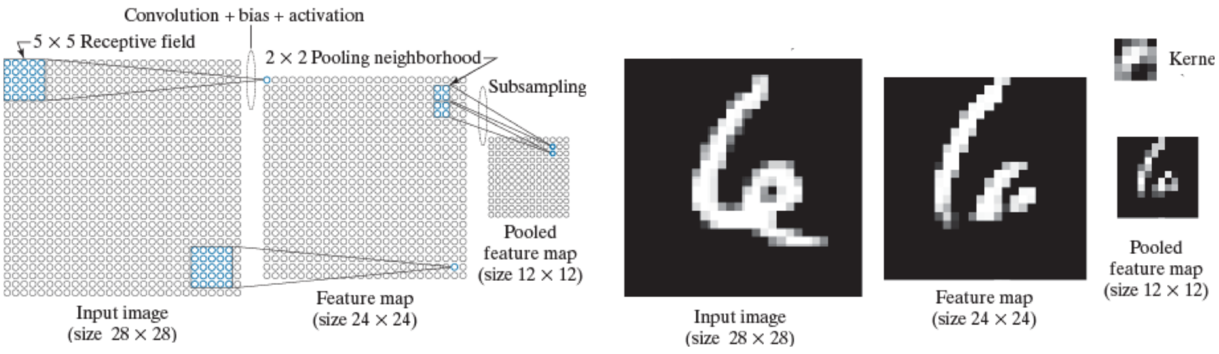
8.4.2 Elements of convolutional neural networks


Input Image
Convolution
ReLU rectified linear units
Pooling
FC Fully Connected layers to support classification
Flower
Cup
Car
Tree
Sliding window
Filters
light and dark
simple shapes
complex shapes
shapes that can be used to define a flower
X1
X2
X3
Every feature map output is the result of applying a filter to the image
The new feature map is the next input
Activations of the network at a particular layer
FC
softmax categorical probability distribution
Probability

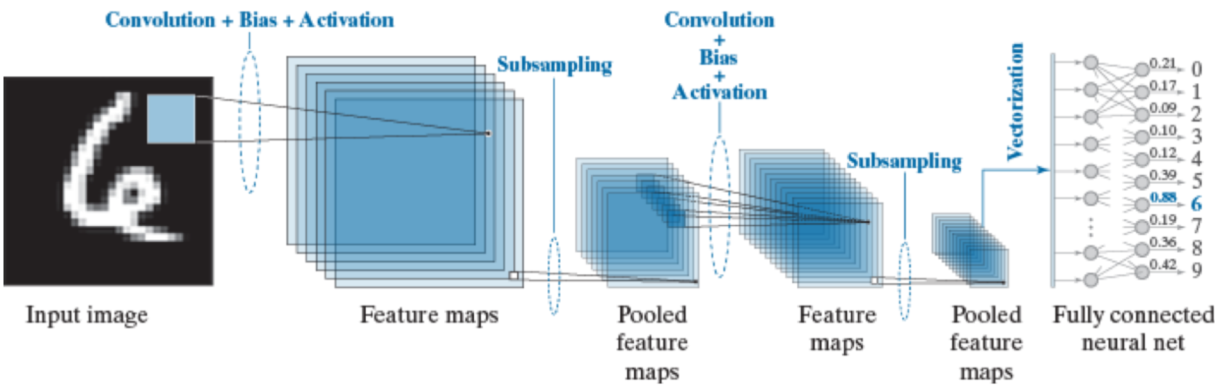
The LeNet Network This is the general functional architecture for the LeNet Network:



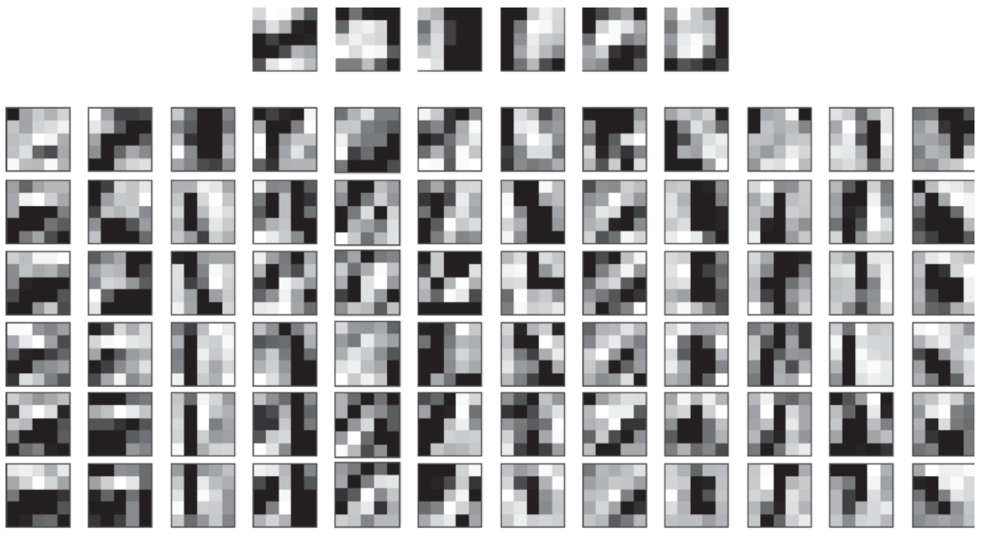
We can see here an example for a handwritten character:


Convolution + bias + activation
5 × 5 Receptive field
2 × 2 Pooling neighborhood
Subsampling
Pooled feature map (size 12 × 12)
Input image (size 28 × 28)
Feature map (size 24 × 24)
Kerne
Input image (size 28 × 28)
Feature map (size 24 × 24)
Pooled feature map (size 12 × 12)

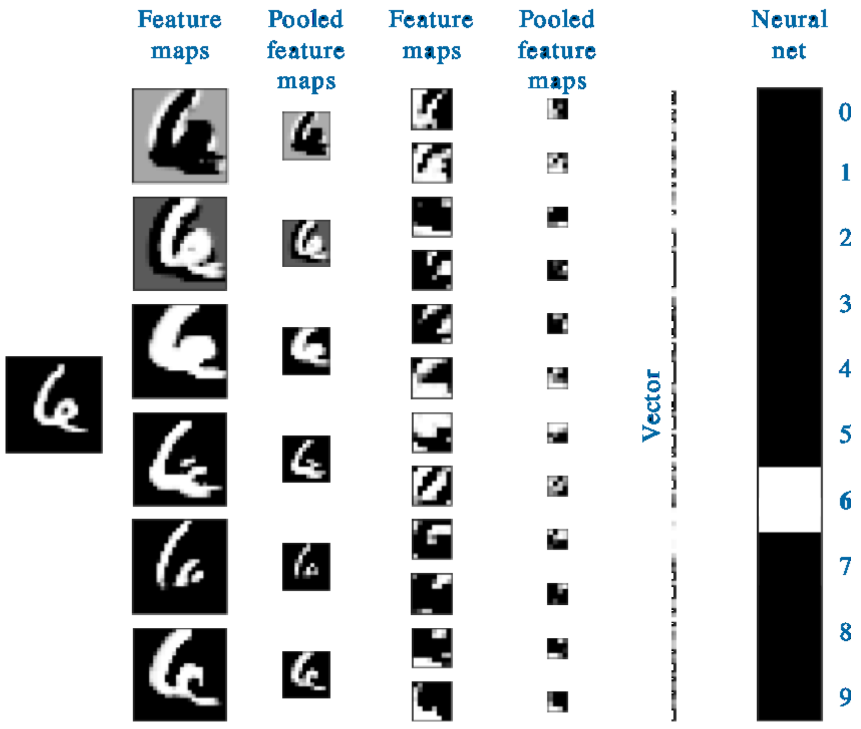
After the training the network is then able to (probably) recognize the character:



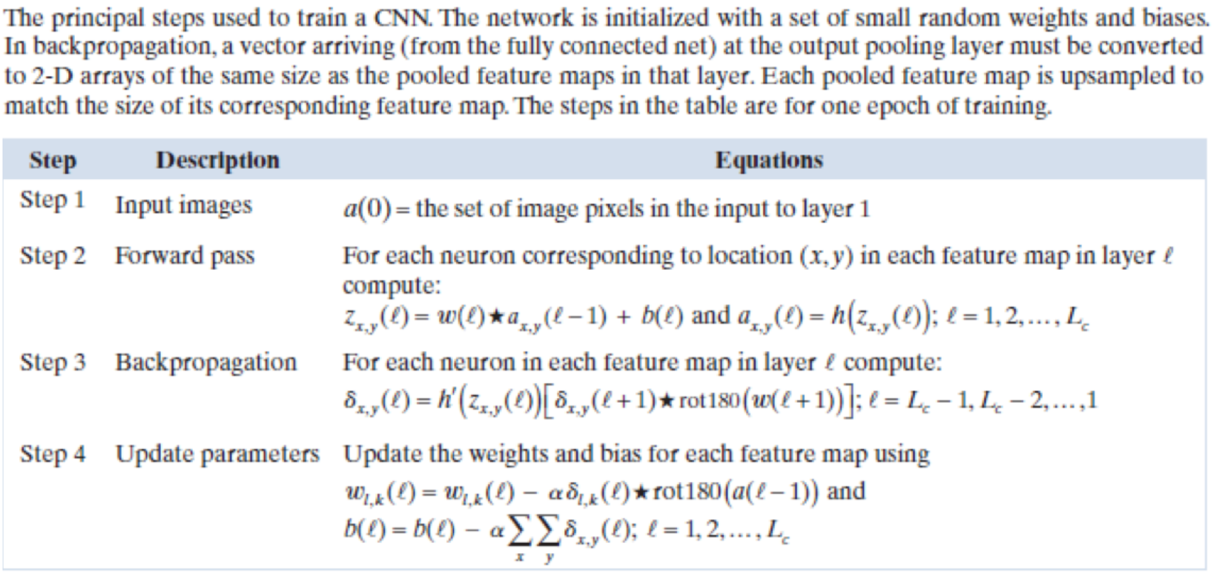
In the following image we can see the the weights of a trained LeNet architecture. On the top there the weights (shown as 5x5pixels images) corresponding to the 6 feature maps in the first layer of the CNN. On the bottom there are the weights corresponding to the 12 feature maps of the second layer:



We show then a visual summary of an input image propagating through the CNN. Here are shown all the results of convolution (**feature maps**) and pooling (**pooled feature maps**) for both layers of the network:



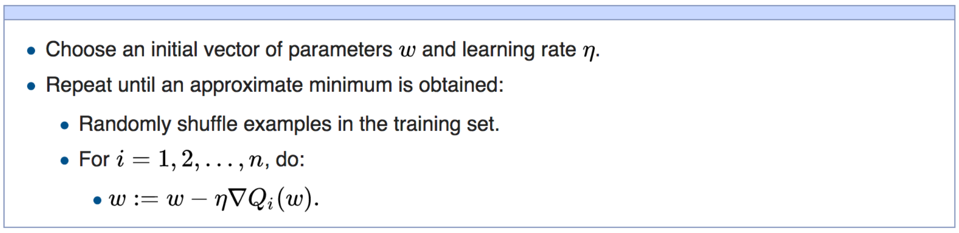
8.4.3 Learning a deep neural network with backpropagation



The principal steps used to train a CNN. The network is initialized with a set of small random weights and biases. In backpropagation, a vector arriving (from the fully connected net) at the output pooling layer must be converted to 2-D arrays of the same size as the pooled feature maps in that layer. Each pooled feature map is upsampled to match the size of its corresponding feature map. The steps in the table are for one epoch of training.

Step	Description	Equations
Step 1	Input images	$a(0)$ = the set of image pixels in the input to layer 1
Step 2	Forward pass	For each neuron corresponding to location (x, y) in each feature map in layer ℓ compute: $z_{x,y}(\ell) = w(\ell) \star a_{x,y}(\ell - 1) + b(\ell)$ and $a_{x,y}(\ell) = h\big(z_{x,y}(\ell)\big)$; $\ell = 1, 2, \ldots, L_c$
Step 3	Backpropagation	For each neuron in each feature map in layer ℓ compute: $\delta_{x,y}(\ell) = h'\big(z_{x,y}(\ell)\big)\big[\delta_{x,y}(\ell+1) \star \mathrm{rot180}\big(w(\ell+1)\big)\big]$; $\ell = L_c - 1, L_c - 2, \ldots, 1$
Step 4	Update parameters	Update the weights and bias for each feature map using $w_{l,k}(\ell) = w_{l,k}(\ell) - \alpha\delta_{l,k}(\ell) \star \mathrm{rot180}\big(a(\ell - 1)\big)$ and $b(\ell) = b(\ell) - \alpha \sum_x \sum_y \delta_{x,y}(\ell)$; $\ell = 1, 2, \ldots, L_c$

Optimization using Stochastic Gradient Descent:



- Choose an initial vector of parameters w and learning rate η.
- Repeat until an approximate minimum is obtained:
  - Randomly shuffle examples in the training set.
  - For $i = 1, 2, \ldots, n$, do:
    - $w := w - \eta\nabla Q_i(w)$.

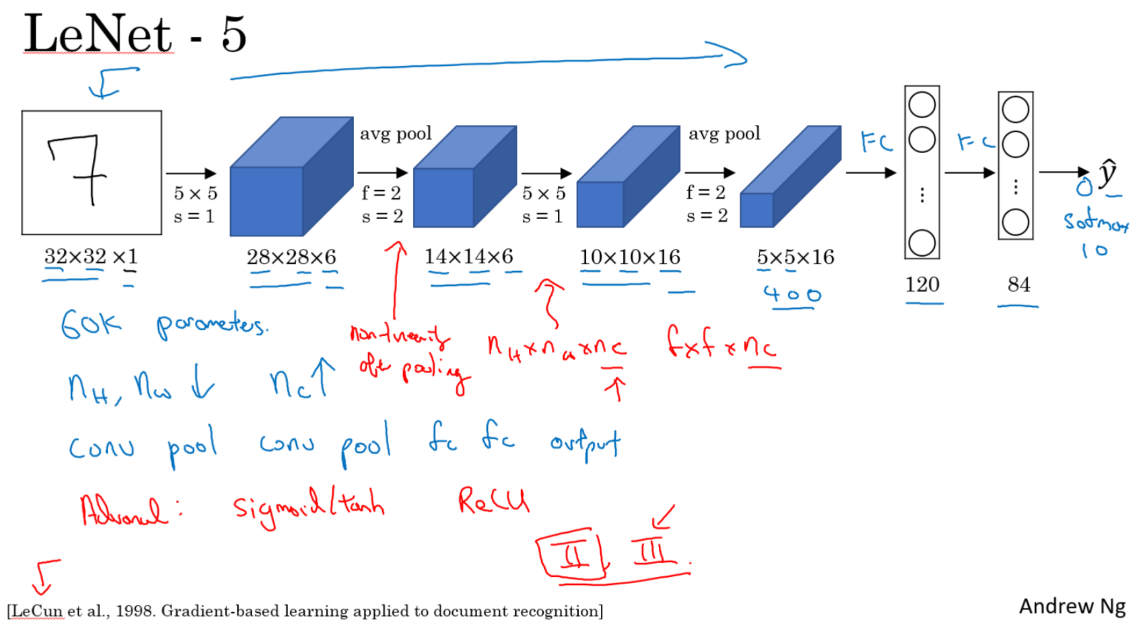
Momentum:

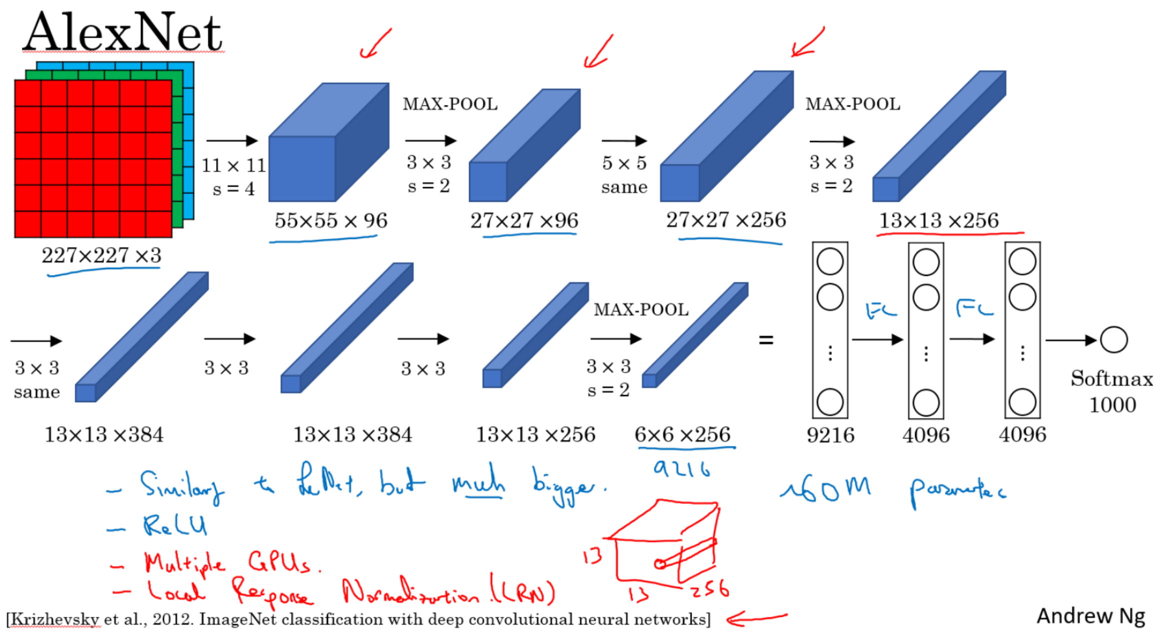
$$w := w - \eta\nabla Q_i(w) + \alpha\Delta w$$

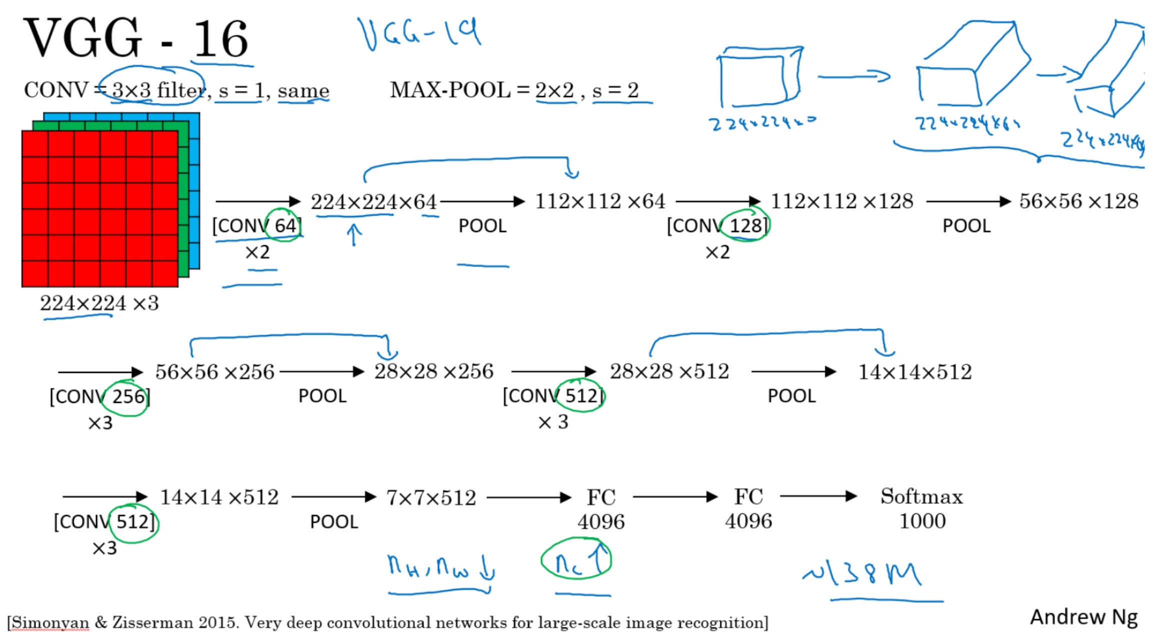
ADAM: Adaptive Moment Estimation:

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1 - \beta_1)\nabla_w L^{(t)}$$
$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1 - \beta_2)(\nabla_w L^{(t)})^2$$
$$\hat{m}_w = \frac{m_w^{(t+1)}}{1 - (\beta_1)^{t+1}}$$
$$\hat{v}_w = \frac{v_w^{(t+1)}}{1 - (\beta_2)^{t+1}}$$
$$w^{(t+1)} \leftarrow w^{(t)} - \eta\frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon}$$

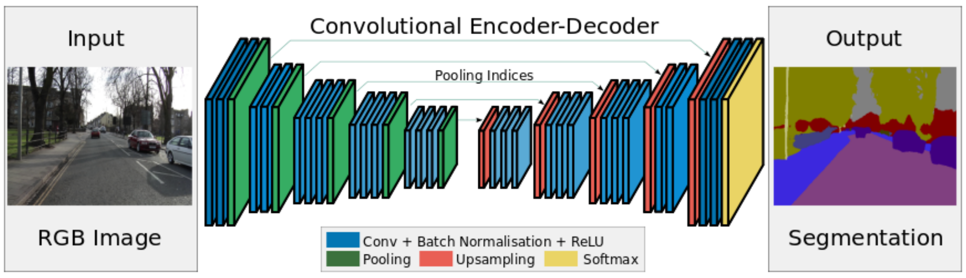
8.4.4 Applications

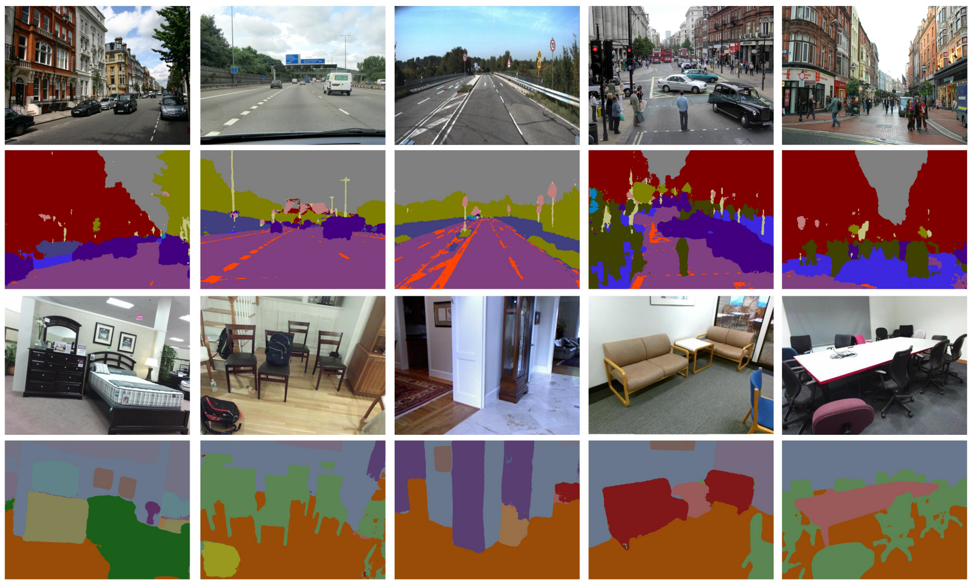






Semantic segmentation: SegNet





Deep Learning Frameworks: Nvidia CUDA drivers + CUDA SDK + CuDNN (https://developer.nvidia.com/); TensorFlow (https://www.tensorflow.org/) + Keras (https://keras.io/)