Light

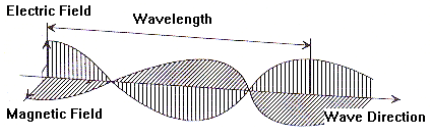
The study of light and its interaction with matter can be carried through at three levels of detail:

1. **geometrical optics** useful for predicting spatial paths of light that interacts with bodies much larger than its wavelength,
2. **physical optics** useful for studying refraction and transmission effects – as light is seen as an electromagnetic wave, or
3. **quantum-mechanical optics** needed to explain aspects of emission and absorption.

Since the analysis of reflected and transmitted light are the essence of computer vision, physical optics is the most suitable approach.

Light as Electro-Magnetic Waves

Characteristic Parameters



The following parameters are required to fully describe a harmonic wave:

1. wavelength
2. direction
3. amplitude $||\boldsymbol{E}||_{\max}$
4. phase
5. direction of polarisation

The strength of the electric and magnetic fields are connected by a simple relation that depends on the medium:

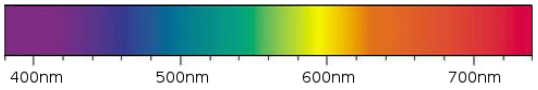
$$\frac{||\boldsymbol{E}||_{\max}}{||\boldsymbol{H}||_{\max}} = \frac{||\boldsymbol{E}||}{||\boldsymbol{H}||} = \sqrt{\frac{\mu}{\varepsilon}}.$$

The wavelength is related to the observed color and the amplitude to the intensity (energy) of the wave.

Wavelength

Electromagnetic waves come in an extremely broad range of wavelengths and no device is able to capture them all. Visible electromagnetic waves lie in the range of

$$\lambda = 380 - 760\,\text{nm}.$$



Interaction with Matter

Through its interaction with matter, light might alter its direction, intensity, state of polarization, and wavelength. Four basic types of interaction are discussed: *absorption, scattering, refraction,* and *reflection.* Diffraction (deviation from the straight propagation in the presence of obstacles) and fluorescence (re-emitted radiation after an atom excited by incoming photons returns to its ground state) are not considered.

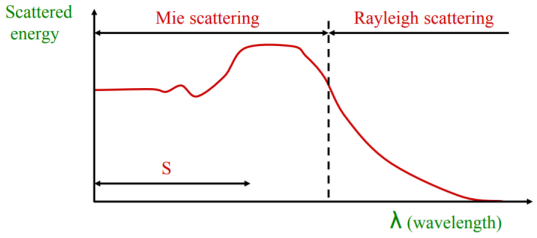
Absorption

As light passes through a medium, even a transparent one, e.g., water or air, it undergoes absorption, as photons collide with particles of matter, give up their energy and disappear. The wavelength of the light influences the amount of absorptance.

Scattering

Light is scattered as it passes through a transparent or translucent medium. One can think of rays being scattered by striking small particles of matter. There are 3 types of scattering depending on relative sizes of particles and wavelengths:

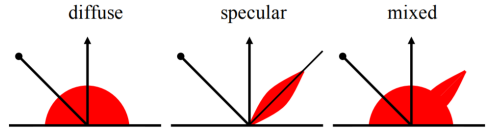
1. **Rayleigh** for small particles, strongly proportional to wavelength (e.g sky),
2. **Mie** for particles of comparable size, weakly proportional to wavelength (e.g colored volcanic cloud),
3. **Non-selective** for large particles, independent of wavelength (e.g clouds).



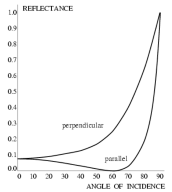
Reflection

When light strikes a surface, its energy is not only refracted into the second medium, but some of it is reflected. There are two types of reflections:

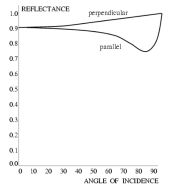
1. **specular** smooth surface, angle of incidence = angle of reflection, keeps polarization, and
2. **diffuse** multitude of small, planar facets, does not keep polarization.



The ratio of reflected power and incoming power is called the reflectance. As with absorption and transmittance, it is possible to look at different wavelengths separately. The direction of propagation of the incoming wave together with the interface normal define the *plane of incidence.* The incoming wave can be decomposed into a component with polarization perpendicular to that plane $\boldsymbol{E}_\perp$ and a component parallel to the plane $\boldsymbol{E}_\parallel$. The reflectance of those components will differ and can help identifying the reflecting medium:



Dielectric Medium

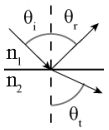


Conductor Medium

Note that for dielectrics there is an angle of incidence, called the *Brewster angle*, for which the parallel component is not reflected, but there is a full reflection at grazing angles.

Refraction

When rays of light pass from a medium of one index of refraction to a medium of a different index of refraction, they are bent, or refracted, unless they strike the boundary between the two media perpendicularly. Assuming that the interface between two media is optically smooth, the direction of refraction is given by *Snell's law*:



$$n_1 \sin\theta_i = n_2 \sin\theta_t.$$

Acquisition of Images

Image formation, i.e. everything that preceeds the actual image processing by a computer, can have a tremendous impact on the robustness, reliability and speed of a vision system, even though it might seem to be a side issue.

Illumination Techniques

There exists multiple illumination techniques to control the environment/limit further processing:

1. **back lighting** light source behind object,
2. **directional lighting** generates shar shadows yielding information about shape (used for crack detection),
3. **diffuse lighting** illuminates uniformly from all directions,
4. **polarized lighting** improves contrast between lambertian and specular reflections & improves contrast between dielectrics (polarizers at brewster angle) and metals (preserve polarization). The intensity of polarized light behaves accourding to the Malus law :

$$I(\theta) = I(0)cos^2(\theta)$$

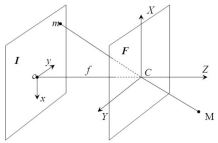
5. **coloured lighting** highlights regions of similar colour with band pass filter
6. **structured lighting** spatially or temporally modulated light pattern,
7. **stroboscopic lighting** eliminates motion blur.

Optics for Image Formation

Camera Obscura (Pinhole Camera)

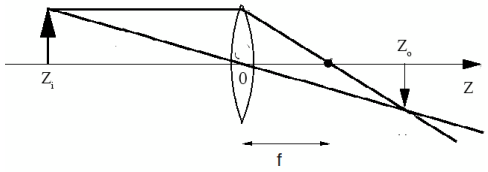
This predecessor of the camera consists of a completely opaque and reflectionless box with a tiny hole in one of its walls. Opposite of that hole is the so–called image plane. The light coming from a point $P_o(X_o, Y_o, Z_o)$ in the direction of the hole $O(0,0,0)$ reaches the image plane in a point $P_i(X_i, Y_i, -f)$. This image formation geometry is referred to as *perspective projection.* From similar triangles one can find

$$\frac{X_i}{X_o} = \frac{Y_i}{Y_o} = -\frac{f}{Z_o} = -m.$$



The Thin-Lens Equation

Since the hole in the pinhole model can not be made arbitrarily small because of insufficient intensity and diffraction effects, lenses are used. Lenses solve these issues, but introduce the problem of unsharp pictures.



For thin lenses, the distance at which sharp images are formed can be calculated as follows:

$$\frac{1}{Z_o} - \frac{1}{Z_i} = \frac{1}{f}$$

The distance f is called the *focal length.* A sharp image will therefore be formed at (negative) coordinate Z_i, i.e., distance $|Z_i|$ from the optical center of the lens. The following assumptions are made:

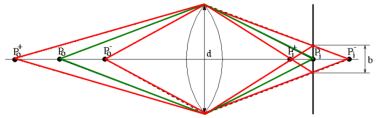
1. The lens surfaces are spherical,
2. incoming light rays make a small angle with the optical axis,
3. the lens thickness is small compared to its radii of curvature, as well as
4. the refractive index is the same for the media on both sides of the lens.

The Depth-of-Field

Given f and Z_i, points at other distances Z_o than specified with the thin-lens equation will become unsharp. In practice, however, a whole **interval** of distances will be projected with acceptable sharpness, due to the finite resolution of image formation and human perception, given by:

$$\Delta Z_0^- = Z_0 - Z_0^- = \frac{Z_0(Z_0 - f)}{Z_0 + f\frac{d}{b} - f}$$

Indeed, an image is called sharp as long as a point projects to a spot with a diameter below some small value b, typically chosen as the resolution of the sensing device. The corresponding points P_o^+ and P_o^- that lie just on the boundary of being projected in the disk with diameter b determine the depth-of-field.



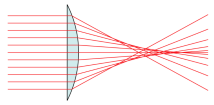
Aberrations

The three assumptions of the model that describes capturing an image are

1. All the rays emerging from a point in space are focused onto the same image point;
2. These image points all fall in a single image plane;
3. Magnification is constant throughout the image plane.

The deviations from this ideal are called aberrations. Two general families of aberrations are distinguished: *geometrical* and *chromatic.* Geometrical aberrations become visible as image distortions or degradations like blurring, whereas chromatic aberrations correspond to the visibly different behaviour of different wavelengths (recall Snell's law). The typical way of reducing these aberrations in practice is by designing composite lenses – up to 10 lenses of different shapes and materials are not uncommon. Chromatic aberration cannot be removed completely.

Radial Distortion



Radial distortion is the most important of geometrical aberrations, not only because it's frequent and often strong, but also because one can undo its effects. Radial distortion is caused by a systematic variation of the optical magnification when radially moving away from a certain point, called the center of distortion. The larger the distance between an image point and the center of distortion, the larger the distortion effect. The center of distortion can usually be approximated to coincide with the principal point and the center of the image. Radial distortion is a non-linear effect and typically modeled using a Taylor expansion. Oftentimes only the even order terms play a role in this expansion, i.e., the effect is symmetric around the center. Let $m_u = (m_{ux}, m_{uy})^\top$ be the image projection of a point in the scene without radial distortion. The distance r from the optical axis is then

$$r^2 = \left(\frac{m_{ux}}{m_{uw}}\right)^2 + \left(\frac{m_{uy}}{m_{uw}}\right)^2.$$

Furthermore, let m_d be defined as

$$m_d = \begin{pmatrix} m_{dx} \\ m_{dy} \end{pmatrix} = \begin{pmatrix} (1 + \kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6) m_{ux} \\ (1 + \kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6) m_{uy} \end{pmatrix},$$

where one typically does not compute more than three parameters ($\kappa_{1:3}$). When the distortion parameters are known, the image can be undistorted.

Cameras

In current times, solid-state cameras dominate the market. Both CCD (charge-coupled device) and CMOS (complimentary metal–oxide semiconductor) image sensors consist of a regular array of light–sensitive cells. The cells collect incoming photons and turn them into electrons through the photoelectric effect. Incident photons increase the energy of electrons,

which absorb the energy from the incoming electromagnetic radiation, leave their original energy band, and become free electrons. The CMOS sensors are more widely used as each photo sensor has its own amplifier (more noise but lower sensitivity); they are cheaper, lower power and allow for random pixel access.

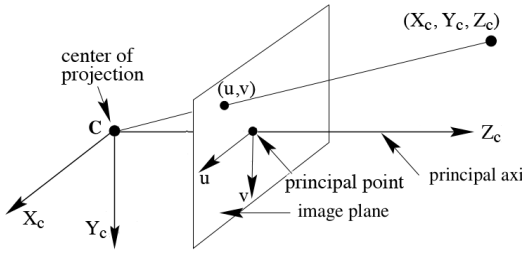
For colour cameras, there are 3 ways to acquire coloured images:

1. **Prism with 3 sensors** for high end cameras, high resolution & framerate,
2. **Filter mosaic** for low end camears, coat filter directly on sensor,
3. **Filter wheel** only for static scenes.

Models for Camera Projection

Perspective Projection

The introduction of lenses allows us to stick to the pinhole camera model, at the expense of limited focus for a given focal length. An image is formed because light rays that pass through the optical center fall onto the image plane. The rays can now be thought to pass through the center of the lens, which acts as the center of projection.



The image plane depicted above is sometimes also called *virtual image plane*, as the physical image plane is located behind the camera center. The usage of the virtual image plane, however, is widely adapted and is justified in the sense that one does not have to deal with the sign reversals of the physical image plane. The coordinate system X_c, Y_c, Z_c has been chosen in a specific way:

- Its origin lies at the center of projection, i.e., in the center of the lens of the camera;
- The Z_c axis coincides with the optical axis of the lens or objective; and
- The plane through the center of projection and perpendicular to the optical axis is the X_cY_c–plane, with the X_c–axis parallel to the main direction. The orientation of the Y_c–axis is fixed as its orthogonal to X_c and Z_c.

This coordinate frame is referred to as the *camera coordinate frame*. The point where the optical axis intersects the image plane is called the *principal point*. The image of a point $\boldsymbol{P}$ in the scene with coordinates $(X_c, Y_c, Z_c) \in \mathbb{R}^3$ is the intersection of the line through $\boldsymbol{P}$ and the center of projection with the image plane, i.e., the plane with equation $Z = f$ (since we are still asuming a pinhole model):

$$u = f\frac{X}{Z} \qquad v = f\frac{Y}{Z}$$

The main assumption of this model is that the object lies at a large distance compared to the focal distance.

Pseudo–Orthographic Projection

If the range of Z values of object points is small compared to Z, f/Z will remain almost constant and the perspective projection equations simplify to

$$u = kX \qquad v = kY.$$

Projection Matrices

The perspective projection is incomplete in at least two ways:

1. The coordinates of points are seldom known in the camera coordinate frame, so transformations have to be applied; and
2. In the perspective projection model, the coordinates of points in the image plane are derived from camera frame coordinates; row and column coordinates (pixels) would be more practical.

Let the position of scene points be described in the world coordinate frame, and let the position $\boldsymbol{C} \in \mathbb{R}^3$ and orientation matrix $\mathbf{R} \in \mathrm{SO}(3)$ of the camera with respect to this frame be known. The coordinates of a scene point $\boldsymbol{P} = (x, y, z)$ with respect to the camera frame are given by

$$\left(\langle \boldsymbol{r}_1, \boldsymbol{P} - \boldsymbol{C}\rangle, \langle \boldsymbol{r}_2, \boldsymbol{P} - \boldsymbol{C}\rangle, \langle \boldsymbol{r}_3, \boldsymbol{P} - \boldsymbol{C}\rangle\right),$$

where $\boldsymbol{r}_i$ denotes the ith column of the matrix $\mathbf{R}$. The corresponding image point has (u, v)–coordinates

$$u = f\frac{\langle \boldsymbol{r}_1, \boldsymbol{P} - \boldsymbol{C}\rangle}{\langle \boldsymbol{r}_3, \boldsymbol{P} - \boldsymbol{C}\rangle} \qquad v = f\frac{\langle \boldsymbol{r}_2, \boldsymbol{P} - \boldsymbol{C}\rangle}{\langle \boldsymbol{r}_3, \boldsymbol{P} - \boldsymbol{C}\rangle}.$$

In *homogeneous coordinates*, this equation can be written as

$$\rho\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} fr_{11} & fr_{12} & fr_{13} \\ fr_{21} & fr_{22} & fr_{23} \\ fr_{31} & fr_{32} & fr_{33} \end{pmatrix}\begin{pmatrix} X - C_1 \\ Y - C_2 \\ Z - C_3 \end{pmatrix}.$$

In computers, however, it is preferable to indicate the position of an image point by means of its pixel coordinates (x, y), which leads to the following transition

$$x = k_x u + sv + x_0 \qquad y = k_y v + y_0 \qquad \frac{k_y}{k_x} = \text{aspect ratio}$$

Where the following nomenclature is used:

- (x_0, y_0) are the pixel coordinates of the principal point;
- k_x is the number of pixels per unit length in the horizontal direction ($1/\text{width of a pixel}$);
- k_y is the number of pixels per unit length in the vertical direction ($1/\text{height of a pixel}$);
- s indicates how strong the shape of a pixel deviates from being rectangular and is usually referred to as the *skewness* of the pixels ($s = 0$ corresponds to rectangular pixels).

A point $\boldsymbol{p}$ in the image with pixel coordinates (x, y) can also be represented in homogeneous coordinates as $(x, y, 1)$. Which allows to rewrite the transition from (u, v)– to (x, y)–coordinates as

$$\boldsymbol{p} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} k_x & s & x_0 \\ 0 & k_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} u \\ v \\ 1 \end{pmatrix}.$$

Defining the *calibration matrix* of the camera as

$$\mathbf{K} := \begin{pmatrix} fk_x & fs & x_0 \\ 0 & fk_y & y_0 \\ 0 & 0 & 1 \end{pmatrix},$$

yields the following *projection equations* for a pinhole camera:

$$\rho\boldsymbol{p} = \mathbf{K}\mathbf{R}^\top(\boldsymbol{P} - \boldsymbol{C}).$$

Internal Parameters

The numbers k_x, k_y, s, x_0 and y_0 are referred to as *internal camera parameters*. When they are known, the camera is said to be *internally calibrated*.

External Parameters

The vector $\boldsymbol{C}$ and the matrix $\mathbf{R} \in \mathrm{SO}(3)$ that describe the position and orientation of the camera with respect to the world frame, are called *external camera parameters*. If $\boldsymbol{C}$ and $\mathbf{R}$ are known, the camera is said to be *externally calibrated*. A camera is said to be *fully calibrated* if it is calibrated both internally and externally

From Object Radiance to Pixel Gray Levels

After the geometric camera model, we now look at the photometric model. There are 2 steps:

1. from object radiance to image irradiance
2. from image irradiance to pixel gray level

We look at the irradiance that an object patch will cause in an image, assuming radiance R is known and the object is at a large distance compared to focal length. After calculations we find that the irradince I is

$$I = \frac{F}{A_i} = R\frac{A_l}{f^2}cos^4(\alpha)$$

We conclude that the irradiance, which is the camera input, is proportional to the radiance if the viewing direction is roughly constant. Now from the irradiance to pixel grey level:

$$f = gI^\gamma + d$$

with g the gain of the camera, γ the nonlinearity of the camera response to incoming light (close to 1 nowadays), and d the dark reference corresponding to the signal with the lens cap on.

Sampling

In order for a computer to process an image, it has to be stored as a series of numbers, each of finite precision. Sampling refers to the spatial discretization of an image (usually in rectangular squares). It is often useful to have resolutions that are powers of 2, e.g., $16 \times 16, \ldots, 512 \times 512$ images. This can be useful for hardware implementations as well as for algorithms such as the FFT.

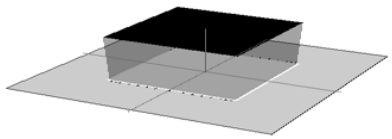
A Sampling Model

The intensity value attributed to a pixel corresponds to the integration of incoming irradiance over a cell of tessellation, which are located at discrete positions. This situation can be modeled in a two–step process:

1. Integrate brightness over regions of the pixel size,
2. Read out values only at the pixel positions.

Let $p(x, y)$ be the *neighbourhood function*

$$p(x, y) := \begin{cases} 1 & \text{inside region with shape of pixel} \\ 0 & \text{otherwise.} \end{cases}$$



The output intensity o can then be written as

$$o(x, y) = \iint i(x', y')p(-(x - x'), -(y - y'))\,\mathrm{d}x'\,\mathrm{d}y',$$

which makes it clear that the output intensity o is the convolution of the input intensity i with $p(-x, -y)$, denoted by $i(x, y) * p(-x, -y)$ and since p is symmetric:

$$o(x', y') = i(x, y) * p(x, y).$$

In a next step, only the values at the pixel locations are selected. This can be achieved by *multiplication with a two–dimensional pulse train* $\delta(x, y)$, i.e., a function consisting of Dirac impulses at the pixel positions. To this end, one can make use of the *sifting theorem*:

$$f(a, b) = \iint_{-\infty}^{\infty} f(x, y)\delta(x - a, y - b)\,\mathrm{d}x\,\mathrm{d}y.$$

Linear, Shift–Invariant Operators

Every linear, shift–invariant operation is essentially a convolution and vice versa.

- **Linearity** Consider a two–dimensional system that produces outputs $o_1(x, y)$ and $o_2(x, y)$ when given inputs $i_1(x, y)$ and $i_2(x, y)$, respectively. The system is called *linear* if the input $ai_1(x, y) + bi_2(x, y)$ produces the output $ao_1(x, y) + bo_2(x, y)$.
- **Shift–Invariance** Consider a system that produces output $o(x, y)$ given input $i(x, y)$. The system is called shift–invariant if it produces the shifted output $o(x - \alpha, y - \beta)$ given the shifted input $i(x - \alpha, y - \beta)$.

Point Spread Function

Suppose a process can be modeled as a linear, shift–invariant operation $\mathcal{O}$. Any arbitrary image i can be considered to be a sum of point sources (Dirac impulses). Knowledge of the operation's output for a point source input could be used to determine the output for i. The output of $\mathcal{O}$ for a point source input is called the *point spread funtion (PSF) of* $\mathcal{O}$. A more general term for the point spread fuction is *impulse response*, the PSF being the impulse response of a focused optical system. The point spread function of an image formation operation $\mathcal{O}$ is denoted by $r(x, y)$. The response to $\delta(x - \alpha, y - \beta)$ is therefore given by $r(x - \alpha, y - \beta)$. Let $i(x, y)$ be an arbitrary input picture, which can always be written as:

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

For the linear, shift–invariant operator $\mathcal{O}$, one can obtain

$$\begin{aligned} o(x, y) &= \mathcal{O}[i(x, y)] \\ &= \mathcal{O}\left[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} i(\alpha, \beta)\delta(x - \alpha, y - \beta)\,\mathrm{d}\alpha\,\mathrm{d}\beta\right] \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mathcal{O}\left[i(\alpha, \beta)\delta(x - \alpha, y - \beta)\right]\,\mathrm{d}\alpha\,\mathrm{d}\beta \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} i(\alpha, \beta)\mathcal{O}\left[\delta(x - \alpha, y - \beta)\right]\,\mathrm{d}\alpha\,\mathrm{d}\beta \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} i(\alpha, \beta)r(x - \alpha, y - \beta)\,\mathrm{d}\alpha\,\mathrm{d}\beta. \end{aligned}$$

The latter expression is the *convolution* of i and r, i.e., $i * r$. A simple change of variable shows that the above expression can also be written as

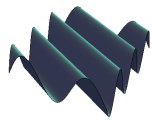
$$o(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} i(x - \alpha, y - \beta)r(\alpha, \beta)\,\mathrm{d}\alpha\,\mathrm{d}\beta,$$

which is no surprise as the convolution is commutative and associative:

$$r * i = i * r \qquad r_2 * (r_1 * i) = (r_2 * r_1) * i.$$

Thus if the image is first processed by convolving with r_1 and later with r_2, that operation can be expressed as the convolution with a single filter $r_1 * r_2$.

The Two–Dimensional Fourier Transform



Because convolution in the spatial domain becomes multiplication in the spatial frequency domain, the transformation to the latter is attractive in the case of linear, shift–invariant operators. For the one–dimensional case, the Fourier transform is based on a decomposition into a set of functions $\mathrm{e}^{\mathrm{i}2\pi ft} = \cos(2\pi ft) + \mathrm{i}\sin(2\pi ft)$ that form an orthogonal basis. Similarly, in the two–dimensional case, we can decompose images into a weighted sum of two–dimensional orthogonal basis functions

$$\mathrm{e}^{\mathrm{i}2\pi(ux - vy)} = \cos(2\pi(ux + vy)) + \mathrm{i}\sin(2\pi(ux + vy)).$$

The frequency now has two components, u and v and the uv–plane is referred to as the *spatial frequency domain*. In contrast, the xy–plane is called the *spatial domain*. The waveforms $\cos(2\pi(ux + vy))$ and $\sin(2\pi(ux + vy))$ correspond to waves in two dimensions. The maxima and minima of $\cos(2\pi(ux + vy))$ lie on parallel equidistant ridges along

$$2\pi(ux + vy) = k\pi$$

for integer k. The two–dimensional Fourier transform $\mathcal{F}$ decomposes a function as a weighted sum of such basis functions. The Fourier transform $\mathcal{F}$ of a function $f(x, y)$ is defined as

$$\mathcal{F}\{f(x, y)\} = F(u, v) := \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)\mathrm{e}^{-\mathrm{i}2\pi(ux + vy)}\,\mathrm{d}x\,\mathrm{d}y.$$

$F(u, v)$ is therefore a complex function $F_R(u, v) + \mathrm{i}F_I(u, v)$ with magnitude and phase angle

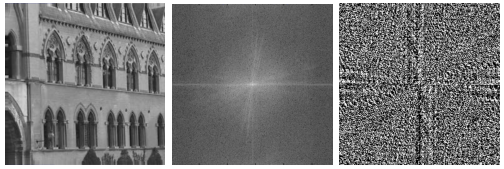
$$|F(u, v)| = \sqrt{F_R^2(u, v) + F_I^2(u, v)}$$

$$\angle F(u, v) = \arctan\left(\frac{F_I(u, v)}{F_R(u, v)}\right).$$

The magnitude squared $|F(u,v)|^2$ is generally referred to as the *power spectrum*. It has peaks at spatial frequencies of repeated texture (can help removing periodic background), and generally decreases with higher spatial frequencies. Even though the Fourier transform outputs a complex function for a real input, it has properties so the total number of independent variables stays the same:

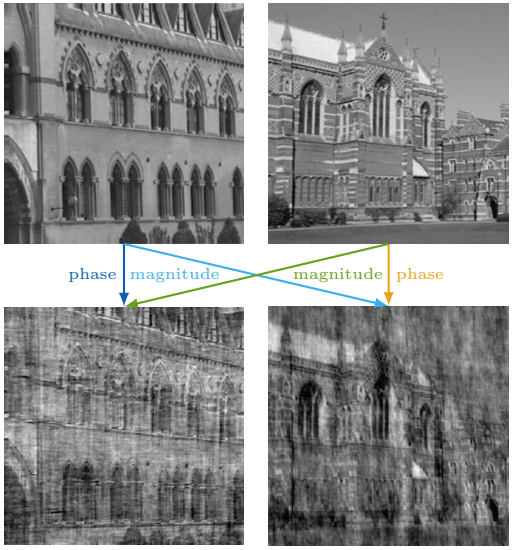
Spatial Domain	Frequency Domain
even	real, even
odd	imaginary, odd
rotation	rotation
scaling	inverse scaling
affine transform	affine transform

At first glance, the magnitude of the Fourier transform seems more informative:



$i(x,y)$ $|I(u,v)|$ $\angle I(u,v)$

When reconstructing the original image from the Fourier transform, however, it turns out that the phase determines the recognizability:



Also, rotation in the spatial domain leads to the same rotation in the frequency domain, whereas scaling in the spatial domain leads to opposite scaling in the frequency domain.

The Convolution Theorem

Let $c(x,y) = a(x,y) * b(x,y)$ and let $A(u,v)$ and $B(u,v)$ be the Fourier transforms of $a(x,y)$ and $b(x,y)$ respectively. The convolution theorem states that

$$C(u,v) = A(u,v)B(u,v).$$

In words: *Convolution in the spatial domain becomes multiplication in the frequency domain and vice versa.* The commutativity and associativity of convolution follow directly from the corresponding properties of multiplication. One consequence of the convolution theorem is that every component basis function of $a(x,y)$ and $b(x,y)$ can be looked at independently, i.e., $A(u,v)B(u,v)$ suffices to know what the result for the frequency component (u,v) will be.

Modulation Transfer Function

Applied to images the convolution theorem states that if we have an input $i(x,y)$ which undergoes a linear, shift–invariant operation with point spread function $r(x,y)$, then the Fourier transform of the output is

$$O(u,v) = \mathcal{F}\{i(x,y) * r(x,y)\} = I(u,v)R(u,v).$$

$R(u,v) \in \mathbb{C}$ is called the *modulation transfer function* and describes how each frequency component of the image behaves under the operation $\mathcal{O}$, telling which frequencies pass and which should be suppressed. Indeed, each frequency component is only changed in these respects, no new frequencies emerge. For each component $I(u,v)$ the corresponding output $O(u,v)$ clearly has the same frequency. The frequency components $e^{-i2\pi(ux-vy)}$ are eigenfunctions of the linear, shift–invariant operator and the eigenvalues are given by $R(u,v)$.

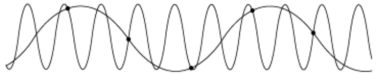
Convolution with Impulse Train

The convolution of a function with an impulse train yields exact copies of the function at the dirac impulse locations. For a single diract impulse:

$$\begin{aligned} i(x,y) * \delta(x,y) &= \mathcal{F}^{-1}\{\mathcal{F}\{i(x,y)\} \cdot \mathcal{F}\{\delta(x,y)\}\} \\ &= \mathcal{F}^{-1}\{\mathcal{F}\{i(x,y)\} \cdot 1\} = i(x,y). \end{aligned}$$

Nyquist Sampling Rate and Aliasing

The lowest possible sampling rate which does not cause overlapping in the frequency domain is called the *Nyquist Sampling Rate*. The problem with frequency overlay is that it makes recovering the original image impossible, as a period of the Fourier domain will no longer be the original Fourier transform. Oversampling is the process of sampling a signal with a sampling frequency significantly higher than the Nyquist rate (twice the highest frequency component in the signal).



Aliasing means that higher frequencies interfere with, i.e., are folded back onto, lower frequencies and the resulting image is degraded. Note that if the input signal is not band–limited (signal frequency domain representation = 0 above certain frequency), aliasing becomes unavoidable.

The Sampling Theorem

The sampling theorem states that if the Fourier transform of a function $f(x,y)$ is zero for all frequencies beyond u_b and v_b, i.e., if the Fourier transform is band–limited, then the continuous function $f(x,y)$ can be reconstructed from its samples as long as the sampling distances w and h along the x and y directions are such that

$$w < \frac{1}{2u_b} \qquad h < \frac{1}{2v_b}.$$

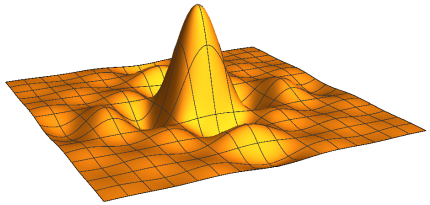
In other words: *All the information of a continuous time signal of finite bandwith can be captured by a discrete sequence of samples.*

Sinc Function

In order to gain a deeper understanding of the sampling model's first step, we need the Fourier transform $P(u,v)$ of the neighbourhood function $p(x,y)$:

$$P(u,v) = wh\left(\frac{\sin(\pi wu)}{\pi wu}\right)\left(\frac{\sin(\pi hv)}{\pi hv}\right).$$

Here, w is the width of the neighbourhood function and h is its depth. Because $p(x,y)$ is real and even, so is its Fourier transform $P(u,v)$. A function of this form is called a *sinc* function.



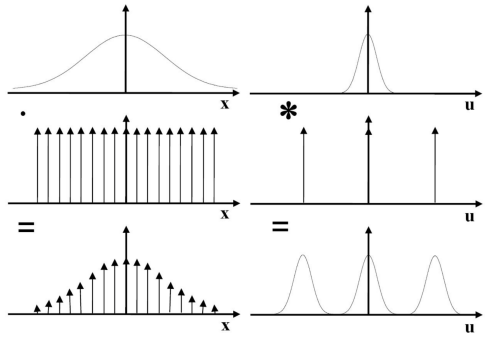
As an important consequence, the neighbourhood "filter" $p(x,y)$ won't change the phase of the frequency components, but only their amplitude. However, there are amplitude sign changes, since $P(u,v)$ becomes negative for some (u,v), which corresponds to a complete phase reversal (shift over π), usually for higher frequencies.

In practice, *sinc* has infinite support, meaning the numerical range where it assigns non-zero values is infinite. Implementing kernels of infinite support on the computer is a difficult task because you would need a very large discretization (essentially a matrix) for the kernel. Finite support kernels however, are easier to implement because you would only have to use a small discretization.

Discretizing the Image

Discretizing the Image in the Spatial Domain

As a second step in the sampling model, only discrete pixel positions of the neighbourhood filtered values are selected. This is modelled as a multiplication with a two–dimensional pattern of Dirac impulses.



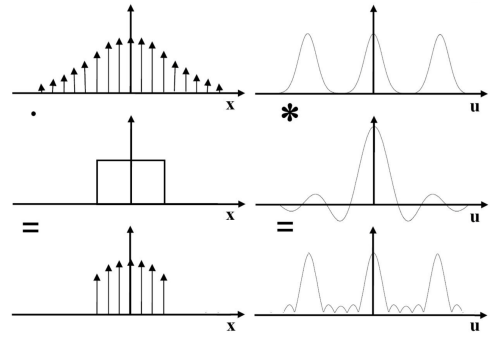
Multiplication in the spatial domain corresponds to convolution in the frequency domain. If

$$\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty}\delta(x-kw, y-lh)$$

is the pulse train pattern in spatial domain (e.g. indicating the location of the pixel centers on a sensor), its Fourier transform is

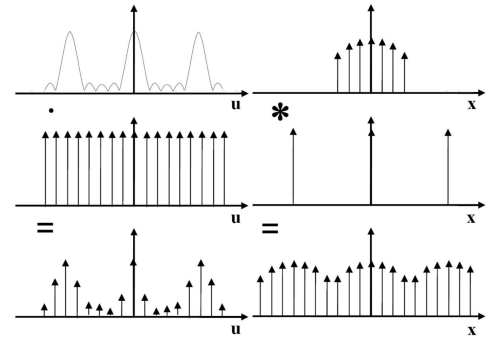
$$\frac{1}{wh}\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty}\delta\left(u-k\frac{1}{w}, v-l\frac{1}{h}\right).$$

A pulse train in the spatial domain yields a pulse train, *with inversely spaced pulses* in the frequency domain (and vice versa). Thus, closer sampling in the spatial domain will further separate the pulses in the frequency domain while coarser sampling will drive them together. As mentioned, the impulse train in the frequency domain (which represents reading out discrete pixel values) is convolved with the Fourier transform of the convolution of the original image with the neighbourhood function. This convolution results in a periodic Fourier transform and it becomes obvious that sampling rates below the Nyquist–rate result in aliasing. Since filtering with the neighbourhood function extended the image, the next step is to limit the spatial extent, which is accomplished by multiplying the discrete filtered values by a rectangle function that is 0 for regions outside the image and 1 for the region of the image.



Discretizing the Image in the Frequency Domain

Analogous to sampling the spatial domain, one also has to sample the frequency domain if is to be analyzed by computer. The sampling theorem also holds in a similar fashion for the case of discrete signals: If the function $f(x,y)$ is zero outside the interval $(-x_b, x_b)$ for x and $(-y_b, y_b)$ for y, then its Fourier domain can be completely reconstructed from samples not further apart than $1/2x_b$ for the u direction and $1/2y_b$ for the v direction.



In conclusion, we can achieve a perfect representation if

1. the signal is band limited
2. the signal is sampled at or above the Nyquist limit
3. the signal is periodic
4. sampling is compatible with the signal period, meaning there is no phase difference after a finite number of periods

The Discrete Fourier Transform

As a result of discretization in the spatial and the frequency domain, one is left with two discrete signals – one in each domain. This pair can be considered together according to what is called the discrete Fourier transform. Sometimes this pair is defined directly, but it can be shown to yield exactly the values of the continuous transform pair at sample positions. The pair is given by

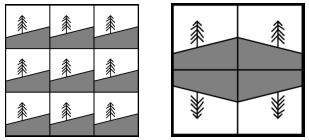
$$F(k,l) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)e^{-i2\pi\left(\frac{km}{M}+\frac{ln}{N}\right)}$$

$$f(m,n) = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} F(k,l)e^{i2\pi\left(\frac{km}{M}+\frac{ln}{N}\right)}.$$

Note that this transform can be done quite efficiently in $\mathcal{O}(N\log_2 N)$ with the FFT–Algorithm.

Interpretation of the Discretized Spaces

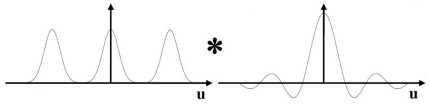
Since discretizing, i.e., multiplying with a dirac pattern, in one domain corresponds to convolving with a dirac pattern in the other domain (implying periodicity), and both domains have been discretized, both domains are periodic. Thus both the image and its frequency content should be interpreted as periods of their respective signals. The discrete Fourier transform is therefore not the Fourier transform of the image as such, but rather of the periodic signal created by repeating the image both horizontally and vertically. For most images, this introduces sharp edges which lead to high frequencies in the Fourier domain that are not originally part of the image.



The picture on the left shows the introduction of sharp edges, whereas if the picture is repeated as shown on the right, the periodic extension would not introduce any new sharp edges.

Leakage

As mentioned before, applying the neighbourhood function in the spatial domain corresponds to the convolution with the sinc function in the frequency domain. Since the sinc has infinite support and the frequency signal is periodic, each period is affected by the ones next to it. This periodicity, however, is just an artifact of the Fourier transform and the alteration of the frequencies in one period by the next period is undesireable and in this context called leakage. It can be compensated for by subsequent sampling in the frequency domain.



Quantization

Quantization refers to the discretization of the pixel brightness. To this end, one can define K intervals in the range of possible measured intensities. K intervals correspond to $\log_2(K)$ bits needed for storage. Note that the intervals do not have to be of the same size, such that the decision levels

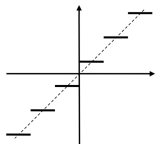
$$z_1, z_2, \ldots, z_{K+1}$$

are ultimately a design choice. Furthermore, one can chose an arbitrary mapping for the representative value of each interval:

$$[z_k, z_{k+1}] \longrightarrow q_k$$

Uniform Quantizer

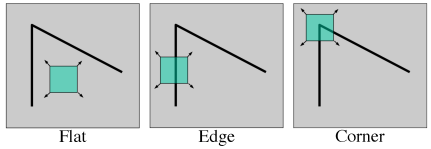
The simplest and arguably most intuitive quantizer has equal interval sizes and choses the representative value of the interval as its mean. This is called the uniform quantizer. Some remarks about the uniform quantizer:



- Its implementation is rather simple;
- Perceptually, a fine quantization is needed;
- Typical number of bits are: 8 for monochrome, 24 for RGB;
- Medical images use up to 16 bits (65536 levels).

Local Feature Extraction

Local features are supposed to help in the task of finding corresponding features in two different images. The challenge is to find benchmarks that match the same features in different images despite large differences in *viewpoint, illumination, background,* and even with *occlusions*. When selecting features, one should considre whether they are complete (describe the pattern unambiguously) or not, the robustness and the ease of extraction, as well as whether it is global or local. Additional requirements are that a feature should capture something discriminative and well localized about the pattern within its local patch. Shifting the patch should make a big difference in the underlying pattern.



Corners are the most prominent examples of so–called **interest points** (points that can be well localised in different views of the scene; blobs are another kind) as there is significant change in all directions.

Uniqueness of a patch

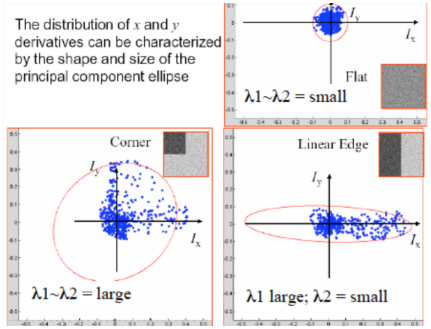
When shifting the window over a patch, we want to compare each pixel before and after by summing up the squared differences (SSD), which defines an SSD error :

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

After a taylor expansion we can rewrite this as:

$$E(u,v) = \begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u & v \end{pmatrix} \mathbf{H} \begin{pmatrix} u \\ v \end{pmatrix}$$

We can then look for the directions [u,v] which will result in the largest and smallest E values, by finding the eigenvectors of **H**. In order to achieve that, we need to apply PCA to H (more details in *Image Decomposition*). The PCA assumes the distribution of (I_x, I_y) to be Gaussian, hence it fits an ellipse to that distribution; the eigenvectors are the long and short axes while the eigenvalues are the ellipse size in these directions.



Since we want E(u,v) to be large for small shifts in all directions, the smaller eigenvalue λ_- of **H** should be large (more details in *Basic Feature Detection*).

Descriptor Problem Formulation

The problem with corners is that usually there are many corners coming out of a corner detector (which yields image points), yet it is hard to tell them apart. In order to discriminate between them, one needs to build a feature vector of the surrounding patch, i.e., a *descriptor*. During a subsequent matching step, the descriptors can then be matched. A requirement for the descriptor is invariance under geometric and photometric changes.

Geometric Invariance

A local patch is small, hence probably rather planar. The question becomes how different views of the same planar contour differ. Suppose (x, y) are the contour coordinates in one image and (x', y') in another. The challenge is to find the transformation of $(x, y) \to (x', y')$. Three different cases are considered:

1. **Similarity:** Suppose the contours are viewed from a perpendicular direction and can only rotate about an axis paralell to this direction. Any 3D translation is allowed. The transformation is

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = s \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}.$$

Note that a similarity transformation has **four** degrees of freedom.

2. **Affinity:** Assume the contour can undergo arbitrary three–dimensional rotations and translations but the projection is simplified to a pseudo–orthographic projection (considering the objet is small compared to its distance to the camera)and scaling. The coefficients can be found using the planer restriction

$$aX + bY + cZ + d = 0$$

and solving for Z. The transformation becomes

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

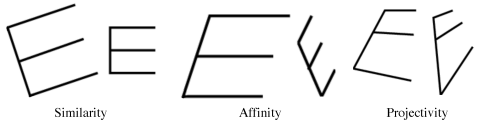
Note that an affinity transformation has **six**degrees of freedom.

3. **Projectivity:** Same assumptions as for the affinity, i.e., arbitrary three–dimensional rotations and translations, this time under the perspective projection model. A projectivity is of the form

$$x' = \frac{p_{11}x + p_{12}y + p_{13}}{p_{31}x + p_{32}y + p_{33}} \qquad y' = \frac{p_{21}x + p_{22}y + p_{23}}{p_{31}x + p_{32}y + p_{33}}.$$

Note that a projectivity transformation has **eight** degrees of freedom. We can also show that:

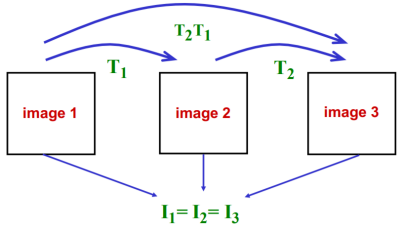
Similarities ⊂ Affinities ⊂ Projectivities



Since local patches are per definition small, one can expect that similarity and affinity transformations will yield acceptable results.

Invariance and Groups

Complexity of the group goes up with the generality of the viewing conditions, and so does the complexity of the group's invariants.Invariance under transformations implies invariance under the smallest encompassing group



Photometric Invariance

A reasonable model for a photometric change is given by

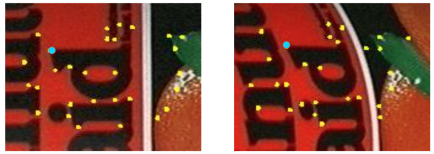
$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{pmatrix} s_R & 0 & 0 \\ 0 & s_G & 0 \\ 0 & 0 & s_B \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} o_R \\ o_G \\ o_B \end{pmatrix}.$$

As descriptor patches cover part of a surface with some texture, the descriptor also have to be invariant photometric changes. Using edge information is often a way to go around photometric changes. The linear part of the equations caters for changes of color and/or intensity of the illumination (the 3 scale factors are the same if the illumination changes intensity only). The non-linear part caters for specularities.

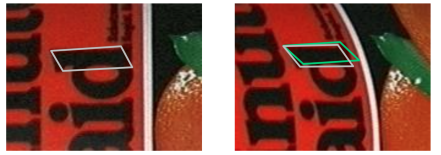
Variable Patch Shape

Handling Different Perspectives

Once a blob has been identified as interesting, the patch should be transformed according to the perspective of the image, such that the same "physical pixels" are analyzed for the feature generation. The shape of the patch therefore has to be adapted. The important thing is to achieve such change in patch shape without having to compare the images, i.e this should happen on the basis of information in one. In order to achieve this, first a Harris corner detector can be run over a patch.



Apply a Canny edge detection and determine the relative affine invariant parameter along two edges by moving away from the corner and considering point pairs that yield equal areas between the edge and the straight line between the points and the corner in opposite directions. This yields a one–dimensional family of pairs. Subsequently one can construct a one–dimensional family of parallelogram shaped regions from the corner point and the point pairs. Select the parallelograms based on an invariant extrema of the average value of the colorband within a patch.



Examples

Typical examples of interest points are:

1. **Edge corners + affine moments**: describe the pattern within the parallelogram with affine invariant moments.
2. **Intensity extrema + affine moments**: search intensity extrema, observe intensity profile along rays, search maximum of invariant function along each ray, connect local maxima, fit ellipse, double ellipse size, and describe elliptical patch with moment invariants.
3. **MSER** (Maximally Stable Extremal Regions) interest points. Starting with the intensity extremum, you move the intensity threshold away from its value and watch the super/sub-threshold region grow. The extrema region is defined such that

$$\forall p \in Q, \forall q \in \delta Q : I(p) > I(q) \text{ or } I(p) < I(q)$$

You can then order regions, following increasing or decreasing threshold:

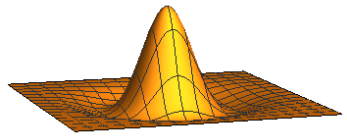
$$Q_1 \subset \ldots \subset Q_i \subset Q_{i+1} \subset \ldots Q_n$$

You then select the regions at thresholds where the growth is slowest (happens when regions are bounded by strong edges). These MSER are the local minimum of

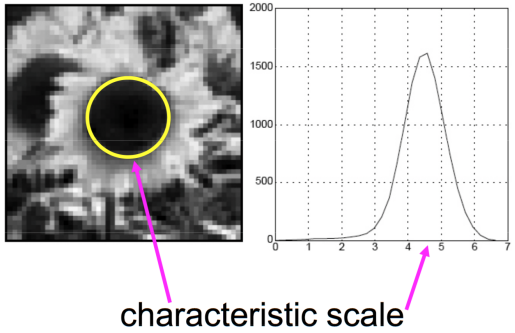
$$q(i) = |Q_{i+\Delta}/Q_{i-\Delta}|/Q_i$$

4. **SIFT Descriptor** SIFT stands for scale–invariant feature transform. It is an interest point detector and descriptor based on *intensity gradients* that is *invariant under similarities, but not under affinities.*

The **SIFT blob detector** is based on a Laplacian of a Gaussian (LoG), or a difference of Gaussians (DoG) respectively. This allows the detection to be robust against changes in scale, rotation and illumination. Recall how the LoG filter looks like:



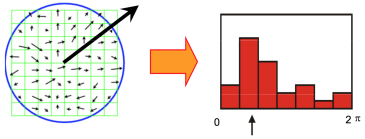
The characteristic scale of a blob is defined as the scale that produces a peak in the Laplacian response in the blob center.



Note that the response of a derivative of a Gaussian to a perfect step edge decreases as σ increases. Since the Laplacian is the second derivative, one has to scale the response by σ^2 in order to get a normalized response. In order to detect interesting points in the image, it is convolved with various Laplacian of Gaussian filters at different scales. This not only yields the blob location but also its characteristic scale.

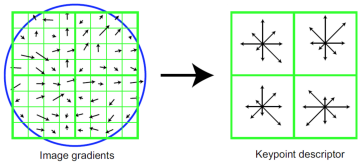
Rotational ambiguity

In order to eliminate the rotational ambiguity, histograms of the local gradient directions in the detected blob are created. The dominant orientation is selected and the patch is rotated accordingly.



Feature Generation and Matching

With the patches rotated, the next step is to generate features that describe the patch. To this end, thresholded image gradients are sampled over a grid. Subsequently, an array of discretized orientation histograms is created within blocks of this grid. The histograms are then used to create 128–dimensional feature vectors $\boldsymbol{f} \in \mathbb{R}^{128}$ ($8 \times 4 \times 4$).



The graphic shows 8×8 and 2×2 grids, in most implementations, however, 16×16 and 4×4 grids are used. Finally, blobs are matched if they have a small distance in feature space. This can be a regular Euclidian distance

$$d(\boldsymbol{f}_1, \boldsymbol{f}_2) = ||\boldsymbol{f}_1 - \boldsymbol{f}_2||$$

or, to be more robust against repetitive features, use this distance

$$d(\boldsymbol{f}_1, \boldsymbol{f}_2) = \frac{||\boldsymbol{f}_1 - \boldsymbol{f}_2||}{||\boldsymbol{f}_1 - \boldsymbol{f}_2'||},$$

where $\boldsymbol{f}_2'$ is the feature vector of the second best match to $\boldsymbol{f}_1$.

5. **SURF**: Speeded Up Robot Features is the efficient alternative to SIFT.

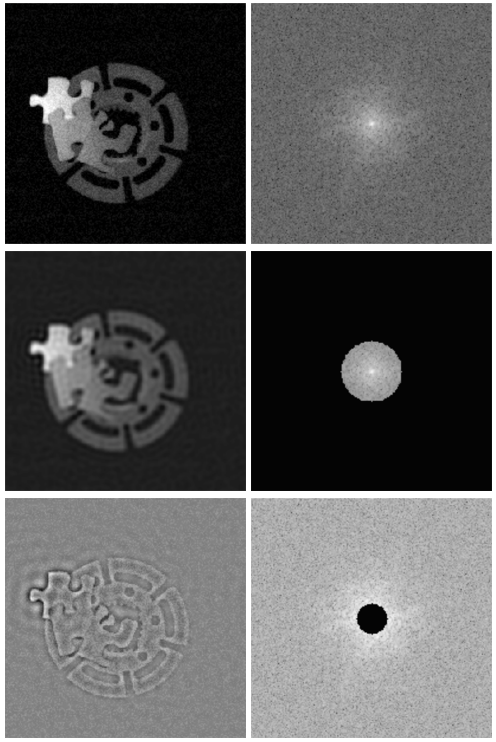
In practice, different types of interest points are often combined.

Preprocessing and Enhancement

Image enhancement usually takes place before features are extracted and often takes the form of an image normalization, where one makes sure that the same levels of contrast and the same range of intensity levels result, even if this is not the case in the input images (due to changing illumination). There are 3 types of image enhancement: *noise suppresion*, *image de-blurring*, and *contrast enhancement*.

Fourier Power Spectra of Images

Usually, nearby pixels in an image have similar gray levels (intensity). This correlation suggests that the power spectra of images mainly contains low frequencies. As is seen in the example below, most of the power is clustered around the DC component ($u = v = 0$ or $k = l = 0$).



However, the object boundaries – which often obtain the essential information about the object shape – are contained in the higher frequencies, meaning they can't just be discarded. The problem is, that the signal to noise ratio(SNR) at higher frequencies is often rather low, meaning that the edge information is submerged in noise. Also, when isolating the lower frequencies, we can see that ripples appear in the second image. This occurs because multiplying in the frequency domain means convolution in the spatial domain, and the inverse Fourier of the disk is the sinc function which has ripples.

Noise Suppression

There are many kinds of noise (white, Gaussian, salt and pepper etc.) and algorithms usually have been optimized to work for one specific type. We only consider 2 general options: *low pass convolution filters*, and *non-linear edge preserving filters* (median filtering and anisotropic diffusion).

Low Pass (Linear, Shift–Invariant) Convolution Filters

The importance of linear, shift–invariant filters lies in a combination of a deep theoretical insight in their effect and the fact that they can be implemented rather efficiently. Consider the following filter usage: A mask, usually square and of small dimensions (e.g. 3×3 or 5×5) is shifted over the image. At each position the mask parameters are multiplied with the grey level of the pixel they cover. The sum of these products is assigned to the central pixel as its new grey level. Large masks, which would be computationally expensive, can often be implemented more efficiently in the Fourier domain.

The goal of low-pass filters is to remove the noise part of the spectrum by multiplying the Fourier domain by a mask, but such spectrum filters yield rippling as previously explained.

Averaging Masks

In order to generate low-pass filters that do not cause rippling, we try to model convolutional filters in the spatial domain to approximate low-pass filtering in the frequency domain. For example, averaging masks are a subset of low–pass convolution filters. At the higher frequency part of the spectrum the signal to noise ratio is typically low and noise can be reduced by suppressing the higher frequencies. Considering linear and shift–invariant filters, such low–pass filtering or "smoothing" can be obtained by calculating local averages of the grey levels, for example by the following 3×3 or 5×5 masks:

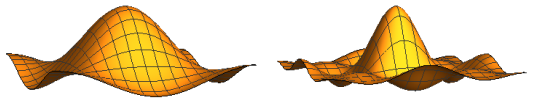
$$\frac{1}{9} \cdot \begin{array}{|c|c|c|}\hline 1&1&1\\\hline 1&1&1\\\hline 1&1&1\\\hline\end{array} \qquad \frac{1}{25} \cdot \begin{array}{|c|c|c|c|c|}\hline 1&1&1&1&1\\\hline 1&1&1&1&1\\\hline 1&1&1&1&1\\\hline 1&1&1&1&1\\\hline 1&1&1&1&1\\\hline\end{array}$$

Although high frequencies have been suppressed more than low frequencies, hence the blurred effect, these filters are not really low–pass: Both low and high frequencies survive. This becomes clear when looking at their Fourier transforms:

$$\mathcal{F}(\text{avg}_{3\times3}) = \frac{1}{9}\left(1 + 2\cos(2\pi u)\right)\left(1 + 2\cos(2\pi v)\right)$$

$$\mathcal{F}(\text{avg}_{5\times5}) = \frac{1}{25}\left(1 + 2\cos(2\pi u) + 2\cos(4\pi u)\right) \cdot \left(1 + 2\cos(2\pi v) + 2\cos(4\pi v)\right).$$

Note that the transform of the 3×3 mask can be found by the convolution of a 3×1 mask and a 1×3 mask and hence the Fourier tranfrom of the resulting 3×3 mask is simply the product of the Fourier transform of these two masks. Analoguous procedure leads to the 5×5 mask.

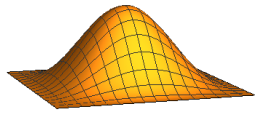


Note that the first cut–off frequency is lower for the larger filter. However, the filters cannot be made to better approximate a purely low–pass filter by making them wider as it also adds more lobes and leads to the sinc filter in the limit. As was seen earlier, the sign reversals result in phase shifts by π and cause the often unwanted "rippling effect". In order to avoid this effect from occurring, it might be better to us a filter without sign reversals in the spatial domain.

Binomial / Gaussian Filters

Binomial filters are a subset of low–pass convolution filters which do not result in sign reversals. Since in image analysis filters with odd dimensions are preferred, the following mask is often used

$$\frac{1}{16} \cdot \begin{array}{|c|c|c|}\hline 1&2&1\\\hline 2&4&2\\\hline 1&2&1\\\hline\end{array}$$



The modulation transfer function of this filter is

$$\mathcal{F}(\text{bin}_{3\times3}) = (2 + 2\cos(2\pi u))(2 + 2\cos(2\pi v)),$$

which is a genuine low–pass filter. Selecting larger binomial filters will render the resulting filters more isotropic. The coefficient (i, j) of an $m \times n$ binomial filter is chosen according to

$$b_{ij} = \frac{1}{2^{n+m-2}} \binom{m-1}{i} \binom{n-1}{j}.$$

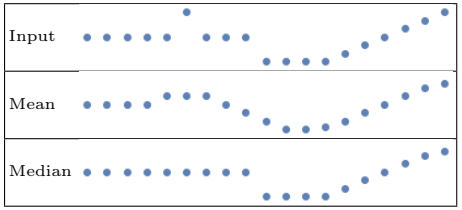
In the limit, the filters will approach a Gaussian. The Fourier transform of a Gaussian is a Gaussian and consequently zero crossings occur neither in the spatial nor in the spatial frequency domain. The above 3×3 filter can be considered an approximation to the Gaussian with minimal spatial extent.

However, no matter how sophisticated, linear filters can never resolve the problem of edge information and noise being intertwined, as they cannot disentangle the contributions from edges and noise at a given frequency. Their effect on these contributions will be identical, i.e., a multiplication with the corresponding gain factor specified by their modulation transfer function.

Non-linear (edge preserving) filters

Median Filters

Median filtering is based on the rank–order of the neighbourhood values of the pixel under scrutiny. The pixel's grey level is replaced by the value, such that there are as many higher as lower values in the rank order. Thus, the new grey level is always already present in the image and no new gray levels emerge. The advantage of this type of filer is its "odd-man-out" effect.



Median filters are non–linear , discard spikes, preserve discontinuities, and produce best results for salt–and–pepper noise; while Gaussian filter is superior for gaussian or uniform noise.

Combined Deblurring and Noise Suppression

The problem with unsharp masking is that it boosts edges and noise alike, since it's a linear method. Similarly, diffusion – approximated by averaging masks or binomial filters – successfully suppresses noise but also blurs edges. The idea is then to devise a process that behaves as usual diffusion on homogeneous intensity areas, and thus smoothes out the noise, whereas it turns into a backward diffusion at places with outspoken edges.

Anisotropic Diffusion Method

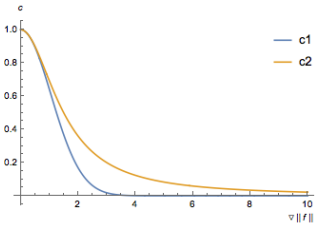
The envisaged effect of combined deblurring and noise suppression is achieved by making the diffusion coefficient in the diffusion equation anisotropic (different properties in different directions), resulting in

$$\frac{\partial f}{\partial t} = \text{div}\left(c(f)\nabla f\right).$$

This operation is similar to diffusion, but now the gray level of a pixel is modified by a weighted sum of the gray level differences with the sourrounding pixels. Possible functions for $c(f)$ are

$$c1(f) = \exp\left(-\left(\frac{||\nabla f||}{K}\right)^2\right) \quad \text{and} \quad c2(f) = \frac{1}{1 + \left(\frac{||\nabla f||}{K}\right)^2}$$

with some constant K. It can be shown, that by using the above suggestions and rewriting the equation so it fits the diffusion equation, the diffusion coefficient is positive when $||\nabla f||$ is small (forward diffusion, smoothing) and negative when $||\nabla f||$ is large (sharpening). The gray level of the central pixel is modified with an amount which is calculated as a weighted sum of the local differences in gray levels.



$$c_1 \cdot \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 1&-1&0\\\hline 0&0&0\\\hline\end{array} + c_2 \cdot \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&-1&1\\\hline 0&0&0\\\hline\end{array} + c_3 \cdot \begin{array}{|c|c|c|}\hline 0&1&0\\\hline 0&-1&0\\\hline 0&0&0\\\hline\end{array} + c_4 \cdot \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&-1&0\\\hline 0&1&0\\\hline\end{array}$$

However, discretization effects and "gaps" in the edges surrounding regions cause the contrast over the edges to diminish gradually until they are no longer visible. As a consequence, the process should be halted at a stage where an acceptable compromise between edge sharpness and contrast is established. This evolution towards a homogeneous solution can be halted by adding a restraining force, which keeps the solution close to the initial image :

$$\frac{\partial f}{\partial t} = \operatorname{div}\left(c(f)\nabla f\right) - \frac{1}{\sigma^2}(f - f_0)$$

Image Deblurring

Unsharp Masking

In contrast to histogram equalization, unsharp masking is a local method, linear and image independent. It enhances local contrast and can effectively undo Gaussian blur. Moreover, it is simple to implement. Suppose a sharp image $f_o(x, y)$ was blurred by a Gaussian filter, resulting in the blurred image $f_b(x, y)$. The process of gradual Gaussian blurring, governed by the so–called heat diffusion equation

$$\frac{\partial f}{\partial t} = c\left(\nabla^2 f\right)$$

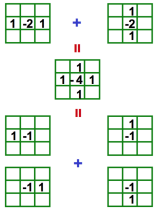
with the diffusion coeffitient $c \in \mathbb{R}_+$ and the Laplacian operator ∇^2 defined as

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

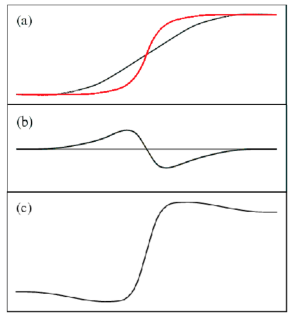
will start at $t = 0$ with f_o as initial condition. After some time Δt, the diffusion process will have reached exactly the degree of blur observed in f_b. To regain the original image, one can use the following approximation

$$f_o = f_b - (c\Delta t)\,\nabla^2 f_b,$$

where in practice, $c\Delta t$ is replaced by an appropriate constant. Note that this unsharp masking technique can be interpreted as a backward diffusion process. The finite difference approximation of the Laplace operator ∇^2 shows that the gray level f is changed by a portion of the average gray level difference with the surrounding pixels.



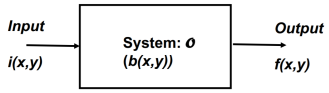
The Laplacian can therefore be approximated as the difference between the smoothed image and its original version. For a 1D example:



(a) Input image in red, smoothed version in black.
(b) Difference between the smoothed version and the input, as an approximation of the Laplacian.
(c) Result of the input image minus the Laplacian of the image. The edge becomes steeper giving a sharpened impression, which is further accentuated by the under- and overshoots.

Inverse Filtering

For deblurring, we assume that b and f are known, with b being the blurring filter, and we are trying to find i.



The effects of a linear blurring process with modulation transfer function $B(u, v)$ of the blurring filter would be undone if the image could be passed through a linear filter with a modulation transfer function $B'(u, v)$ such that

$$B(u, v)B'(u, v) = 1.$$

However, frequencies that have been completely suppressed, i.e., for which $B(u, v) = 0$, cannot be recovered. Noise – especially if it was introduced after the blurring – is another serious problem, because the signal to noise ratio can be small in some areas, and it may lead to spurious high frequencies. As an intuitive approach, consider an inverse filter with modulation transfer function

$$B'(u, v) = \frac{1}{B(u, v) + C}.$$

Note that as long $B(u, v) \gg C$, this MTF closely approximates the ideal MTF $1/B(u,v)$. For small values of $B(u, v)$ it shows a saturation behaviour, i.e., it will tend to $1/C$ for $B(u, v) \to 0$.

This inverse filter still lets frequencies pass that have totally been filtered out by $B(u, v)$, i.e., frequencies with $B(u, v) = 0$. In that case, pure noise is added. This effect can be eliminated by the following choice:

$$B'(u, v) = \frac{B(u, v)}{B^2(u, v) + C}$$

with C a constant. As long as $B(u, v)$ is sufficiently large to suppress the influence of C we have $B'(u, v) \approx 1/B(u,v)$. Once $B(u, v)$ gets too small however, then $B'(u, v)$ also decays to zero.

The Wiener Filter

The Wiener filter is a theoretically optimal deblurring technique. It directly builds on the inverse filtering ideas but requires additional knowledge about the image statistics in exchange for this optimality. Suppose the sum of a signal $b(x, y)$ and noise $n(x, y)$ is given and the goal is to recover the signal $b(x, y)$. The measure of how well the recovery is, can be chosen as the integral of the square of difference between the output $o(x, y)$ and the desired signal $d(x, y)$, where $d(x, y)$ is usually just $b(x, y)$. Therefore, one has to minimize

$$E = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (o(x, y) - d(x, y))^2 \,\mathrm{d}x\,\mathrm{d}y.$$

If we use a linear system for the filtering operation, we can characterize it by means of its point spread function $h(x, y)$. Under the assumption of white noise, it can be shown that the optimal $h(x, y)$ in the sense of minimizing the squared error above has a Fourier transform of

$$H = \frac{\Phi_{bb}}{\Phi_{bb} + \Phi_{nn}} = \frac{1}{1 + \frac{\Phi_{nn}}{\Phi_{bb}}}.$$

Here Φ_{bb} and Φ_{nn} are the Fourier transforms of the autocorrelations of the original signal $b(x, y)$ and the noise $n(x, y)$ respectively and have to be estimated. In words: When the signal to noise ratio Φ_{bb}/Φ_{nn} is high, the gain is almost 1, whereas when the noise dominates, the gain is approximately the signal to noise ratio, i.e., very low. If the signal $b(x, y)$ is passed through a system with point spread function $h'(x, y)$ before the noise $n(x, y)$ is added, the result

$$i = b * h' + n$$

is to be filtered by a system with point spread function $h(x, y)$ with Fourier transform

$$H = \frac{H'\Phi_{bb}}{H'^2\Phi_{bb} + \Phi_{nn}}$$

in order to minimize the squared error.

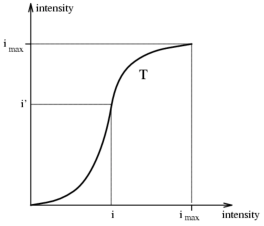
The limits of the Wiener filter are that if the SNR is low, the filter tends to become low-pass blurring instead of sharpening; also $H(u,v)$ must be known very precisely

Contrast Enhancement

When an image is over– or underexposed, the dynamic range (range of possible intensities) of the camera is not effectively used.

Histogram Equalization

Histogram equalization is a global method for better utilizing the range of possible intensities. For an inappropriately exposed picture, or for a picture where a large and interesting part contains intensities that are close to each other, the majority of pixels will be found in a relatively small portion of the intensity range $I = [0, i_{\max}]$. Thus the histogram will have a peaked outlook. The idea of histogram equalization is to produce a new histogram which is flatter. Histogram equalization carries out one of many possible maps from I onto itself. Pixels are given new intensities according to some specific transition. The goal now is to find such a map which will produce a flat histogram. As histograms can also be interpreted as empirical probability densities for pixel intensity, the search for a flattening map can be simplified by looking at the cumulative probability distributions



$$C(i) = \int_0^i p(v)\,\mathrm{d}v.$$

$C(i)$ maps the interval I on the unit interval, such that $C(0) = 0$ and $C(i_{\max}) = 1$. For a flat histogram, (i.e., a probability density function which is constant), the cumulative probability distribution will be a straight line between these points. An image with a flat histogram therefore ought to have a cumulative probability distribution value of $1/i_{\max}$ for each graylevel i. The actual distribution will deviate from this target distribution in that a different proportion of pixels is found in the grey level intervals $[0, i]$. This difference can be eliminated by mapping a grey level i in the original image to a new grey level i' such that

$$\frac{i'}{i_{\max}} = \int_0^i p(v)\,\mathrm{d}v,$$

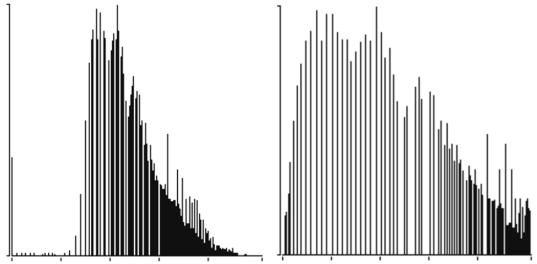
and therefore

$$i' = i_{\max} C(i).$$

The gray level transformation forces the resulting histogram to be flat. Indeed, the new probability density function is found as

$$p' = p\frac{\mathrm{d}i}{\mathrm{d}i'} = p\frac{1}{pi_{\max}} = \frac{1}{i_{\max}},$$

which is the desired constant value. This means that grey levels with many pixels packed together, i.e., a high slope of the cumulative probability distribution, are expanded and intervals with few pixels are compressed.



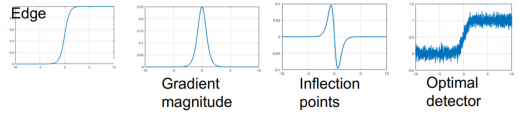
As can be seen, the histogram is not really flat after the transformation. This is due to the discretization effects because discontinuous jumps in the discretised cumulative probability distribution lead to discrete jumps in the equalization map. As a result some intensities are skipped.

Basic Feature Detection

Edges correspond to places in the image with an elongated discontinuity in intensity or colour. If the discontinuity has a high curvature there's a 'corner'. Lines are two edges that are close to each other and more or less parallel.

Edge Pixels

Ideally, a gray level edge is a border between two regions, each of which has approximately uniform gray levels. Taking a cross section of the image along a line at right angles to an edge, the resulting gray level profile will often only approximate a step discontinuity. Also, some edge transitions are better modeled as step transitions in the first derivative of brightness, rather than in the brightness itself. Such edges are called *ramp edges* or *roof edges*. The edge–enhanced images must be processed further to extract one–pixel thick, connected curves, which often is the hardest part.



Gradient Operators

Whatever the edge detection operator, it ought to be isotropic. A first approach to the problem is to look at the local change of intensity and to situate edges where this change is sufficiently large. The gradient magnitude is obtained as

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

Whereas the partial derivatives themselves are clearly not rotation invariant, the magnitude of the gradient is easily shown to be invariant using the rotated coordinates (x', y')

$$x = x' \cos\theta + y' \sin\theta$$
$$y = x' \sin\theta + y' \cos\theta.$$

Differentiation with respect to the rotated coordinates yields

$$\frac{\partial f}{\partial x'} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial x'} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial x'} = \frac{\partial f}{\partial x}\cos\theta + \frac{\partial f}{\partial y}\sin\theta$$
$$\frac{\partial f}{\partial y'} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial y'} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial y'} = -\frac{\partial f}{\partial x}\sin\theta + \frac{\partial f}{\partial y}\cos\theta.$$

The direction for which the change in intensity is steepest is given by the θ that extremizes $\partial f/\partial x'$. Simple differentiation with respect to θ yields

$$\theta_{\text{extr}} = \arctan\left(\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}\right) + k\pi$$

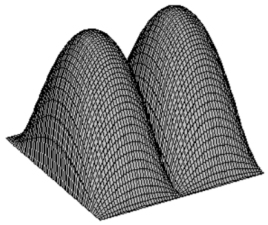
with k an arbitrary integer. The edges are then perpendicular to θ_{extr}. The gradient is obtained by a non–linear operation. The partial differentiations $\partial f/\partial x$ and $\partial f/\partial y$, however, correspond to linear and shift invariant operations and can therefore be written as simple convolutions.

$\frac{\partial}{\partial x}$ [-1 1] $\qquad$ $\frac{\partial}{\partial y}$ [-1 / 1]

However, we cannot apply them directly because they are prone to noise, and we want something that will smooth and compute gradients; often 3×3 masks are used. One example thereof are the "Sobel masks":

$$\text{horizontal: } G_x = \begin{array}{|c|c|c|}\hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline\end{array} * f \qquad \text{vertical: } G_y = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline\end{array} * f$$

The mask at the left yields high output where vertical edges can be found, the one at the right will yield high outputs for horizontal edges. We can indeed see that the MTF of the left one works a band pass in the *u-direction* and a low pass in the *v-direction*.



Their outputs are then combined according to the formulas above, taking the square root of the sum of their squares and then taking the *arctan* of their ratio in order to obtain edge orientation. In principle, their outputs should be divided by 8 to estimate the partial derivatives. In practice only their relative outputs is of interest, however. Note that these masks are separable. Consider:

$$\text{horizontal: } G_x = \begin{array}{|c|}\hline -1 \\ \hline 0 \\ \hline 1 \\ \hline\end{array} \cdot \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline\end{array} * f = \begin{array}{|c|c|c|}\hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline\end{array} * f,$$

$$\text{vertical: } G_y = \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 1 \\ \hline\end{array} \cdot \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline\end{array} * f = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline\end{array} * f,$$

meaning one can first calculate the finite difference with a $(-1, 0, 1)$ or $(-1, 0, 1)^\top$ mask and smooth the results by a $(1, 2, 1)^\top$ or $(1, 2, 1)$ binomial mask in a second step. Sobel masks are the optimal 3×3 filters with integer coefficients for the detection of step edges. They will yield acceptable approximations to both edge contrast and orientation.

Zero–Crossings

One can also consider edges to lie on the inflection points of the intensity function $f(x, y)$. Such points are found at the zero–crossings of the Laplacian:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0.$$

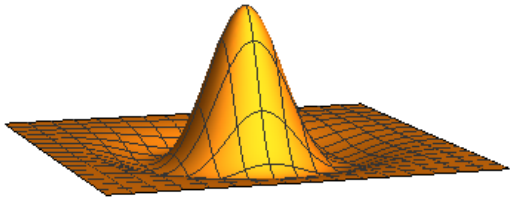
The Laplacian is a linear and shift invariant operator and can therefore be implemented as a simple convolution. Two masks are suggested:

$$\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline\end{array} \qquad \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & -12 & 2 \\ \hline 1 & 2 & 1 \\ \hline\end{array}$$

The mask on the left is based on finite difference approximations of the second order derivatives, the one on the right is another approximation which is more isotropic. Because this operator is based on second–order derivatives, it is highly sensitive to noise. Therefore, the operator is usually combined with a smoothing operator, e.g., a Gaussian filter. If G represents the Gaussian convolution filter, and L the Laplacian convolution filter, then

$$L * (G * f) = (L * G) * f.$$

This means one can combine both operations into a single convolution with a (usually large) mask $L * G$, which is often referred to as the "Mexican Hat"– or "Laplacian of Gaussian (LoG)"–filter.



Moreover, the LoG filter can be well approximated by a difference of a narrow Gaussian and a wide Gaussian with the same volume underneath. Such an approximation is referred to as a "Difference of Gaussian (DoG)"–filter. The edges found this way often do not really correspond to what a human observer would regard as an edge. Zero–crossings are found where neighbouring pixels have opposite signs for the output. As can be seen, this operation leads to one–pixel thick edges, as opposed to the results obtained with the gradient masks.

The appendix includes more detailed information on the differences between the gradient operators and the zero–crossing techniques.

Matched Filter Theorem

The *matched filter theorem* states that the best convolution mask (in this context often called a template) to find a certain reference pattern (edge, corner, flat region) in a signal, will often be that reference pattern itself. The "often" indicates that certain conditions have to be fulfilled.

The Canny Edge Detector

The process of Canny edge detection can be broken down in five seperate steps:

1. Apply a Gaussian filter to smooth the image;
2. Find the intensity gradients of the image;
3. Apply non–maximum suppression to get rid of spurious response to edge detection;
4. Apply double threshold to determine potential edges;
5. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

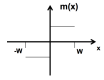
Canny's Optimality Terms

Let $\mathbf{h}(\cdot)$ be the response of the system $h(x)$ to input $(\cdot)$. Canny proposed three optimality criteria to judge a filter's performance:

1. **Good Detection:** Choose the filter that maximizes the signal–to–noise ratio (SNR) of the response at the edge location, subject to the constraint that the response to a constant signal is zero. The SNR is defined as

$$\text{SNR} = \frac{|\text{response to signal at edge}|}{\text{standard deviation of response to noise at edge}} = \frac{\int_{-W}^{W} h(-\hat{x})m(\hat{x})\,\mathrm{d}\hat{x}}{\sigma\sqrt{\int_{-W}^{W} h^2(\hat{x})\,\mathrm{d}\hat{x}}}.$$

Here, $h(x)$ is the filter of support radius W to a deterministic signal model $m(x)$ (step function) with edge at $x = 0$. The denominator is the expected value of the filter response given white noise.



2. **Good Localization:** Suppose the maximum of the system response to $m(x)+n(x)$, i.e. the maximum of $\mathbf{h}(m+n)$, where $n(x)$ is white noise, is at x_0. Since $n(x)$ is of stochastic nature, the localization is quantified through the expected value:

$$\text{LOK} = \frac{1}{\sqrt{\mathbb{E}[x_0^2]}} = \frac{\left|\int_{-W}^{W} h'(\hat{x})m'(-\hat{x})\,\mathrm{d}\hat{x}\right|}{\sigma\sqrt{\int_{-W}^{W}[h'(\hat{x})]^2\,\mathrm{d}\hat{x}}}$$

The idea is now to maximize both qualities, i.e., SNR · LOK. According to the *matched filter theorem*, this is achieved with a filter that is a scaled version of the signal to be detected, i.e., a step function. Unfortunately, the noise results in many local maxima, hindering the unique detection of edges. Therefore a third criteria is introduced:

3. **Sparse Peaks:** The average number of maxima within the filter support $[-W, W]$ can be shown to be (Rice theorem)

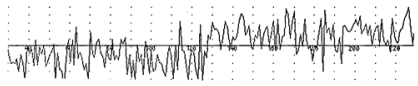
$$N_{\max} = \frac{2W}{x_{\max}} = \frac{W}{x_{\text{ave}}}.$$

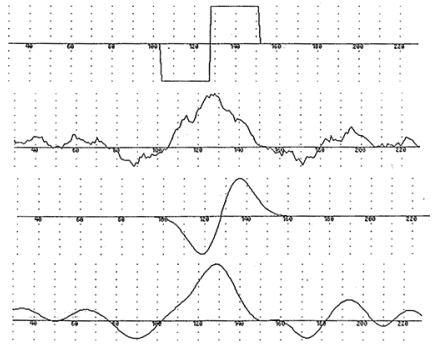
Here, x_{ave} is

$$x_{\text{ave}} = \pi\sqrt{\frac{-\Phi_{ff}(0)}{\Phi''_{ff}(0)}}$$

with $f = \boldsymbol{h}'(n)$ and Φ_{ff} the Fourier transform of the cross correlation of f.

Ultimately, the goal now is to find a filter $h(x)$ that maximizes SNR · LOK subject to $x_{\text{ave}} \leq \varepsilon$. It can be shown that the solution to this optimization problem is very close to a Gaussian derivative for large ε.





Step 1: Gaussian Filtering

Since the gradients of the image intensity are sensitive to noise, it is reasonable to smooth the image before calculating the image gradients. Any smoothing filter can be applied here; note, however, that the size of the filter kernel plays a key role in the trade off between noise suppression and edge localization.

Step 2: Intensity Gradient

Any gradient operator (Sobel, Prewitt) returns an approximation for the first derivative in horizontal and vertical direction, i.e., G_x and G_y respectively. From this, the edge gradient G and its direction θ can be determined by

$$G = \sqrt{G_x^2 + G_y^2} \qquad \theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Subsequently, the edge direction is rounded to one of the following four angles:

$$0^\circ, \qquad 45^\circ, \qquad 90^\circ, \qquad 135^\circ.$$

This is done as preparation for the non–maximum suppession step.

Step 3: Non–Maximum Suppression

Non–maximum suppression is an edge thinning technique. After the intensity gradient, the edge is still rather blurry, which is not desireable. The algorithm for each pixel *in the gradient image* is:

1. Compare the edge strength G of the pixel under scrutiny with the edge strengths in the positive and negative gradient directions.
2. If the edge strength of the pixel under scrutiny is the largest compared to the other pixels in the mask with the same direction, the value is preserved, otherwise it is suppressed.

Note that the gradient is perpendicular to the edge direction. Therefore as an example, if the rounded gradient is 0°, i.e., the edge is in north–south direction, the pixel will only be considered an edge–pixel if its gradient is greater than the gradient of the immediate pixels to the east and west.

Step 4: Double Threshold

At this point, most edge pixels will represent actual edges in the image. However, noise and color variation might still cause unwanted edge detection. To counteract such responses, two thresholds are set to identify the different types of edge pixels:

- **High Threshold:** If the pixel's gradient magnitude is larger than the high threshold, the pixel is marked as a strong edge–pixel.
- **Low Threshold:** If the pixel's gradient magnitude is smaller than the low threshold, it is suppressed.
- **In Between:** If the pixel's gradient matnitude is between the high and low thresholds, the pixel is marked as a weak edge–pixel.

The two threshold values are empirically determined.

Step 5: Edge Tracking

Strong edge pixels are now considered to represent true edges whereas the weak edge pixels could stem from true edges or noise/color variation. To distinguish between the two cases, a weak edge–pixel is only kept, if it is (possibly through other weak edge–pixels) connected to a strong edge–pixel.

Harris Operator

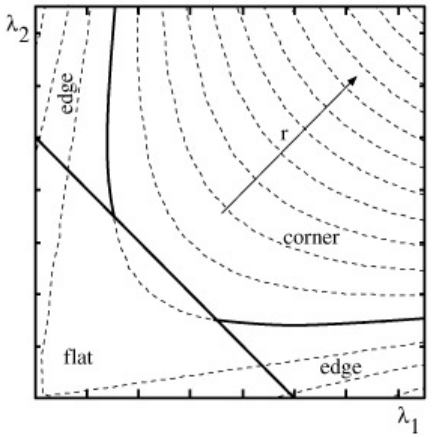
One might not only wish to find edges, but also identify corners and flat regions. Flat regions show little variations in intensity, edges correspond to intensity profiles with major changes in one direction and small changes in the orthogonal direction and corner regions have large changes in two orthogonal directions. In a mathematical approach, the environment of the pixel under scrutiny is examined with the help of the following averages of the second order moments:

$$\begin{pmatrix} \widehat{\left(\frac{\partial f}{\partial x}\right)^2} & \widehat{\frac{\partial f}{\partial x}\frac{\partial f}{\partial y}} \\ \widehat{\frac{\partial f}{\partial x}\frac{\partial f}{\partial y}} & \widehat{\left(\frac{\partial f}{\partial y}\right)^2} \end{pmatrix}$$

The direction of the largest changes in the intensity is calculated as the eigenvector of this matrix with the largest corresponding eigenvalue (PCA). Similarly, the other eigenvector, with the smallest eigenvalue, yields the direction of least changes in the intensity, which is orthogonal to the first. Again, the eigenvalue for this direction quantifies the amount of change observed in that direction. Flat regions have two small eigenvalues, edges have one large and one small eigenvalue and corners have two large eigenvalues. A possible criterion therefore is to take the determinant of the 2nd–order moment matrix, which corresponds to the product of the two eigenvalues. Its value will mostly be large if both eigenvalues are large. However, if one eigenvalue is very large, this criterion will still propose a corner, even if the other eigenvalue is rather small. One way to circumvent this is to use iso–lines for

$$\lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

as a measure. This allows to distinguish flat regions, edges and corners quite robustly.



Typically, the next step is to look for local extrema of the operator's output, in order to localize the corners and edges with better precision.

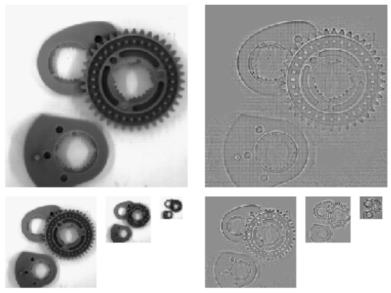
Image Decomposition

As the Fourier transform has already shown, images need not be decomposed as pixels with their individual intensities, but can also be interpreted as a linear combination of sine patterns. The Fourier transform, however, is just one way of describing images as a combination of patterns, and this chapter will introduce several new approaches.

Scale Space / Image Pyramids

To illustrate the concept, suppose you are flying at high altitude over a forest. It will look like a homogeneously textured region. Flying lower, you will distinguish the individual trees, and even lower the individual leaves To summarize, scenes contain information at different levels of detail, and we wish to develop hierarchical descriptions and increase efficiency by working on lower resolutions. The process is, for an image I_i:

1. Smoothing I_i with Gaussian will result in S_i
2. Take the difference image: $L_i = I_i - S_i$. This difference of Gaussians (DoG) can be approximated to the Laplacian of a Gaussian (LoG). The difference between two images smoothed (i.e low pass filtered) to a different degree yields a band-passed image (where the edges can be easily identified).
3. Given sufficient smoothing (following the Nyquist theorem, reduce the size of the smoothed image: I_{i+1} = down-sample(S_i)



We usually find corners in down-scaled images and then precise their location with the original image. However, since we are working with discrete approximations of the images and the levels in scale space, we will have to build discrete approximations of the Gaussian filter. This implies, e.g, if we apply a small smoothing filter with positive coefficients c_{-1}, c_0, c_1, then $c_0^2 = 4c_{-1}c_1$ has to hold. So the binomial filter [1,2,1] is a valid scale space filter but [1,1,1] is not.

Unitary Transforms for Compression

A unitary transform is a transform that preserves the inner product, i.e $U^*U = UU^* = I$, which is only possible for real functions iff the columns of U are orthonormal(i.e the inner product of all components with self=1, others=0). Unitary is for complex functions what orthogonality is for real functions. This section discusses unitary transforms for image crompression. Suppose one can write an image as a linear combination of basis images (similarly to what the Fourier transform achieves with eigenfunctions). If one could truncate this decomposition after a small number of basis images while still having a good approximation of the image and only keep the weights for the remaining basis images, this could constitute a good compression. Before, we had seen:

1. **Pixelwise decomposition**: 1 Dirac impulse at the corresponding pixel in each basis image (perfect localization in space domain, none in frequency).
2. **Fourier decomposition**: 1 oriented cosine/sine pattern in each basis image (perfect localization in frequency domain, none in space).

All these basis images were orthogonal, i.e., their inner product (correlation) is zero. Going from the original Dirac-based decomposition to a linear combination of unitary basis images amounts to describing the image in terms of one set of orthogonal basis vectors to another such basis. This means that the unitary transform essentially applies a rotation to the data. In compression applications, the hope is that after this rotation the projection of the image (a vector) onto many of the new axes vanishes. The unitary transforms also compromise localization in space/frequency.

Inner Product for Images

The inner product for two $N \times N$ basis images i_1 and i_2 is defined as

$$\langle i_1, i_2 \rangle = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} i_1(x,y)i_2^*(x,y)$$

with $(\cdot)^*$ denoting the complex conjugate. This is a generalization of the orthogonality concept for vectors.

Basis Images

The orthogonality of functions as discussed in the appendix (**see appendix**) is confined to the one–dimensional case. Orthogonal function sets of higher dimensions can be easily obtained from there, however. Suppose the images are of size $M \times N$ and one has a one–dimensional set of M orthogonal functions $\phi_i(x)$ and another of N such functions $\psi_j(x)$. A set of MN orthogonal basis functions B_{ij} can then be constructed as

$$B_{ij}(x,y) = \phi_i(x)\psi_j(y) \qquad \Longleftrightarrow \qquad B_{ij} = \phi_i\psi_j^\top.$$

These B_{ij} are called basis images. Note that there are plenty of basis images that are not separable into products of one–dimensional basis functions. When it is possible, however, the basis images are said to be *separable*.

Decomposition of Images

Let $f(x,y)$ be an image that is to be decomposed in terms of the basis images $B_{uv}(x,y)$. Since the basis images $B_{uv}(x,y)$ are orthonormal, finding the corresponding weights w_{uv} such that

$$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} w_{uv}B_{uv}(x,y)$$

is an easy task. In order to calculate a particular weight $w_{u'v'}$, we can multiply both sides of the equation above with $B^*_{u'v'}(x,y)$ and take the sum over the whole domain:

$$\begin{aligned}
&\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)B^*_{u'v'}(x,y) \\
&= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} w_{uv}B_{uv}(x,y)B^*_{u'v'}(x,y) \\
&= \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} w_{uv}\underbrace{\left(\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} B_{uv}(x,y)B^*_{u'v'}(x,y)\right)}_{\delta_{u'v'}} \\
&= w_{u'v'}.
\end{aligned}$$

The last equality holds since the inner product of the two basis images is zero for all $u \neq u'$, $v \neq v'$ and one otherwise. One interpretation of this operation is as a correlation of the given image f with the basis images, making it clear that the weight factors w_{uv} will be large for basis images that strongly resemble the given image and smaller otherwise.

Transformation

In the context of basis images, one can define the transformation F of image f in terms of the weights for the respective basis images:

$$F(u,v) = w_{uv}.$$

Note that if f lives in $M \times N$, so does F.

Forward Transformation

The forward transformation of f is given by

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)B^*_{uv}(x,y).$$

Backward Transformation

The backward transformation, backtransformation or inverse transformation is given by

$$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} \overbrace{F(u,v)}^{w_{uv}}B_{uv}(x,y).$$

Truncation Property of Unitary Image transforms

The forward transform yields the weight w_{uv} to be used for decomposition. If data compression is the goal, one has not gained much, since there are still MN such weight. This section deals with the selection of weights to keep if not all are retained. Suppose the truncated decomposition

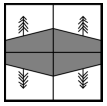
$$\hat{f} = \sum_{u=0}^{M'-1}\sum_{v=0}^{N'-1} c_{uv}B_{uv}(x,y)$$

with $M' < M$ and $N' < N$ wich minimizes the sum of the errors squared is to be found. The least–square approximation $\hat{f}(x,y)$ of $f(x,y)$ should therefore minimize

$$e_{M'N'} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left|f(x,y) - \hat{f}(x,y)\right|^2.$$

As it turns out, one should just stick to the weights w_{uv} of the original decomposition and truncate below a certain threshold. Indeed, if we wish to use a smaller amount of bases, we should not change the weights to compensate because the bases are orthogonal.

The Cosine Transform



The Fourier transform of an even, real function consists of cosine terms only. Extending the image as shown to the right not only eliminates false high frequencies from the DFT, but also makes the image even. The forward discrete cosine transform (DCT) F of this extended, even $2N \times 2N$ image f is given by

$$\begin{aligned}
F(u,v) = \frac{4C(u)C(v)}{N^2}&\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y) \\
&\cdot \cos\left(\frac{\pi}{N}u(x+\frac{1}{2})\right)\cos\left(\frac{\pi}{N}v(y+\frac{1}{2})\right)
\end{aligned}$$

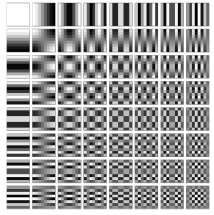
where

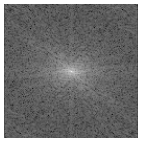
$$C(w) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } w = 0 \\ 1 & \text{for } w = 1,2,\ldots,N-1. \end{cases}$$

The backward DCT is defined as

$$\begin{aligned}
f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1}&C(u)C(v)F(u,v) \\
&\cdot \cos\left(\frac{\pi}{N}u(x+\frac{1}{2})\right)\cos\left(\frac{\pi}{N}v(y+\frac{1}{2})\right)
\end{aligned}$$

Just like the FFT, the DCT has a $\mathcal{O}(N\log_2 N)$ fast version of implementation compared to the $\mathcal{O}(N^2)$ of the straight forward implementation. Whereas the DFT will always contain high frequencies from the boundary discontinuities, such high frequencies do not occur in the DCT. As a result the DCT will be more strongly concentrated at low frequencies and the resulting compression efficiency is higher. Its different components are, on average, more decorrelated than those obtained with other unitary image independent transforms. Moreover, the DCT only requires real computations.



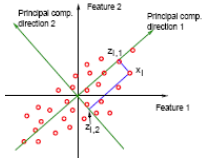


The center picture shows the FFT of the left image and the right picture shows its DCT. Other types of Unitary transforms include:

1. **Slant transform**: discrete sawtooth-like basis vectors which efficiently represent linear brightness variations along an image line
2. **Haar Transform** example of a wavelet transform, with an analysis both in space and frequency; which gets smaller in space for higher frequencies
3. **Hadamard Transform** only 1s and -1s, so no multiplication needed; generates minimally correlated binary blocks

Principle Component Analysis (PCA/KLT)

If two variables are significantly correlated, knowing one yields pretty precise information on the value of the other. Instead of keeping track of two variables, one could store only one and thus save storage cost. However, just keeping the value of one of the variables is suboptimal, since one can take advantage of the correlation by a simple rotation of the coordinate axes.



Typically, the PCA will be applied to higher–dimensional cases. In the case of image compression, the PCA is sometimes called the Karhunen-Loève Transform (KLT) and the dimension of the space equals the number of pixels in the images.

Principle Component Analysis of Images

Generally, in a set of p–tuples, the p components have substantial correlations among them. As described above, one can do a p–dimensional rotation, which is a unitary transformation, to capture most of the variance in the data in a few coordinates. Rotation preserves the overall variance in the data, i.e., for original coordinates x_i and rotated coordinates z_i:

$$\sum_{i=1}^{p} \sigma_i^2 = \sum_{i=1}^{p} \tilde{\sigma}_j^2$$

with σ_i the variance in x_i and $\tilde{\sigma}_i$ the variance in z_i. This should come as no surprise, as rotation maps the centers of gravity onto centers of gravity. The PCA algorithm allows a rank–ordering of new coordinates z_i such that the first coordinate z_1 captures the largest variance, the second the next largest and so on. These coordinates are called the first, respectively second, *principle components* (PCs). In other words: Principle Component Analysis identifies the optimal coordinate rotation to obtain complete decorrelation and thus often allows a drastic reduction in the dimensionality of the problem while retaining most of the variation, i.e., information, in the data. In the case of image analysis, $N \times N$ images are represented by $N^2 \times 1$ vectors where each element represents the intensity at a certain location. Therefore one should consider the variances of pixel values and covariances for pixel pairs over the family of images to be considered. This text assumes that the necessary statistics can be retrieved.

The PCA–Algorithm for Images

Suppose that x is a column vector of p random variables. The PCA–Algorithm can be broken down in the following steps:

1. Look for a linear combination $c_1^\top x$ of the components of x which has maximum variance.
2. Look for a linear combination $c_2^\top x$, uncorrelated with $c_1^\top x$, which has maximum variance.
3. Continue looking for the linear combination $c_i^\top x$ that is uncorrelated with all previous linear combinations and has maximum variance.

This will yield p linear combinations with the property that $c_k^\top x$ is the kth PC. In order to find $c_1^\top x$ consider the following setup. Let C be the covariance matrix of x, which means that

$$\mathrm{Var}\left(c_1^\top x\right) = c_1^\top C c_1 .$$

As it stands, the maximum will not be achieved for finite c_1, so a normalization constraint $c_1^\top c_1 = 1$ is introduced. Using Lagrange multipliers, one now has to maximize

$$c_1^\top C c_1 - \lambda(c_1^\top c_1 - 1).$$

Differentiation with respect to the components of c_1 yields

$$(C - \lambda \mathbb{I}_{p\times p})c_1 = 0.$$

Thus, λ is an eigenvalue of C and c_1 is the corresponding eigenvector. Since we want to maximize the variance, we should chose the largest eigenvalue:

$$c_1^\top C c_1 = c_1^\top \lambda c_1 = \lambda c_1^\top c_1 = \lambda.$$

In general, the kth PC $c_k^\top x$ has a variance $\mathrm{Var}\left(c_k^\top x\right) = \lambda_k$ corresponding to the kth largest eigenvalue of the covariance matrix C.

KLT: PCA–Based Image Compression

The steps to use PCA for the sake of image compression are as follows.

1. Consider the image as large vector of pixel intensities;
2. Calculate the covariance statistics over the set of images considered;
3. Look for the eigenvalues and eigenvectors of the covariance matrix. The eigenvectors can be interpreted as so–called *eigenimages* or in more general terms *basisimages*.

Combining only the eigenimages with the largest eigenvalues will in general yield an acceptable approximation to the original image. However, there are three important problems with this approach:

1. **Computational Problems** Applying PCA on $N \times N$ images by interpreting them as $N^2 \times 1$ vetors of pixel intensities leads to the calculation and analysis of $N^2 \times N^2$ covariance matrices.
2. **Specifying Image Statistics** The availability of the necessary statistics implies that these can be calculated over a set of representative samples. In many applications, however, image content can take all sorts of forms.
3. **Image Dependence** The resulting transformation will only be finetuned towards use for the set which was specified as the set of representative samples. This means that the eigenimages for the set also have to be stored or transmitted which is not in the sense of image compression.

The first problem can be alleviated by considering that the number of images used to extract the statistics will typically be much smaller than $p = N^2$ (the dimensions of the "image space"). As soon as the number of samples $n < p$ is smaller than p, the complete $p \times p$ covariance matrix is singular and cannot have rank higher than n. This means that there exists a shortcut in the derivation of the first n principal components: Consider the $(p \times n)$ data matrix X that contains the n sample vectors, which are assumed to be normalized towards zero–mean. The $(p \times p)$ sample covariance matrix is given by $C = 1/nXX^\top$. However, one can find the meaningful, first n eigenvectors of C as well as their eigenvalues without actually having to deal with its huge dimensions by introducing the $(n \times n)$ matrix $S = X^\top X$:

$$X^\top X c_i = \lambda_i c_i$$
$$XX^\top X c_i = X\lambda_i c_i$$
$$\left(\frac{1}{n} XX^\top\right)(Xc_i) = \frac{\lambda_i}{n}(Xc_i).$$

This implies that Xc_i is an eigenvector with eigenvalue λ_i/n of C if c_i is an eigenvector with eigenvalue λ_i of S.

Independent Component Analysis (ICA)

Suppose we have n signals/images i, which are linear combinations of n underlying signals/images u such that

$$i = Au \qquad (1)$$

ICA tries to extract the u_i. It is not a unitary transformation (i.e not a rotation), but it is a general linear transformation. The algorithm is based on the assumption that the underlying signals u are statistically independent (and not just decorrelated as with PCA).
An example for the ICA would be an image of a shop window in which one has the superposition of a reflection in the window and what's behind. If two images were taken under different eliminations, we could separate the reflection from what's behind.

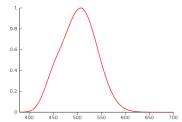
Perception

Brightness (Luminance)

Brightness addresses the perceived intensity of light and is not a measure for the actual power in the incoming light. Radiometric units are therefore not apt for specifying observed brightness.

Relative Luminous Efficiency

The relative luminous efficiency of monochromatic light with wavelength λ is defined as the ratio $c(555\,\mathrm{nm})/c'(\lambda)$ where $c(\cdot)$ denotes the spectral radiant flux (power) required to achieve a certain brightness. Since the human eye is most sensitive to light with a wavelength of 555 nm (yellow–green light), this fraction is always between 0 and 1.



Link radiometry-photometry

Photometry is the subjective impression (expressed in lumens), while *radiometry* is the objective, physical measurement (expressed in watts). For light with spectral composition $c(\lambda)$ we have :

$$l = k \int_{\lambda=0}^{\infty} (\lambda)\nu(\lambda) d\lambda \qquad (2)$$

Monochromatic Light

Light that consists of only one spectral component (wavelength) is called monochromatic light.

Relative Luminous Efficiency Function

The relative luminous efficiency as a function of λ is denoted by $v(\lambda)$. Two monochromatic lights with $c_1(\lambda_1)$ and $c_2(\lambda_2)$ appear equally bright to an observer when

$$c_1(\lambda_1)v(\lambda_1) = c_2(\lambda_2)v(\lambda_2)$$

Color

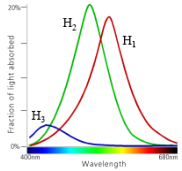
Human observers can distinguish many more colors than they can intensity levels (shades of grey).

Representation of Color

The perceptual dimensions of color are *luminance (brightness)*, *hue* and *saturation* or *chroma*. The hue of a color refers to the aspect that we usually refer to as "the color", i.e., the overall color type such as red, green, etc. Saturation defines how much white is added to any monochromatic light.

More quantitatively, it has been shown, that an arbitrary color can be generated with an appropriate mixture of three primary colors. This does not mean that a combination of primary colors can create any monochromatic light, but rather, that the mixture can not be distinguished from the monochromatic light *by a human observer*. The human retina contains three different types of cones with distinct spectral sensitivity – the corresponding response curves are denoted $H_1(\lambda)$, $H_2(\lambda)$ and $H_3(\lambda)$. A source with spectral radiant flux $C(\lambda)$ will produce responses R_i, $i = 1, 2, 3$, for the three cones as the integrated response over the spectral radiant flux:

$$R_i(C) = \int H_i(\lambda)C(\lambda)\,\mathrm{d}\lambda, \quad i = 1, 2, 3.$$



The perception of color, however, can not be reduced to the outputs of the three cone types as the brain is interpreting the signals in ways that are not yet completely understood.

Contrast

When the impression of brightness is tested with respect to a background, *relative luminance* turns out to be crucial.

Surface Features

The Representation of Color

Generating Colors from Primaries

In the view of the three–cone theory, the technology of color displaying is normally based on three different types of sources. These three light sources are *primary sources* or *primaries*. In general, they are given, rather than being free to choose. First, one needs to know their spectral radiant flux $P_j(\lambda), j = 1, 2, 3$. The primary sources recommended by the CIE are three monochromatic sources with wavelengths $\lambda_1 = 700\,\mathrm{nm}$, $\lambda_2 = 546.1\,\mathrm{nm}$ and $\lambda_3 = 435.8\,\mathrm{nm}$, corresponding to the colors *red*, *green*, and *blue*. Given a source with spectral composition $C(\lambda)$, the question is what proportion, i.e., what amount $m_j, j = 1, 2, 3$, of primaries result in the same perception. The human cone responses R_i to the source $C(\lambda)$ should therefore be the same as the responses to the mixture.

$$R_i(C) = \begin{cases} \int H_i(\lambda)C(\lambda)\,\mathrm{d}\lambda & \text{for the source} \\ \sum_{j=1}^{3} m_j \int H_i(\lambda)P_j(\lambda)\,\mathrm{d}\lambda & \text{for the primaries.} \end{cases}$$

Here, the H_is are the cone sensitivities as explained in *Perception*. Using the shorthand notation

$$l_{i,j} = \int H_i(\lambda)P_j(\lambda)\,\mathrm{d}\lambda,$$

one can see that these integrals can be determined as soon as the radiant fluxes P_j of all three primaries have been chosen. For the monochromatic CIE primaries, this simplifies to $l_{i,j} = H_i(\lambda_j)$. Knowing the human responses R_i to the source, one can obtain a linear system of equations to be solved for the m_j:

$$\sum_{j=1}^{3} m_j l_{i,j} = R_i.$$

These equations specify the linear transformation between the use of the human "primaries", i.e., the spectral characteristics of the cones, and the given set of technological primaries. Notice that solving this system implies inverting the matrix of the $l_{i,j}$. This will be possible as long as the primaries are independent with respect to human vision, i.e., none of them can be produced as a linear combination of the other two.

Tristimulus and Chromaticity Coordinates

Usually the amount w_j to produce a standard white source is used as reference when specifying relative values with respect to the amounts m_j of the primaries needed to create a certain perception. These *tristimulus values* for the source $C(\lambda)$ are then $T_j = m_j/w_j$. By definition, the CIE standard white source W has tristimulus values $w_1 = w_2 = w_3 = 1$. In order to achieve this, the relative power $P_1 : P_2 : P_3$ of the primaries R, G, B has to be in the proportion 72.1 : 1.4 : 1.0 (cf. the spectral sensitivity curves, where the area below the blue is clearly much less than for the other primaries). Since white is defined to have a flat spectrum $R_i(W) = \int H_i(\lambda)\,\mathrm{d}\lambda$, and thus

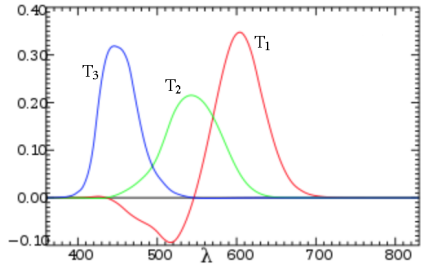
$$\sum_{j=1}^{3} w_j P_j H_i(\lambda) = \int H_i(\lambda)\,\mathrm{d}\lambda = R_i(W).$$

The *spectral matching curves* $T_j(\lambda')$ give the tristimulus values for monochromatic sources $C_{\lambda'}$ with wavelength λ':

$$R_i(C_{\lambda'}) = H_i(\lambda') = \sum_{j=1}^{3} w_j l_{i,j} T_j(\lambda').$$

These are very useful, since knowing the spectral matching curves $T_j(\lambda')$, the tristimulus values for an arbitrary source $C(\lambda)$ are found as

$$T_j(C) = \int C(\lambda)T_j(\lambda)\,\mathrm{d}\lambda.$$



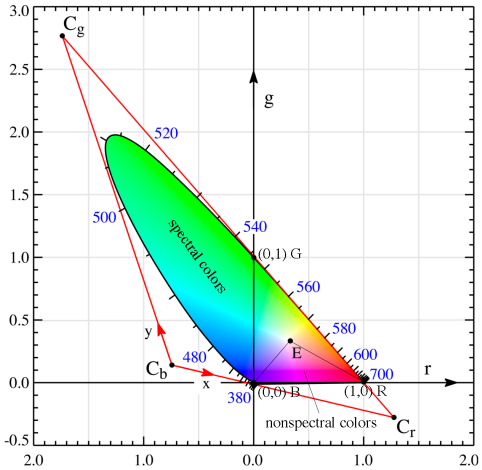
Negative values indicate colors that cannot be reproduced with the CIE primaries. The corresponding values indicate how much of the corresponding primary should be added *to the given light source* to produce the combination of the other two primaries with positive values. Not being able to produce some colors is by no means a specific drawback of the CIE choice of primaries. In fact, there will always be colors that cannot be produced, no mather the choice of primaries. The tristimulus representation of color still contains brightness information. Multiplying the spectral composition of the color $C(\lambda)$ by some factor, the tristimulus values will simply scale accordingly. These values therefore still don't capture pure chrominance information. This can be remedied by normalizing the tristimulus values, resulting in the *chromaticity coordinates*:

$$t_j = \frac{T_j}{T_1 + T_2 + T_3}.$$

It follows immediately that $t_1 + t_2 + t_3 = 1$. This linear dependence allows the elimination of one coordinate, leading to a two–dimensional color space (e.g $t_3 = f(t_1, t_2)$. One can think of this space as a horizontal section through a squeezed version of the color solid. A pair of chromaticity coordinates specifies saturation and hue, but doesn't contain information about the luminance. The chromaticity coordinates (r, g) corresponding to the CIE primaries are calculated on the CIE tristimulus values R, G, B:

$$r = \frac{R}{R+G+B} \qquad g = \frac{G}{R+G+B}.$$

This results in the *r–g–color space*:



Note that due to the negative portion of the $r(\lambda) = t_1(\lambda)$ curve, negative values along the spectral lotus are encountered. The spectrum locus is the contour of the color space, showing the most satured colors. The line connecting the extremes of the locus (Wavelengths 380 nm to 700 nm) is called the *line of purples*. Colors in the triangle formed by the reference white E and the line of purples are *nonspectral colors*. All colors visible to the human eye are encompassed by the spectrum locus and the line of purples. Colors inside this region but outside the triangle of nonspectral colors are called *spectral colors*. Colors that can be produced as a mixture of the primaries are confined to the triangle $R - G - B$. Other colors have at least one negative chromaticity coordinate and can therefore not be produced. This suggests choosing primaries that minimize the area outside of their triangle. Notice however, that this criteria is flawed as the area does not correspond to the amount of distinguishable colors; The scale on the spectrum locus is not linearly spaced.

x, y–Coordinates

In order to get rid of the negative values with the CIE primaries, a virtual or normalized tristimulus colour system X, Y, Z of purely positive coordinates has been defined. The corresponding primaries are purely hypothetical, they can in fact not be physically realized. Nevertheless, it is a convenient coordinate system, since all the colors have corresponding chromaticity coordinates between 0 and 1. The linear transformation from the R, G, B to the X, Y, Z coordinates is given by

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.490 & 0.310 & 0.200 \\ 0.177 & 0.813 & 0.011 \\ 0.000 & 0.010 & 0.990 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}.$$

The transformation has been chosen as to make Y represent luminance. The coordinates of the reference white remain invariant: $R = G = B = 1$ is mapped to $X = Y = Z = 1$. The chrominance coordinates x, y are found as

$$x = \frac{X}{X+Y+Z} \qquad y = \frac{Y}{X+Y+Z}.$$

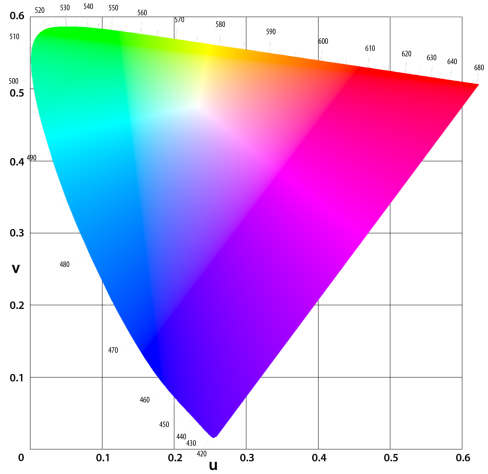
Note that changing the primaries corresponds to considering a new triangle in the chromaticity diagram, with the source colors as its vertices. Two sides of such a triangle are to be used as the basis of a new coordinate frame.

u, v–Coordinates

In order to represent the perceptual distance more truthfully, the following transformation has been introduced:

$$u = \frac{4x}{-2x + 12y + 3} \qquad v = \frac{6y}{-2x + 12y + 3}.$$

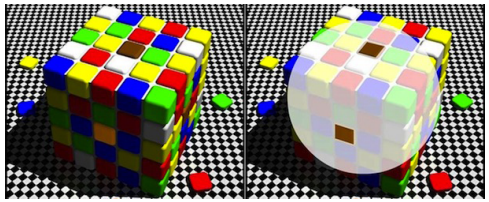
This results in the following color space:



So, colour coordinates need for their definition 3 primaries + white (4 points) in order to define a projective frame (to map 4 points to 4 different points with a different choice of primaries and white).

Color Constancy

Unfortunately, the response for the three color bands are not only a function of the reflectance characteristics of a surface, but also of the spectral composition of illumination. The former is an inherent property of the object's surface and therefore useful for tasks like object recognition or tracking. The latter, however, may change between views and interferes with any attempt to gain knowledge about the surface itself. Human color perception is far more intricate than the collection of the three color cone responses R_i. The color constancy phenomenon amounts to objects keeping approximately the same perceived color under changing illumination, even though the reflected – and ultimately received – light changes it's spectral composition.



The human visual system succeeds in transforming the triplet of cone reponses into a percept that is almost independent of illumination. In other words, it can guess the color of the light source as well as the color of the object just looking at the reflected color, which is a product of the two.

Illumination Invariant Color Features

Extracting the true surface color under varying illumination – as the human visual system can – is very difficult for computers. A less ambitious goal is to extract color features that do not change with illumination. There are three different types of such features:

- **Spectral Changes:** Sometimes also called "internal changes". Consider the irradiance for the three spectral bands I_R, I_G and I_B. A change in the spectral signature of the sources will change the irradiance by three pixel–independent factors, say α, β, and γ, i.e., $(I'_R, I'_G, I'_B) = (\alpha I_R, \beta I_G, \gamma I_B)$. It follows that for the irradiance of two different points 1 and 2 in the scene:

$$\frac{I'_{R1}}{I'_{R2}} = \frac{\alpha I_{R1}}{\alpha I_{R2}} = \frac{I_{R1}}{I_{R2}}$$

The ratio of corresponding irradiances for the R–band at two different points is therefore not affected by the change, and thus invariant. This holds for the other two bands as well. A similar result can be obtained for non–linear responses

$$\frac{(\alpha I_{R1})^\gamma}{(\alpha I_{R2})^\gamma} = \frac{(I_{R1})^\gamma}{(I_{R2})^\gamma},$$

as well as for human–like log responses

$$\log(I'_{R1}) - \log(I'_{R2}) = \log\left(\frac{I'_{R1}}{I'_{R2}}\right) = \log\left(\frac{I_{R1}}{I_{R2}}\right) = \log(I_{R1}) - \log(I_{R2}).$$

- **Geometric Changes:** In case of a geometric change of the illumination as, e.g., the light comes form a different direction, the three irradiances change with the same factor, say s, but this time the factor changes over the scene: $(I'_R, I'_G, I'_B) = (s(x,y)I_R, s(x,y)I_G, s(x,y)I_B)$. Once more ratios can be used as features that remain unchanged, however, in this case ratios are formed from different irradiances at the same point:

$$\frac{I'_R}{I'_G} = \frac{s(x,y)I_R}{s(x,y)I_G} = \frac{I_R}{I_G}$$
$$\frac{I'_R}{I'_B} = \frac{s(x,y)I_R}{s(x,y)I_B} = \frac{I_R}{I_B}$$

Again the response of the camera would have to be subtracted rather than divided if a log–responsitivity profile applies.

- **Spectral and Geometric Changes:** Consider a combination of the above cases. The following still holds:

$$\frac{I'_{R1}I'_{G2}}{I'_{R2}I'_{G1}} = \frac{\alpha s(x_1,y_1) I_{R1} \beta s(x_2,y_2) I_{G2}}{\alpha s(x_2,y_2) I_{R2} \beta s(x_1,y_1) I_{G1}} = \frac{I_{R1}I_{G2}}{I_{R2}I_{G1}}.$$

For log–response systems this translates to

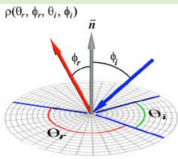
$$\log\left(I'_{R1}\right) + \log\left(I'_{G2}\right) - \log\left(I'_{R2}\right) - \log\left(I'_{G1}\right)$$
$$= \log\left(I_{R1}\right) + \log\left(I_{G2}\right) - \log\left(I_{R2}\right) - \log\left(I_{G1}\right).$$

Another way to deal with spectral and geometric changes is to focus on information at the two sides of edges. It is assumed that the edge is due to a discontinuity in surface reflectance and not to a discontinuity in surface normal. In that case, points on both sides of the edge share approximately the same factors $s(x_1, y_1) \approx s(x_2, y_2)$. Consequently, one can again use the simpler invariants

$$\frac{I'_{R1}}{I'_{R2}} = \frac{I_{R1}}{I_{R2}}$$

BRDF

BRDF is the Bidirectional Reflection Distribution Function, for different wavelengths. It is a 4D function, specifying the radiance for and outgoing direction, given an irradiance for an incoming direction, relative to the normal and ideally for 1 wavelength at the time. It is thus hard to find the radiance.



Surface Texture

Texture analysis typically probes for three attributes:

- **Regularity** Discerns stochastic and regular patterns.

- **Orientation** Texture direction can be a feature of value.

- **Coarseness** The coarseness of a pattern holds information as well.

Fourier Features

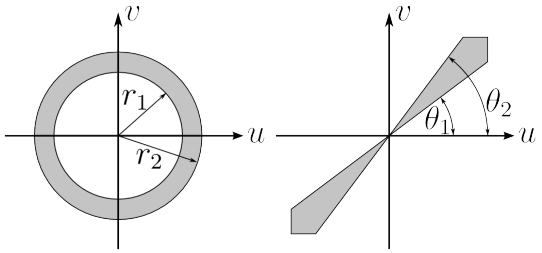
- **Regularity** If a texture is highly regular, it can be expected to have dominant peaks in its power spectrum such that the detection of regularity amounts to the identification of those peaks.
- **Orientation** The power spectrum integrated over sectors provides information about the directionality of the texture.

$$\theta_1 \leq \arctan\left(\frac{u}{v}\right) \leq \theta_2$$

- **Coarseness** The power spectrum integrated over rings centric around the origin (zero frequency) gives an indica-

tion of the coarseness of the texture. A texture is coarse if the power is concentrated in the inner rings and fine if its mainly found in the outer rings.

$$r_1^2 \leq u^2 + v^2 \leq r_2^2$$



The problem with Fourier features is that they collect information over the image globally. Any sort of spatial information is lost and segmentation or inspection is therefore not possible.

Cooccurrence Matrices

A *cooccurrence matrix* is a similar data structure to a histogram, but it preserves information on some aspects of the spatial configuration. Suppose a two dimensional vector is shifted over the image. For each position of its tail, there is a corresponding pixel at its head. These two pixels (tail and head), define ordered pairs of image points. The cooccurrence matrix C is then simply formed as follows: Let the column coordinate of the matrix correspond to the intensity of the tail pixel and use the intensity at the head pixel as the row of C. Suppose the intensities of tail and head pixels are i and j, respectively. One simply increments the element $C(i, j)$ each time one finds a pixel pair with the corresponding intensities for the tail and head pixel. Dividing all matrix elements by the total number of such pixel pairs then yields a probability estimation for the cooccurrence of the corresponding intensity values (i, j) at points with the specified relative position. Obviously one can build several such matrices, each for a different choice of vector. One can also use several vectors at once and put the results in a single matrix. If, e.g., vectors of the same length but different orientations are applied, the resulting cooccurrence matrix is immune to changes in the orientation of the texture. Vectors of texture–length yield diagonally dominated cooccurrence matrices. The vector length can be used to determine the coarseness of the texture. Typically, only a number of features are extracted from such matrices, as keeping the whole matrices is not efficient. The following is a sample of the so–called *Haralick features*:

Feature	Expression
Energy	$\sum_i \sum_j C^2(i,j)$
Entropy	$-\sum_i \sum_j C(i,j) \log C(i,j)$
Contrast	$\sum_i \sum_j (i-j)^2 C(i,j)$
Homogeneity	$\sum_i \sum_j C(i,j)/(1+\lvert i-j\rvert)$
Max. Probability	$\max_{i,j} C(i,j)$

Energy is high if the coocurrence matrix has strong peaks. Entropy reaches its highest values if all entries are equal. Maximum probability yields an indication of the strongest entry. Contrast and homogeneity are more specifically oriented towards the assessment of whether the entries are concentrated near the diagonal, in which case contrast is small and homogeneity is large.

Filter Banks

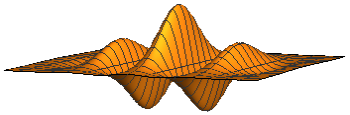
A simple method for extracting texture information is to convolve the image with a series of convolution filters. The outputs of such a "filter bank" are then combined into a feature vector and subsequently used for classification.

- **Laws filters** The *Laws filters* are a series of 3×3, 5×5, and 7×7 filters, created by the convolution of row and column filters with specified coefficients. Usually the output of those filters is non–linearly transformed and then averaged over larger regions in order to get an 'energy'. The energies are then combined into a feature vector that can be used for segmentation or classification.

- **Gabor Filters** These filters sample the local spatial frequency content of the image with optimal resolution. They are constructed by modulating a Gaussian with a cosine function. For a vertically oriented Gabor filter, this yields:

$$g(x, y) = \mathrm{e}^{-\frac{x^2+y^2}{4\Delta_{x,y}^2}} \cos\left(2\pi u^* x + \varphi\right).$$

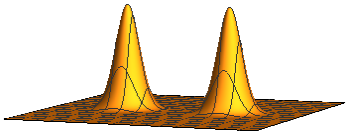
The parameter $\Delta_{x,y}$ determines the width of the Gaussian envelope, u^* specifies the frequency of the modulating cosine, while φ is its phase.



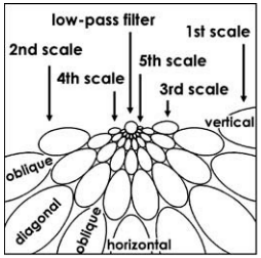
A Gabor filter samples the Fourier domain, as can be seen from its Fourier transform.

$$G(u, v) = \frac{1}{4\pi\Delta_{u,v}^2}\left(\mathrm{e}^{-\frac{(u-u^*)^2+v^2}{4\Delta_{u,v}^2}} + \mathrm{e}^{-\frac{(u+u^*)^2+v^2}{4\Delta_{u,v}^2}}\right)$$

where $\Delta_{u,v} = 1/4\pi\Delta_{x,y}$. Since the Gabor filter is a product of a Gaussian and a cosine, its Fourier transform is known to be the convolution of a Gaussian and a pair of Dirac impulses.

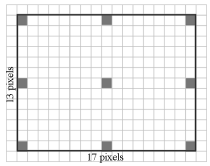


The two Gaussians are positioned oppositely at $(u = u^*, v = 0)$ and $(u = -u^*, v = 0)$. As features the outputs of a bank of Gabor filters are used, with different scales (Δ) and different orientations, allowing to probe for directionality and different scales.



Again, these outputs are non–linearly transformed and averaged over some neighbourhood. Note that $\Delta_{x,y}\Delta_{u,v} = 1/4\pi$. The *uncertainty principle* states that it is not possible to get better localization in both the spatial and the frequency domain, i.e., the Gabor filters are optimal with respect to simultaneous localization in both domains.

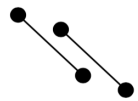
- **Eigenfilters** The Laws and Gabor filter banks have fixed coefficients. If a specific type of texture should be analysed, it can be benefitial to finetune the filter set for that type. Suppose one takes a 3×3 neighbourhood mask and orders the pixels in top–left to bottom–right fashion in a vector. When shifting the mask over the whole image many samples of this nine–dimensional vector are obtained and one can calculate the corresponding covariance matrix. The principal components yield a basis of mutually orthogonal vectors with uncorrelated output, the eigenvectors. Arranging the nine components of such an eigenvector in the original 3×3 mask structure yields one filter for every eigenvector (principle component), i.e. a so–called eigenfilter. A mask with these values will have the largest swinfs when moving over the image. The filters coming with the highest eigenvalues reflect the structure of the texture on which the system was trained. Instead of using masks of fixed size, one can also adapt the mask size to the period of the texture. In order to get the period, autocorrelation is usually applied. As large convolution masks are not very practical, sparse variations can be considered. For a horizontal period of 17 pixels and a vertical period of 13 pixels, consider the following mask



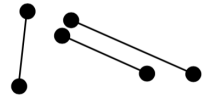
where gray shaded pixels indicate non–zero values. After applying the eigenfilters to the image, simple thresholding is used to identify flawed textures.

Stochastic Models

Similarly to the cooccurrence matrix method, this approach looks at pixel pairs. Every different type of pair is referred to as a clique type.



Cliques of same type



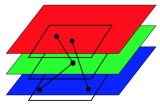
Cliques of different type

Instead of increasing a matrix counter as in the cooccurrence matrix method, the signal difference Δ for the pixels of each type is measured subsequently and plotted in a histogram. The number of samples with a specific difference Δ is called the appearance frequency $f(\Delta)$ of said difference. This can also be used for texture synthesis. For all types t of cliques, the histogram distance to the reference texture is calculated as

$$d_t = \sqrt{\sum_{\Delta} \left(f_t(\Delta) - f_t^0(\Delta)\right)^2} \quad \forall\, t.$$

The clique types can be selected according to the following algorithm:

0. Collect reference histograms for all clique types t (restriction on maximal clique length).
1. Collect histograms for the current synthesized texture (initially white noise).
2. Select the clique type with maximal distance d_t.
3. If maximal distance $<$ threshold, stop.
4. Add clique type to the texture model.
5. Synthesize texture based on current model.
6. Go to step 1.



Note that clique types can be defined as spanning over color channels.

Segmentation

Segmentation is a chicken–and–egg problem: How is one to decide which pixels form an entity without knowing what one is looking at? Yet, in order to identify an object, a reasonable segmentation must have preceeded.

Thresholding

Thresholding amounts to somehow determining a critical intensity level and assigning pixels with intensities on either side of this level to an "objects" and "background" class respectively. Once an image has been thresholded, problems like edge detection and the measurement of simple shape characteristics become almost trivial. Thresholding provides serious bandwidth reduction and simplification for further processing. However, thresholding will generally not provide a satisfactory segmentation of images and loses structural information due to pixel-by-pixel decision.

Bimodal Histograms

A first technique applies to bimodal histograms. In order to try to distinguish between object and background, one could select the threshold as the minimum between the peaks.

Relative Area Method

In some applications the relative area of the object is given. If for example the object is known to occupy 40% of an image and is supposed to be brighter than the background, one could simply label the brightest 40% of the image as object and the rest as background.

Otsu's Criterion

Otsu's criterion selects the threshold that divides the image into two maximally homogeneous pixel populations. Each choice for the threshold separates the histogram into two populations of pixels, each with their overall probability p_1 and p_2 and variance σ_1^2 and σ_2^2. The threshold that minimizes the weighted variance

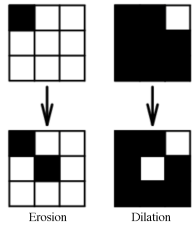
$$p_1\sigma_1^2 + p_2\sigma_2^2$$

is selected.

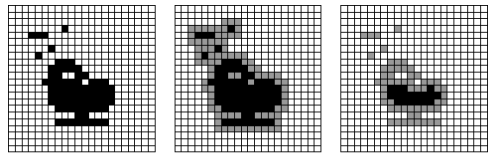
Mathematical Morphology

Even in high–contrast cases, some pixels can fall on the wrong side of the threshold. We thus wish to enhance binary images. When such errors occur isolated or in small groups, they can be corrected by a combination of *erosion* and *dilation*. These operations are based on pixel neighborhood defined by structural elements. If the image is defined as A and the binary structural element as B, we have:

$$\text{Dilation: } A \oplus B = x|B_x \cap A \neq \emptyset \text{Erosion: } A \ominus B = x|B_x \subseteq A \tag{3}$$



In the graphic above, white and black represent object and background respectively. Often dilation and erosion are applied subsequently using the same structural element, as dilation not only gets rid of erroneously classified background pixels but also deforms the object and introduces false object pixels. The subsequent erosion reverses the deformation to an extent, without reintroducing the false background pixels. Erosion + dilatation is known as *opening* and mostly removes islands. Dilatation + erosion is known as *closing* and mostly removes holes.



Note that contrary to a convolution, erosion and dilation are *not linear*; and median filtering is very useful and commonly used as edge preserving smoothing.

Multiple Objects: Connected Components

When multiple objects appear on an image, we would like to separate them. In order to do so, we proceed with a connected component analysis. Discreticizing the image space, we evaluate the neighborhood on a Cartesian image raster. A component is said to be *connected* if all its pixel pairs are connected through a chain of pixels all within the same component (we usually used 4 or 8 neigbors).

One method is the single-pass connected component labeling. At every pixel its neighborhood is divided into past and future. If no label is found for this pixel, start a new label; if 1

label is found copy it; if > 1 label is found note their equivalence. At the end, co-label equivalent components (connected but initially labeled as different).

There are different ways to define the distance between pixels (e.g euclidean distance), which we can use to establish "distance maps" based on distance propagation along neighborhoods.

Edge Based Segmentation

When thresholding is not enough, edges can help. Indeed another intuitive approach is to segment an image after having done an edge analysis. Once one has closed, one–pixel thick edges, segmentation is trivial. However, quite often the boundaries of interest are fragmented and we have a set of "cluttered edges".

The Hough Transform

The first edge linking technique is the Hough transform, which uses the symmetry of shapes to extract them in lower dimensional spaces (a straight line remains invariant when shifted along its direction, a circle is rotation–invariant).

Straight Line Detection

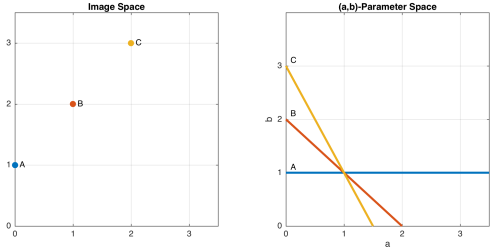
Suppose one would like to detect straight lines in all possible positions and orientations. In general, at least three dimensions have to be considered: Two components of translation and one of rotation. Straight lines however remain invariant under a one–dimensional subgroup of the translations. In fact, the image projection of a straight line is fully characterized by two parameters, whatever the transformation might be, as long as it preserves colinearity (endpoint positions are not taken into account). The equation for a straight line is

$$y = ax + b,$$

and it's clear to see that two real numbers a and b fully characterize the line. On the other hand, fixing a point (x, y) all (a, b) pairs of lines containing the point are found to be related by a linear relationship as well:

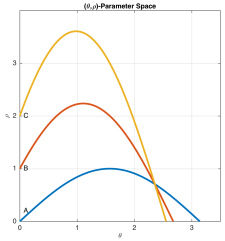
$$b = (-a)x + y.$$

Thus, to detect straight lines, one can scrutinize all points of interests, i.e., points with high probability of lying on an object outline. For each such point we draw the above line in (a, b)–parameter space.



The lines in parameter space indicate all (a, b) pairs yielding a straight line through one of the points at a time. Where the three lines intersect, the corresponding (a, b) parameter pair results in a straight line through all three of the image points. As this is implemented in a computer, the parameter space will have to be discretized. A finite number of cells are created so that when creating the straight lines a counter will be incremented at each cell where the lines pass, i.e., the cells receive "votes". In order to detect straight lines, one then has to look for cells with high votes, as this corresponds to many lines passing through. The problem with this version of the Hough transform is, that the (a, b)–domain is not bounded. In particular, vertical or near–vertical lines result in $a \to \infty$ and thus cannot be stored. This problem can be circumvented by redefining the parameters to polar coordinates. In that case, a straight line is expressed as

$$x \cos\theta + y \sin\theta = \rho.$$

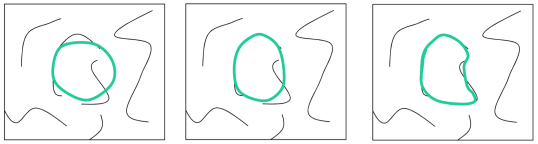


Each point will add a cosine function in the (θ, ρ) parameter space. The values of θ is restricted to the interval $[0, \pi[$ and the magnitude of ρ never exeeds the largest distance an image point can have to the origin.

The Hough transform works when other objects are present, and although it is time consuming, it is robust to noise in the image.

Deformable Contour Models: Snakes

Snakes rely on shape priors, which most often have to be provided by humans. Deformable contour models are in general an interactive approach and seldom completely automated.



Moreover, snakes are of iterative nature. They are initialized near the contour of interest and iteratively refined so as to be near image positions with high gradients and to satisfy shape preferences (e.g. low curvature) or contour priors. The refinement occurs according to a "cost"– or "energy"–function that quantifies how good a fit is. The general snake energy is defined as

$$E_{\text{total}} = E_{\text{internal}} + E_{\text{external}}.$$

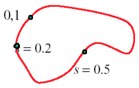
- **Internal Energy:** The internal energy encourages prior shape preferences, e.g., smoothness, elasticity, known prior shape.
- **External Energy:** The external energy stimulates the contour to fit interesting image structures, e.g., edges and corners.

A *good* fit between the current shape and the target shape in the image yields a *low* energy value.

Parametric Curve Representation

In the continuous case, the shape is defined by a normalized parametric curve of the form

$$v(s) = \begin{pmatrix} x(s) \\ y(s) \end{pmatrix} \qquad 0 \le s \le 1.$$



External Energy Definition

External (image) energy can be thought of as gravitational pull towards areas of high image contrast. Once the directional derivatives G_x and G_y have been computed (using e.g. finite differences), the external energy at any point on the snake is defined as

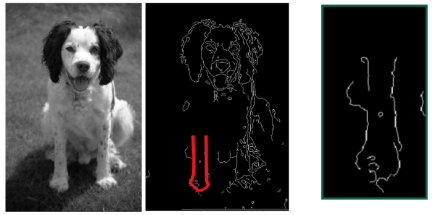
$$E_{\text{external}}(v(s)) = -\left(|G_x(v(s))|^2 + |G_y(v(s))|^2\right).$$

The total energy then equates to

$$E_{\text{external}} = \int_0^1 E_{\text{external}}(v(s))\,\mathrm{d}s.$$

Internal Energy Definition

A priori, one wants to favour smooth shapes, contours with low curvature in order to balance what is actually observed in the gradient image.



For a continuous curve, a common internal energy term is the *deformation energy*. At a specific point $v(s)$ on the curve, it is defined as

$$E_{\text{internal}}(v(s)) = \alpha \left|\frac{\mathrm{d}v}{\mathrm{d}s}\right|^2 + \beta \left|\frac{\mathrm{d}^2 v}{\mathrm{d}s^2}\right|^2$$

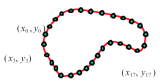
The first summand, weighted by α, mimics tension and penalizes stretching of the snake. The second summand, weighted by β, is termed the curvature and punishes bending. The total internal energy of the snake can be computed as

$$E_{\text{internal}} = \int_0^1 E_{\text{internal}}(v(s))\,\mathrm{d}s.$$

Discretization

Since this should be processed with a computer, one has to discretize the method. For the snake, one usually chooses a fixed number n of equidistant points along the continuous form, resulting in a sequence

$$v_i = (x_i, y_i) \quad i = 0, \ldots, n-1.$$



And the derivatives used for the internal energy amount to

$$\frac{\mathrm{d}v}{\mathrm{d}s} \approx v_{i+1} - v_i,$$

$$\frac{\mathrm{d}^2 v}{\mathrm{d}s^2} \approx (v_{i+1} - v_i) - (v_i - v_{i-1}) = v_{i+1} - 2v_i + v_{i-1}.$$

The integrals for the total energies are replaced by sums and the derivatives by finite differences, leading to

$$E_{\text{external}} \approx -\left(\sum_{i=0}^{n-1} |G_x(x_i, y_i)|^2 + |G_y(x_i, y_i)|^2\right)$$

$$E_{\text{internal}} \approx \alpha \left|v_{i+1} - v_i\right|^2 + \beta \left|v_{i+1} - 2v_i + v_{i-1}\right|^2.$$

Energy Minimization

Several methods exist to fit snakes:

1. **PDEs**: from the snake energy equation, establish a differential equation which we can solve in an iterative scheme using image gradients at curve positions in the previous iteration.
2. **Greedy search**: for each point, place a search grid (e.g 5x5) of discrete positions around it and pick the location where the energy function is minimal. Stop when a predefined number of points have not changed in the last iterations (convergence not guaranteed).
3. **Dynamic programming**: rewrite snake energy as a cost function and munimize using dynamic programming.

Extension of the Energy Model

- **Shape Prior:** If the object is a smooth variation of a known shape $\hat{v}$, one can add a term to the total energy that penalizes the deviation from this shape

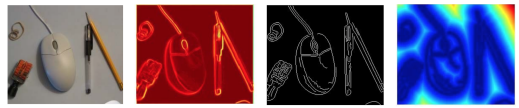
$$\delta \sum_{i=0}^{n-1} (v_i - \hat{v}_i)^2$$

- **Balloon Pressure:** Instead of forcing a curve to shrink, one can push the curve to extend with a force of constant magnitude in the direction of the snake normal.
- **Gravity:** If there is a preferred direction, one can add a constant force to the energy model.

Limitations

- Depending on the parameters α and β the snake may oversmooth the boundary.
- The snake cannot follow topological changes of the object.

- Snakes are only locally optimal. The snake cannot sense object boundaries in the image unless it gets very close to them, since only then are the image gradients large. A remedy for this is instead of using the gradient potential as external energy, one can first identify the edges and subsequently use a distance map to make the snake less short–sighted.

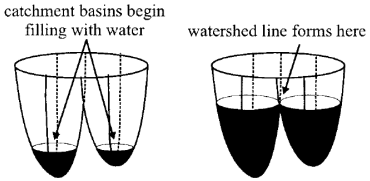


Region Based Segmentation

Instead of edge based segmentation, one can also identify different regions based on homogeneous regions. In order to do so, first very homogeneous regions have to be identified, i.e., regions with low variance that can act as "seeds". These are subsequently grown as long as a certain *homogeneity criteria* is satisfied. The choice of this criteria, however, is not straight forward. The problem is that low–contrast edges often leak, meaning that the region is grown over the edge. Also, region and edge based methods can be combined in *hybrid approaches*.

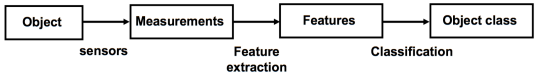
Watershed Algorithm

One possible way to define the homogeneity criteria is through the watershed algorithm. The algorithm starts by identifying local gradient minima. These are the seeds for the subsequent region growing. A region is grown until a local maxima of the gradient magnitude is encountered.



Statistical Pattern Recognition

Statistical pattern recognition follows this scheme:



In general, feature extraction is one of the hardest and most significant steps, as it has a major influence on classification results. An example would be **Object**: Frog, **Sensors**: Cameras, **Measurements**: Pixel Intensity, **Features**/ Color, **Object classes**: Foreground/Background. There are 3 alternatives:

1. Unsupervised clustering
2. Supervised generative modeling
3. Supervised discriminative modeling

Notation

The set of $s + 1$ possible classes is denoted with Ω, where the $+1$ comes from the introduction of a rejection class for when an object can not be classified with a sufficiently large probability

$$\Omega = \{\omega_1, \omega_2, \ldots, \omega_s, \omega_r\}.$$

Each feature is structured in a so–called feature vector

$$\boldsymbol{v} = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n.$$

These vectors span a linear space and are of non–determinisitc nature. They can be characterized by the joint probability function $p(\boldsymbol{v}, \omega_i)$. The a priori probability of a class occurrence is

$$P(\omega_i) = \int p(\boldsymbol{v}, \omega_i)\, \mathrm{d}\boldsymbol{v}.$$

Similarly, the probability of observing a specific feature can be calculated from

$$p(\boldsymbol{v}) = \sum_{i=1}^{s} p(\boldsymbol{v}, \omega_i).$$

Stochastic Characterization

The feature distribution for a class ω_i is given by

$$p(\boldsymbol{v} \mid \omega_i) = \frac{p(\boldsymbol{v}, \omega_i)}{P(\omega_i)},$$

whereas the probability of class ω_i given feature $\boldsymbol{v}$ is observed – the a posteriori probability – is computed as

$$P(\omega_i \mid \boldsymbol{v}) = \frac{p(\boldsymbol{v}, \omega_i)}{p(\boldsymbol{v})}.$$

Maximum A Posteriori Decision

Intuitively one wants to classify a sample as the most probable class for its given features. This cannot be done directly, but one can use Bayes theorem

$$P(\omega_i \mid \boldsymbol{v}) = \frac{p(\boldsymbol{v} \mid \omega_i) P(\omega_i)}{p(\boldsymbol{v})}.$$

Here, $P(\omega_i)$ comes from prior knowledge, $p(\boldsymbol{v})$ from a specific observation (usually approximated with histograms) and $p(\boldsymbol{v} \mid \omega_i)$ has to be learned from examples. Learning the right distribution is crucial for the accuracy of the segmentation.

Unsupervised Clustering

We want to distribute measurements to classes. The goal is homogeneity within classes, reducing variance over features. We only know the features and have no information on the classes (which is to be found as a result of this unsupervised process). We use a **K-means** algorithm:
Choose K centers/means

$$m_i \in \mathbb{R}^n, i = 1, ..., K$$

Repeat until centers (m's) do not change; for all measurements j, assign to nearest center i

$$c_j = \arg_i \min ||\vec{v}_j - m_i||^2$$

For all centers i, update it to center-of-mass

$$m_i = \sum_j^M \delta_{i=c_j} \vec{v}_j / \sum_j^M \delta_{i=c_j}$$

Note that the choice of K has a major influence (results in K classes) and initialization is important as K-means can get stuck in local minima.

Supervised Generative Modeling

Similarly to simple thresholding, earlier probabilistic model considers each pixel independently. We wish to do this with graphicals models. To do this, knowing the class for each pixel and the joint distribution of an individual pixel, we can define the joint distribution of all pixels when they are not independent as

$$p(\boldsymbol{c}, \vec{v}) = \prod_{j=1}^M p(\vec{v}_j | c_j) P(\boldsymbol{c})$$

Markov Random Fields sets up this distribution based on the Markovian property (i.e future states depend only on the present state and not on past states). Hence the probability of a class for a certain pixel only depends on its neighbors if all others are given. A common way to define it is through defining an energy based on the distance between c_j and c_k. We can then define a *Gibbs Probability Distribution* such that

$$P(c) = \frac{1}{Z} \exp(-E(c))$$

We can then perform segmentation through posterior maximization such that

$$arg_c max P(c|\vec{v}) = arg_c max P(c, \vec{v}) = \prod_{j=1}^M p(\vec{v}_j | c_j) \exp(-E(c))$$

To conclude, optimization of these models are hard, but they are flexible, general, and can be extended to various features.

Supervised Discriminative Modeling

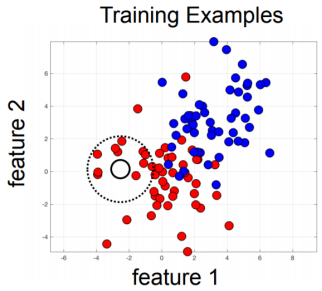
Now, we wish to take the features as input and predict the segmentation class assignment.

$$f(\vec{v}_j) = c_j$$

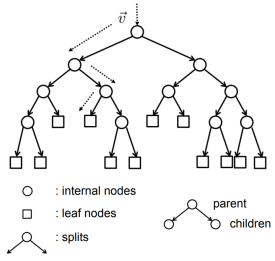
This mapping is learn from a training set of examples, composed of images the corresponding segmentations of the objects you are interested in.

KNN

Mapping is a parametric model based on the K nearest neighbors. For the *learning/training phase*, we estimate the parameters of the model in order to obtain the best possible segmentation in the training examples. Optimization is done by minimizing the discrepancy between algorithm segmentation and ground truth. We then move on to the *segmentation testing phase* where we extract the features used during training and use the mapping learned before. We can then find the K nearest neighbors within the training dataset with minimum distance. You can then define the mapping $f(\vec{v}))f(c_1, ..., c_k)$ either with majority voting or probabilities with uncertainties. KNN is very simple to implement but highly depends on the definition of K and needs a lot of training samples and memory.



Random Forests



At each internal node n there is a binary question on the features

$$f_n(\vec{v}) = \begin{cases} 0 \implies \text{go left} \\ 1 \implies \text{go right} \end{cases}$$

At each leaf node there is a prediction and it changes from leaf node to leaf node

$$f_l(\vec{v}) = c$$

These two levels of parametrization are learned during training. To deal with high dimensionality you can use an ensemble of decision trees. They are very efficient but need a large number of data and parametric choices.

Motion Extraction

Optical Flow

The motion of gray level patterns as observed in the image when a camera is moving relative to the objects in a scene or vice versa is called *optical flow*. Ideally, the optical flow will correspond to the projection of the three–dimensional velocity vectors on the image plane, but is not necessarily the case. Consider, e.g., a perfectly homogeneous sphere rotating about a fixed axis. There will be no changes in the shading pattern and therefore zero optical–flow, even though there is motion. Also, if there is no motion but the lighting changes, the optical flow will be non-zero. Fortunately, in typical cases, the projected velocities and the motion of the gray level patterns are well correlated.

Determination of the Optical Flow

Suppose two subsequent frames out of a sequence have been taken. The determination of the optical flow amounts to assigning to each pixel in the first image the corresponding pixel *with the same gray level* (intensity) in the second picture. The displacement between these pixels yields the local velocity. Denoting the gray level at (x, y) and time t by $I(x, y, t)$, then yields the following constraint for each pixel, called the *optical flow constraint equation*:

$$\frac{\mathrm{d}I}{\mathrm{d}t} = \frac{\partial I}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial I}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t} + \frac{\partial I}{\partial t} = 0.$$

Using the abbrevations

$$u = \frac{\mathrm{d}x}{\mathrm{d}t} \quad v = \frac{\mathrm{d}y}{\mathrm{d}t} \quad I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y} \quad I_t = \frac{\partial I}{\partial t},$$

the same constraint can be written as

$$I_x u + I_y v + I_t = 0.$$

The derivatives I_x, I_y and I_t are estimated from the image(s), with I_t being the change of intensity between two frames at the same pixel (x,y). Since there is a linear relation between u and v, we can rewrite the constraint as

$$\begin{pmatrix} I_x \\ I_y \end{pmatrix} \cdot \begin{pmatrix} u \\ v \end{pmatrix} = -I_t,$$

which determines the component of optical flow in the direction of the brightness gradient. The magnitude of this component is:

$$\frac{I_t}{\sqrt{I_x^2 + I_y^2}}.$$

The component of the optical flow perpendicular to this direction, however, cannot be easily resolved. The underdetermined nature of this formulation is referred to as the *aperture problem*, it is the inability to solve for both u and v. In order to get some intuition about this problem, one can think of looking at a line through a small hole. Suppose the line moves in some arbitrary direction. The movement of the line in direction of the line itself can never be seen, assuming the endpoints are not in the field of view. Notice that if the line was curved or had a corner, both velocity components could be extracted.

In order to retrieve the other component, an additional constraint is introduced in a process known as *regularisation* which enables us to solve ill-defined problems. Usually motion varies smoothly in most parts of an image, i.e. u and v are mostly constant in x and y. This observation underlies the usual choice for what is referred to as the *smoothness constraint*, which is imposed by minimizing some measure for the changes in the flow field. An example of such a measure is

$$e_s = \iint \left(\left(u_x^2 + u_y^2\right) + \left(v_x^2 + v_y^2\right) \right) \mathrm{d}x\,\mathrm{d}y,$$

where similar abbrevations as above are used:

$$u_x = \frac{\partial u}{\partial x} \quad u_y = \frac{\partial u}{\partial y} \quad v_x = \frac{\partial v}{\partial x} \quad v_y = \frac{\partial v}{\partial y}.$$

On the other hand, the optical flow constraint should also be small, i.e., one also tries to minimize

$$e_c = \iint (I_x u + I_y v + I_t)^2 \, \mathrm{d}x\,\mathrm{d}y.$$

Overall, then, the optimal functions $u(x, y)$ and $v(x, y)$ – which make up the optical flow – are sought that minimize

$$e_s + \lambda e_c,$$

where λ is a parameter that should be large if brightness measurements are accurate and small if they are noisy.

Euler–Lagrange Equations for Optical Flow

We reformulate the optimization over a function into a classical optimization over a scalar (the deviation from the solution which we want to equal 0), which we know how to solve. The solution to the minimization problem above can be found by solving the following Euler–Lagrange equations

$$F_u - \frac{\partial}{\partial x} F_{u_x} - \frac{\partial}{\partial y} F_{u_y} = 0$$
$$F_v - \frac{\partial}{\partial x} F_{v_x} - \frac{\partial}{\partial y} F_{v_y} = 0,$$

where

$$F = \left(u_x^2 + u_y^2\right) + \left(v_x^2 + v_y^2\right) + \lambda \left(I_x u + I_y v + I_t\right)^2.$$

This coupled pair of partial differential equations can be solved iteratively by solving the evolution equations

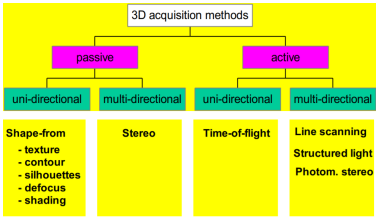
$$\frac{\partial u}{\partial t} = \nabla^2 u - \lambda \left(I_x u + I_y v + I_t\right) I_x,$$
$$\frac{\partial v}{\partial t} = \nabla^2 v - \lambda \left(I_x u + I_y v + I_t\right) I_y.$$

These equations are iteratively updated at each pixels and their derivatives can be approximated using finite diffferences. The solution, found at equilibrium $\partial u/\partial t = \partial v/\partial t = 0$ clearly depends on the parameter λ. More than two frames allow for a better estimation of I_t and the information spreads from edge- and corner-type patterns.

Note: We will obtain errors at object boundaries, because the smoothness constraint is no longer valid. Other approaches include *model-based tracking* (application specific) and *feature tracking* (i.e corners,blobs,regions etc...).

3D Data Extraction

There are multiple 3D methods:

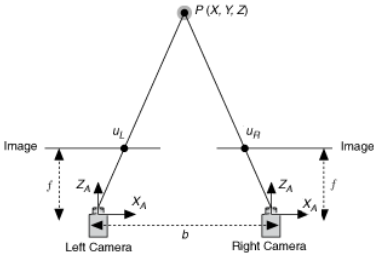


Stereo with Calibrated Cameras

Stereo–based 3D reconstruction is a passive multi-directional 3D acquisition method. The principle behind it is simple: Given two projections of the same point, its 3D position is found as the intersection of the two projection rays through its two images. This method of construction is referred to as *triangulation* and requires complete knowledge of both the internal and the external parameters of the cameras.

Simple Stereo Setup

A simple stereo setup that is widely used involves two cameras with identical internal parameters with $s = x_0 = y_0 = 0$. Moreover, the optical axes of the cameras are parallel and their image planes are coplanar, with coincident x–axes.



The distance b between the two camera centers is called *baseline*. Furthermore, suppose the world coordinate frame coincides with the left camera. The images of a point with coordinates (X, Y, Z) are then

$$\text{Left Camera:} \qquad \rho \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = K \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

$$\text{Right Camera:} \qquad \rho' \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = K \begin{pmatrix} X - b \\ Y \\ Z \end{pmatrix}.$$

With $\mathbf{K}$ the calibration matrix:

$$K = \begin{pmatrix} fk_x & 0 & 0 \\ 0 & fk_y & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

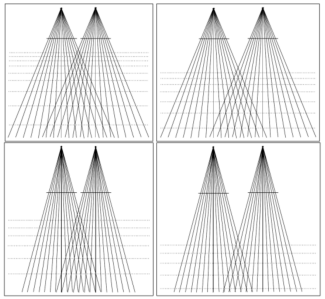
And hence,

$$\begin{cases} x = \frac{fk_x X}{Z} \\ y = \frac{fk_y Y}{Z} \end{cases} \qquad \begin{cases} x' = \frac{fk_x (X-b)}{Z} \\ y' = \frac{fk_y Y}{Z}. \end{cases}$$

Note that $y = y'$ for all point projections. Solving these equations for X, Y, and Z yields

$$X = b\frac{x}{x - x'} \qquad Y = b\frac{k_x}{k_y}\frac{y}{x - x'} \qquad Z = bk_x\frac{f}{x - x'}.$$

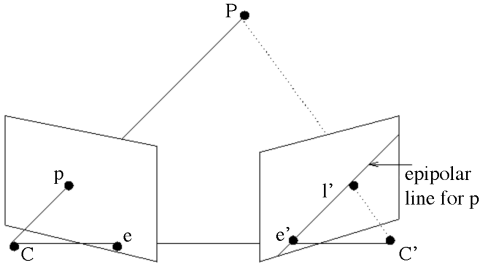
The distance $(x - x')$ is the so–called *disparity*. The distance Z, also referred to as the *range*, is inversely proportional to the disparity. Beyond a certain distance, depth measurements will therefore become very coarse. Human stereo depth perception is limited to distances of about 10 meters. Beyond, depth impressions arise from other cues. Note that the disparity is directly proportional to b and f; depth resolution can therefore be increased by increasing one or both of these variables.



Note: Same-disparity positions can be found on the same depth planes. Increasing b and/or f increases depth resolution but reduces simultaneous visibility. But the real problem is finding correspondences.

General Configuration – Epipolar Constraint

Given the projection $\boldsymbol{p}$ in the first image of a point $\boldsymbol{P}$, its position in the scene could be anywhere along the ray of projection connecting the camera center $\boldsymbol{C}$ with $\boldsymbol{p}$. The projection $\boldsymbol{p}'$ in the second image must lie on the projection $\boldsymbol{l}'$ of this ray in the second view.



The parameter equations for the ray of sight connecting the camera center $\boldsymbol{C}$ with the projection $\boldsymbol{p}$ is

$$\boldsymbol{P} = \boldsymbol{C} + \mu \mathbf{R}\mathbf{K}^{-1}\boldsymbol{p}.$$

Every scene point $\boldsymbol{P}$ satisfying this equation for some real number μ projects onto $\boldsymbol{p}$ in the first image. If $\mathbf{K}'$, $\mathbf{R}'$, and $\mathbf{C}'$ are the camera parameters of the second camera, then the projection $\boldsymbol{p}'$ of $\boldsymbol{P}$ is given by

$$\rho'\boldsymbol{p}' = \mathbf{K}'\mathbf{R}'^{\top}\left(\boldsymbol{P} - \boldsymbol{C}'\right),$$

for some nonzero $\rho' \in \mathbb{R}$. Substituting $\boldsymbol{P}$ with the first expression yields

$$\rho'\boldsymbol{p}' = \mu\mathbf{K}'\mathbf{R}'^{\top}\mathbf{R}\mathbf{K}^{-1}\boldsymbol{p} + \mathbf{K}'\mathbf{R}'^{\top}\left(\boldsymbol{C} - \boldsymbol{C}'\right).$$

The last term in this equation corresponds to the projection $\boldsymbol{e}'$ of the position $\boldsymbol{C}$ of the first camera in the second image

$$\rho'_e\boldsymbol{e}' = \mathbf{K}'\mathbf{R}'^{\top}(\boldsymbol{C} - \boldsymbol{C}').$$

This projection $\boldsymbol{e}'$ is called the *epipole* of the first camera in the second image. To simplify notation, an *infinity homography* matrix $\mathbf{A}$ is introduced as

$$A := \frac{1}{\rho'_e}\mathbf{K}'\mathbf{R}'^{\top}\mathbf{R}\mathbf{K}^{-1},$$

allowing to write

$$\boldsymbol{p}'^{\top}\left(\boldsymbol{e}' \times \mathbf{A}\boldsymbol{p}\right) = 0.$$

The consequence of this is that all epipolar lines intersect in the epipole. Introducing the skew–symmetric matrix notation $(\cdot)^{\times}$ as

$$\boldsymbol{a}^{\times} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}^{\times} = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix}$$

allows to further simplify the equation to

$$\boldsymbol{p}'^{\top}\boldsymbol{e}'^{\times}\mathbf{A}\boldsymbol{p} = 0.$$

Note that $(\cdot)^{\times}$ has rank two for any nonzero vector. The 3×3 matrix

$$\mathbf{F} = \boldsymbol{e}'^{\times}\mathbf{A}$$

is called the *fundamental matrix* and has rank two since $\boldsymbol{e}'^{\times}$ has rank two. Moreover, each pair of corresponding image points $\boldsymbol{p}$ and $\boldsymbol{p}'$ brings one homogeneous equation

$$\boldsymbol{p}'^{\top}\mathbf{F}\boldsymbol{p} = 0,$$

that is linear in the entries of the fundamental matrix $\mathbf{F}$.The 3-vector $\boldsymbol{p}'^{\top}\mathbf{F}$ contains the line coordinates of the epipolar line of $\boldsymbol{p}'$ (i.e a line in the 1st image that contains its corresponding point $\boldsymbol{p}$. The 3-vector $\mathbf{F}\boldsymbol{p}$ contains the line coordinates of the epipolar line of $\boldsymbol{p}$ (i.e a line in the 1st image that contains its corresponding point $\boldsymbol{p}'$. Thus, $\mathbf{F}$ *can be computed linearly, up to a non–zero scalar factor, from (at least) 8 correspondences between the two images*. Due to noise, the computed matrix $\mathbf{F}$ will not be of rank two. Imposing the rank two constraint for the computation of $\mathbf{F}$, results in a non–linear constraint, essentially reducing the number of required correspondences to 7. In conclusion, for 3D reconstruction, knowing all internal and external camera parameters as well as the corresponding point pair $(\boldsymbol{p}, \boldsymbol{p}')$ one has 6 linear equations

$$\boldsymbol{P} = \boldsymbol{C} + \mu\mathbf{R}\mathbf{K}^{-1}\boldsymbol{p}$$

$$\boldsymbol{P} = \boldsymbol{C}' + \mu'\mathbf{R}'\mathbf{K}'^{-1}\boldsymbol{p}'$$

in the 5 unknowns X, Y, Z, μ and μ'. Because of image noise and discretization errors, the computed projection rays may not intersect. One usually takes the point in the middle of the points of the rays where they are closest to each other.

Note: Epipolar lines are in mutual correspondence which allows us to match points on an epipolar line to points on the corresponding epipolar line. We can also separate 2D correspondence search problems to 1D search problem using two view geometry. One could also use more than 2 images, i.e 3, where $\boldsymbol{p}''$ is found at the intersection of epipolar lines, but it fails when the epipolar lines coincide.

Note2: From 2 views, if the camera translates, an affine reconstruction can be made, and a projective reconstruction is always possible (if no pure rotation). From 3 general views taken with the same camera parameters, a metric reconstruction is possible up to unknown scale.

RANSAC

Oftentimes, the fundamental matrix $\mathbf{F}$ is found using an algorithm called random sample consensus (RANSAC). is a general way to fit a model to data which contains outliers. The basic algorithm is iterative and has two steps:

1. A sample subset containing *minimal* data items is randomly selected from the input. A fitting model and the corresponding model parameters are computed using only the sampled subset.
2. The algorithm checks which elements of the entire dataset are consistent with the model instantiated by the estimated model parameters obtained in the first step. A data element will be considered as an outlier if it does not fit this model within some error threshold that defines the maximum deviation attributable to the effect of noise, and as inlier otherwise.

The set of inliers is called the consensus set. The algorithm iteratively repeats the above two steps until the obtained consensus set has a certain threshold cardinality.

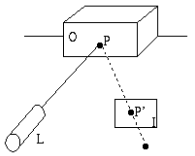
The Correspondence Problem

The correspondence problem refers to the task of finding the point in the second image which corresponds to a specified point in the first i+mage or vice versa. The epipolar constraint restricts the area in the image over which one has to search. There exist two general approaches:

- **Correlation–based:** The correlation method considers a window around the point in the first image. This window is used as a template to be shifted over the second image. Where the correlation between this template and the underlying intensity pattern is maximal, the corresponding point is assumed to lie. Small windows can be handled faster, but the result can be highly noisy and several wrong correspondences might result. Larger windows less accurately localize the corresponding point and are more computationally expensive but better withstand noise.
- **Feature–based:** One could look for corners or sharp edges in both images and try to retrieve correspondence for these features only. Not so many of such features will occur typically, but still there is the danger of finding several corners or edges along the epipolar line. Further disambiguation can be achieved by looking at attributes like edge contrast and polarity. Another useful constraint may be that the ordering between edges in both views often won't change. Finally, not finding many correspondences means not being able to calculate the depth of many points. The depths of all other pixels will have to be interpolated in one way or another.

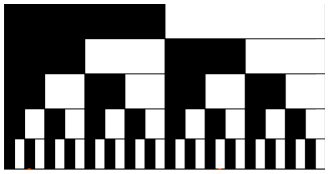
Active Triangulation

The correspondence problem with stereo vision can be solved by replacing one of the cameras by a projection device. If the projector is, e.g., a laser, then a single spot is projected onto the scene. The spot formed by reflection will be easily detectable in the image. If one knows the position and orientation of both the laser ray and the camera's projecting ray, then the corresponding three-dimensional point is known to be their intersection. However, due to noise, it is very hard for the two lines to intersect. Hence, by placing a cylindrical lens in front of the laser, the spot can be extended to a line so that scanning only has to be performed in one direction and the viewing ray only has to intersect with a plane. Sometimes a camera stereo pair is used in combination with a laser. The laser position and orientation of projection then no longer are to be known as its sole purpose in this setup is to simplify the correspondence problem.



Structured Light

With this method, patterns of a special shape are projected onto the scene, and deformations of the patterns yield information on the shape. We wish to combine good resolution with a minimum number of pattern projections.



This sequence of patterns is called *serial binary patterns* and yields 2^n identifiable lines for only n patterns. Also, using colour will yield 3^n identifiable lines for only n patterns.

Phase shift?

Time-of-flight

This method measures the time a modulated light signal needs to travel before returning to the sensor. This time is proportional to the distance. One can use radar, sonar, lidar, optical radar... If we are using lasers, we are also able to detect the intensity of each reflected laser pulse and color it.

Shape-from...

- **Texture**: assumes a slanted and tilted surface to have a homogeneous texture. Inhomogeneity is regarded as the result of projection.
- **Contour**: makes assumptions about contour shape. The deviation from the selected "true" shape yields information on 3D orientation.
- **Shadows**: uses directional lighting often with known direction. Local intensity is brought into correspondence with orentation via reflectance maps. The orientation of an isolated patch cannot be derived uniquely. However, we have to assume surface smoothness and known normals at the rim.
- **Silhouettes, Defocus**

Photometric stereo

We eliminate constraint propagation by using light from different directions. When the light sources are given different colours, the differently directioned sources can be applied at the same time.

Specific Object Recognition

The complexity of the object recognition problem depends on whether the 'what' and 'where' part of the challenge can be separated.

Model–Based Approaches

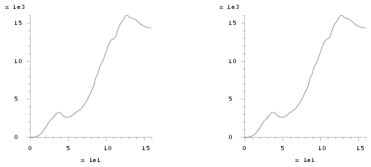
Model–based methods use topological constraints imposed on the features. Graphs are well–suited for describing such *structural* or *relational models*. Recognition in those cases is a question of (sub)graph matching. Structural methods strongly rely on model data while extracting the features, in contrast to invariant based approaches. The recognition problem is also heavily intertwined with the localization problem.
The general idea is as follows: Assuming that both the model and the scene are composed of features such as vertices, holes, straight lines, etc., and a specified object is searched for, one knows in advance what types of features to search for in the image. After making an initial feature match hypotesis, the arrangement of other features in the model is used to verify the assumption: the other features have to be found near their expected locations. Notice that the assumption is not only about type but also about pose. Algorithms usually cycle trough this hypothesize–and–verify loop, which can be very slow.

Invariant-based recognition of planar shapes

The crucial advantage of this method is that it decouples the object type and pose problem. It is assumed that perspective effects can be neglected, i.e., that an affine transformation models the actual transformation well. Furthermore it is assumed that it is possible to calculate the area of shapes. The following expressions can be shown to be invariant under affine transformations:

$$\int_{t_2}^{t_1} \left| \boldsymbol{x}_1 - \boldsymbol{x}_2 \boldsymbol{x}_1^{(1)} \right| \, \mathrm{d}t \qquad \int_{t_2}^{t_1} \left| \boldsymbol{x}_1^{(1)} \boldsymbol{x}_1^{(2)} \right|^{\frac{1}{3}} \, \mathrm{d}t.$$

Here, $\boldsymbol{x}_2$ is a fixed point, which is also used as the starting point of the integrations (t_2 is the parameter value at $\boldsymbol{x}_2$). $\boldsymbol{x}_1$ is a point that varies with t and t_1 represents the corresponding parameter value at any particular position of $\boldsymbol{x}_1$. Hence, for every position of $\boldsymbol{x}_1$ one obtains two values, one for each integral. Expressing the values as a function of each other results in invariant features.



Pros and Cons

- **Pros**: Compact model and can deal with clutter
- **Cons**: Slow analysis-by-synthesis, models difficult to produce, and works only for limited object classes

Appearance–Based Approaches

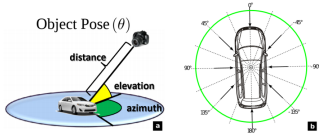
This method is only used with simple changes in object pose, e.g., pure translations of the shape. Template matching consists of comparing, usually by correlation or subtraction techniques, sections of the image with images of the objects one is looking for. The images of the objects are called templates. Hence, the "model" of an object is simply its images. This approach is very time consuming, especially if the orientation or scale of the pattern are unknown.

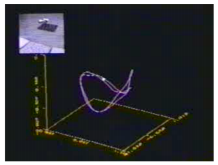
Appearance manifold approach

Training: for every object:

1. sample the set of viewing conditions (viewpoints)
2. use these images as feature vectors
3. apply PCA over all the images
4. keep the dominant PCs (10-20 usually)
5. sequence of views for 1 object represents a manifold in the space of projections

By looking at the appearance changes projected on the PCs, we see that the PCs evolve in a continuous way. This is sufficient characterization for recognition and pose estimation (works particularly great for patterns with a lot of redundancy, e.g faces).





Pros and Cons

- **Pros**: Efficient, models easy to produce, and works for wide classes of objects
- **Cons**: Large models and cannot deal with clutter

Hybrid Approaches

These techniques combine appearance and model based strategies and are sometimes called *view–based techniques*. Usually, this class of methods models objects on the basis of several views, but rather than taking the images directly as the components of the model, features are extracted from them.

Euclidean Invariant Features

The method can be separated in **training**:

1. look for corners (with Harris detector)
2. take circular regions around these points, of different radii (to cope with scale changes)
3. calculate, from the regions, invariants under planar rotation
4. do this from different viewpoints, where the invariance cuts down the number of views needed (no in-plane rotation necessary)

and **testing:**

1. compare these invariants with those found for images in the database (find matches)
2. look for consistent placement of candidate matches/ epipolar geometry
3. decide which object based on the number of remaining matches (best matching image : type + approximate pose)

Euclidean invariant features are features that remain constant under a similarity transformation. Suppose G is a Gaussian convolution filter and I represents image intensity. Then, $G_x = \partial/\partial x(G*I) = G*\partial I/\partial x$ and $G_y = \partial/\partial y(G*I) = G*\partial I/\partial y$ represent smoothed values for the first derivatives of the intensity pattern, often referred to as the first *Gaussian derivatives*. Similarly, higher order Gaussian derivatives can be calculated. The profile of the Gaussian drops off to nearly zero, thereby effectively representing a circular neighbourhood. The following combinations of Gaussian derivatives can all be shown to be invariant under image rotation (note that the gradient is not rotation invariant, but the gradient magnitude is):

$$G_x^2 + G_y^2 \qquad G_{xx}G_x^2 + 2G_{xy}G_xG_y + G_{yy}G_y^2$$

$$G_{xx} + G_{yy} \qquad G_{xx}^2 + 2G_{xy}G_{yx} + G_{yy}^2.$$

In order to also deal with objects appearing at different scales in the images, these rotation invariants are extracted for Gaussians of different sizes. For each size the above invariants form a feature vector. If one wants to deal with changes in viewing direction as well, then multiple images need to be taken from different directions, as in the case of the appearance–based methods. The invariance of the descriptors can be improved using feature extraction methods as previously seen.

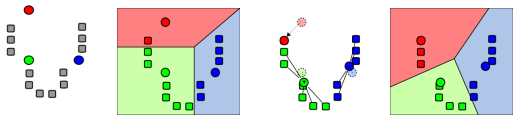
Pros and Cons

- **Pros**: Efficient, can deal with clutter and partial occlusion, models easy to produce (take images, fewer than in pure appearance-based method), and works for rather wide classes of objects
- **Cons**:Problems with uniform (untextured) objects

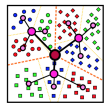
Visual Words and Vocabulary Trees

Regardles of descriptor type, at one point one will have to compare the descriptor of a new test image to a descriptor database. Obviously **this can be a tedious task, if done naively**. One way to do this more efficiently is to cluster the descriptors into "visual words" and match the words hierarchically. In the machine learning community this is called k–means clustering. The algorithm is fairly straight forward:

1. k initial "means" are placed randomly within the data domain;
2. Every sample is associated to its nearest mean, generating k clusters;
3. The centroid of each of the clusters becomes its new mean;
4. Steps 2 and 3 are repeated until convergence is reached.

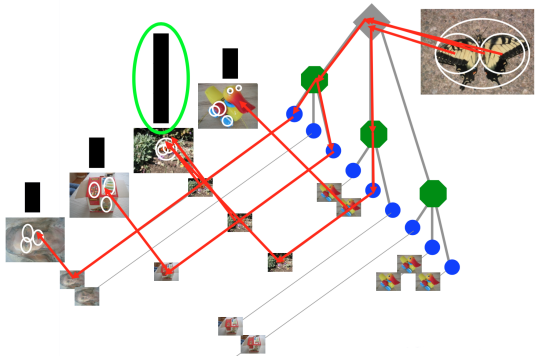


For each of the clusters created with the k–means algorithm, one can again do the same trick. This results in a *hierarchical k–means* structure, a so–called "vocabulary tree", allowing for much greater vocabulary and thus better classification results.



Inverted File Index

For text documents, an efficient way to find all pages on which a word occurs is to use an index. The same idea can be applied to find all images on which visual words appear. A new query image is mapped to the database and the image in the database that shares the most visual words is suggested as having the object in it.



RANSAC for Image Processing

However, we need to support the matching step, as it **will often fail when only based on descriptor matching.** Typically, many "matches" are wrong, so–called *outliers*, and one needs to add a test on the configuration in order to remove the outliers and only keep the correct matches, i.e., the *inliers*. The tests used for 3D data extraction are usually:

- **Epipolar Geometry** The test on epipolar geometry assumes that there is a fundamental matrix that matches are in agreement with.
  This assumes rigidity in the scene, i.e., that objects do not move with respect to each other.
- **Projectivity** The test on projectivity assumes that there is a projectivity that maps points in the first image onto the matching points in the second image.
  This assumes that the scene is largely planar.

The RANSAC algorithm for solving the correspondence problem is

1. Randomly select the minimum number of matches to formulate an initial constraint hypothesis: 7 for the epipolar geometry to compute the fundamental matrix **F** and 4 for a projectivity. These numbers should be small, since the selected tuples must not contain any outlier match.
2. Check how consistent other matches are with these hypotheses, i.e., how far they are supported.
3. Use all supporting matches to refine the hypotheses and discard the rest.

Finally, RANSAC selects the hypothesis with maximal support after a fixed number of trials or after sufficient support was reached. Suppose

- n is the minimum number of data required to fit the model,
- k is the number of iterations performed by the algorithm,
- t is the threshold to determine when a match fits the model,
- d is the number of inlier needed for a model to be a good fit.

Typically, t and d are chosen beforehand. The number of iterations k can then be calculated. Let Let p be the probability that RANSAC selects inliers for the n data units generating a hypothesis at least once, i.e. the probability that the algorithm gets a good output. Let w be the the *estimated* proportion of inliers. The probability that no good hypothesis is selected is then

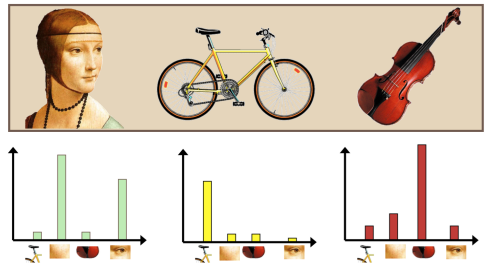
$$1 - p = (1 - w^n)^k.$$

Note that RANSAC does typically not withstand more than 30% outliers, so if the initial matches are bad, alternative solutions should be considered.

Object Category Recognition

Classification: Bag–Of–Words

In this problem we are looking at categories, i.e how to recognize ANY car for example. Therefore, on top of factors affecting specific object recognition, we have the added complexity of intra-class variation. We see two main tasks: *classification*: is there this object in this image, and *detection*: where is this object in this image. Any object that one wishes to recognize can be analyzed in terms of the visual words that it contains. Appearances of objects vary a lot within a category but appearance of local parts varies less! Subsequently, a histogram of the words can be created for every object. Thereby, every image is summarized based on its distribution, i.e., histogram of word occurences. The advantage of histograms is that irrespective of the number of local features, the size is always the same.



Subsequently, any histogram similarity measure can be used to compare two histograms. One example of such a measure is the normalized scalar product:

$$\mathrm{sum}(\boldsymbol{d}, \boldsymbol{q}) = \frac{\boldsymbol{d} \cdot \boldsymbol{q}}{|\boldsymbol{d}| \, |\boldsymbol{q}|} = \frac{\sum_{i=1}^{t} w_{i,d} w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,d}^2} \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

Hence BoW enable to describe the unordered feature set with a single vector of fixed dimensionality. It provides an easy way to use distribution of feature types with various learning algorithms requiring vector input. We can learn a decision rule (classifier) assigning BoW representations of images to different classes (e.g zebra and not zebra). Any decision rule divides input space into *decision regions* separated by *decision boundaries*. The classifier can be Nearest Neighbor, Neural Networks, SVM, Boosting etc... Note that this approach removes the spatial layout of the visual words, which is both a strength and a weakness. There exist some ideas to bring the spatial information back into the descriptors:

- "Visual phrases": frequently cooccurring words can be formed into visual phrases
- Semi–local features can be used to describe the neighbourhood
- The position of the visual word can be part of its descriptor
- The bags of words can be counted only on sub–parts of the image

The **Spatial Pyramid Representation** does the latter, separating an image into multiple ones and doing a histogram for each of them (works particularly well for capturing scene categories).

Pros and Cons

- **Pros**: Flexible to geometry/deformations/viewpoint, compact summary of image content, provides vector representations for sets, and empirically good recognition results in practice
- **Cons**: Basic model ignores geometry (can verify afterwards or embed within feature descriptors), background and foreground mixed when bag covers whole image, no guarantee to capture object-level parts with interest points or sampling, and optimal vocabulary formation remains unclear

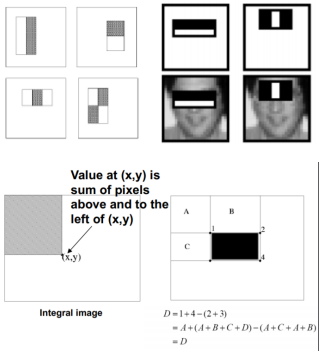
Detection: Sliding-Windows

The basic idea is to create a binary classifier (e.g car or not car) and slide a window around looking for it. Therefore, we need to obtain training data, define features, and train this classifier. Feature extraction is based on global appearance. We could create a simple grayscale/color histogram which could be translated into a vector of pixel intensities as the description of image content, but this would be sensitive to illumination and intra-class appearance variation (e.g albino koala).

A better method is *Histogram of Oriented Gradients*. This technique counts occurrences of gradient orientation in localized portions of an image for the purpose of object recognition. The features are computed on a dense grid of uniformly spaced cells. This method offers invariance to small shifts and rotations (locally orderless) an tries to correct for variable illumination (contrast normalization).

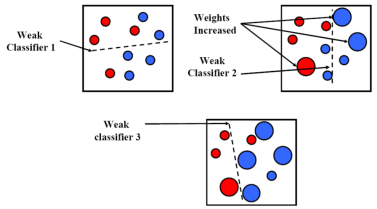
Rectangular Integral Image Filters

After computing the intensity in the white region and the black region, the feature output is the difference between them. It is efficiently computable with integral image, and scales features directly for the same cost. There exists a large library of filters, and you need to select the informative ones to form the classifier.



AdaBoost

The goal is to build a strong classifier by combining many "weak" classifiers, which need only be better than chance. It is a sequential process: at each iteration, we add a weak classifier. Let's consider a 2D feature space with positive and negative examples. Each weak classifier splits the training examples with at least 50% accuracy. Examples misclassified by a previous weak learner are given more emphasis at future rounds. This is the base learning algorithm for the Viola-Jones face detector.



AdaBoost is both to select the informative features (from the large library of filters) and to form the classifier. We want to select the single rectangle feature and threshold that best separates positive (faces) and negatives (non faces) training examples, in terms of weighted error.

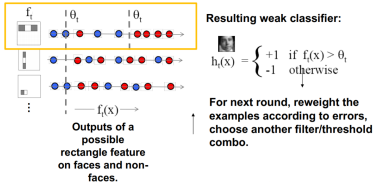


Image features are weak classifiers. For each round of boosting:

1. Evaluate each rectangle filter on each example
2. Sort examples by filter values
3. Select best threshold for each filter (min error)
4. Select best filter/threshold combination
5. Weight on this feature is function of error rate
6. Reweight examples

Note: even if the filters are fast to compute, each new image has a lot of possible windows to search. In order to make the detection more efficient, we could implement a cascade of classifiers of increasing complexity. (first conservative classifiers trying to keep all positives, letting pass some false positives as price to pay; then complex classifiers checking the fewer surviving patterns). For the Viola-Jones detector, the first two features selected are the eyes.

Pros and Cons

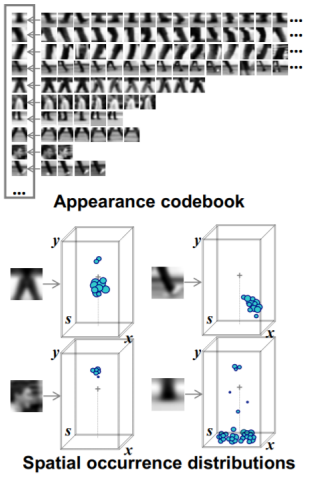
- **Pros**: Simple detection protocol to implement, good feature choices critical but part of the process, past successes for certain classes, good detectors available (Viola-Jones, HoG, etc..), with so many windows, false positive rate should be low
- **Cons**: High computational complexity (more than 100,000 locations in an image putting tight constraints on the classifiers we use: if training binary detectors independently for different classes, this means cost increases linearly with number of classes; can be solved by focusing on promising locations only), typically need fully supervised training data, not all objects are box shaped, deformable objects (e.g animals) not captured well, when considering windows in isolation context is lost, sensitive to partial occlusions

Generalized Hough Transform

Implicit Shape Model

The implicit shape model uses visual words to not only detect but also localize an object. It does so by indexing votes for an object position, given its visual words.

We first learn the appearance codebook (extract local features at interest points, the agglomerative clustering consists in the codebook), then learn spatial distributions (match codebook to training images, and record matching positions on the object).



In other terms: each visual word has assigned to it the possible locations of the object center (see displacement vectors), weighted by their appearance in the training data. When an object is detected, the localization is achieved by looking at which position gets the most votes of the visual words.

Pros and Cons

- **Pros**: Works well for many different object categories (deformable too), has a flexible geometric model, learning is done on few training examples, and it is optimized for detection (good localization properties)
- **Cons**: Needs supervised training data (bounding boxes for detection), only weak geometric constraints, and is purely a representative model (no discriminative learning)

Tracking

Tracking methods are used to follow a specific object through an image sequence. Such methods will often not only produce the position as output, but also additional information, such as the direction or type of motion, the velocity, etc. This allows to predict the objects future position and hence restrict the area to be searched over.

Problem Setup

Depending on the application, points, lines, contours, regions, or whole 3D models could be used as state vectors $x \in \mathbb{X}^n$. The state vector x is assumed to evolve according to a *system model*

$$x_t = f_{t-1}(x_{t-1}, w_{t-1}),$$

where f_{t-1} is the *system transition function* and w_{t-1} is a noise vector with a known distribution. At discrete times, measurements $z_t \in \mathbb{X}^{\mathcal{Y}}$ based on a segmentation become available. These observations are related to the state vector by the *measurement model*

$$z_t = h_t(x_t, v_t),$$

where h_t is the *measurement function* and v_t is the measurement noise. The history of all observations is denoted as $\mathcal{Z}_t = \{z_1, \ldots, z_t\}$. In general one is interested in deriving an estimate for the state vector x_t at time t from the previous state vector x_{t-1} and the measurement z_t.

Recursive Bayesian Filter

As there is uncertainty about the objects state, it is represented by a probability distribution. The system model allows for the computation of the probability distribution for the next time step, which is called the *a priori* distribution $p(x_t \mid \mathcal{Z}_{t-1})$. The update of this distribution with the given new measurement is then the *a posteriori* distribution $p(x_t \mid \mathcal{Z}_t)$. Thus, one can recursively construct the a posteriori probability density of the state conditioned on all measurements.

Prediction

The system model is described by

$$p(x_t \mid \mathcal{Z}_{t-1}) = \int p(x_t \mid x_{t-1}) p(x_{t-1} \mid \mathcal{Z}_{t-1}) \, \mathrm{d}x_{t-1}$$

where $p(x_t \mid x_{t-1})$ is the *process density* or *transition probability*. An integration must be solved as each x_{t-1} can make a contribution to $p(x_t \mid \mathcal{Z}_{t-1})$ via $p(x_t \mid x_{t-1})$. Note that this integral can only be solved for a small group of probability densities.

Update

The prediction at each time step should be updated by incorporating the new measurement z_t. With the help of Bayes' rule, one can determine the a posteriori distribution

$$p(x_t \mid \mathcal{Z}_t) = \frac{p(z_t \mid x_t) p(x_t \mid \mathcal{Z}_{t-1})}{p(z_t \mid \mathcal{Z}_{t-1})},$$

where $p(z_t \mid \mathcal{Z}_{t-1})$ is a normalization factor and the *observation density* $p(z_t \mid x_t)$ defines the likelihood that a state x_t causes the measurement z_t.

Particle Filtering

This is a sample based solution to the recursive Bayesian filter. The key idea is to represent the probability distribution by a weighted sample set $S = \{(s^{(n)}, \pi^{(n)}) \mid n = 1, \ldots, N\}$. Consequently, the distribution of the samples constitutes a discrete approximation of the probability distribution. Each of the N samples consists of an element s which represents the state vector and a corresponding numerical weighting factor, i.e., a discrete sampling probability which is given by π.

Prediction

One starts with a sample set S_{t-1} computed in the previous iteration step, which approximates the *a posteriori* density $p(x_{t-1} \mid \mathcal{Z}_{t-1})$. The evolution of each sample is described by the application of the system model

$$s_t'^{(n)} = F s_{t-1}^{(n)} + B w_{t-1}^{(n)},$$

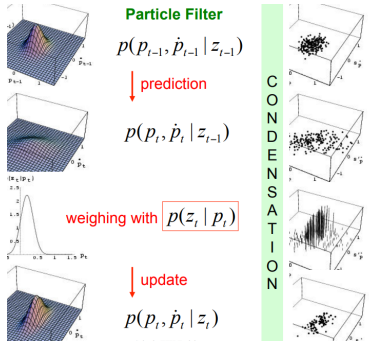
where F determines the deterministic and B the stochastic component of this difference equation. This generates a new sample set S_t' which approximates the *a priori* density $p(x_t \mid \mathcal{Z}_{t-1})$.

Update

As before, the new measurement z_t is used to update the prediction by weighting each element of the sample set S_t' on the basis of

$$\pi^{(n)} \propto p\left(z_t \mid s_t'^{(n)}\right).$$

By resampling the set S_t' we obtain a new sample set S_t which approximates the new *a posteriori* density $p(x_t \mid \mathcal{Z}_t)$. A particular sample $s_t'^{(j)}$ is drawn, with replacement, by choosing it with probability $\pi^{(j)}$.



Traditional (simple) tracking, is then just comparing the position in the previous frame to candidate new positions and finding the best new position.

Feature Tracking

Region Tracking

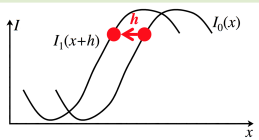
Background Modeling
Background modeling in its simplest form is only applicable for stationary cameras and illumination. A snapshot of the background is necessary. Once the system is set up, one simply subtracks the background from the current frame and the difference of the two is assumed to be the object.

Mean Shift Tracker
The mean shift tracker follows a region with a predescribed (color) distribution. It can be used to find the object from background modeling, by assuming that the object is formed of a large group of densely located pixels (in contrast to noise as fewer scattered foreground pixels). The similarity between the tracked region and a target region is maximized through evolution towards a higher density in parameter space. Usually after quite few iterations, the maximum likelihood position of the object in the new frame will have been found. We can initialize multiple means and pick the location where many converge.

Point Tracking

Consider a simple one–dimensional problem of an intensity function $I(x)$. It can be shown that

$$h \approx \frac{I_1(x) - I_0(x)}{I_0'(x)}.$$



One kind of problem includes getting stuck in local minima and having small derivatives $I_0'(x)$. Another problem family is based on the aperture problem. For tracking to be well definied, nonzero gradients in all possible directions are needed. If no gradient along one direction, we cannot determine relative motion in that axis. The indirect result is that the frame-rate should be faster than the motion of half-wavelength (Nyquist rate). Whereas Optical Flow recovers (smooth) motion everywhere, **tracking seeks a single motion for a region**. We can handle the local minima problem by using the same regularization constraint as in the case for optical flow, i.e., spacial coherence (a pixel's neighbours all move together, we assume a single motion for a region). We thus have a least squares problem which resembles the Harris corner detector. Hence, good image features are also good for tracking, and it will be a lot easier to track feature-dense objects.

Template Tracking

The idea is to keep a template image to compare with each frame, usually applied for small patches. We generalize the motion model from translation to other parametric models, with the *warp* being any kind of transformation (affine, rotation, shear etc..). We use the **Lucas-Kanade Template Tracker** in the following way:

1. Warp I to obtain $I(W([x, y]; P))$
2. Compute the error image $T(X) - I(W([x, y]; P))$
3. Warp the gradient ΔI with $W([x, y]; P))$
4. Evaluate $\partial W/\partial P$ at $([x, y]; P$ (Jacobian)
5. Compute steepest descent images $\Delta I \cdot \partial W/\partial P$
6. Compute Hessian Matrix
7. Compare with the error
8. Compute the change in parameter ΔP
9. Update $P \longleftarrow P + \Delta P$

Model Tracking

Tracking-by-Detection

This method is used to track a specific target, or of an object class.

Specific target
Using 3D object detection, we can find the feature descriptors of the object to detect and then find them again in the test image. First we have to detect keypoints which are invariant to scale, rotation or perspective. Then we can build feature descriptors which identify those specific points. From the test image, we are then able to search in the database and match the keypoint descriptors of the query. Using RANSAC we can finally test the geometric soundness and remove outliers.

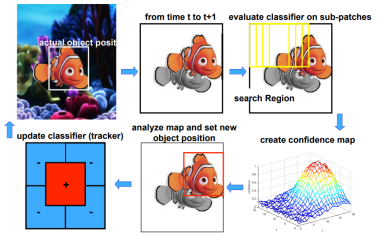
Object class
Here we detect independently in each frame and then associate detections over time into tracks. This works for multiple object tracking. In order to get the detections and classify the objects, we can use SVM, Random Forests, Neural Networks...

Model-Based Body Articulation

The model has 2 components: *parts* (2D image fragments) and *structure* (configuration of parts). Parts based analysis allows us to detect humans and determine body pose. Given parts and pairwise linkages between parts, we have h possible poses for each part. We can then label each part by its pose, each labelling having a cost. We then fit the model as labelling with the lowest energy. We can use this to track articulated motion. Walking cycles have one main periodic DOF, and we can use regressors (e.g PCA) to learn this space and its variation.

On-line Learning

Using classification we can learn current object appearance vs. local background.



Progressively changing the lighting or the background is not a problem as the model progressively learns. However, since the model is self learning, we can experience drift, as the model will start to track an unwanted object (i.e face tracker becomes hand tracker).

Tracking and detection

Avoiding Drift

In order to avoid drift, we have to refine our object model. The only thing we are sure about the object is its initial model (e.g appearance in first frame). Therefore, we can "anchor" our model with this information, in order to help avoid drift.

Context in tracking

Humans use context to track objects which change their appearance very quickly, occluded objects or objects outside the image, and small or low textured objects. For example, in order to track an object which becomes occluded, one could use supporters, which are points which seem to move together with this object but remain seen.

In practice

If we are able to add tracking info (e.g optical IR markers) then it is easier to do so. If the object is known but modification is impossible, then we should use known info (e.g template image/known object features). If the object is unknown but resides in a static environment then we can use background modeling. If none of the above then we can simply follow the object from the initial image/location.

Applications
For safety next to blade we can use articulated model or background modeling. To track the motion of ultrasound probe we can use marker + feature. For autonomous driving it is best to use learning based detection (as we don't know all types of cars etc..).

Tracking Issues

- **Prediction vs. Correction** If the dynamics model is too strong, the algorithm will end up ignoring the data, whereas if the observation is given too much weight, tracking is reduced to repeated detection.
- **Fast motion**
- **Multiple object tracking** What if we don't know which measurements to associate with which tasks?
- **Nonlinear dynamics** Non–linear dynamics require a non–linear dynamics model. Abrupt changes in direction usually gives rise to problems.
- **Data Association** Background and appearance change cause problems because the correction gives poor results. If online adaption is implemented, however, the algorithm can start learning to track the wrong object.
- **Drift** Errors caused by the dynamics model, the observation model and data association tend to drift over time and become larger.

Introduction to Neural Networks

Basic Concepts

Machine learning for visual perception implies discovering and leveraging **patterns**, by seeing lots of examples and making predictions. The two main types of learning are

1. **supervised learning**: specific task (e.g segmentation) and examples have both features and labels whereas at prediction we only have the images.
2. **unsupervised learning**: no specific task and examples only have images.

As for the basic notation

1. **Features**: $\mathbf{x} = x_1, x_2, ..., x_d$, usually observed raw intensities
2. **Labels**: $\mathbf{y} = y_1, y_2, ..., y_m$, can be numerical (regression) or categorical (classification)

We wish to create a mapping between features and labels, using parameters θ such that

$$y = f(\mathbf{x}; \theta)$$

Learning implies determining the "best" parameters using the examples. Labeled examples used for learning are called *training set*. The examples form a paired dataset

$$\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$$

The "best" parameters are the ones that minimize a *cost function* defined between predictions and ground-truth labels, which is task dependent

$$\theta^* = arg_\theta min \sum_{n=1}^{N} \mathcal{L}(\mathbf{y}_n, f(\mathbf{x}_m; \theta))$$

Regression

We can use general p-norm to establish the cost:

$$\mathcal{L}(\mathbf{y}_n, f(\mathbf{x}_n; \theta)) = ||\mathbf{y}_n - f(\mathbf{x}_n; \theta)||_p$$

Prediction will have errors, and can be written as

$$\hat{\mathbf{y}} = f(\mathbf{x}; \theta^*)$$

The most commonly used techniques for prediction are Mean Squared Error and Mean Absolute error:

$$\text{MSE} = \frac{1}{T}\sum_t^T ||\hat{\mathbf{y}}_t - \mathbf{y}_t||_2^2 \,; \text{MAE} = \frac{1}{T}\sum_t^T |\hat{\mathbf{y}}_t - \mathbf{y}_t|$$

Classification

We can use cross-entropy to establish the cost:

$$\mathbf{y}_n \in \mathcal{C}, \mathcal{C} \text{ : set of possible classes}$$

$$f(\mathbf{x}; \theta) = [f_{c_1}(\mathbf{x}; \theta), f_{c_2}(\mathbf{x}; \theta)...], \sum_{k \in \mathcal{C}} f_k(\mathbf{x}; \theta) = 1$$

$$\mathcal{L}(\mathbf{y}_n, f(\mathbf{x}_m; \theta)) = -\sum_{k \in \mathcal{C}} log(f_k(\mathbf{x}; \theta))\mathbf{1}(y_n = k)$$

where predictions are considered as class probabilities. Predictions will have errors and can be written as

$$\hat{\mathbf{y}} = \arg_k \max f_k(\mathbf{x}; \theta^*)$$

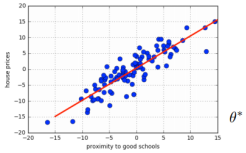
The most commonly used technique for prediction is classification error:

$$\text{Cerr} = \frac{1}{T}\sum_t^T \mathbf{1}(\hat{\mathbf{y}}_t \neq \mathbf{y}_t)$$

Linear and Logistic Regression Models

The linear model is the main building block (for continuous labels), it assumes a linear relationship between features and labels. For simplicity, if we assume a one dimensional label $\mathbf{y} = y$ we have

$$\theta_1 x_1 + \theta_2 x_2 + ... + \theta_d x_d = \theta^T \mathbf{x}$$



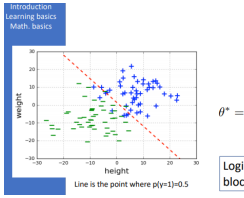
$$\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$$

$$y = f(\mathbf{x}; \theta) = \theta^T \mathbf{x}$$

Learning

$$\theta^* = \arg_\theta \min \sum_{n=1}^{N} \|y_n - \theta^T \mathbf{x}\|_2^2$$

The logistic regression is the extension to binary labels



$$\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$$

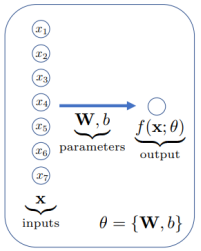
$$p(y = 1) = f(\mathbf{x}; \theta) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}}$$

Learning

$$\theta^* = \arg_\theta \min \sum_{n}^{N} -\log(f_k(\mathbf{x}_n; \theta))y_n - \log(1 - f_k(\mathbf{x}_n; \theta))(1 - y_n)$$

Logistic regression is the building block of the perceptron model

Perceptron model



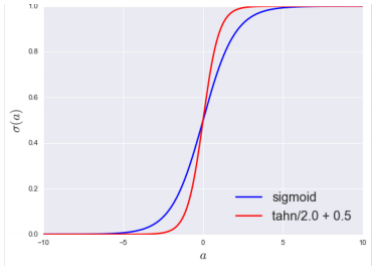
Given $\mathbf{x}$ inputs, and the d+1 parameters $\mathbf{W} \in \mathbb{R}^{dx1}$ & $b \in \mathbb{R}$, the model of binary classification is written as

$$p(y = 1) = f(\mathbf{x}; \theta) = \sigma(\mathbf{W}\mathbf{x} + b)$$

It is formed of two different parts:

1. Activation: $\underbrace{a}_{activation} = \underbrace{\mathbf{W}}_{weights}\mathbf{x} + \underbrace{b}_{bias}$
2. Nonlinearity: $f(\mathbf{x}; \theta) = \sigma(a)$

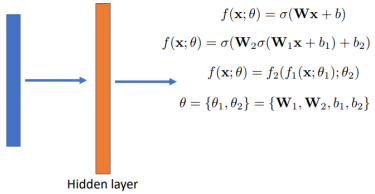
The activation function determines the threshold (decision boundary) whether the neuron should be activated or not, i.e whether the element belongs to the class. The nonlinearity maps real line to probabilities, a sigmoid ($\sigma(a) = \frac{1}{1+\exp(-a)}$) or tangent hyperbolic function can be used



This model is essentially a linear model, and can only model linear boundary separation.

Multilayer extension

By using multilayer perceptron (MLP) we are able to perform non-linear separation.

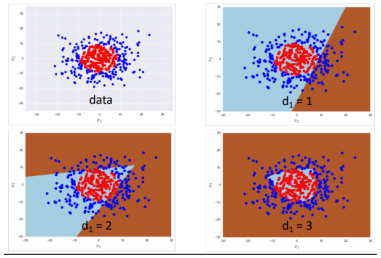


$$f(\mathbf{x};\theta) = \sigma(\mathbf{W}\mathbf{x}+b)$$
$$f(\mathbf{x};\theta) = \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x}+b_1)+b_2)$$
$$f(\mathbf{x};\theta) = f_2(f_1(\mathbf{x};\theta_1);\theta_2)$$
$$\theta = \{\theta_1,\theta_2\} = \{\mathbf{W}_1,\mathbf{W}_2,b_1,b_2\}$$

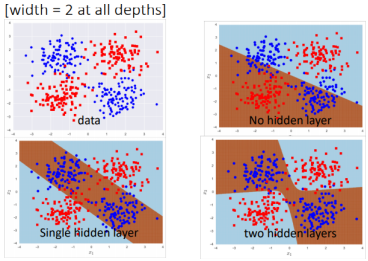
There are two main characteristics which define MLP (choosing them is an *architectural* design choice):

- **width** of the hidden layer d_1
  - $d_1 = d$ – Only nonlinear transformation
  - $d_1 > d$ – Mapping to a higher dimensional space makes it easier to model complicated decision boundaries in higher dimensions. There is a larger number of model parameters.
  - $d_1 < d$ – Compression/bottleneck, possible information loss

$$\mathbf{W}_1 \in \mathbb{R}^{d_1 x d} \longrightarrow \sigma(\mathbf{W}_1\mathbf{x}+b_1) \in \mathbb{R}^{d_1}$$
$$\mathbf{W}_2 \in \mathbb{R}^{d_1 x 1} \longrightarrow f(\mathbf{x};\theta) \in \mathbb{R}$$



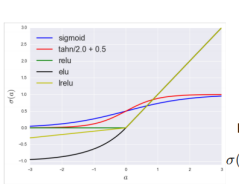
- **depth**:
  - No hidden layer ⇔ logistic regression/linear
  - Increasing depth allows more complicated models. There is a large number of model parameters and we need more samples to fit reliably. It also may become difficult to train.



The activation functions $\sigma(\cdot)$ have to be non-linear. If they were linear, the final map would be linear same as a single linear model like logistic regression:

$$f_l(h_{l-1}) = \mathbf{V}(\mathbf{W}_l h_{l-1}+b)+c = \widetilde{\mathbf{W}}h_{l-1}+\tilde{b}$$
$$y = f_l \circ f_{l-1} \circ \ldots \circ f_1(\mathbf{x}) = \hat{\mathbf{W}}\mathbf{x}+\hat{b}$$

More recent non-linear models which are often used in connections between internal layers are:



Rectified Linear Unit (Relu)

$$\sigma(a) = \begin{cases} a, & a \geq 0 \\ 0, & a < 0 \end{cases}$$

Leaky Relu

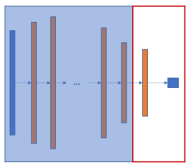
$$\sigma(a) = \begin{cases} a, & a \geq 0 \\ \alpha a, & a < 0 \end{cases}$$

Exponential Linear Unit (Elu)

$$\sigma(a) = \begin{cases} a, & a \geq 0 \\ \exp(a)-1, & a < 0 \end{cases}$$

The main difference between non-linear functions are their derivatives. Sigmoid and tanh saturate for high values, their derivatives diminish. Relu doesn't saturate and has constant derivative for positive and 0 for negative values. It yields good empirical performance.

Feature space view separates the layers:



- There is a linear model at the very final layer
  $f(\mathbf{x};\theta) = f_L(\phi(\mathbf{x});\theta_L) = \sigma(\mathbf{W}_L\phi(\mathbf{x})+b_L)$
- There is a feature map that transforms the input
  $\phi(\mathbf{x}) : \mathbb{R}^d \mapsto \mathbb{R}^{d_{L-1}}$
- Automatically determined non-linear feature map

A Fully Connected Classification Network is a MLP model with multiple outputs (labels). The last layer of the network is constructed to make the network perform binary or multi-class classification, and is usually linear. Multi-label classification if we have K classes:

$$a_L = \mathbf{W}_L h_{L-1} + b_L \in \mathbb{R}^k$$
$$p(y=k) = f_k(\mathbf{x};\theta) = \frac{e^{a_{L,k}}}{\sum_{l' \in \mathcal{C}} e^{a_{L,k}}}; \sum_{k \in \mathcal{C}} p(y=k) = 1$$

A Fully Connected Regression Network is very similar, with the final layer usually being linear. For an M dimensional regression problem:

$$y \in \mathbb{R}^M \quad f(\mathbf{x};\theta) \in \mathbb{R}^M$$

Note: Since the information flow is *forward* when predicting, we have a *feed-forward network architecture*. It is important to choose the network architecture correctly for the best prediction accuracy, this is done on a validation set. Models can interpolate between samples they have seen, but they do not extrapolate well.

Training of Multilayered Networks

Three Sets: Training, Validation, Test

- **Training set**: determining the best model parameters

$$\mathcal{D}_{training} = \{\mathbf{x}_n, \mathbf{y}_n\}$$
$$\theta^* = arg_\theta min \sum_{n=1}^{N} \mathcal{L}(\mathbf{y}_n, f(\mathbf{x}_m;\theta))$$

We want the model parameters that minimize a cost for the training set. The size of the training set is important, large number of training examples are needed for models with larger number of parameters.

- **Validation set**: determining the hyper-parameters

$$\mathcal{D}_{validation} = \{\mathbf{x}_n, \mathbf{y}_n\}$$

The hyper parameters are
  - Architecture – number of layers, width, non-linearities..
  - Number of training iterations
  - Using/not using regularization, regularization coefficients
  - Batch size during training

Ideally there is no overlap with the training set.

- **Test set**: estimating model prediction (generalization) accuracy

$$\mathcal{D}_{test} = \{\mathbf{x}_n, \mathbf{y}_n\}$$

Model parameters / hyper parameters should **not** be changed based on test accuracy. Ideally should come from the same distribution as training and validation sets. Variations in distributions can reduce prediction accuracy. There must be absolutely **NO** overlap with the training or the validation set.

Note: Generalization accuracy computed on the test set will often be lower than on the training set. Too much difference between test and training accuracy points out to *over-fitting*, where the model fits to noise in training set that is by chance correlated with labels. For small datasets, one can use data augmentation such as bootstrap resampling etc.

Cost function

The sums are over the training samples

- Regression
  - Mean Squared Error

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}(y_n - f(\mathbf{x}_n;\theta))^T(y_n - f(\mathbf{x}_n;\theta))$$

  - Mean Absolute Error

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{M}|y_{n,j} - f_j(\mathbf{x}_n;\theta)|$$

- Classification
  - Binary classification

$$\mathcal{L} = \sum_{n=1}^{N} -y_n \ln f(\mathbf{x}_n;\theta) - (1-y_n)\ln(1-f(\mathbf{x}_n;\theta))$$

  - Multi-label classification

$$\mathcal{L} = \sum_{n=1}^{N}\sum_{\mathbf{k}\in\mathcal{C}} -\mathbf{1}(y_n = k)\ln f_k(\mathbf{x}_n;\theta)$$

Optimization

For both classification and regression we wish to optimize the weights and biases of the network.

Backpropagation

Let us use gradient based optimization – gradient descent

$$\theta_i^{t+1} = \theta_i^t - \eta\frac{\partial\mathcal{L}(f(\mathbf{x};\theta))}{\partial\theta_i}$$
$$\text{with} f(\mathbf{x};\theta) = f_L \circ \ldots \circ f_2 \circ f_1(\mathbf{x})$$
$$f_l = \sigma(a_l) = \sigma(\mathbf{W}_l f_{l-1} + b_l)$$

We can calculate the gradient via chain rule (e.g for 1 weight and one bias:)

$$\frac{\partial\mathcal{L}}{\partial w_{ij,l}} = \frac{\partial\mathcal{L}}{\partial f_{l,i}}\frac{\partial f_{l,i}}{\partial a_{l,i}}\frac{\partial a_{l,i}}{\partial w_{ij,l}} \quad \frac{\partial\mathcal{L}}{\partial b_{i,l}} = \frac{\partial\mathcal{L}}{\partial f_{l,i}}\frac{\partial f_{l,i}}{\partial a_{l,i}}\frac{\partial a_{l,i}}{\partial b_{i,l}}$$

The individual terms are equal to

$$\frac{\partial f_{l,i}}{\partial a_{l,i}} = \sigma'(a_{l,i}); \quad \frac{\partial a_{l,i}}{\partial b_{i,l}} = 1; \quad \frac{\partial a_{l,i}}{\partial w_{ij,l}} = f_{l-1,j}$$

Defining the error signal as

$$\delta_{l,i} f_{l-1,j} = \frac{\partial\mathcal{L}}{\partial a_{l,i}}$$

Gradients with respect to the error signal are

$$\frac{\partial\mathcal{L}}{\partial w_{ij,l}} = \delta_{l,i} f_{l-1,j} \qquad \frac{\partial\mathcal{L}}{\partial b_{i,l}} = \delta_{l,i}$$

If we calculate the error signal we find

$$\delta_{l,i} = (\sum_k \delta_{k,l+1} w_{ki,l+1})\sigma'(a_{l,i})$$

We can see that the error signal at layer l is a function of the error signal and the weights at layer $l+1$; the error signal starts at the final layer L and propagates backward. The error signals can be written in matrix form :

$$\delta_l = (\delta_{l+1,l}, \ldots, \delta_{l+1,d_{l+1}})\mathbf{W}_{l+1}\begin{bmatrix}\sigma'(a_{l,1}) & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \sigma'(a_{l,d_l})\end{bmatrix} \quad (4)$$

Stochastic Optimization

Training with a large number of samples, the forward and backward pass required for each sample for each update step would be very slow. Instead, at every step a random batch of training samples are chosen, and parameters are updated accordingly, summing over the batch and not the training set.

$$\mathcal{B} \subset \mathcal{D}_{training}$$

Notes on optimization techniques: The learning rate is crucial, often set by empirical tests on the validation set. Variations of the gradient descent technique exist, including momentum and second order gradient approximations. Also, different algorithms have different convergence properties with more or less parameter tuning.

Initialization

If we apply zero (constant) initialization, i.e

$$w_{ij,l}^0 = 0 \quad b_{i,l}^0 = 0$$

It becomes very problematic as we will have complete symmetry in the network. No matter what the input is, the values of the hidden units will be the same, and all the weights in one layer will get the same updates; effectively we will have a one neuron thick network. If only bias is initialized with 0, it does not cause symmetry problems.
The common approach is random initialization for the weights and zeros for biases. The distribution used can be normal or uniform, but should consider that the variance of a signal at the input and output of a layer should be similar:

$$w_{ij,l} \propto \mathcal{N}(0, \sqrt{\frac{2}{d_{l\pm1}}})$$

Avoiding Over-fitting

If we have the following network

$$\underbrace{10}_{\text{Input dimension}} \longrightarrow \underbrace{20 \longrightarrow 30 \longrightarrow 10}_{\text{Hidden dimensions}} \longrightarrow \underbrace{1}_{\text{Output dimension}}$$

So the total number of parameters is

$$10x20 + 20 + 20x30 + 30 + 30x10 + 10x1 + 10 = 1170$$

To determine the large number of parameters, we need a large number of samples. Overfitting is when the model has too many parameters and not enough training samples. The model can learn noise in the training samples, perfectly predict the training data, but won't be able to generalize. The best strategy is to have more data; one can also reduce the size of the network or regularize. Another strategy is *drop-out* where we randomly set some of the activations to zero during training.

Convolutional Neural Networks

Now the input x is no longer a vector but an image. This causes problems in the activation $a_l \mathbf{W}_l h_{l-1} + b_l$:

1. Fully connected links leads to too many parameters
2. Images are composed of a hierarchy of local statistics
3. Lack of translation invariance

We had projections $a_{l,k} = \sum_j w_{l,k} h_{l-1,j} + b_{l,k}$

- Each $h_{l-1,j}$ is a scalar and so is $a_{l,k}$
- Each h_l is a vector of neurons and a_l is a vector of activation
- There is a separate w_l, kj that is linking each neuron $h_{l-1,j}$ to each $a_{l,k}$
- High dimensions of h_{l-1} and a_l lead to high number of weights

Now we use convolutions $a_{l,k} = \sum_j w_{l,k} * hl-1, j + b_{l,k}$

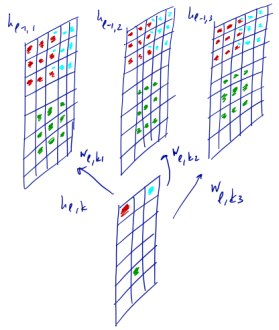
- Each $h_{l-1,j}$ is an image of neurons and so is $a_{l,k}$
- There is a separate convolutional kernel linking images $h_{l-1,j}$ and $a_{l,k}$. Same kernel applies to the entire image of neurons

- $h_{l,j}$ are channels
- $w_{l,kj}$ are convolutional filters

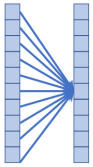
Nonlinearities following activations remain similar.

Number of Parameters

In the convolutional achitecture, image of neurons form a channel, and a vector of channels form a layer (each neuron in a fully connected link corresponds to a channel). We thus have a tensor representation of layers $d_l x N_l x M_l$. The same filter is used for all neurons in the same channel: **weight sharing**.



We thus have **larger** layers with **sparser** connections with lower number of parameters



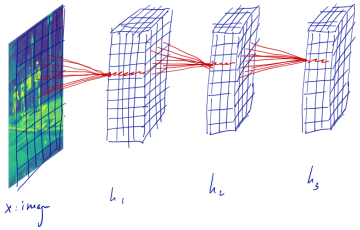
Fully connected	Convolutional
$x \in \mathbb{R}^{N_0 \times M_0}$	$x \in \mathbb{R}^{N_0 \times M_0}$
$h_1 \in \mathbb{R}^{d_1}$	$h_1 \in \mathbb{R}^{d_1 \times (N_1 \times M_1)}$
$\mathbf{W}_1 \in \mathbb{R}^{d_1 \times (N_0 \times M_0)}$	$\mathbf{W}_1 = \{w_{1,1}, w_{1,2}, \dots, w_{1,d_1}\}$
	$w_{1,j} \in \mathbb{R}^{k_1 \times k_2}$
	$(k_1 \times k_2)$: kernel size
	$\mathbf{W}_1 \in \mathbb{R}^{d_1 \times (k_1 \times k_2)}$

For example for a 3x3 filter, $k_1 = 3$ and $k_2 = 3$. The number of parameters per layer for convolutional filters is $d_1 x k_1 x k_2$, and for fully connected $d_1 x N_0 x M_0$. The number of neurons for convolutional filters is $d_1 x M_1 x N_1$, and for fully connected d_1. In a convolutional architecture, local neighborhoods are represented in a single vector, so you need a small d_1 to retain information. In a fully connected architecture, you need a very large d_1 to retain a similar level of information.

When the kernel is placed on the boundary, it is not well defined. We have two options:

1. **Valid convolution**: We only evaluate convolution when all the elements are defined. We thus lose some pixels and have $M_l = M_{l-1} - k_1 + 1$ and $N_l = N_{l-1} - k_2 + 1$
2. **Same padding**: We pad the boundaries (usually with 0s) so that the results of the convolution will have the same size. $M_l = M_{l-1}$ and $N_l = N_{l-1}$

Hierarchy of Local Statistics



Extracting task specific features from the images. As the layers progress, more global information is encoded (e.g layer 1: first order derivatives, layer 2: second order derivatives etc...)

Translation Invariance

For these two images, we want identical outputs for classification, but different outputs for localization and segmentation. These images will lead to very different activations in the hidden layer, but it is possible to teach a fully connected network to be invariant to translation.

The convolution operation is translation equivariant (i.e $f(T \circ x) = T \circ f(x)$), but not translation invariant (i.e $f(T \circ x) = f(x)$). The problem is that dimensionality reduction requires many parameters. A solution is *strides*, where instead of moving one pixel in each convolution step, we skip multiple pixels. The new channel size is thus $M_l = \frac{M_{l-1} - k_1}{s_1} + 1$ and $N_l = \frac{N_{l-1} - k_2}{s_2} + 1$. Although we are losing information, we want to keep information that is useful for the task. We do not gain directly translation invariance, but we do when there is a lot of them.

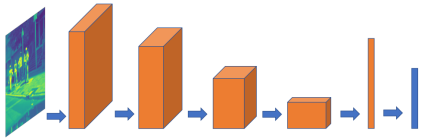
Pooling Layers

We want to reduce the size of the channel, and pooling accelerates that reduction. We pool information in a neighborhood. We represent the region with one number which summarizes the information. This is applied to each channel separately. The most commonly used version is max pooling, which is a nonlinear operation.

Max Pooling

We represent the entire region with the neuron that achieves the highest activation. This leads to partial local translation invariance, but not complete. It is often applied with strides equal to the size of the kernel. This leads to a substantial dimensionality reduction, and the bigger the pooling kernel, the bigger the reduction increase. Only the most prominent activation is transmitted to the next layer.

CNN Architecture



We see convolution, followed by nonlinearity, followed by max pooling. And then we have a fully connecte layer where we see a transformation followed by non-linearity. For the output, the number of neurons is equal to the number of classes. We also have a forward pass for prediction and a backward pass for computing gradients. Local features are extracted and aggregated throughout the network. The last layers "sees" the entire image and the features encode global information.

Historical Evolution

1998: first one. 2012: halved error rate in ImageNet Challenge. 2015: ResNet used residual modeling (i.e present layer is a function of the previous one + the previous one). 2016: classification error: 3%. GPU implementation speeded up the process. Very deep networks with different kernels (e.g Inception combines multiple convolutional filters of different sizes).

Extended use

Region Classification

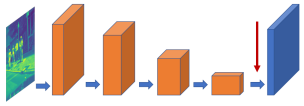
So far we have only seen image-wide classification. Before, if an object was in a cluttered scene, we could slide a window around looking for it, but this is a brute force approach with many local decisions. Instead of sliding windows, we can use **region proposal** by **selective search** and warp regions to a fixed size. We then use CNNs to extract features and SVM to classify. But this can be slow. It can be made faster by unifying the feature extraction and classification into one step.

Segmentation

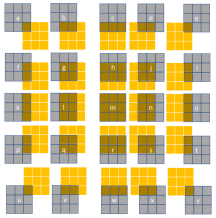
Classification CNN progressively reduces the size of the network but segmentation needs to keep the size constant. So we need to use pixelwise classification. There are two competing factors:

1. **Integrating larger context**: decision for each pixel should look at a large portion of the image to decide the semantic label. But this means information for neighboring pixels are very similar – difficult to disambiguate boundaries.
2. **Distinguishing neighboring boundary pixels**: network should be able to distinguish neighboring pixels – focusing on fine level details. Only focusing on fine level details cannot integrate the larger image context and has trouble determining the semantic labels.

The naive approach is to do the sliding window approach, but it can be slow in test time and not very good in boundaries Instead we can use a fully convolutional network.

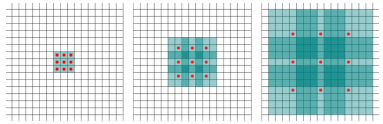


The penultimate layer is deep but has a very small channel size. The last, output, layer has a very large channel size – as large as the input image. There are two basic ways of doing this: transposed convolutions (see fig) and upsampling.

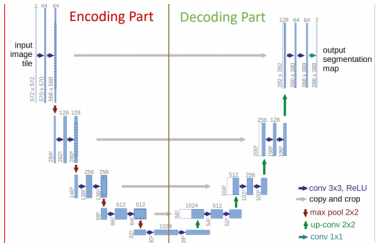


The penultimate channel must see large areas of the image so we can determine semantic labels for the pixels. Due to the preceding convolutions and pooling, each neuron in this channel sees a large area of the image – it has a **receptive field**. Receptive field grows progressively as we go through the network. However, we have trouble distinguishing between boundary pixels.

The first alternative is using dilated convolutions



The second alternative is using Skip Connections. The encoding part of the network is like the full convolutional network. It pools local information and achieves global representation at the bottom layers. The skip connections passes high-resolution information to retain information necessary to distinguish neighboring boundary pixels.



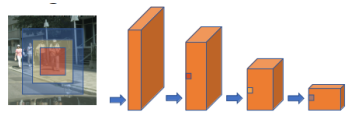
Other Possibilities

- **Data augmentation**: random transformation of the observed samples to augment the training set.
- **Transfer Learning**: start with weights of a trained network and retrain the network for the specific task.

Advanced Topics in Deep Learning

Visualization and Diagnostics

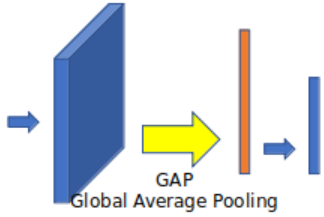
We wish to understand how a network works. For this purpose, we will visualize features (i.e go backwards to see what pattern in the input image caused a particular activation for a neuron in a particular layer). Indeed, different inputs will lead to activation at different levels due to the non-linearity. The size of the input pattern changes with respect to the receptive field, which is dependent on the layer.



In the forward pass, in order to compute the activation in a particular neuron we do three operations: convolutions with a set of filters, non-linearity with ReLu function, and max-pooling. Now we will run an input image forward and compute all the features; then keep the activation of the wanted neuron and set the rest to 0. Starting from the same layer, run the operations in reverse order: unpool, rectify, transposed convolution.When unpooling, we place the values to the respective positions and zero values are placed where activations are discarded during forward pass (in max-pooling we only keep the max locations). The end result is feature maps, where we can see that in each category the features are very similar, even in deeper layers.

Localization and classification

With slight modifications, classification networks can identify approximate locations, based on global average pooling idea. In a normal classification network, the last fully connected layers summarize the feature maps. We also have seen fully convolutional networks where we replace the final fully connected layers with image size outputs. Combining these ideas, using Global Average Pooling, we average all the information to a single number and then continue as usual.

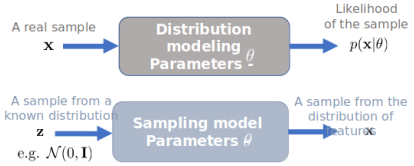


Per-class weighted sum of all the channels before global average pooling yields the class-specific activation map. Note that the network achitecture preceding the GAP layer can change. This method is a form of weak-supervision for localization, and has possible applications in medical imaging.

Unsupervised Learning

Supervised leaning is based on patterns between two types of data, examples have both kinds and the goal is to predict one from the other. We assume a mathematical model between features and labels, and estimate the parameters of the model to best predict labels from features in the training examples.

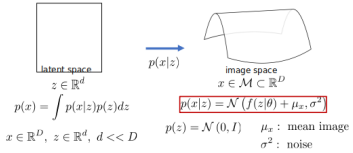
In unsupervised learning, there are no labels. There are two kinds: unsupervised learning of features, and of distributions. In the first one, we determine features in an unsupervised manner. In the second one, we use patterns within the data to describe the variability in the data and estimate its distribution; there is still a training dataset. We will focus on this latter one. The algorithm assumes a mathematical model for the future and ideally this is the probability distribution of the features.



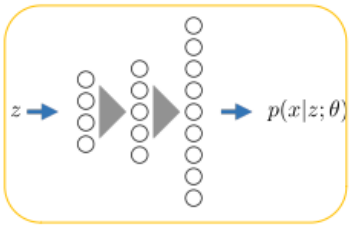
This is useful as we sample from the distribution of images to generate images. In other words, we generate an image (unknown distribution that we want) from a known distribution trained on all images, and sampled on a class specific way, e.g: give me a dog.

The most straightforward way is Kernel Density Estimation (KDE). We place a "kernel" around each training sample and determine the likelihood of a new sample based on these kernels. If kernels depends on euclidean distance, then likelihood is related to the distance in euclidean space. But the KDE is a bad idea due to the dimensions of images which are often quite large. For the KDE to work, we need to "fill" the space (e.g of 4096 dimensions if the image is 64*64).

So we assume that images live in a lower dimensional subspace and we build a mapping between this latent space and the image space. Assuming the mapping is a linear one, we have a clear link to PCA. However, whereas in PCA the eigenvectors made sense (most prominent features), in latent space it doesn't necessarily. However, linear maps are not enough like in supervised learning.



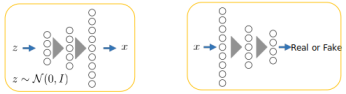
We can stack these models in density networks:



Two avenues are possible: Generative Adversarial Network (GAN), where we generate a sample from the distribution of features with input: a sample from a known distribution; and Variational Auto-encoders, where we find the likelihood of the sample with input: a real sample. The latter builds on density networks concept but instead of Monte-Carlo uses variational infence with a network parameterized samping (approximate) distribution. We maximize the evidence lower bound instead of real likelihood.

To the left is the encoding model which takes an image and maps it to the posterior distribution in the latent space (we encode to the lower dimensional space). To the right is the decoding model which takes the lower dimensional representation and maps to an image. It can be used as a sampler of a reconstruction tool. Both are gaussian models. The difference with the PCA is that we have a lot less components.

Now, instead of an explicit probabilistic model,a GAN is a sampling tool that generates samples from the data distribution. To the left is the generator (G) which generates realistic looking images from random samples in the latent space. To the right is the discriminator (D) which tries to classify images into two categories: real or generated. During training they compete as we try to optimize both.



$$\min_{\theta} \max_{\phi} \mathbb{E}_{x \sim p_{\text{real}}}(x) \left[\ln D(x;\phi)\right] + \mathbb{E}_{z \sim p(z)} \left[\ln(1 - D(G(z;\theta))\right]$$

Appendix

Orthogonality Condition for Functions

Two (possibly complex) square integrable functions $\phi_i(x)$ and $\phi_j(x)$ are called *orthogonal* on the interval $[a, b]$ if their inner product is zero. A common definition of the inner product for such two functions on $[a, b]$ is

$$\langle \phi_i, \phi_j \rangle := \int_a^b \phi_i(x)\phi_j^*(x)\,\mathrm{d}x$$

Here $(\cdot)^*$ denotes the complex conjugate. The trivial function $\phi(x) = 0$ is clearly orthogonal to all other functions, but we are only interested in orthogonal functions that also satisfy

$$\int_a^b |\phi_i(x)|^2 = \alpha_i,$$

with $\alpha_i \neq 0$ some positive, real number. Particular interest for this course lies in sets of *orthonormal* functions such that

$$\int_a^b \phi_i(x)\phi_j^*(x)\,\mathrm{d}x = \delta_{ij},$$

with δ_{ij} the Kronecker delta, i.e., $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise. Note that sets of orthogonal functions can be modified to sets of orthonormal functions via a simple scaling factor of $1/\sqrt{\alpha_i}$

Completeness Condition

; As vectors can be characterized by their projections on the basis vectors, square integrable functions can be characterized by their correlations with the basis set of orthonormal functions. In the continuous case, such characterization needs not be unique though. There are orthogonal basis sets of functions that do not allow unambiguous descriptions of the original function via such a projection mechanism. Therefore, the orthonormal functions sets should also observe the *completeness condition*, i.e., one should make sure they allow such unambiguous description. As an example, the set of orthogonal functions $\cos(m\omega x)$ is orthogonal but not complete, as the linear combination of even functions can never produce odd functions. For discrete problems this completeness issue vanishes, as a function can be represented as $(N \times 1)$ sample vector and any combination of N orthogonal basis functions will be a complete basis.

Fourier Orthogonality Relations

Orthogonality of Sine and Cosine

It can be shown that the following holds:

$$\int_{-L}^{L} \cos(nx)\cos(mx)\,\mathrm{d}x = \begin{cases} 0 & m \neq n \\ L & m = n \neq 0 \\ 2L & m = n = 0 \end{cases}$$

$$\int_{-L}^{L} \sin(nx)\sin(mx)\,\mathrm{d}x = \begin{cases} 0 & m \neq n \\ L & m = n \neq 0 \\ 0 & m = n = 0 \end{cases}$$

$$\int_{-L}^{L} \sin(nx)\cos(mx)\,\mathrm{d}x = 0$$

Orthogonality of Fourier Basis Functions

The Fourier transform is based on a decomposition into a set of functions $\mathrm{e}^{\mathrm{i}2\pi ft} = \cos(2\pi ft) + \mathrm{i}\sin(2\pi ft)$, *with different frequencies* f. According to the orthogonality of sines and cosines, these functions form an orthogonal basis.

Linear Operators

An operator L is said to be linear if for every function f and g and all scalars $a, b \in \mathbb{R}$ the following holds:

$$L\{af + bg\} = a \cdot L\{f\} + b \cdot L\{g\}$$

Eigenfunctions and Eigenvalues

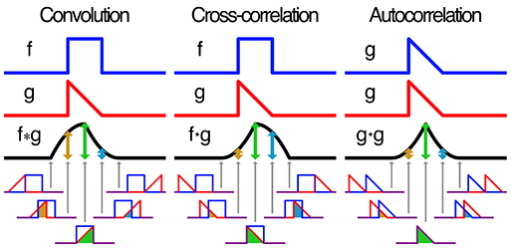
Given a linear operator L the eigenfunctions of L are the solutions to the equation

$$L\{f\} = \lambda f.$$

In words: The output of the linear operator L given one of its eigenfunctions f_e as an input is simply a scaled version of the input f_e.

Cross– and Autocorrelation

Convolution, crosscorrelation and autocorrelation are closely related to each other.



Crosscorrelation

Crosscorrelation is a measure of similarity of two series or functions as a function of the lag of one relative to the other. The definition in the two–dimensional case is

$$\phi_{ab}(x, y) = (a \star b) := \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} a^*(\xi - x, \eta - y)b(\xi, \eta)\,\mathrm{d}\xi\,\mathrm{d}\eta.$$

The difference with convolution lies in the direct use of $a(x, y)$ without prior reflection.

Autocorrelation

Autocorrelation is the crosscorrelation of a function with itself:

$$\phi_{aa}(x, y) = (a \star a) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} a^*(\xi - x, \eta - y)a(\xi, \eta)\,\mathrm{d}\xi\,\mathrm{d}\eta.$$

Informally, it's the similarity between observations as a function of the time lag between them. The autocorrelation of a function is symmetric, i.e., $\phi_{aa}(x, y) = \phi_{aa}(-x, -y)$, which implies an extremum at $(x, y) = (0, 0)$. More particularly, it can be shown that this is always a global maximum. Note that if $b(x, y) = a(x - x_0, y - y_0)$, then $\phi_{ab}(x, y)$ attains its maximum at (x_0, y_0).

Fourier Transforms

The Fourier transforms of the crosscorrelations and autocorrelations are often informative. They are called *power spectra* and are denoted as $\Phi_{ab}(u, v)$ and $\Phi_{aa}(u, v)$ respectively. If the Fourier transform of $a(x, y)$ is $A(u, v)$, then

$$\Phi_{aa}(u, v) = |A(u, v)|^2 = A^*(u, v)A(u, v),$$

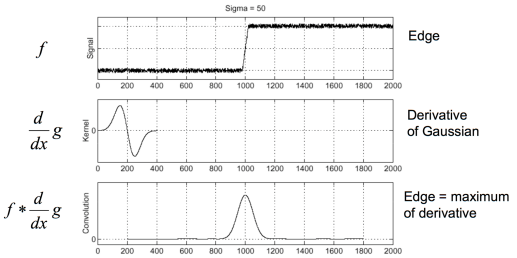
where $A^*(u, v)$ is the complex cojugate of $A(u, v)$. Note that Φ_{aa} is always real and shift–invariant, i.e., Translating the spatial pattern doesn't alter the power spectrum (it would only alter the phase).

Gradient Operators vs. Laplacian Of Gaussian

Both edge detection methods first smooth the image in order to better deal with noise. In principle, any smoothing mask will do, in practice often a Gaussian filter is applied. Moreover, both methods rely on derivatives for edge detection. Since the derivative and convolution with a smoothing mask are both linear, shift–invariant operations, they can be combined into a single mask. The most prominent advantage of the zero–crossings approach is that it can not only detect edges but also blobs. Informally, blobs are a cluster of pixels that are more or less homogeneous.

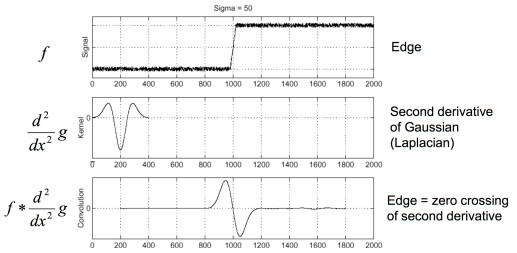
Gradient Operators

The gradient operator takes the first derivative of the smoothed image and detects edges where the response is largest.

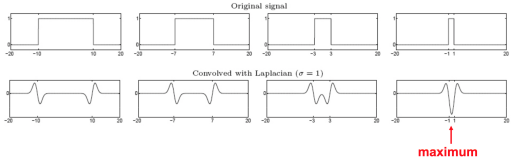


Laplacian of Gaussian

The zero–crossings approach takes the second derivative – the Laplacian – of the smoothed image and detects edges where the response has a zero–crossing.



For blob detection, not the zero–crossing but the peak value is used.



Note that this requires a normalized Laplacian response, meaning it has to be multiplied by σ^2.