Computer Vision – Summary

Please report any spotted mistake to: *jborn@uos.de* Last update: 06.11.2016

1. Introduction

Image Processing: Improving images to ease analyse by humans, filtering irrelevant information → compression

- Repair, compensate bad acquisition conditions, highlight information (improve perceptibility)
- Often requires computer vision

→ Changes images

Computer Vision: Recognition of parts of the image by the computer (→ Does not change images)

Subtasks:

- Feature Detection : regions, boundaries, attributes
- Motion Detection : Object/background, object tracking, direction & velocity
- Classification : colours, shapes, objects
- 3D : representation, reconstruction

Applications: Industrial quality control, character recognition, driver assistance, medical image processing, surveillance, robotics

Human Vision :

- Many processes like object recognition not yet understood, though 25% of brain deals with vision
  - Technical solutions only for special cases
  - Biological understanding only for special cases
- Understanding vision = result & foundation of computer vision research

Problem: Match <u>features of pixels</u> (luminance, colour, position) to <u>local patterns (</u>texture, corner, edges)

1. Analyse pixels to find local patterns
2. Analyse local patterns to find scenes

- We want 2 mappings: $image \rightarrow \ scene \quad \wedge \quad scene \rightarrow image$
- Often we need <u>context</u> of image to apply heuristics.
- Humans perceive context-sensitive and sometimes in strong contrast to real pixel values
- Images are interpretable as a non-continuous 2D function with input x, y to get value at position x, y
    - $I: [1:M] \times [1:N] \rightarrow [0,255]$

Hash-Code: Assign hash-code to binary 8×10 "image" that represents characters → Hash table holds meaning

- $h(f) = \sum_{pixel} f(x,y) \cdot 256^{y \cdot M + x}$ dimensions: M, N position: x, y Luminance: $f(x,y)$
- # Hash codes: $2^{10 \cdot 8} \approx 10^{24}$ → Useless
- Compressed hash code? Number of codes = number of different meanings (rest blend out)
    - How to get a hash function that maps similar meaning to similar code?

- Image interpretation impossible without expectation (context, task, assumption)
- We need an appropriate model that:
    - Provides knowledge about scene, objects,
    - Provides sufficient but limited degree of freedom for adaptation of real scene to expectation
- <u>Task</u>: Fit a model to data such that the image is most likely explanation
- <u>Procedure</u>: Fit model, validate results, correct model (loop)

Model: Its Purpose: Explain an image entirely form the scene. Model must contain:

- All parts of image (objects, persons, liquids, air, dust etc.)
- All light sources (+ geometry, location, direction, spectrum)
- Reflection properties of surfaces (also skin, liquids)

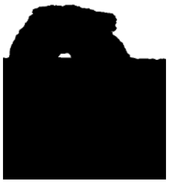
- **State of Art: Todays models cover low-level-features (close to signal) but no high level concepts as human use**
    - **Semantic Gap:** Hope of correlation between high level concepts and (evaluated) low level features
    - Also few advantages in low-level-feature-analysis:
        - Object recognition becomes robust against occlusions and viewpoints
    - $scene \rightarrow image$ possible. $image \rightarrow scene$ not
    - Example models:
        - **Image restoration** : input: noisy image. Output: estimated clean image (model generation)
        - **Image enhancement** : emphasize features of image → more pleasant (model perceptibility)
        - **Recognition** : try to detect features by models? (models of objects, persons)

Strategies of fitting models to data:

1. **Bottom up (data driven):** Start with data, search increasingly complex features and connections (edges, interpolations etc.) until they fit models
2. **Top down (model driven):** Develop model (based on expectation) → Try to find model within data. Possible: adaptation by a loop

<u>Example Pictures:</u> ⟶

Object-background separation using colours is not useful



2. Image Acquisition

Image acquisition:

Chronology of photography:

1822: first photo (Joseph Niepce, France)
1839: first commercial photo technique (Daguerreotype, Louis Daguerre, France)
1889: first roll film (George Eastman, USA)
1895: first cinema (Auguste & Louis Lumiere, France)
1908: first color-photography (Lumiere, France)
1920: first television (u.a. Vladimir Zworykin, Russia)
1939: TV in Germany (~ 500)
1970: first commercial CCD camera (100×100 pixel) (Boyle, Smith, USA)

Pinhole camera: Firstly mentioned by Aristoteles, used by da Vinci, Descartes, Locke

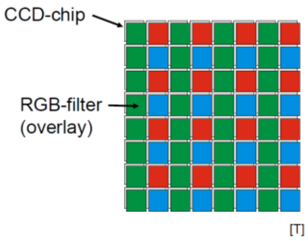
- Each object reflect light, goes through pinhole and hits the wall at the carton side opposite of the hole
- Image appears at wall upside down, same effect as in eye
- Need hole (= lens) otherwise each object point reflects to each point on wall → no virtual image
- Size of image at wall is inverse proportional to the distance of the object to the whole
  - The larger the distance the smaller the image (→ invers proportional)
- 3D-point (x, y, z) is mapped onto 2D point (x', y') $x' = f' \cdot \frac{x}{z}$, $y' = f' \cdot \frac{y}{z}$ f' =dist(wall,hole)
- The smaller the hole the sharper the image
- The smaller the hole the darker the image
- Digital cameras: Wall = CCD
  - CCD = Charge Coupled Device, light-sensitive chips, basis of all today's digital cameras
  - Pixel size: $5 - 20\mu m$

Human eye:

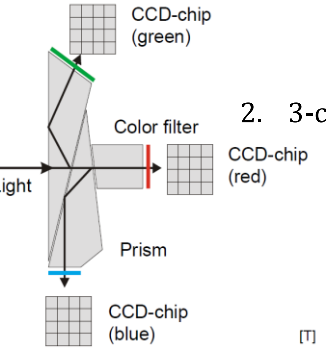
- Focusing by changing sharp of lens
- Iris controls brightness: opens in darkness, closes in brightness (protection)
- 2 light receptors
  - Rods, many, mainly in periphery, luminance separation, motion detection
  - Cones: low, mainly in fovea, colour perception, bright light necessary
  - Funfact: Cones are larger than rods
- Colour perception:
  - Visible wavelength for humans: $400 - 750nm$
  - 3 types of cones S(hort): Blue, M(iddle): Green/Yellow L(ong): Yellow/Red
  - Blue: 450nm, Green: 540nm, Red: 570nm. Overlapping distributions (trichromatic coding)

CCD-Camera: Colour coding

- To take nice photos, we need 3 colour values (RGB) for each image point
- 2 approaches:
  1. 1-chip-camera: (naïve)



- Light sensors are only sensitive to brightness: Put colour filter in front to get colour-sensitivity
- Arrange R-G-B-sensitive sensors side by side on a chip
- Overlay the sensors by using filters → estimate colour separations
- Seems to be unprecise but might yield equivalent results to 3-chip
- Half the filters = green, other half = blue + red (humans more sensitive in green spectrum)
- Compact, cheap
- Limited resolution, unprecise (colour recorded only at particular locations)



[T]

2. 3-chip-camera (advanced)
   - Put a prism between light source (lens) and sensors
   - 3 chips, one for each colour (RGB)
   - Prism separates light in dependence of wavelength and directs it to the respective chip
   - Better spatial resolution, each colour recorded at each location
   - Bigger, more expensive
   - Better in low-light or low-contrast situations. In normal images advantage becomes negligible due to compression

- CCD-chip are **analogue** (brightness for special colour = certain electrical charge)
- The signal is digitalized within the camera but not by the CCD-chip itself
- Transmission from (digital) information in camera to computer
  1. **Analogue:** Camera converts digital signal back to analogue, transmits it to the computer where it is again digitalized (loss of quality)
  2. **Digital:** higher quality

Image representation – Introduction in basic terms

Images (or scenes) have 3 types of information:

- Spatial information (pixel)
- Intensity information (colour)
- Temporal information

Modern sensors map intensity (analogue electrical information) onto a matrix (digital information)

Neighbourhood:

- Especially important for connectivity – pixels of connected regions have neighbours that are part of the object
  1. Squared Pixels Disadvantage: Neighbourhood is ambiguous
     - 4-neighbourhood = pixels with common edge are neighbours
     - 8-neighbourhood = pixels with common corners are *also* neighbours
  2. Hexagonal pixels Clearly defined neighbourhood

Digression: If we demand 8-neighbourhood for regions (i.e. foreground) than we implicitly assume a 4-neighbourhood for background pixels (and vice versa)

Distances: Often an important information

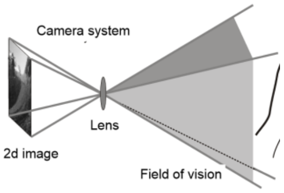
- There are several ways to define distances of pixels
- All types are well defined metrics from maths

1. Euclidean Distance : Real Life $D_e(x_1, y_1; x_2, y_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
2. Cityblock/Taxicab/Chessboard Distance: 4-neighbourhood $D_4(x_1, y_1; x_2, y_2) = |x_1 - x_2| + |y_1 - y_2|$
3. Chessboard Distance : 8-neighbourhood $D_8(x_1, y_1; x_2, y_2) = \max(|x_1 - x_2|, |y_1 - y_2|)$

Loss of information:

- There are 2 basic types of information loss we can separate:

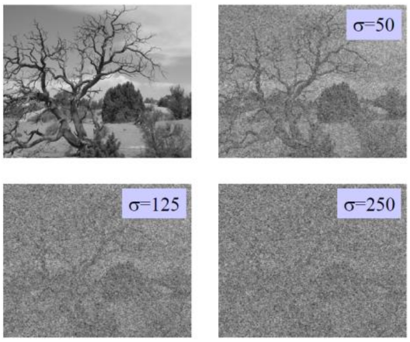
1. **Deterministic loss**
   - Unchangeable physical limitations
   - Problems that concern all pixels to the same degree
   - Causes of deterministic loss:
     1. **Projection & Sampling**. Some information is lost:
        - Occluded points
        - Depth information ($\mathbb{R}^3 \rightarrow \mathbb{R}^2$)
        - Points outside pyramid of sight
     2. **Motion blur**



[T]

        - During exposure camera moved by Δs
        - Point p on object now maps on many points on CCD-chip ($- \rightarrow\ ---$)
        - Intensity decreases (lower light incidence)
        - Motion blur is describable by linear filters (convolution)
    3. Over/Underexposure (Belichtung), **Focus**
        - Wrong focus yield inaccurate images or even misleads about information
            - e.g. in a geometric figure

2. **Stochastic loss** Not all pixels have the same deviation
    - Random variation of brightness or colour information in an image
    - Best example: **Noise** (non-deterministic loss)
    - Some pixels are disrupted, some not. Mostly unknown: **probability** to be distorted
    - Undesired co-product of image acquisition
    - 2 basic noise types:
        1. **Quantum/Gaussian noise**
            - Object photons hit CCD with certain probability.
            - Disruption is spatially independent but follows a normal distribution
            - Pixels of image = $g(x, y)$ are either
                - Undistorted : $g(x, y) = f(x, y)$
                - Distorted : $g(x, y) = f(x, y) + \eta(x, y)$ ← Noise! (η =Eta)
                    - Noise is **additive**
                - $P(\eta) = \frac{1}{\sqrt{2\,\pi\sigma}} \exp(-\left(\frac{\mu}{\eta}\right)^2)$
            - Caused e.g. by poor illumination
            - Reducible by spatial filters: Mean filtering (convolution), median filtering, Gaussian smoothing
            - But: This blurs sharp edges and fine details



    - **SNR =** Signal to Noise Ratio = strength of signal compared to noise
        - $SNR_{Max}(f) = f_{max}/\sigma$ Highest ratio is where undistorted signal is highest…
        - $SNR_{Avg}(f) = \frac{1}{MN} \cdot \sum_{pixel} f(m, n)\ /\sigma$
        - Mean pixel value divided by σ (standard deviation of noise)
        2. **Impulse noise (Salt & Pepper)**
            - Each pixel is disrupted with a certain probability
            - Value of disrupted pixel is independent from original pixel value
                - Either black or white (Pepper & Salt)
            - Might appear by converter errors
            - Reducible by median or morphological filter

Other image acquisitions: pixel value has interpretation different from grey value (luminance)

- Reconstruction of 3D scan – types:
    - PET – concentration of tracer (radioactive isotope attached to molecule)
    - MRI – provides contrast between different soft tissues
    - CT – computed from 2D X-Ray.

	X-Ray	**Reconstr. 3D scan**	**Reflection time image**	**Depth map/image**
Description	Projection of X-ray absorption	Types: PT, MRI, CT	Time to receive an echo	Distance to digital camera
Frequency	Monochromatic	Monochromatic	Monochromatic	Monochromatic
Resolution	High: >4 MP	Low: 0,02 – 0,25 MP	Varying	Good (like camera)
Value range	8-12 bit	8-12 Bit	8 Bit	8-16 bit
Problem	No depth info	Artifacts, noise	Artifacts	Depth errors

3. Basic Operations

- Basic Operators transforms image based on a specific algorithm – mainly to reduce noise & detect edges
- Those operators can be distinguished by properties like locality, linearity, number of input images etc.
- Usually the transformations imply loss of information
- If Information is preserved ⇒ ∃ inverse transformation (as e.g. in Fourier transform)

Functional/Output:

1. Image restauration & enhancement (see above):
    - Smoothing, edge enhancing, removing disruptions, transforming intensity etc.
    - Results in an image
2. Basic pattern recognition
    - Edge extraction, binarization, object/background separation (object *keeps* original value!)
    - Semantic level of image is not solely the grey values but the pattern it makes up

Input on pixel level: Operator O transforms grey value of input to a new value

1. **Point-Operators**: $g'(x,y) = O\big(g(x,y)\big)$ result only depends on input at same position
    - very rough object-background may be possible with PO. 2 Implementations:

    A. Function $O(x)$
    - **+** Easy representation, parameterizable (i.e. we may put in parameter to get value)
    - **-** repetitive computation (pixelwise) object/background
    - E.g. $g'(x,y) = a \cdot g(x,y) + b$ or $g'(x,y) = \theta(g(x,y) - \xi)$, ξ=threshold

    B. Lookup Table
    - **+** Arbitrary "functions" representable, fast execution
    - **-** high memory consumption (in small kernels like camera), no parametrization
2. **Local-Operators**: $g'(x, y = O(g(x,y), g\big(surr(x,y)\big))$ result depends on input + surrounding
    - Recognition of local patterns (details see below)
3. **Global-Operators**: $g(x,y) = O\big(g(all\ pixels)\big)$ result depends on *all* input pixels.

Input on image level:

1. **Monadic** $g \to g'$ 1 input image
2. **Dyadic** $g_1, g_2 \to g'$ 2 input images
    - Formula: $g'(x,y) = O(g_1(x,y), g_2(x,y), x, y)$
    - E.g. g_1: car, g_2: binary text. → $g'(x,y) = |g_1(x,y) - g_2(x,y)|$ To save readability

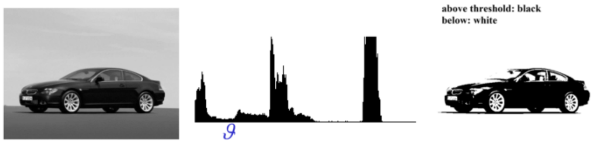
Transformation type:

1. **Linear vs. Non-Linear** Linear ⇒ Homogeneous
    - Linear = Homogen + additive → $O(a \cdot g) = a \cdot L(g) \quad \wedge \quad O(g_1 + g_2) = O(g_1) + O(g_2)$
    - Meaning: LO are invariant against scalar multiplication to image and against addition of images
2. **Homogeneous vs. non-homogeneous**
    - Homogeneous: **translation invariant**: The mapping is independent from location (like point operators) Application: E.g. Luminance - / Contrast enhancement
    - Inhomogeneous: $g'(x,y) = O(g(x,y), x, y)$ Mapping might depend on location

Histogram:

- Plotting grey value distribution of images (one bin per value)
- Bin value is number of occurrences of gray value in image
- $H(i) = \#p_i, i \in [0{,}255] \to p_i$ number of pixels with grey value i
- High ranges in histograms are good → better differentiation in observing image

For real scenes, thresholds are far more difficult to find:



Applications of Histogram:

- Useful to find thresholds(object background separation), K for K-Means, Hufman Coding (compression)

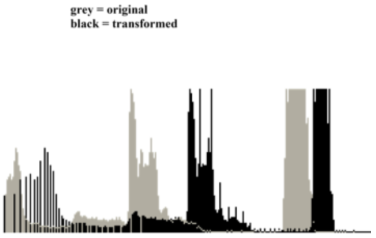
Object-Background-Separation: 2 Options, **point operator** both

1. Under threshold: black, Above: keep colour. $g'(x,y) = g(x,y) \cdot \theta(g(x,y) - \xi)$
   Under: $0 \cdot g(x,y) = 0 =$ black Above: $1 \cdot g(x,y) = g(x,y)$

2. Under threshold: keep, Above: white $g'(x,y) = 255 \cdot \theta(g(x,y) - \xi) + g(x,y) \cdot \theta(\xi - g(x,y))$
   Under: $255 \cdot 0 + g(x,y) \cdot 1 = g(x,y)$ Above: $255 \cdot 1 + g(x,y) \cdot 0 = 255$ =white

Logarithmic brightening:

- $g'(x,y) = a \cdot \log(s \cdot g(x,y) + 1)$ $s = \frac{1}{2}$, a =Normierung
- Better differentiation for low values (dark area), worse in bright areas
- No inverse function (though continuous, since grey values are discrete not real)



Dyadic Operators

Contrast enhancement:

- E.g. histogram equalization (i.e. pulling a small gray value range to the [0,255] scale)
- Object-background-separation:
  - *Naïve approach*: global (homogenous) threshold → imprecise, defective
  - *Better*: inhomogeneous (local) threshold → hardly to compute
  - *Solution*: delete the objects from the inhomogeneous background (not described how to do that), then remove this background by dividing pixelwise: g_1 / g_2

Temporal Smoothing

- Idea: Apply n times a specific noise t to a given image and finally divide the result by n to get smooth
- $g(x,y,t) = \frac{1}{n} \cdot \sum g(x,y,t')$ E.g. Salt & Pepper noise for 1% of pixels Noise: $\delta(x,y,t)$

Difference Images:

- Try to detect motion (separate moving objects from static background)
- Normal difference image: $g'(x,y,t) = |g(x,y,t) - g(x,y,t+1)|$
  - Bad: Shows only where the object was before and where it is now.
- Search for regions in DI: $g(x,y,t) = \theta(g_d(x,y,t) - \xi_1) \cdot \theta\big(g_{d_B}(x,y,t) - \xi_2\big)$
- Formula: Binarization of difference image minus threshold times binarization of difference image to background image (without objects!) minus threshold

Fixed camera, moving objects

[J]

Local Operators

- Def: Operator L transforms value at position (x,y)depending on surrounding $U(x,y)$ → homogenous
- **Applications**: Smoothing, noise reduction, gradient computation, edge extraction, pattern recognition
- Adding dependence on position (x,y) yields an inhomogeneous local operator (not handled at all)
- **Procedure**: Move operator window R over image and compute L at each location
  - Problem: Undefined values at the border (like in convolution) 2 Options:
    - Ignore border → no artifcats, easy, information loss for large R
    - Enlarge image with fix value like 0 → artifacts (FT), easy, no information loss
    - Periodic continuation of image → same. No artificial edges but still artifacts

Linear Local Operator: Almost all linear operators are local operators → LLO. Representable by **convolution**

Convolution

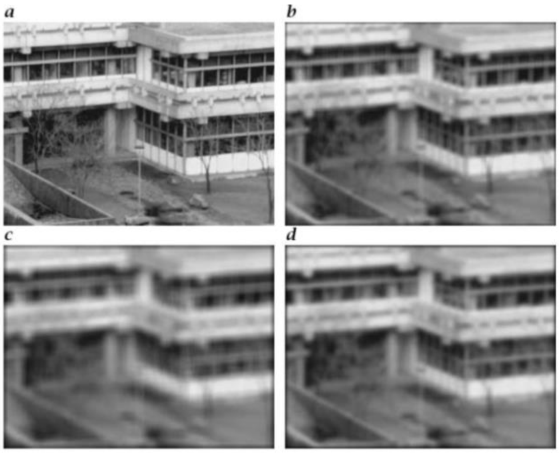
Visualisation

- Very famous homogenous linear local operator principle
- Problem: Pixel-based image restoration ignores relations between neighbouring pixels.
- Common notation indicating convolution: $(g * f)(x,y)$
  - Within this summary $*$ denotes convolution, $\cdot$ multiplication
  - Properties: Associative, commutative, distributive

- **Idea**: We define a certain kernel (i.e. a matrix of size $(2m+1)\times(2n+1)$ commonly 3×3 or 5×5)
- We slide this kernel over the image. Result is scalar product of the kernel with the same-sized image patch
- $\boldsymbol{g'(x,y) = \sum_{i=-m}^{m}\sum_{j=-n}^{n} k(m+i,n+j)\cdot g(x+i,y+j)}$ (Result has same size like image)
  - For real values we would use the integral. Discrete → Sum
  - We compute always the value for the pixel that is covered by the *center* of the kernel (odd size)
- Computational effort with $m\times n$ kernel is: $O(mn)$
- Some kernels (Gaussian, Sobel) are **separable** :
  - Meaning k can be computed by a product of a convolution of a row and a column vector
  - $k(i,j) = k^C(i)\cdot k^R(j) \quad\rightarrow\quad g'(x,y) = \boldsymbol{\sum_{i=-m}^{m} k^C(i+m)(\sum_{j=-n}^{n} k^R(n+j)\cdot g(x+i,y+j))}$
  - Use first row kernel, then column kernel (or v.v.) → reduces computational effort to $O(m+n)$

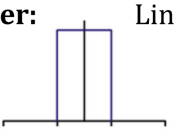
Filter Kernels: Kernels are simply matrices

- There exist many different kernels for applications like noise reduction, edge detection etc.
- Use small kernels. Large ones make computations hard and yield blurred results (especially at edges)
- If not explicitly indicated, they're all implemented by convolution



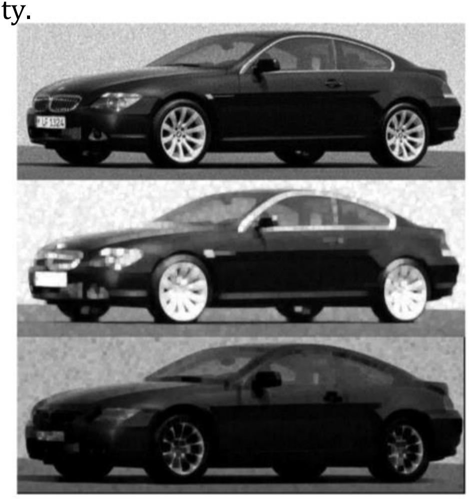
(a) Original; (b) 5x5 box filter; (c) 9x9 box filter; (d) 17x17 binomial filter.

Mean-Filter/Box(car)-Filter: Linear Noise reduction

- $\frac{1}{9}\cdot\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix}$ 
- same weighting for all pixels → yields bad results

Gaussian Filter: Linear Noise reduction

- Idea: Weight pixels around the center stronger than pixels farer away → Binomial kernels: Much better performance
- Family of filters based on binomial coefficients. Exists in 1D & 2D
- 1D: n-th order: $\frac{1}{2^n}\cdot nth$ row of pascal's triangle.
- Formally: $B^K = B^{K-1} * B^1$. $B^1 := \frac{1}{2}\cdot(1\quad 1)$
- 2D: n-th order: $\frac{1}{2^{2n}}\cdot nth$ row $\cdot\ nth$ row. Formally: $B^{K,K} = B^K * B^{K^T}$
- $\frac{1}{16}\cdot\begin{pmatrix}1&2&1\\2&4&2\\1&2&1\end{pmatrix}$
- Kernel is a discrete approximation of a 2D-gaussian: $\frac{1}{2\pi\sigma}\cdot e^{-(x^2+y^2)/2\sigma^2}$
- Hard borders of box filter smoothed by Gaussian multiplication
- Real gaussian filter is the result of a binomial filter with an order of infinity.

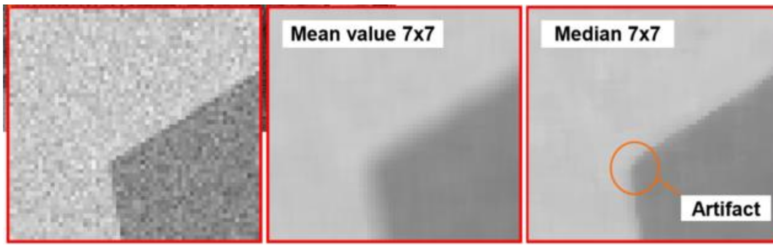


Min-Filter / Max-Filter: Non-Linear

- Blurs the image holistically, but reduces noise & preserves edges
- Kernel is not a matrix; but defines the area I to consider pixels:
- $g'(x,y) = \min(\max)\big(g(x+i,y+j)\big) \qquad i := [-m,m], j := [-n,n]$
- min destroys bright regions but preserve dark regions → image dims
  - opposite for max
- E.g. Salt & Pepper noise: min → Black dots enhanced, whites vanished
  - opposite for max

Median Filter: Non-Linear Noise reduction

- Combination of Min-&Max-Filter.
- Sort the grey values in area I.
- Take the **median** (usually ≠mean) as new value
- + Outlier-robust, edge-preserving
- - High computational effort for sorting

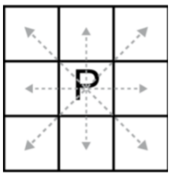


K-nearest-neighbour-Filter (KNN): (Non-Linear)

- Famous approach in Machine Learning, pattern recognition etc.
- In Computer Vision a hybrid of Box- & Median-Filter
- Nearest neighbour relates to distance of pixel values not to distance of pixels in image
- **Idea:** Take the centre pixel, compute the mean of the k values of the environment I, whose value is closest to those of the central pixel.
- Convention: $k > (2m + 1)(2n + 1)/2$
- High computational effort (sorting), low smoothing effect
- The larger the k the larger the smoothing effect, the higher the information loss

Symmetric-nearest-neighbour-Filter (SNN): (Non-Linear)

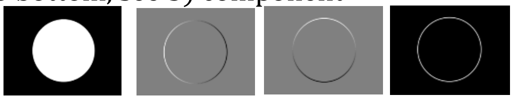
- Special computer vision approach to reduce noise (as usual)
- Similar to KNN, but less computational effort.
- Opposing pixels of 8-neighbourhood build pairs (tupels)
- Take from each pair the pixel closer to the centre (yields 4 pixels)
    - Compute **mean** of 4 pixels
- Possible for colour, but ordering in 3D impossible
    - Use the absolute value



Before
After

Gradient magnitude

- An image is a pseudo-continuous function – it has obviously discrete values
- Often we are interested in the *change* (jumps in grey values) of neighbouring pixels, so we need the derivative of this pseudo-continuous function (impossible due to discreteness)
- So, we approximate the derivative by a convolution with special kernels
- The resulting gradient image has huge applications (e.g. in border detection)
- The gradient has a horizontal (left to right, see 2) and a vertical (top to bottom, see 3) component
- Shows how much the grey values change in the respective direction
- Grey: no change, white: black → white, black: white → black
- Combine both components to get *gradient magnitude* (see 4)



<u>Conclusion</u>: Gradient requires derivative. Problem: Image is discrete ($\rightarrow \nexists$ derivative). Solution: Use convolution kernel(s) to approximate.

Gradient computation: Filtering image with one kernel for each orientation

- Measures change of intensity at a given point in a particular direction
- It is possible to select any 2 not-collinear directions.
    - Convention: x, y direction (handy)
- We can either use separated kernels (one for each direction, below) or use a combined kernel (e.g. Sobel Filter or Prewitt Operator)
- Horizontal direction: $D_X = [-1 \quad 0 \quad 1]$ Vertical: $D_Y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$
    - Apply convolution: $G_X \coloneqq I * D_X$
    - Gradient vector points in direction of largest intensity increase
    - Length of vector corresponds to rate of change in that direction

Figure 3.1 Initial Image

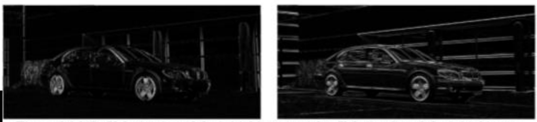


Figure 3.2 left: X-derivative of the initial image; right: Y-derivative of the initial image

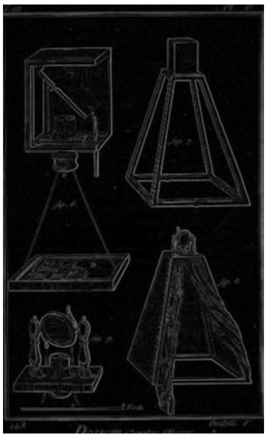
  - Sum up partial derivatives in a vector: $\Delta f(x,y) = \begin{pmatrix} G_X \\ G_Y \end{pmatrix}$, Δ denotes the gradient
  - Can be used for edge detection: Largest values in gradient image are best edge candidates
- Gradient direction (orientation): $\theta \coloneqq \arctan(\frac{G_Y}{G_X})$ direction of strongest change
- Gradient magnitude: $|M| \coloneqq \sqrt{G_x^2 + G_Y^2}$ intensity in direction of strongest change

Prewitt-Operator Edge Detection

- Vertical extension of the 3×1-gradient filter $(-1 \quad 0 \quad 1)$
- Formula: $\frac{1}{3} \cdot \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$
- Operator exists for both directions (x,y) and is separable into 2 vector operations (row +column)
- Output: 2 gradient images, one in x-, one in y-direction
- Good edge detection, but same weighting ∀ pixels (improved in Sobel)

Sobel-Operator: Edge Detection

- Technically: Overlaying Prewitt-Operator with Binomial Filter (weighting of grey values)
- Approximates the 1st derivative. Edges of image are minima/maxima of Sobel image →
- Practically: Like Prewitt-Operator but weight the central column/row (here row) double
- Exists in 4 directions and finds edges corresponding to that direction
  → not direction invariant
- $\frac{1}{4} \cdot \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$ For Horizontal-direction.
  - For vertical: Transpose.
  - For diagonals: make 0 in diagonal.
- Better than Prewitt → Applied in normal software like Photoshop
- Disadvantage: Might miss some edges (False Negatives)



Isotropic derivative Filter: Isotropy = direction independence

- Finds edges independent of direction
- Second order derivative filter
- 1D-kernel exists: $(1 \quad -2 \quad 1)$ but can be extended to 2D: Laplace Operator

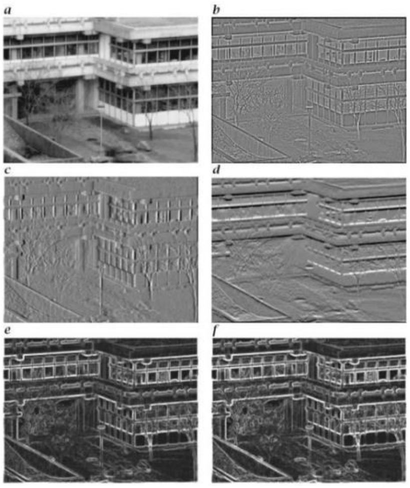
Laplace-Filter: Edge Detection

- Motivation: Continuous Laplace-Operator: $\Delta g \coloneqq \frac{\delta^2 g}{\delta x^2} + \frac{\delta^2 g}{\delta y^2}$ → Approximates 2nd derivative
  - Laplace Operator computes sum of diagonal in Hesse Matrix (General notation: Δ^2)
- Discrete: $\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$
- Advantage comparing with Sobel: Direction invariant (finds edges independent from direction)
- Edges are zero-crossings in 2nd derivative (value min/max in 1st derivative)
- Generally the edge detection is better than in Sobel, but the signal is noisy, i.e. it is hard to differentiate between real edges in noise and the most common threshold definition generate an overset of possible edges (False Positives)
- To avoid this preprocessing might be useful → **Laplace-Of-Gaussian-Filt.**
- ∃ Special version to detect also diagonal edges (45°)
- Horizontal derivative → vertical edges (+ v.v.)

Laplace-Of-Gaussian-Filer:

- Special case of Laplace-Filter that tries to avoid the noise-sensitivity of the normal Laplace-Filter

- Can be used to preprocess images before using Laplace Filter
- Can also be used instead of using a Gauss-Filter first and subsequently using the Laplace Filter
- Commonly a larger matrix (7×7) than Laplace-/Gauss Filter
- The LoG Filter can itself be approximated by a Difference of Gaussian (DoG) as in SIFT (chapter 12)



a) Original
b) Laplace filter applied
c) Horizontal derivative D_x
d) Vertical derivative D_y
e) Gradient magnitude of c) and d): $(D_x^2 + D_y^2)^{1/2}$
f) Sum up values of c) and d $|D_x| + |D_y|$

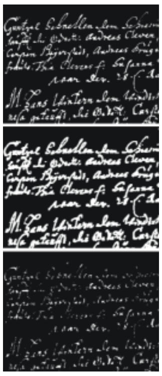
[J]

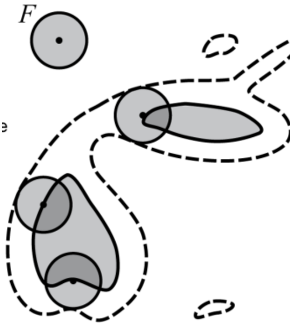
4. Morphological Operators

- Morphological operators try to remove errors in segmented images (e.g. object-background images)
  - Applications: disruption removal, smoothing of segmented regions, simple pattern recognition, remove unwanted segment connections, holes in boundaries
  - Skills: MO can change shapes and search for specific shapes
- Erosion & Dilation are basis for all further morphological operators
- MO are: **Homogenous** (translation invariant) & **Nonlinear**
- Recall LO: Structuring element was a kernel; in MO we have a *binary kernel*. Centre pixel = *anchor point*
- MO corresponds mostly to one of the set-operations unification ∪ , intersection ∩ (Boolean Operations)

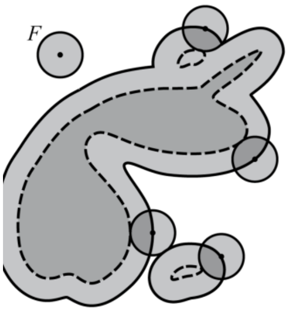
Some basics of morphological operators:

- Defined for binary images (object $\coloneqq 1$, background $\coloneqq 0$) but extendable to gray value images
- Recall: $1 \cup 0 = 1$ but $1 \cap 0 = 0$ (use either set theory with ∩,∪ or Boole with ∨,∧)
- $B \coloneqq$ binary kernel, normally $ones(m,n)$ m,n odd
  - Gaps of a particular shape may be filled (e.g. stripe of specific angle in a text)
- Computation is pixelwise like in convolution

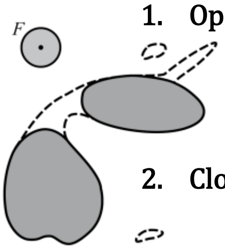




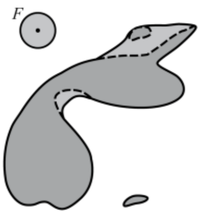
1. **Erosion** reduces segments (cuts off edges of objects)
   - Notation: Short: $g' = g \ominus B$ (minus → remove)
   - Long: $g'(x,y) = \bigcap_k \bigcap_l g(x+k, y+l) \wedge B(k+m, l+n)$
   - Assign 1 to output pixel ⇔ $\forall$ pixel p of patch: $B(g(p)) = 1 \rightarrow g(p) = 1$
   - Smoothes the boundary of large objects, remove small (unimportant) objects
   - Enhance holes in objects
   - Adaptable to greyscale images by replacing ∩ with min
2. **Dilation** enhances segments (traces edges of object)
   - Notation: Short: $g' = g \oplus B$ (plus → add)
   - Long: $g'(x,y) = \bigcup_k \bigcup_l g(x+k, y+l) \wedge B(k+m, l+n)$
   - Assign 1 to output pixel ⇔ $\exists$ pixel p of patch: $B(g(p)) = 1 \wedge g(p) = 1$
   - Marks object boundaries, unify close objects, enhance small structures, close background-holes
   - Adaptable to greyscale images by replacing ∪ with max
   - $\oplus$ does not denote the XOR-Operation



Both, erosion & dilation change the size of objects (bad) but smooth noise (dilation remove disrupted pixels in foreground, erosion in background). We concatenate both to keep the object size.

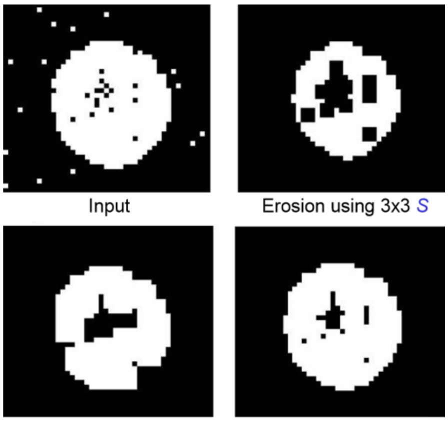


1. **Opening:** $g \circ B \coloneqq (g \ominus B) \oplus B'$ $B' \coloneqq B$ point-mirrored at anchor point (you may also use B here)
   - First apply erosion to delete small objects, then apply dilation to enhance remained objects
   - Remove noisy pixels in background without making object smaller
   - Opening: *Enhance* (open) noisy pixels in object without making object smaller
2. **Closing:** $g \cdot B \coloneqq (g \oplus B) \ominus B'$



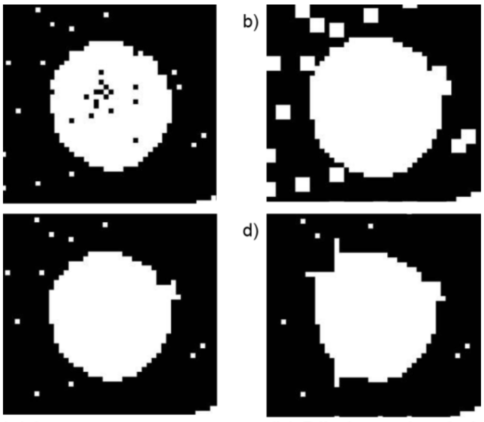
- First apply opening to remove background noise, then apply erosion to reduce the enhanced objects
- Closing: Remove (close) noisy pixels in object without making object larger

Examples:



Input   Erosion using 3x3 *S*

Opening using 5x5 *S*   Opening using 3x3 *S*



(a) Input, (b) Dilation using 3x3

(c) Closing using 3x3 *S*, (d) Closing using 5x5

Boundary extraction: $g_{bound} = g\backslash(g \ominus B)$ If g is binary: $g\backslash(g \ominus B) = g - (g \ominus B)$

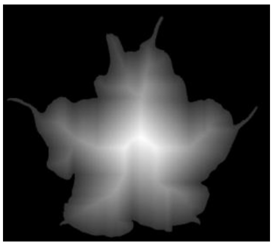
- No division, but complement operation
- Boundaries carry relevant information about an object (shape etc.) → interesting to extract them!
- Input: object + boundary. Apply erosion → object without boundary. Compute xor to get boundary
- To extract boundary based on 8-neighbourhood use "normal" structuring element B ($ones(3,3)$)
- 4-neighbourhood boundary: $B := \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$

Boundary extraction →

This is current driver circuit.
Phil.

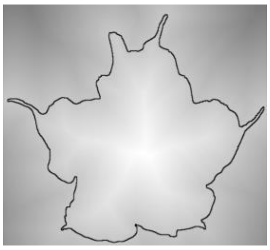
Distance transform

- g_{bound} yields boundary of object (i.e. pixels with distance 0 to boundary)
- Applying the g_b operation consecutive n times we get pixels with distance n to boundary
- So, the boundary extraction formula is a special instance ($n = 0$) of the general formula:
    - $g_b^n = (g(\ominus B)^n)\backslash(g(\ominus B)^{n+1})$ $(\ominus B)^n =$ n consecutive erosions applying B
- The result of a distance transform is a **distance image**
    - binary input g → greyscale distance image D
    - Distance image is obtained by unifying all single boundary images g_b^n
    - Each iteration gets a unique grey value → brighter the farer from boundary
    - Formula: $D = \bigcup_n n \cdot g^n$ (point wise multiplication to weight distance)
    - If necessary, normalize values to range [0,255]
    - Intensity (grey value) of all pixels of the distance image indicate cityblock-distance.



Generalized distance transform/image:

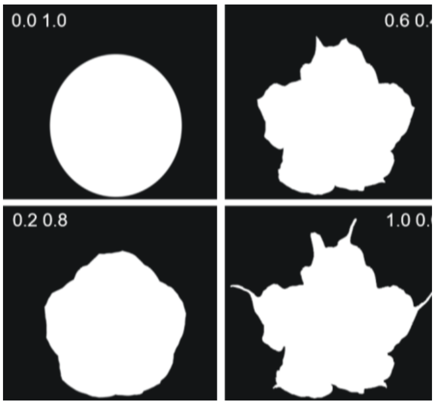
- Additionally compute distance to boundary for all background pixels
- Not only $b \rightarrow D_1$, but also inverted input $b^I \rightarrow D_2$ Then: $D = D_1 - D_2$
- Note I: Distance images are greyscale images, not binaries anymore
- Note II: Inverted image $b^I \neq b^{-1}$ = inverted matrix b



Morphing:

- Generalized distance transform allows to transform a binary image b to a binary image b'

- **Idea:** Compute distance image b_D, b_D' of both images, add them by a specified weighting and binarize back with a suitable threshold
- Allows for theoretically infinitely smooth morphings b_i (depending on the choose of N) The i-th morph is:
  - $b_i = \theta(\frac{i \cdot b_D + (N-i) \cdot b_D'}{N}) \quad \Rightarrow \quad b_0 = b_D' \quad \wedge \quad b_N = b_D$
  - Binarization operator θ is the inverse operation of the generalized distance transform

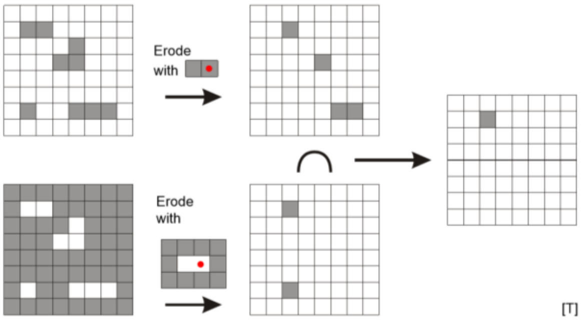


Using Morphological Operators to search shape:

- Specific shapes can be searched by adjusting the binary kernel B (normally $ones(m,n)$)
- Changing B (i.e. filling 0 at some positions) not only allows to e.g. fill gaps in text but also to find patterns

Hit-or-Miss-Operation: $\otimes$

- Detect all image patches with *exact* shape of compared shape S
- 2 steps (Hit & Miss)
  1. Find *candidate locations* by eroding with S. Result holds positions with *foreground(!)* of shape S
  2. Remove impossible locations by eroding inverted image b^I with the inversion S^I



[T]

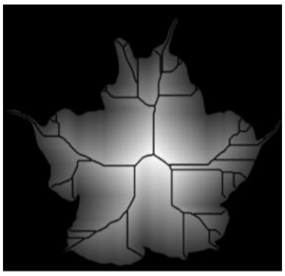
- Intersect the *hit* with the *miss* result to get all anchor points of the pattern S
- Formula: $b \otimes (S, S^I) := (b \ominus S) \cup (b^I \ominus S^I)$
- *Hit-* & *Miss*-Operator (S and S^I) can normally be merged. Here: $\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Improvement:

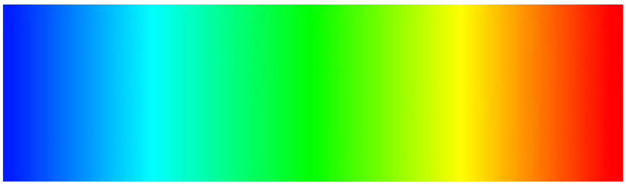
- To additionally detect structures *similar* to our target shape S, we can adjust the robustness by keeping the *hit-* and *miss*-operator different and/or dynamic. Easy example: $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & X & 1 & 1 & 1 & X & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$
- This operator searches horizontal lines of length {3,4,5}
- Generally: Increasing the tolerance to the size of the target S leads unfortunately to additional detection of large objects that *include* the target S (e.g. large disrupted circles)

Skeletonization: yields compact shape description of an object

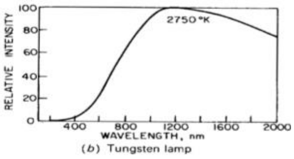
- We need a compact, meaningful shape-description of a binary segment to facilitate classification
- Skeleton = Set of all centers of *largest* circles *within* the segment that touch the boundary at at least 2 points
- Skeletons are always branched and connected
- The skeleton is the set of ridges of the distance image (see above)
- How to compute this skeleton?
  - Since the ridges are normally unknown, we use *hit-or-miss*-operation
  - Computation is iteratively:
    - Remove all not-ridge pixels (darkest pixels?)
    - Apply 8 *hit-or-miss*-operations to remove boundary pixels for 8 directions
- This principle is extendable to greyscale images (with a binary structuring element)



5. Color



- Visible range of light-waves for human eye: $380 - 780nm$
  - Blue Light: short wavelength, high energy
  - Red Light: long wavelength, low energy
- We perceive objects red iff they *reflect* red, the rest is absorbed

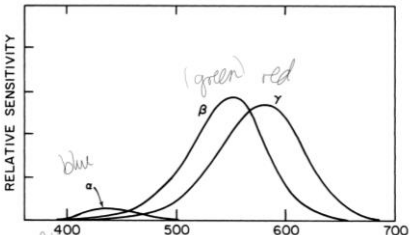


(b) Tungsten lamp

- Reflection of red, green and blue is *independent*
- Reflection depends not only on wavelength but also on material, angle of incidence etc.
- Different light sources (sun, lamp, laser, neon) emit light in different spectra
  - Some spectra are continuous: Tungsten Lamp, diodes
  - Some spectra are lines/discrete: Helium-Neon, Mercury arc

Colour vision in beings:

- Humans are *trichromatic* with 3 types of cones S: Blue, M: green/yellow L: yellow/red
  - Overlapping sensitivity (best in range: $530 - 560nm$ i.e. green/yellow)
  - Reason for best sensitivity: Much blue in natural light
  - Reason for overlapping: guarantee unambiguitiy of colour perception
  - Otherwise impossible to distinguish *some* clear red from *much* unclear red



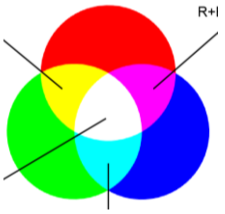
- Many beings have no colour vision at all, birds are *tetrachromatic* (ultraviolet)
- Monochromatic: light with exactly one wavelength (not perfectly possible)
- The visual input is nearly infinite, but reduced to 3 values (3 cones)
  - $\mathbb{R}^\infty \rightarrow \mathbb{R}^3$ $\phi_i(\lambda) \coloneqq$ receptor sensitivity, $C(\lambda) \coloneqq$reflected range, $i \in \{S, M, L\}$
  - Formular: $\int \phi_i(\lambda) \cdot C(\lambda)\, d\lambda$
- Metamerism: 2 colours appear identical under one spectrum but different under another (→ Different light spectra may cause same stimulation of cones, called valence)

Color spaces General

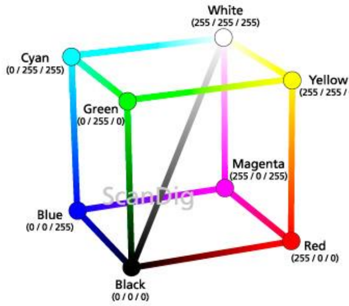
- Color spaces are vector spaces with mostly 3 dimensions to describe the area of perceived colours
- The target is, that each colour is mixable by primaries: $C = xP_1 + yP_2 + zP_3$ $x = y = z$ $\Rightarrow$ grey
  - $x + y + z = 1$ colors have identical intensity independent from x, y, z $\Rightarrow$ x describable by y, z
- The 3 primary colors should be orthogonal (i.e. it stimulates only *one* receptor type).
- This is impossible due to overlapping cone spectra → some perceivable colours are not realizable by primaries.

Specific color spaces:

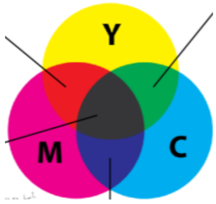
1. **RGB-Color-Space:** Red-Green-Blue
   - Additive colour-space with 3 primaries based on trichromatic theory (defined after CIE-Experiments)
   - Additive since pixels are self-luminous → Used for screens
   - Values in RGB-Cube (3D) in [0,1] or [0,255].
     Additive mix (RGB, light) →



RGB-Cube →

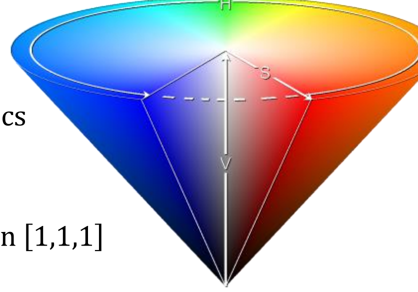


2. **CMYK-Space** Cyan-Magenta-Yellow-Key
   - Subtractive color mix: Colors are not self-luminous (printers)
   - E.g. Cyan: absorbs red. Rest (green + blue) is reflected.
   - Technically Key is not needed but mixable by CMY. Practically it is cheaper to use Key
   - Complementary to RGB → Transformation from RGB to CMYK or v.v. is easy:
     - $C = 1 - R$ $M = 1 - G$ $Y = 1 - B$ → Subtract complementary color from 1



3. **HSV-Space:** Hue, Saturation, Value Most important in CV Color-Cone
   - Very different from all other spaces → dimensions of color spaces are not colors

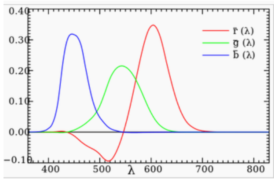
1. Hue: Angle on color circle: 0° = red 120° = green 240°= blue
2. Saturation: 0% no colour 100% pure colour
3. Value: (brightness) 0% darkness 100% brightness



- Advantage: More intuitive to mix ad-hoc than RGB due to well defined semantics
- HSV-space can be represented in RGB-cube (unpractical)
  - Hue are planes with different angles that go through [0,0,0] and [1,1,1]
  - Saturation are cones with peak at [0,0,0] that become wider in direction [1,1,1]
  - Value are planes orthogonal to the diagonal through [0,0,0] and [1,1,1]
- Cylindrical representation is redundant: low brightness → saturation & angle can be ignored → cone

4. Lab-Space: Famous in computer graphics.

- Advantage: Describes all perceivable colors
- Colors are defined independently from light source on standardized observers and brightness conditions
- *All* colors (also theoreticals) are mathematically defined → unambigious → global code
- Advantage: Intuitive distance: Perceived distance = Euclidean distance in Lab-Space
- Parameters: L ≔Luminance, Color-dimensions: **Green-Red** & **Blue-Yellow**



5. **CIE-RGB-Space:** (Commission internationale de l'éclairage)

- Find relation between physical & perceived colors
- Developed 1931 but still most used system to describe range of perceived colors
- Test: Let subjects mix 3 monochromatic primary colors to match a presented color

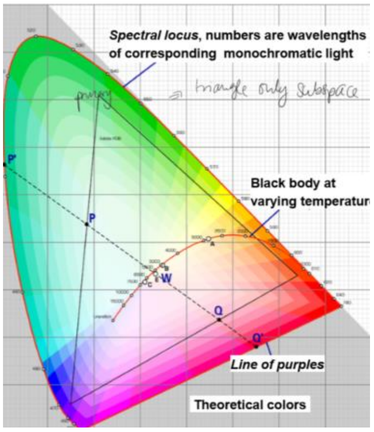
  Result: Sometimes impossible. Orthogonality isn't enough to reach any point in $\mathbb{R}^3$: Reason: $0 < x, y, z < 1$
  Then an increase of the intensity of one primary (red) of the presented color was necessary (see graph, increase of primary = decrease presented under 0)

Conclusion: Some turquoise-tones are not representable in full saturation.

6. **(CIE) XYZ Space:** Historically the precursor of the CIE-RGB space.
   - Not practical for application (artificial, not representable colors)

Background: CIE Chromaticity Diagram:

- 2D-representation of 3D-colour space (z is computed by $x + y + z = 1$)
- Spectral locus: *pure* colors, maximal saturation
- Line of purples: Maximally saturated colours that are perceivable but not mixable
- $W ≔ x = y = z = \frac{1}{3}$ P' and Q' are opponent colors


Spectral locus, numbers are wavelengths of corresponding monochromatic light
primary
→ triangle only subspace
Black body at varying temperatur
P'
P
W
Q
Q'
Line of purples
Theoretical colors

Colors in CV: Widespread applications! Used for low-level-processing

- Segmentation (partitioning colors), object detection, e.g. skin-detection,
- But not for shape recognition → Better use brightness (resolution in human eye), coloring book theory: shape is recognized independent from color in V1
- Sometimes criticized: Color contributes to structure with 40%, coloring book theory questionable

6. Segmentation

Def: Partitioning an image into multiple segments. **Target**: make image more meaningful & easier analysable

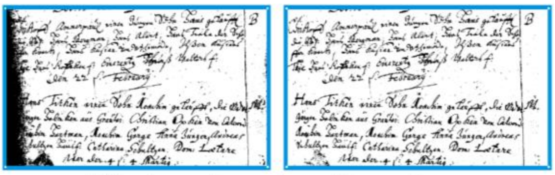
Basis: Complete – connected – no overlap:
Means: Each pixel is assigned to exactly one segment and segments are connected regions

Segmentation may be combined with user interaction to solve complex tasks but is still a low-level step in CV

Histogram-based segmentation: Use histogram in binary segmentation to find a suitable threshold
Problem: non-uniformed brightness → no global threshold input image: $g(x, y)$

Solution 1: Multiply shading (background) image with original. Then: binary segmentation possible

- Background image $h(x,y)$:
  - Background are large connected regions
  - Region needs to contain at least 1 fore-/background pixel
- $g'(x,y) = \frac{g(x,y)}{h(x,y)}$ Then normalized, then binary segmentation



Without correction / with correction

Solution 2: **Locally varying threshold**

Idea: Decompose the image into fields and compute a binary threshold $\theta(x,y)$ **for each pixel** based on the histogram and the neighbouring fields

1. Compute binary threshold $\theta(x,y)$ for each field. Assign threshold to center of field (if bimodal)
   i. If histogram is not bimodal → fields are too big, define fields smaller
   ii. If unimodal → assign no threshold (unclear whether fore-/background)
2. ∀ fields without threshold: Find 4 closest fields t_i with distance d_i assigned threshold and compute missing threshold by *bilinear interpolation*. $\theta(i,j) = \sum(\theta(t_i) \cdot d_i) \,/\, \sum d_i$
3. Centers of fields now have threshold, rest not: Compute $\theta(x,y)$ ∀ missing values by step 2

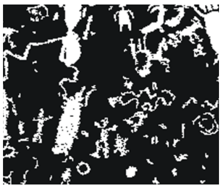
Region labelling:

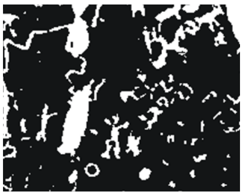
Result →

- Assign a **unique** region label to each set of connected (4- or 8-neighb.) foreground-pixels
- Connected regions can be visualized by pseudocolors or by pulling on range [0,255]
- **Postprocessing** is *crucial* to remove isolated "regions" that are created by noise or errors.

There are 3 approaches of postprocessing (input is a labelled image):

1. **Remove Regions** with $< n$ pixels: + shape of large regions remains
2. **Median Filter:** Advantage: + errors within large regions are removed





3. Relaxation Labelling:

Probabilistic approach with a recursive definition

- **Idea:** Bases on uncertainty: $P^0(p_i, l_k) :=$ Probability P for pixel p_i to have label l_k at time 0
- RL is an iterative process to find most likely label assignments
- Relaxation: Probability changes iteratively fewer

1. **Initialisation:** for each pixel and each label assign a probability: 0.8 $if\ \ l_k = l(p_i)$, $\frac{0{,}2}{k-1}\ else$

- So, the probability of a pixel to have the label it actually has after the normal region labelling process is 0.8. The probability to have any other label is equally distributed between the remaining 0.2.

2. **Compatibility coefficient:** All pixels influence all other pixels. Coefficient tells how compatible labels are:

- Only identical labels are compatible: $r(l_k, l_l) = 1\ if\ k = l\,,\ \ 0\ else$ k=l since labels should be unique
- If more information available: r might depend on p_i and p_j → e.g. supress label change for close areas

3. **Computation:** Is iterative (reciprocal influence). Stops after change is under threshold.
   Assign max $P^N(p_i, l_k)$

- Influence of **one pixel (p_j)** on label l_k of another pixel (p_i) in iteration n: $q_j^n := \sum_l P^n(p_j, l_l) \cdot r(l_k, l_l)$
  - "Sum of influence of all labels l_l of p_j weighted by their probability for that iteration."
- Total influence (of all pixels p_j) on label l_k of a pixel (p_i): $Q^n(p_i, l_k) := \sum_j q_j^n\,(p_i, l_k) \cdot c_{ij}$
  - c_{ij} is neighbourhood relation (e.g. $\frac{1}{4}$ or $\frac{1}{8}$)
- Final step: The $n+1th$-probability ∀ labels, ∀ pixels is

$$P^{n+1}(p_i, l_k) := \frac{P^n(p_i,l_k)\cdot(1+Q^n(p_i,l_k))}{\sum_l P^n(p_i,l_k)\cdot(1+Q^n(p_i,l_l))}$$

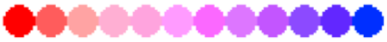
  - Division normalizes to range [0,1]

Confidence map may be a visual help to decide when n is high enough: It shows the reliability on the current label decision: The darker the pixel on the map, the more fragile the decision. The brighter the better.

Region-based-Segmentation:

Segments are homogeneous

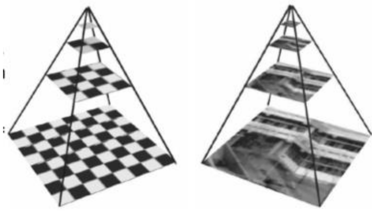
- Trying to detect high-level-concepts (sky, water etc.) not low-levels like before.
- Applying multiscale strategies (i.e. rely on homogeneity conditions for multiple scales)
- Hope on correlation between local & global features, but avoid causal chains!
- Segmentation of homogeneous regions possible with respect to properties like color, texture, grey value
- Ensures homogeneity within regions and hopes for sharp boundaries



The segment is locally homogeneous, but globally inhomogeneous.

Multi-Scale Representations (Gaussian pyramid)

- Used e.g. to reduce & expand images: (level 0 = bottom)
  - Reduce Operator $R(g)$: Each pixel of level $i+1$ replaces 4 pixels of level i
    - 2^N pixels, level 0 = bottom (original), level $N = 1 \times 1$ "image"
    - Can be done by Gaussian or Binomial filters: e.g. $\frac{1}{16} \cdot (1 \quad 4 \quad 6 \quad 4 \quad 1)$
    - Required space: $O(\frac{4}{3} \cdot img)$
  - Expand Operator $\mathrm{E}(g)$: Generate pixels of level i by interpolation of pixels of level $i+1$
    - Expanding blurs since reducing isn't invertible (→ loss of information)



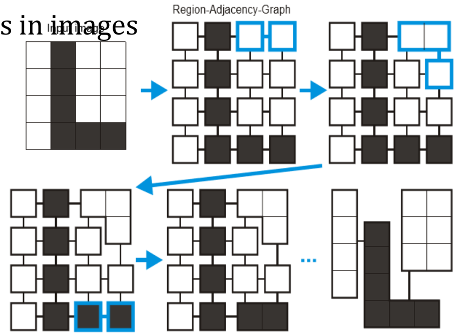
Laplacian Pyramid

- Computed by the difference of 2 neighbouring planes of a Gaussian-pyramid.
- Expand the smaller plane to get planes of same size.
- LP is now the difference: $\mathrm{L_I} := \mathrm{G_i} - \mathrm{E}(\mathrm{G_{i+1}}) = \mathrm{G_i} - \mathrm{E}(\mathrm{R}(\mathrm{G_I}))$
- Advantage: Nearly free of redundancies. → Used for data compression
- Recall: Zero Crossings of 2nd derivative correspond to edges. If crossings are found on multiple planes of Laplacian pyramid they likely correspond to real edges, not to noise.

We already know how to postprocess region labelled images (e.g. Relaxation labelling see above). But how to get the region labelled image itself? There are several approaches:

Region Merging: Region labelling approach to find connected objects in images

- Simple bottom-up method with high runtime
  1. Define each pixel as own segment. (Bottom-Up approach)
  2. Check if 2 neighboured nodes satisfy the same homogeneity condition
     If so: merge them. → Heuristic approach
  3. Stop if no more segments exist that may be merged.
- Homogeneity conditions (e.g. color, texture) can be defined by the user and must be computable for pixels *and* regions
- Image is mapped to RAG *(Region Adjacency Graph)* where segments corresponds to nodes and neighbourhood to edges
- Does not guarantee an optimal solution (lowest possible number of segments)
  - Reason: Region Merging is ambiguous since homogeneity condition **changes** by each merging
  - Intuitive idea: Merge most similar regions → Produces fragments (not lowest poss. number)



Region Merging + Multiscale Strategy: *explicit* multi-scale representation

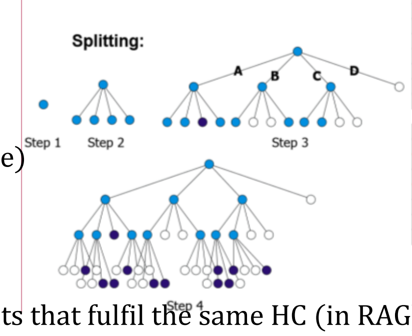
1. Start on clearly defined (high) level of Gaussian Pyramid (low enough that HC manifests)
2. Estimate on basis of the grey values of the segments of the start-level the grey value distribution of the lower level (higher resolution)
   Homogeneity condition: probability that both distributions are equal
3. Use expand ($\mathrm{E}(g)$) to propagate result to lower level. Back to step 2. (Stop on lowest level)

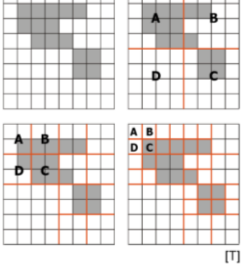
Split & Merge algorithm: Advanced region labelling approach to find connected objects in images

- Combined Top-Down/Bottom-Up approach with better runtime and an *implicit* multi-scale strategy

1. <u>Initialisation</u> : The entire image is a single segment
2. <u>Splitting</u> (Top-Down-part): If HC not fulfilled:
   Divide segment iteratively into 4 sub-segments

Internally represented by a **Quad-Tree**:





- Special tree where each node has exactly 4 children
- Mainly used to handle 2D-Data (3D → Octtree)

Example image: White node → White pixel(s), Blue node → grey pixel(s) cyan → both

Traverse Quadtree to obtain RAG

3. Merging: (Bottom-Up-part): Merge neighbouring segments that fulfil the same HC (in RAG)

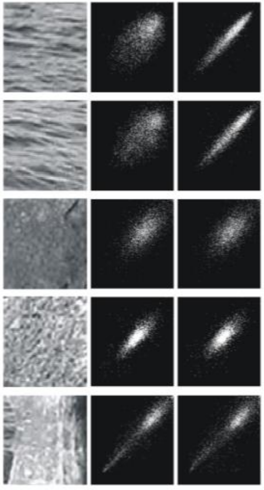
Conclusion: (In comparison to Region Merging)

- Split & Merge again doesn't guarantee an optimal solution
- Much better runtime that Region Merging (splitting stops before pixel level)
- Better results: Large segments improve the estimation of the distribution of grey values
- Yields complete, unique segmentation

Texture: "structure" or "patterning" of a surface

- No clear definition – No general algorithm to detect texture
- Texture is bound to a certain scale and periodic (but not strictly)
- Texture can be treated by 3 different mechanisms:
  - Its structural composition out of texels (i.e. texture elements)
  - As result of a parametric, stochastic (colour distribution) process
  - Defined by the properties of a spectral/frequency representation

Co-Occurrence-Matrix: (COM) Important tool to recognize textures. Represents correlation between pixels



- Describes the co-occurrence of 2 grey values g_1, g_2 with distance Δ and angle α to the horizontal axis (e.g. 90°: North-Pixel). From that we can derive a probability *that* the tupel occurs (given Δ, α). Δ is commonly chosen 1 or 2 (Close pixel have strong correlation → COM useful to detect correlations over short distances)
- Can be interpreted as a 2D-histogram with pairs of pixels (pairs are built by a relation p)

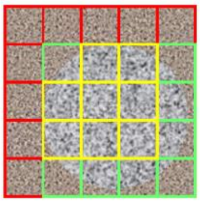
Example: Relation p (shortcut for Δ, α) may be "look at the right neighbour" ($p(x,y) = (x+1, y)$)

- Help-Matrix H may describe the set of possible tuples(!) (e.g. $< 0,0 >, < 1,0 >, < 0,1 >$...
  - Yields a quadratic $n \times n$ Matrix (with n different values in image g)
- COM P_p holds ∀ tuple the number of occurrences, e.g. $P_{p(1,0)} = 3$ → 3 times a 0 is right to a 1
- Strong main diagonal denotes large monotone regions (neighbouring pixels have close values)
- Strong bottom-left and top-right corners indicate strong contrast

Formula: $P_{\Delta,\alpha}(g_1, g_2) = \frac{1}{N^2} \cdot \sum_{\vec{p}\in W} \delta(g(\vec{p}) - g_1) \cdot \delta\big(g(\vec{p} + \vec{\Delta}) - g_2\big)$

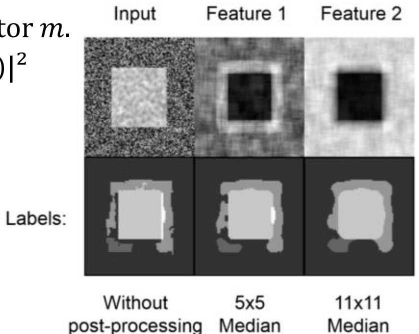
$W \coloneqq$ examined window of g, $\vec{\Delta} \coloneqq \Delta \cdot \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix}$ $\delta(x) \coloneqq 1\ iff\ x = 0, \quad 0\ else$

- Co-Occurrence-Matrix can be evaluated by different texture features after Haralick: Contrast, Entropy, Homogeneity, Energy, Inverse difference moment:



Texture Segmentation: Aim: Merge groups of pixels of similar texture to obtain regions

- So far, homogeneity conditions were computed based on pixel values
  - Texture is a feature of groups of pixels → Define a homogeneity measurement
- Homogeneity measure can also be reached by combining texture features in a vector m.
  - Def: distance between pixels for all features: $H(p_1, p_2) \coloneqq |m(p_1) - m(p_2)|^2$
  - Region R is homogeneous iff: $\forall p_i, p_j \in R : H(p_i, p_j) < H_{\min}$
- Texture Segmentation using Split & Merge is possible.
  - Easy for pixels that are clearly within or outside region



    - Hard at boundary: Different degrees of overlap, neighbouring pixels are unclear
    - → Post-processing necessary:
        - Median Filter or Relaxiation Labelling (20 iterations) are possible techniques for that

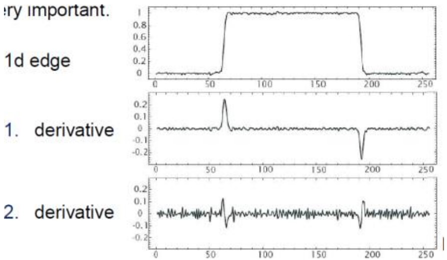
Edge-based-Segmentation:

- Inverted approach to *region-based*: segments are defined by clearly detected boundaries, but it hopes for homogeneity within regions
- Steps: Start with edge point detection (e.g. gradient magnitude), region labelling.
- The obtained edges are not yet connected

<u>Solutions:</u> Edge Linking Canny Operator Detection of Zero-Crossings Watershed transform

1. **Edge Linking:** Uses gradient magnitude to link edges
    - The stronger the gradient value at position (x, y) the more probable that it is a real edge, not just noise
    - If (x, y) belongs to an edge, there should be more edge pixels orthogonal to the gradient direction

    <u>Algorithm:</u>
    1. Initialization: Mark all edge-pixels as unprocessed
    2. As long as there are unprocessed pixels, mark the first as a start pixel of a new edge
    3. Search in both directions perpendicular to gradient direction for edge pixels with same magnitude & direction
        1. If pixel is unprocessed: Mark it as member of the edge, make it new start pixel, back to 3.
        2. Else (if processed): Note that an edge junction has found, go back to 2.
2. **Canny-Operator** optimal edge detection & localization + no multiple responses (like Laplacian)
    - Conditions: Mean of noise = 0, focus may be blurred
    1. Edge enhancement: Smoothing by Gaussian filter
    2. Compute the gradient magnitude. Use gradient operator to locate edges
    3. Find edges: Define 2 thresholds (θ_1, θ_2, with $\theta_2 < \theta_1$)
        1. All pixels with gradient magnitude $> \theta_1$ are start points of edges (supress noise, e.g. Non-Maxima-Suppression)
        2. Neighbouring pixels are added if gradient magnitude is $> \theta_2$



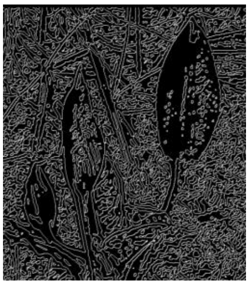
Edge Linking & Canny Operator don't guarantee closed edge contours.

To guarantee that use Zero-Crossing-Detection or Watershed Transform.

3. **Zero-Crossing-Detection**
    - Zero-Crossings of 2nd derivative are contours. → Use Laplace –Operator (see above)

Preprocessed

No preprocessing



<u>Algorithm:</u> 0. Preprocessing with e.g. Gaussian to blur noise

1. $g_k(x, y) = \Delta g(x, y)$ Produce edge image using Laplace-Operator
   May use LoG − Filter to sum up step 0. and 1
2. Evaluate if: $g_k(x, y) \cdot \text{shift}(g_k(x, y)) < 0$ $[- \cdot\ + = \ -]$ shift denotes 4-/8-neighb.
   N_4: Consider east- & south-pixels. N_8: N_4+southeast+southwest
   To avoid double findings consider only south and east pixel in 4-neighborhood

4. **Watershed Transform** Finds segments included by edges
    - Watershed: E.g. in the alps the mountain crest where water flows either to Nordsee or to Mittelmeer
    - Gradient magnitude image representing the heights of the relief/watershed as segment boundaries
    - Result: yields segments enclosed by edges, but ignores the differing strength of edges (noise)
    1. **Rain**: Compute for each pixel the local minimum where a rain drop would end (following the strongest slope. Hard to compute)
    2. **Flood**: Starting at local minima, the groundwater floats the relief.

    <u>Algorithm:</u> 0. Compute absolute gradient image $|\Delta(g(x,y)|$

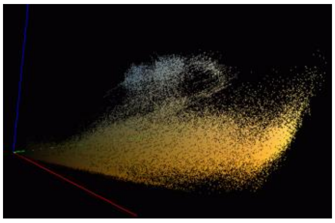
1. Initialization: Level $h = 0$, assign no label to all pixels
2. Increase h as long as: $h < \max(|\Delta(g(x,y)|) =$
3. $\forall x,y$ check: $|\Delta(g(x,y))| > h \land \text{no\_label}(x,y)$ → (x,y) is flooded just now. Check:
    1. <u>Isolated</u>: (x,y) has no flooded neighbours. → (x,y) is minimum of new segment
    2. <u>Segment</u>: (x,y)'s neighbours have identical labels → assign same label to (x,y)
    3. <u>Watershed:</u> (x,y)'s neighbours have different labels → assign label: *"watershed"*

Post-processing: Assign "*watershed"*-pixels to a real value.

Problem: Oversegmentation, since every local minima is a valley (noise sensitive)

→ Multiscale Strategy: Apply some subsequent watersheds (**Hierarchical Watershed Transform**)

- The 1st WST finds many local minima that are just noise
- After a trial each region gets the average grey value of the respective image area
- Trial by trial more regions merge, whereas "real" minima remain several trials
- Get a new gradient image by estimating gradient by the neighbouring segments:
    - E.g.: $gr_M(S) = \frac{1}{N} \cdot \sum_i |g_M(S) - g_M(S_i)|$ Average of diff. to neighb. regions
- Segments stepwise increase: Hard to find automatically the right point to stop.



Color-Segmentation:

Target: Find segments of constant colour

- Homogeneity condition: Similarity measure (distance) in a colour space
- 2 algorithms: : K-Means-Clustering, Mean-Shift-Segmentation
- Grey value segmentation: Analyse histogram, assign segment to pixel based on gray value, hope that regions with similar grey values are close
- Colour segmentation: Look for accumulations in colour space, partition image and assign segments and hope again that regions with similar colours are close (i.e. connected)

Clustering:

One approach to find colour classes. Most popular: K-Means-Clustering

Vector Quantization (VQ): Given a N-dimensional Data D find for each vector $\vec{d_\imath} \in D$ the best match (i.e. the closest distance to one of the cluster centers $\vec{w_j} \in \mathbb{R}^N$. The cluster centres just *represent* the cluster C_i

The more cluster centers, the more computational effort but the better the approximation

→ **Clustering for images finds individual colour-classes (but no segments!)**

K-Means-Clustering: <u>Minimize</u> iteratively error measure: $E(D, \vec{w_j}) = \frac{1}{|D|} \cdot \sum_i ||\vec{d_\imath} - \vec{w}_{m(\vec{d}_i)}||^2$

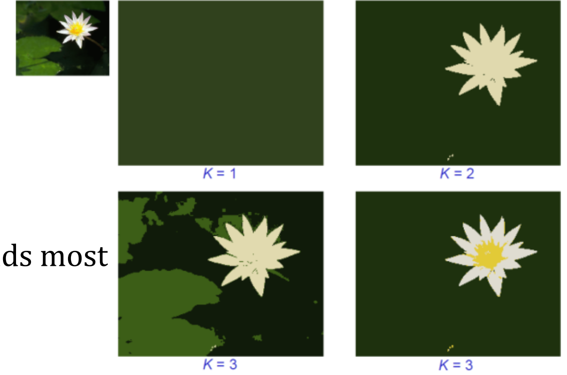
- Start with randomly chosen cluster centres.
- Update cluster centre $\vec{w}_j$by shifting it to mean of data vectors $\vec{d}_i$ belonging to that cluster
- Terminates with a local (not global) minimum → no unique solution

<u>Algorithm:</u>

1. Initialisation: $t = 0$, choose K cluster centres $\vec{w}_1 \dots \vec{w}_K$ randomly (within suitable range)
2. Assign "No value" $\forall$ cluster C_i (e.g. with *for*-loop)
3. Find closest cluster centres: *for* $i = 1 \dots |D|$ ($\forall$ data vectors) : $\arg\min_{k=1\dots K} ||\vec{d}_i - \vec{w}_k||$
   Add data vector $\vec{d}_i$ to the cluster C_k *endfor*
4. Update cluster centres: *for* $k = 1 \dots K$ ($\forall$ cluster centres): $\vec{w}_k = \frac{1}{|C_k|} \cdot \sum_{\vec{c}_i \in C_k} \vec{c}_i$ New cluster centre is mean of vectors that belong to that cluster. *Endfor*
5. Check if termination point reached: *if*$\left(\exists\, k = 1 \dots K : \left|\left|\vec{w}_k - \vec{w}_{k_{ALT}}\right|\right| > \varepsilon\right) : t = t + 1$ go back to step 2. (If change in one cluster was $> \varepsilon$, continue)

K-Means for coloured images:

- $N = 3$ (RGB-Space), $K =$ #colours in ouput image,



Another local minimum

- After termination assign the label $m(\vec{d}_i)$ to each pixel.
- That allows to compute the mean colour $\vec{w}_m$ ∀pixels of the respective cluster (→ visualisation)
- With Only one parameter K (provided) the algorithm finds most frequent colours
- Pixel connectivity is not considered for K-Means
- Segmentation is unique (∃ global minimum), but random initialization mostly finds local min.
- Some alternatives like K-Means++ that try to find better initializations

Mean-Shift-Segmentation (MSS) finds connected segments (but no colour classes)

- Nonlinear, Non-parametric ($\neq K$-Means), iterative algorithm
- Segmentation of homogeneous color regions → considers location AND colour-value
  - Uses *spatial-range-representation* $\in \mathbb{R}^5: (x, y, R, G, B)$
  - Scaling necessary such that both components become comparable:
    - $\sigma = 0{,}04$: Distance 100 in colour space is worth a distance of 4 in image space
- The clusters after segmentation (in $\mathbb{R}^5$) holds neighbouring pixels with similar colours

<u>Algorithm:</u> 0. Preprocessing: Apply non-linear edge preserving smoothing filter

1. **Mean Shift Filtering**: (Non-linear) [Very powerful filter in comparison to e.g. Gaussian]
   Finds local density maxima (pixels similar in space & color) and assigns to each pixel the colour $\vec{c}$ of the closest local density maximum. → Filtered image g^*
   Computes iteratively center of gravity $\vec{q}_{t+1}$ in unit sphere S_t

```
For ∀ i ∈ MN :          $t = 0, \vec{q}_t = \vec{p}_i$
    While $|\vec{q}_{t+1} - \vec{q}_t| > \varepsilon$:
        $\vec{q}_{t+1} = \frac{1}{|S_t|} \cdot \sum_{\vec{p}_j \in S_t} \vec{p}_j$
    $g^*(i) = \vec{c}_i^{\,*}$      (Assign colour of closest local density maximum)
```

   Result: Much easier segmentable image
2. **Clustering:** Principle: ∀ pixel $\in g^*: \left|\left|x, y, R, G, B\right|\right| < \theta$ → common label
   - Completely different from K-Means-Clustering, but ensures connectivity

Evaluation of Segmentations: Mostly manually (by human judgements).

- Target: Improve segmentation algorithmically using domain-independent criteria → Improve correspondence between segments and meaningful entities.
- Quality measurement of segmentation can be *globally* (∀ segments) or *locally* (single segments)
- Meaning cannot be evaluated: Evaluate stability locally against: viewpoint, illumination, disruptions

Segment boundaries with strong colour contrast correspond likely to borders of entities: →
<u>Hypothesis:</u> The stability against the 3 treatments increases with the average colour contrast of neighb. regions

Evaluation Measure: Calculate average color $\bar{C}(R_i)$ of all regions R_i

Region saliency $S_R(R_i)$ = average color difference to neighbouring regions $B(R_i) :=$ boundary pixels

Region saliency: $S_R(R_i) := \frac{1}{|B(R_i)|} \cdot \sum_{p_j \in B(R_i)} \frac{1}{|N_4(p_j) \in R_k|} \cdot \sum_{p'_j \in (N_4 \in R_k)} ||\bar{C}(R_i) - \bar{C}(R_k)|| \; k \neq i$

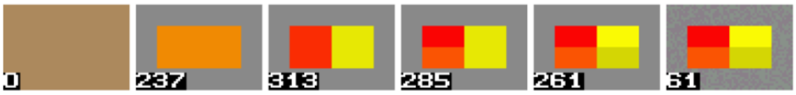
The difference of average colour of the inserted region and the neighbouring regions is weighted by the length of the boundary (i.e. computed n-times in case of n-boundary pixels) The result is then divided by the sum of overall boundary pixels times the sum of boundary-neighbor pixels that belong to another region.

Automatic Parametrization of segmentations important to reduce runtime, improve results

Saliency (generally): average region saliency of all regions: $S = \frac{1}{N} \cdot \sum_{R_j} S_R(R_j)$

- A goodness function to find the optimal parameters for an algorithm (e.g. K for K-Means)

Influence of K on S using K-Means-Algorithm:



- Decay in S as soon as noise becomes modelled
→ much boundaries with small difference

Testing stability against 3 treatments (disruptions (i.e. noise), illumination, varyied viewpoint)

- The first 2 are easy: Record changed image, try to find corresponding segments of both images, count (un)changed assignments of pixels.
- Testing stability against varyied viewpoint is very hard

Conclusion of automatic parametrization:

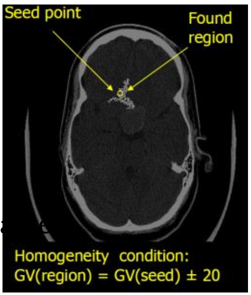
- Maximizing S leads to useful segmentations (by human judgements)
- These segmentations are robust against 3 treatments (correspondance defined manually for viewpoint)
- Optimal parameters are **similar for similar images** (E.g. time series or images of same type)
- High computational effort

Interactive Segmentation

- Regions with simple connectivity conditions (edges, homogeneity) can now be segmented
- Impossible for complex conditions (shadow) → Model-based-segmentation (1 for each model)
- Interactive since user defines model im-/explicitly from image.
    - E.g. Template Matching, Hough-Transform

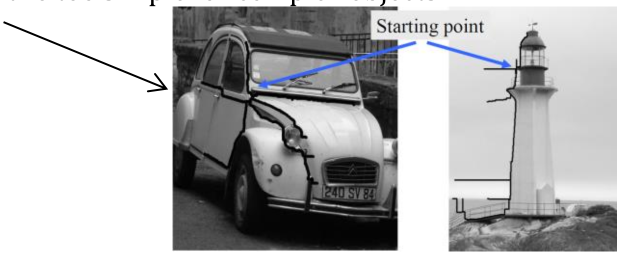
Flood-Fill: Interactive Region-Growing

- Use a single seed-point to make a homogeneous (user-defined) region visible with a label
- Start at seed-point, test all unprocessed neighbours on homogeneity condition
- Homogeneity conditions are flexible: 4-/8-neighbourhood with same grey value, range of v
- Problems: Leakage of regions, uncertainty about size, noise …
- For inhomogeneous regions use n seeding points and strict HC. Merge n fragments to get result



Interactive Edge Seach: Search for exactly n particular edges

- Edge filtering yields list of possible edges; but not all candidates are needed, boundary may have gaps…
- 3 approaches: edge following, optimal edges and Hough-Transform
    1. **Edge Following:** Set starting point manually, then adapt **Edge-Linking**
        - Preprocessing (smoothing) or deletion of edges with false gradient or intensity may be useful
        - Search orthogonal to current gradient direction.
            - Edge points are close to each other, have similar gradient magnitudes & directions
        - Problems: varying gradient strength, sudden changes in direction of edge
    2. **Optimal Edges:** Start & End of edge are known, search in between by optimality measure
        - Possible optimality measure: Minimze change of direction, maximize average gradient magnitude
        - Principle: Transform image to graph. Neighbouring pixels are connected via edges
        - Edge costs are normally difference of pixel value to start-/end-point values
        - Find a paths with minimized costs → High comp. effort
        - Results are still rather bad. Even manually defined HC are too simple for complex objects



Alternative: Machine Learning: Train a classifier!

- Input: manually selected image patches
- Output: Desired labelled image (manually computed)
- Classifier learns mapping $Patch \rightarrow Labels$
- Complex patterns as HC, combines color + texture
- Generalization, training patches must be very similar
- Powerful approach! Postprocessing (noise reduction,

gaussian filter, K-Means with $K = 3$ would yield perfect result

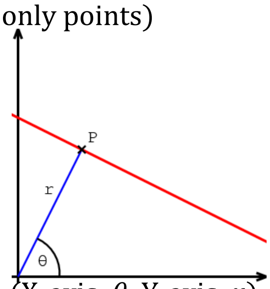
User assigns the labels sky, city, water to the selected patches.
Trained classifier is applied to all pixels.
[H]

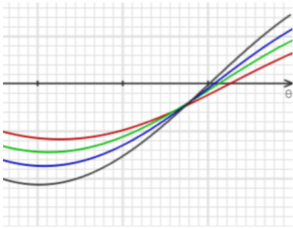
7. Hough-Transform (HT)

- 1 of the 3 approaches for interactive Edge-Search (other are Edge Following & Optimal Edges)
- HT is a technique “between local and global” (Heidemann) to detect lines (extendable to circles, all figures) in binary images
- Input: Binary Image + specification of desired figure (default: lines)
- Output: Double-Image with white accumulations for each found figure
  - Accumulations are plotted in an accumulator space (= Hough- = Dual-space)
  - Position of accumulation holds unique information about shape and location of line
- It is useful to preprocess with Cannel- or Sobel-Filter to reduce comp. effort (reduce line-candidates)

Algorithm/Technique:

1. Image g. Compute gradient magnitudes G and gradient direction Θ
   - Normally gradient direction is named θ (not Θ), but I want to avoid ambiguities with the angle θ (see formula and graphs below)
   - Θ guarantees that accumulator space is unique (bijective. No sine curves, only points)
2. Threshold G to get all “clear” edge points, saved in binary image called G^*
3. Initialize accumulator space $A(r, \Theta)$
4. Compute $\forall$ edge points P in G^* all possible lines m through this point
   - Implemented by changing angle θ from 0 to 180
   - Measure the distance r from each line to origin
   - The origin is normally(!) in the upper left corner
5. Each point P transforms into a line (if no Θ is provided) in the accumulator space (X-axis: θ, Y-axis: r)
   - So, the accumulator space represents distance of m to origin (coded in r) in dependence of angle θ
   - Now, that we got a graphical idea about what happens, we can introduce a formula which generates exactly the representing line: The *sinusoidal* function $r(\Theta)$ is:
     - $\boldsymbol{r(\Theta) \coloneqq x \cdot \cos(\Theta) + y \cdot \sin(\Theta)}$**Note: x and y are parameters now**
6. If multiple points P_n are aligned, the sinusoidal functions $r_{P_n}(\theta)$ intersect in (x, y)
7. Why? Rotating θ yields $\forall$ points P_n many different lines (represented in acc-space)
   - But one line m_i with unique(!) angle θ and distance r occurs $\forall$ points P_n
   - Due to this, all points P_n go through the intersection/accumulation point (r, θ)





Evaluation:

- $A(r, \theta) = n$ means: n edge points are located on a line, whose properties are uniquely coded in r, θ
- Horizontal-X-axis: θ Vertical-Y-axis: r Accumulation point (x, y)
- X-axis θ: x value centered in acc-space for vertical lines. At the boundary for horizontal lines
  - x corresponds to slope of line
- Y-axis r: The lower the distance of line m to origin the closer is the acc-point to the middle of y-axis
  - Holds only if acc-space was shifted (see below), else acc-point is closer to corner of origin (mostly upper left corner)
  - y corresponds to position of line in space (up vs. down)
- The higher the n the longer the line (the brighter the accumulation)

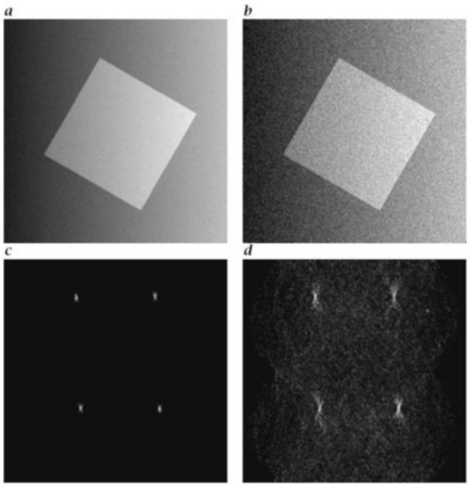
Properties:

- Accumulator space can be represented by matrix (dimensions: θ, r). → Maxima in matrix = lines in g
- Origin seems to be lower right corner (in example below lowest line has smallest r)
  - However it’s unknown whether the acc-space is shifted such that $r = 0$ is centred (e.g. in MATLAB)

- Missing Θ increases comp. effort dramatically. No Θ → "simple HT"
- Since vertical lines can't be described by $y = mx + b$ we could make:
  - a Hough-Transform of the 90° turned image (complicated)
  - represent lines by their Hesse-Form → Leads to axis α (angle) and r (distance) in acc-space
- Broad edges on image lead to flat maxima in accumulator space
- HT is form of model-based search.

Disadvantages:

- Brute-Force → High comp. effort → no realtime analysis (only with Fast HT)



(a) Input with inhomogeneous illumination

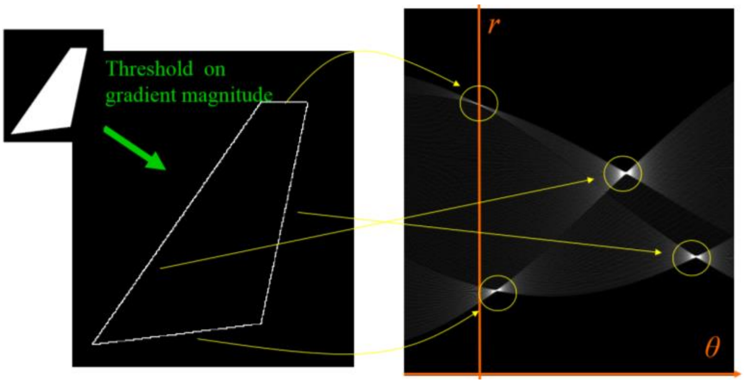
(b) (a) + noise

(c) HT of (a). Inhomogeneous illmination has no effect

(d) HT of (b). Noise makes maxima broader

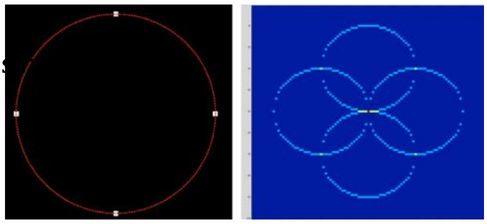
Horizontal axis: r,
vertical axis: θ.

[J]



Circular Hough-Transform: (CHT)

- $(x - x_c)^2 + (y - y_c) = r^2$ → 3D-accumulator space: x_c, y_c, r
- May be simplified by only searching circles of radius r → 2D-acc-space
- Procedure: $\forall\ P \in g$: define circle in Acc-space around (x_c, y_c) with radius
- Intersection of circles (maxima in A) is center of original circle



8. Fourier-Transform (FT)

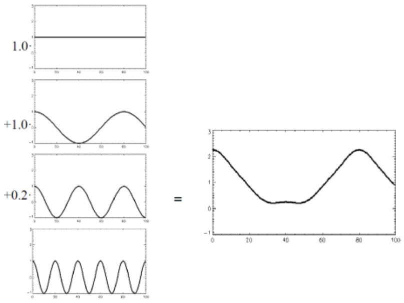
→ I didn't really get this topic, so be especially critical here!

What is a Fourier-Transform?

- Generally, we can decompose an arbitrary function into a sum of sine- & cosine function
- In CV: A bijective (invertible), global operation to decompose an arbitrary greyscale image into periodic waves of different frequencies & directions
  - High frequencies model borders (grey value jumps) and low frequencies model structure

What are the functions of a FT in CV?

- Detects general structures (e.g. texture) independent from size & orientation
- May supress noise in images
- Filtering of irrelevant data (→ compression)



Superposition of four sine functions with different weights.

Smoothie-Analogy :

- Given a smoothie, the Fourier Transform finds the recipe.
- How? Run the smoothie through filters to extract each ingredient.
- Why? Recipes are easier to analyse, compare, and modify than the smoothie itself.
- How do we get the smoothie back? Blend the ingredients.

Abstract idea: (may be skipped)

- The FT generates a different representation of an image. Precisely it uses a linear combination of sine & cosine functions to create an orthogonal basis transform
    - Note: Basis transformation doesn't change image, but just its coordinates

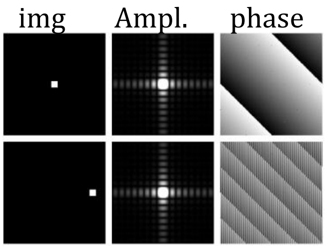
That sounds like lots of Maths, let's start easily again:

- So far, images are represented as matrices, i.e. 2D-arrays
- We can understand images as functions $x: D \to C$ with D as set of pixels and C as set of grey values
- Like any polynomial can be represented as a vector in $\mathbb{R}^n$: (axis $x^0, x^1, \dots$) , this is also possible with images: an image with N pixels may be understood as a vector in $\mathbb{R}^N$
  $\begin{pmatrix} 3 & 4 \\ 6 & 2 \end{pmatrix} = 3\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + 4\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + 6\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} + 2\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = g = \sum_{\forall i} w_i \cdot b_i$ (superposition yields image)
    - These n matrices (precisely their unambiguous vectorial representation) are linearly independent, so they build a basis (minimal generating system) of $\mathbb{R}^N$
    - Basis $\Leftrightarrow$ all vectors are orthogonal $\Leftrightarrow$ $v_i \perp v_j \; \forall i,j \Leftrightarrow < v_i , v_j > = 0 \;\; \forall i,j$
    - For this "image", all bases have 4 vectors and the basis needs to be orthonormal
    - The result of above computation may be transformed to a vector $\vec{x}$ with all image values
    - Multiply $\vec{x}$ with orthogonal matrix A (orthogonal matrix $\mathrm{A} \coloneqq A^T = A^{-1} \Leftrightarrow AA^T = I$)
    - Each row of A is a basis function. → A contains set of basis functions
    - How do the basis functions look like? → Depends on type of transformation
        - FT: basis functions are **complex** (co)sine function
        - Cosine Transform: basis functions are cosine functions
        - They are normally periodic functions → to represent oscillation (frequency) of image
- Problem: Functions are continuous but our image is discrete.
    - Measure the values of each function only at some points (samples or Abtastpunkte)
    - The more samples we have, the higher the computational effort but the better the result
    - For arbitrary close approximations, sum has to be replaced by integral (continuous signal)

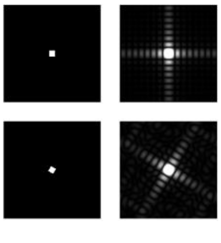
Practical information:

- The transformation of an image (spatial space) to frequency space: $n \times m$ image → nm waves
    - So each pixel requires exactly one wave
- High frequencies model fine structures (details ,edges) low frequencies model constant colouring
- The basis transformation is information preserving (→ invertible since bijective). FT is in a sense a circular operation (viewpoint rotation of 90° in space when observing a function)
    - $\mathcal{F}^2(f) = f(-x)$ $\qquad\qquad$ $\mathcal{F}^4(f) = f(x)$
    -
- Result: image $f(x,y)$ → $\mathcal{F}(u,v)$ We get 2 types of images
    - Amplitude image: Grey value variation in the part represented by a particular wave
        - For a given value $\mathcal{F}(u,v)$ the amplitude $\boldsymbol{A} \coloneqq |(\boldsymbol{u},\boldsymbol{v})|$ $\quad$: $(u,v) \in \mathbb{C}$
        - Gives information about orientation of borders
    - Phase image: For a given value $\mathcal{F}(u,v)$ the phase $\theta \coloneqq$ angle of vector to $\mathbb{R}$-Axis
        - Structural information about image
    - Frequency of wave = distance to point of origin = $\sqrt{(u^2 + v^2)}$

Example: Fourier Transform of a binary image with a single object



1. Change position of object → Same amplitude (same frequencies!)
    - Phase:Translation leads to phase shift of each contributing wave
    - Amount of shift depends on absolute value of shift (clear) + angle between direction of shift & wave



2. Change rotation of object:
    - Same phase image

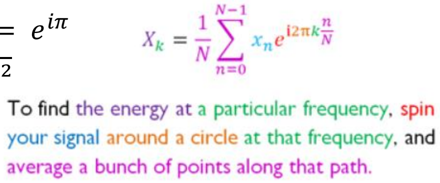
- Different amplitude (rotated in same angle)

Showing a FT-image:

- Showing just the amplitude of all waves is sufficient. Phase image is discarded for the analysis
- Amplitude image is still hardly analysable
  - Preprocess it with the logarithm (reason: Many values map on 0: white centre, black periphery)
  - $\log x$ and x close to 0 yields strong separation! ($\approx\ -5$ bis -20)
  - Shows big compression potential
- Point of origin is shifted to the centre of the image (Matlab: $fftshift$)
  - Low frequencies are around the centre of the amplitude image
  - Increasing distance to point of origin → increasing frequency of wave (in $\mathbb{C}$-space directions)
  - Think of the 4 directions of the $\mathbb{C}$-space
    - Horizontal axis: number line of real values $\mathbb{R}$
    - Vertical axis: number line of pure complex values $c \coloneqq a + ib$ with $a = 0$
    - All values in between: $c \in \mathbb{C} \backslash \mathbb{R}$ (real & complex part $\neq 0$)
  - Bright colouring = high amplitude for this frequency. Dark colouring = low amplitude
- Though amplitude image is easier to analyse, the phase image contains most information (but shows just randomly appearing noise in most cases)
  - Reconstruction of image only with amplitude image is impossible!
  - Reconstruction of image with phase image only is possible (but not perfect)

Formulas:

1. Euler-Identity: $\cos(x) + i\sin(x) = e^{ix}$ $\quad x \coloneqq \pi \quad \Rightarrow \quad -1 = e^{i\pi}$
2. Amplitude : $|c|, c \in \mathbb{C} \quad c \coloneqq a + ib \quad \rightarrow \quad |c| = \sqrt{a^2 + b^2}$
3. Phase: $\phi(u,v) = \arctan(^{Im(\mathcal{F}(u,v))}/_{Re(\mathcal{F}(u,v))})$



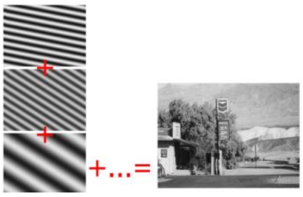
$$X_k = \frac{1}{N}\sum_{n=0}^{N-1} x_n e^{i2\pi k\frac{n}{N}}$$

To find the energy at a particular frequency, spin your signal around a circle at that frequency, and average a bunch of points along that path.

4. 1D-Fourier-Formula: $c_n = \sum_{x=1}^{L} f(x) \cdot \exp(^{2\pi inx}/_{L})$
5. 2D-Fourier-Formula: $\mathcal{F}(u,v) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} f(m,n) \cdot exp(-i2\pi(\frac{um}{M} + \frac{vn}{N}))$

1D-Fourier-Transformation:

- With N Abtastpunkten we have N different frequencies that build up a N-dimensional space
- Basis functions are of type: $\cos(x) + i \cdot \sin(x)$
- **Formula:** $\mathcal{F}(u) = \frac{1}{s} \cdot \sum_{n=0}^{N-1} f(n)\left[\cos\left(\frac{2\pi un}{N}\right) - i\sin\left(\frac{2\pi un}{N}\right)\right] = \frac{1}{\sqrt{N}} \cdot \sum_{n=0}^{N-1} f(n)\exp(-iun\frac{2\pi}{N})$
- Reverse: replace $f(n)$ by $\mathcal{F}(u)$ and delete – in e^x
- $A(\cos(x+\theta) + i \cdot \sin(x+\theta)) = Ae^{i(x+\theta)} = Ae^{ix}e^{i\theta} = Ke^{ix} \quad | K \coloneqq e^{i\theta}$.
- $|K|$ = amplitude $\quad\quad \measuredangle K, \mathbb{R}$(point of origin) = phase



2D-Fourier-Transformation: Transformation of images

- $n \times n$ image has n^2 pixel → n^2 waves. Complexity for one pixel is $O(n^2)$ so total complexity is $O(n^4)$
- Basis functions are now of type: $(\cos(x) + i\sin(x)) \cdot ((\cos(y) + i\sin(y)) = e^{ix} \cdot e^{iy} = e^{i(x+y)}$
- **Formula**: $\mathcal{F}(u,v) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} f(m,n)exp(-i2\pi(\frac{um}{M} + \frac{vn}{N}))$ $\quad m, n$ = image point
- Reverse: Same as for 1D, additionally put $\frac{1}{MN}$ in front of double-sum

Fast Fourier Transformation (FFT):

- Used for data compression, JPG, MPG etc.
- Uses some tricks to reduce complexity of $\mathcal{FT}$: $O(n^4) \rightsquigarrow O(n^2 \log n)$
  1. **Property of periodicity** (holds for all dimensions)

        - $\mathcal{F}(u) = \mathcal{F}(u+N) \qquad f(n) = f(n+N)$
        - Holds for all dimensions
    2. **Property of symmetry:** (holds for all dimensions)
        - The real part $\mathcal{F}(u)$ is point symmetric to the point of origin
        - The imaginary part of $\mathcal{F}(u)$is mirrored point symmetric to the point of origin
        - Only half the values are computed, other half derivable by complex conjugated $a - ib$
    3. **Property of separability:** (holds for all dimensions)
        - Transformation of P-dimensional transformation = series of P 1D transformations
        - Complexity reduces from $O(N^4)$ to $O(N^3)$ (Pixelcomplexity: $O(N)$ makes no sense…!?)
        - Practically for CV: First compute basis functions for all rows, then for all columns
    4. **Divide & Conquer Strategy** (seems to hold only for 2D)
        - Transformation of N values (pixels) replaced by 2 transf. with K, L values ($K + L = N$)
        - Formula is slightly modified, return are values that are used to compute transformation for N pixels

Convolution theorem:

- Convolution of 1D functions was defined as: $(f * g)(n) := \sum_{n=-\infty}^{\infty} f(k) \cdot g(n-k)$
    - $\mathcal{F}\big((f * g)(n)\big) = \mathcal{F}(\sum_{n=-\infty}^{\infty} f(k) \cdot g(n-k)) = F(u) * G(u)$ ($\mathcal{F}$ of f, g respectively)
    - **Convolution in space = Pointwise multiplication in Fourier space**
- Fourier after convolution in spatial space is equal to multiplication of the resulting frequency spaces
- Property holds for discrete & real functions for all dimensions, but requires same size of image & kernel
- Very nice property, if convolution kernels are large, the FTs may be faster and save *much* runtime
- Procedure:
    1. Fourier of image: $\mathcal{F}\big(g(x,y)\big) = G(k_x, k_y)$
    2. Fourier of kernel: $\mathcal{F}\big(m(x,y)\big) = M(k_x, k_y)$ iff $>$ 1 kernels, add indices to M
    3. Pointwise multiplication $G'\big(k_x, k_y\big) = G\big(k_x, k_y\big) \cdot M(k_x, k_y)$
    4. Transform back to image space $f(G'\big(k_x, k_y\big)) = g'(x,y)$ $f$denotes back-transformation

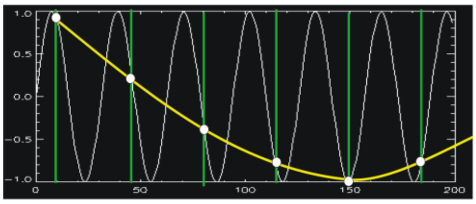
Purpose of Fourier-Transformation

- FT transforms an image into its frequencies: Low frequencies highly contribute to overall shape of image, high frequencies especially represent sharp borders
- We can detect texture in images or compress images
- So far, noise reduction was achievable by applying convolution with special kernels
- Now: Many kernels can be designed in Fourier space (due to convolution theorem)
    - Assume surrounding pixels of noised area have same grey values
    - Add up noise $\eta(x,y)$with mean $= 0$
    - Remove noise by local averaging
    - Sharp lines in image can easily be removed
- Disadvantage: Implicit assumption of FT: waves are at all positions in the image
    - This is not the case since each wave represents only one pixel
    - Wavelets: Amplitude of waves is $\neq 0$ only in very small area (**where** it is needed)
        - Allows in addition to frequential also **spatial resolution**
        - Wavelets can be seen as local FT

9. Sampling Theorem & Image Enhancement

Problem: Cameras record signals analogically (CCD-chip), but the signal gets digitized within the camera before the final result (image) exists. It is a crucial point to decide about the manner of discretization: How many samples (Abtastpunkte) of the analogous signal do we take? Or, more precisely, in which frequency do we take samples to capture all relevant information in the analogous signal?

This problem is called **Alias-Effect:** Discretisation of a continuous signal and subsequent interpolating may causes an incorrect data interpretation if sample frequency is lower than signal frequency.



Example: White line: analogous signal white dots: samples yellow: supposed underlying signal after discretisation (wrong → loose high frequencies → artifacts)

Sampling Theorem

If B is the highest frequency of a measured signal, the sampling rate has to be higher than $\frac{1}{2\mathrm{B}}$ to capture enough information to unambiguously reconstruct the underlying signal.

Fourier-transform of discrete signal: Convolve FT of continuous signal (frequency $< W$) with FT of peak positions (of distance Δx + height) → Result: Superposition of shifted copies of FT of cont. signal.

Overlap → Alias Band limited $:=$ $\Delta x \leq \frac{1}{2W}$

Image Enhancement

Point based image enhancement:

Various techniques: E.g. mapping gray values to others (pseudoclors), homogeneous operations etc.

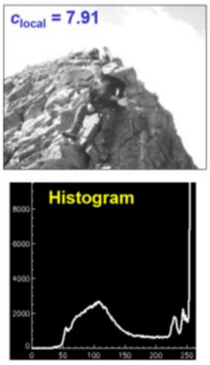
Need: Automatic quality measurement (→ Quantification) Is enhancement necessary, when is it enough?

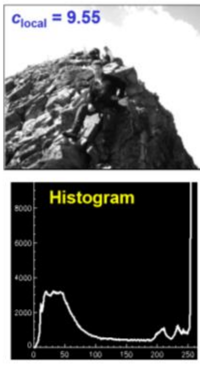
Ideas: Local/Global contrast, entropy etc. Under-/Overexposure: Unter-/Überbelichtung

Global contrast: $c_{global}(g) = \frac{1}{R}(\max g(x,y) - \min g(x,y))$ $R := 255$ (mostly)

This operation is a linear transform

Maximize: $g' = \frac{(g - g_{\min})(w_{\max} - w_{\min})}{g_{\max} - g_{\min}} + w_{\min}$ usually: $w_{max} = 255, w_{min} = 0$





Local contrast: average gray value difference of neighbouring pixels (N_4 or N_8)

$$c_{local}(g) = \frac{1}{MN} \cdot \sum\sum |g(x,y) - \frac{1}{4} \cdot \sum_{x',y' \in N_4} g(x',y')$$

Improving by nonlinear monotonic transfer function (Problem: [0,255] is used, but still too bright/dark at some positions)

$$g' = w_{max} \cdot \left(\frac{g}{w_{max}}\right)^{y}$$

+ stronger contrast than global - enhancement of irrelevant patterns

Information content & Entropy

- Quantitative, intuitive measurement of information. Information content called I

Idea: More surprising (i.e. less probable) events hold more information

- Properties of I: positive, additive, zero for save events, increasing for decreasing probability
- $\boldsymbol{I(n,N) := -\log_n \frac{1}{N}}$ $n :=$ # of possible events (e.g. 2 in binary image) $N :=$ # of pixels
- E.g. Image: $N = 256, n = 2$ (bits) $I(2{,}256) = -\log_2\left(\frac{1}{256}\right) = \mathbf{8}$

Entropy: Degree of suprise while observing an average pixel $\boldsymbol{E = -\sum_{i=1}^{N} p(i) \cdot \log_n p(i)}$

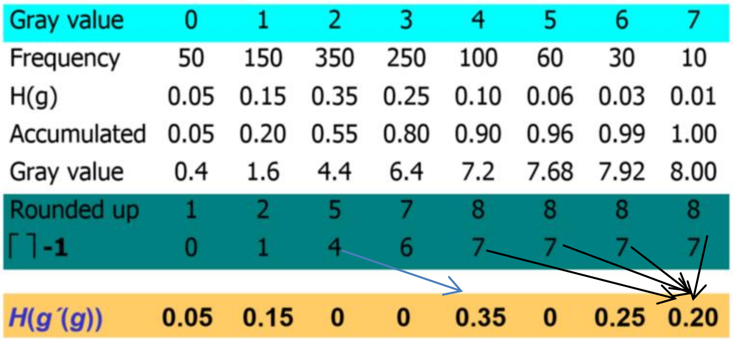
- Formula: Sum over all grey values, the probability of gray value times the $\log_2$ of this value. Change sign.
- To define the probability $p(i)$ of all pixels we need a (normalized) histogram
    - ○ Gives probability for gray values: $H_{norm}(i) = \frac{H(i)}{MN}$ Normalize: Divide by pixel number

Maximizing information content: Maximizing entropy E ⇔ Optimal Image

Transfer function for maximizing: $g'(g) = \int_{0\ldots g} H_{norm}(w)dw$ Bad: Output ∈ ℝ → rounding

Gray value: $g'(g) = N_g \cdot \sum_{w=0\ldots g} H_{norm}(w)$

Accumulated relative prob. · #gray values



Gray value	0	1	2	3	4	5	6	7
Frequency	50	150	350	250	100	60	30	10
H(g)	0.05	0.15	0.35	0.25	0.10	0.06	0.03	0.01
Accumulated	0.05	0.20	0.55	0.80	0.90	0.96	0.99	1.00
Gray value	0.4	1.6	4.4	6.4	7.2	7.68	7.92	8.00
Rounded up	1	2	5	7	8	8	8	8
⌈⌉-1	0	1	4	6	7	7	7	7
H(g´(g))	**0.05**	**0.15**	**0**	**0**	**0.35**	**0**	**0.25**	**0.20**

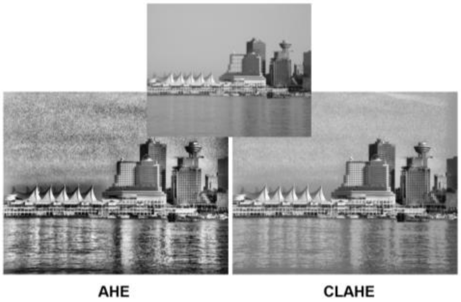
Histogram equalization, separately for each color channel [T]

Transformation is homogeneous (i.e. independent from image content)

Adaptive Histogram Equalization (AHE): Individual transfer function ∀ pixel based on predefined surroundings

Contrast Limited AHE (CLAHE): Improves AHE by preventing excessive local contrast (enhancement up to predefined maximum)

Non-monotonic transfer functions may *strongly* enhance contrast (make details visible) but yields artifacts at edges and isn't invertible (E.g. Pseudocolors)



10. Template Matching

Match similarity of a prototypic template with the image by moving template like a convolution kernel over image

→ A basic approach of object finding which is commonly in use in industry. $T \in \mathbb{N}^{mxn}$ (odd), $g \in \mathbb{N}^{MxN}$

1. Mean Absolute Difference (MAD) Basic approach

- Compute Mean Difference of gray values at kernel anchor point $MAD(x,y) = \frac{1}{mn} \cdot \sum_{ij} |g(x+i,y+j) - T(i,j)|$
- Low results → small difference → High matching

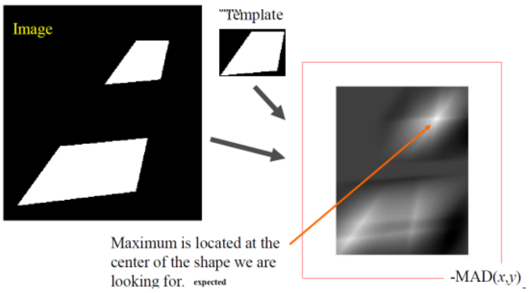


Image
Template
Maximum is located at the center of the shape we are looking for. expected
-MAD(x,y)

2. Correlation Coefficient (CC)

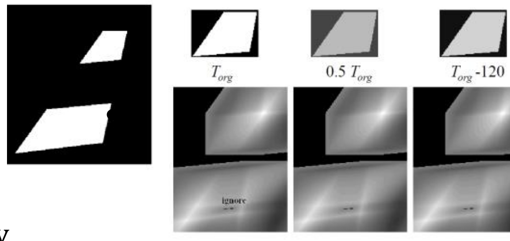
- Template: size is odd, Mean is 0
- Standard deviation: Root of variance (i.e. Mean squared distance to mean value) → σ^2
- Covariance: Mean product of difference of i-th x-value to $\bar{\bar{x}}$ times i-th y-value to $\bar{\bar{y}}$
    - Measures degree of linear relation between 2 variables. $Cov(X,X) = Var(X)$
    - $Cov(X,Y) = \frac{1}{n} \cdot \sum_i (x_i - \bar{x}) \cdot (y_i - \bar{y})$ $X = Y \rightarrow COV(X,X) = \frac{1}{n} \cdot \sum_i (x_i - \bar{x})^2 = \mathrm{Var}(X)$
- Formula: Covariance of image patch with template divided by product of standard deviatons

$$CC_{g,T}(x,y) = \frac{\sigma_{T,g_{patch}}}{\sigma_T^2 \cdot \sigma_{g_{patch}}^2} = \frac{1}{\sigma_T^2 \cdot \sigma_{g_{patch}}^2} \cdot \sum_{i=-m} \sum_{j=-n} (g(x+i,y+j) - \bar{g}) \cdot T(m+i,n+j)$$

- Result is $\in [-1,1]$: 0 → no correlation, 1 → identical up to linear transform −1 → inverted
- Computationally hard: Compute in Fourier space & transform back
- Even Human Vision isn't invariant against all rotations

	Noise	Gray Intens.	Contrast	Rotation	Small bound	Size
MAD	+	-	-	-	-	-
CC	+	+*	+	-	+	+

- Preceding edge detection not useful: exact edge matching necessary
- * Invariant against scaling, but ratio between parts of template & parts of image have to be identical



Results of correlation for optimal / scaled / reduced template values.

Discussion about adaptation of Template Matching to Human Vision:

- Human vision has low-level templates (e.g. edge detecting cells in V1)
- But due to combinatorial explosion (viewpoint, rotation, illumination) grandmother cells can't exist

11. Pattern Recognition

The problem of PR: We hardly have clear concept of patterns (e.g objects like cars). Image taking away components stepwise → where is the border between a car and a non-car?

Supervised Learning: Classifiers in Machine Learning: For train images, user has to *label* whether input is target or not. Classifier learns by tons of these binary decisions to predict unseen input correctly

Reinforcement Learning: Learning by maximizing reward. Often used in AI.

Unsupervised Learning: E.g. ANN, PCA, *K*-Means: No labelling of input data, algorithm clusters the data into different groups by different algorithm-dependant properties. → No feedback/reward system!

Procedure: Sensor for world analysis. Unsupervised Learning: Feature extraction. Classifier: Supervised Learning Task: We need nice classifier to distinguish input signals based on nice features

Classifier properties (→ Machine Learning)

- Divide feature space into well-separated partitions
- Provide generalization for arbitrary inputs of images with same feature
- Separatrix classifier: Represent knowledge explicitly by boundaries. Obtain knowledge from examples
- Prototype classifier: Use examples to derive prototypes (i.e. cluster centers)

Feature extraction

- Good features are more important than good classifier! (Heidemann → <3)
- Discriminative (i.e. well separated)
- Stable against signal variations (noise, rotation, viewpoint, illumination) → Real invariance
- The more sample views the better

Difficulty to transform signal into feature depends on:

1. Distance (signal quality and feature choice: lines vs. object)
2. Abstraction (stylized vs. handwritten letters)
3. Variation (illumination, viewpoint etc.)

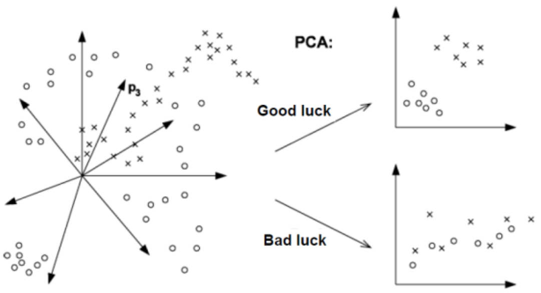
Object recognition

- Object = set of all possible views $\in \mathbb{R}^{M \times N \times C}$ $C :=$ #colour channels
- Object is represented by distribution in this space.
- Complexity depends on variation in appearance (i.e. visual degree of freedom, "vDof")
    - Different rotations, illuminations, occlusions etc. increase variation
    - Symmetry, constant colouring etc. decrease variation

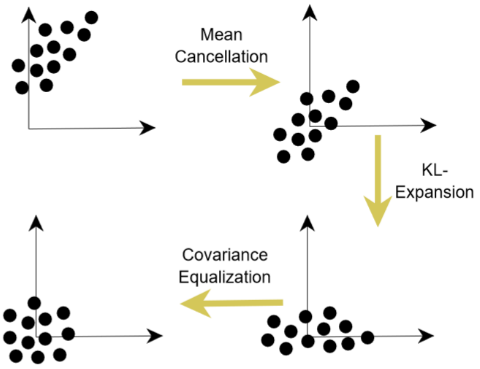
Principal Component Analysis (PCA)

- = Karhunen-Loeve-Transformation (/Expansion)
- A general tool for unsupervised feature extraction

Idea: Map a high-dimensional set of possibly correlated variables (observations) to a set of linearly independent (uncorrelated), low-dimensional variables (so called principal components, PC)



PCA compared to other transformations:



Generally:

- Non parametric method to extract relevant data by computing the most meaningful basis
- Linear dimension reduction method, useful tool for preprocessing of data
- Advantage: Compact and non-redundant representation that even may reveal hidden structure
- Disadvantage: PCs are hardly interpretable
- Project data onto the most meaningful basis that is a linear combination of the original basis vectors
- Assumption: Direction of largest variance of dataset contain interesting information

When can PCA be used? High correlation (similarity) → high redundancy → high effect of PCA

- CV: Pixel space → Feature space (orthogonal transformation)
- We measure/observe d properties for n persons/trials → Each observation vector (point) in $\mathbb{R}^d$
- d-dimensional data $X = \{\vec{x}_1, \vec{x}_n\} \in \mathbb{R}^{d\times n} \rightarrow_{project} m < d$ orthogonal vectors $\{\vec{p}_1, \vec{p}_m\} \in \mathbb{R}^{m\times m}$
- Data is saved in a matrix $X \in \mathbb{R}^{d\times n}$ where rows describe all values of a property, columns of a trial
- Target: Lose as low information as possible, by discarding as many irrelevant information as possible
  - Minimize correlation of observations, shift the basis and construct a vector subspace
  - Principal component vectors $\vec{p}_i$ are orthonormal and in direction of largest variance

Procedure Shortcut: (How PCA is performed in real applications) Long version below

Definitions: $X \in \mathbb{R}^{d\times n} \coloneqq$ data $d \coloneqq$ properties $n \coloneqq$ trials $X_{2,3}$: Value of 2nd prop in 3rd trial

1. Set mean to 0 ∀ properties: Compute the mean for each property d_i, subtract it from all property values
2. Compute covariance (autocorrelation) matrix $C_X = \frac{1}{d-1} XX^T | \in \mathbb{R}^{d\times d}, X =$ data, mean $= 0$ ∀ property d_i

   Eigenvectors of covariance (alternatively autocorrelation) matrix are principal components p_i
3. Compute matrix $P_{C_X} \in R^{n\times d}$ containing eigenvectors of C_X and Eigenvalues $W_{C_X} \in \mathbb{R}^{d\times 1}$

   W contains variances of all properties. Each row $p_i \in P$ is a principal component of X
4. Sort the Eigenvalues $w_i | i \leq d$ in decreasing order and the princ. Comp. $\vec{p}_i$ to match EV with PC

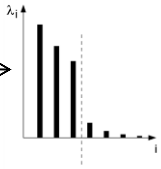
   The higher the eigenvalue the larger the variance, the more important the $p_i$to describe the data
5. Project data X: $R \coloneqq PX | \in \mathbb{R}^{d\times n}$

Procedure (with explanations): Source

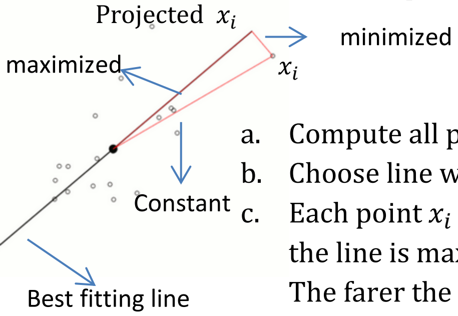
1. Compute Covariance Matrix $C_X \coloneqq \frac{1}{d-1} \cdot XX^T \mid \in R^{dxd}$ (normal variance definition)

- Symmetric, squared matrix: $d = 3$: $$C_X = \begin{pmatrix} Var(d_1) & Cov(d_1,d_2) & Cov(d_1,d_3) \\ Cov(d_2,d_1) & Var(d_2) & Cov(d_2,d_3) \\ Cov(d_3,d_1) & Cov(d_3,d_2) & Var(d_3) \end{pmatrix}$$
- Value at position i, j is covariance between properties d_i, d_j Recall: $(Cov(X,X) = Var(X))$
- Captures correlation between all possible pairs of properties
- Diagonal: Large values → interesting/"principal" (high variance) → Target was to maximize that!
- Off-Diagonal: Large values → High redundancy within data → Target was to minimize that!

2. Target: Transforming C_X to a diagonal matrix C_Y (all off-diagonals are 0).
   - Precisely: Find $P: Y = PX \quad \wedge \quad C_Y = \frac{1}{n-1}YY^T = \frac{1}{n-1}P(XX^T)P^T$ is diagonalised
   - So, we need to choose P such that $P(XX^T)P^T$ is a diagonal matrix
3. Theorem: Symmetric matrix XX^T is diagonalised by $XX^T = EDE^T | D$ ≔diagonal, E ≔Eigenvector ofXX^T
   - Choose P to be E^T: (Eigenvectors of XX^T) → $C_Y = \frac{1}{n-1}P(P^TDP)P^T = \frac{1}{n-1}(PP^{-1})D(PP^{-1}) = \frac{1}{n-1}D$
     - $P(P^TDP)P^T = PP^TDP^TP$
     - Eigenvectors of symmetric matrices are orthogonal → P is orthogonal → $P^T = P^{-1}$
4. Conclusion:
   - So the eigenvectors of XX^T, saved in P are the principal components $p_i$and diagonalize C_Y
   - Length of eigenvector is $= d$ ($=$ #pixels of image → Eigenface)
   - The i-th (diagonal) value of C_Y is the variance of X along p_i
   - $P = \{\vec{p}_1 \dots \vec{p}_n\} \in \mathbb{R}^{d \times n}$ is the set of principal components sorted by relevance for describing the data
   - Often using $m < d$ principal components is sufficient to describing the data appropriately. It's nice if the spectrum of eigenvalues contain a clear cut → easy to find m ⟶
   - The principal components can be interpreted as features, filter kernels or receptive fields

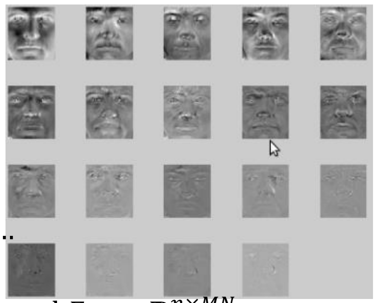


Graphical intuition: Computing the principal components is finding the direction of largest variance.
Think about an iterative process that computes the orthonormal principal components $p_i$that are rows of P



a. Compute all possible lines through mean data point $\bar{x}$ & save average distance to data point $\vec{x}_i$
b. Choose line with minimal average euclidean distance → First principal component p_1
c. Each point x_i is orthogonally projected on the chosen line: Minimizing the distance from x_i to the line is maximizing the distance along the line.
   The farer the projections from $\bar{x}$, the higher the variance in that direction → maximized variance
d. Search i-th line that is orthogonal to 1st ,…i-1-th-PC, goes through $\bar{x}$ and minimizes euclidean distance. → Result is p_i Done until we have $d - 1$ principal components

(p_i are in direction of largest variance. Maximizing it is graphically finding best rotation for old basis)
p_1 has highest variance, $p_2$has 2nd highest…



Eigenfaces (PCA for Faces) Γ: contains faces $\Psi =$ Mean face$= \frac{1}{M}\sum_i \Gamma_i$

- How PCA is actually used in Computer Vision
- $n =$ #faces in dataset, all faces of size MN
- Faces are recorded under artificial conditions: fixed position, size, viewpoint…
- Compute difference face $\Phi_i = \Gamma_i - \Psi$
- Matrix containing eigenvectors of autocorrelation/covariance matrix C_Φ is named $E_{C_\Phi} \in \mathbb{R}^{n \times MN}$
  - Compute eigenvectors and sort them by relevance (for that look at the eigenvalues)
  - Plot rows of E_{C_Φ} → Each plot is one eigenface, last plots are negligible
- Spatial frequency increase with decreasing eigenvalue (low frequencies contribute more to image)

- To compare a new (reshaped) face $F \in \mathbb{R}^{MN\times 1}$ to all test faces from the dataset Γ, project the new face into the eigenspace (i.e. $E_{C_\Phi} \cdot F$) and compare the resulting vector $\in \mathbb{R}^{d\times 1}$ with all test-face-vectors (obtained by same procedure) → comparing = Measuring Euclidean distance

Object/Face recognition by PCA:

- PCA can be treated as a classifier to assign test images to a specific group (*What* object? *Whose* face?)

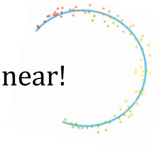
Example: COIL Columbia Object Image Library

- Split data into train data and test data. Compute the principal components (e.g. Eigenfaces) of train data
- Project the train data into the eigenspace (sounds hard, is easy: just multiply image with PCs)
- The obtained point in eigenspace represents the test image
- Compute distance to eigenspace points of all train image → The smaller the distance the closer the images
- COIL: Recognition rate ~ 85%   Train - & test data separated by rotation angle of objects

Improving PCA

1. <u>Spline Interpolation</u>: pieces of continuous functions that are used to connect isolated points in eigenspace that represent the same object under different viewpoints, rotation angle, illumination, etc.
   Why?   Approximation of intermediate views (between sampled test data)
   Interpolation may be projected back from feature to normal space → patterns might become visible
   But: Interpolation is highly ambiguous (high-dimensional space) → rarely of use

2. <u>Non-Linear:</u> Generally hard to match the linear distribution obtained by PCA
   Principal curve can approximate shape distribution of data & maps all data points onto that curve → linear!
   But: Cannot handle clustered data, computationally hard



3. <u>Local PCA:</u>
   a. Make one LPCA for each data cluster. Problem: Define clearly cluster centers?
      - Despite this problem, the approach is normally sufficient
   b. Better: Search for global optimum, approximate data by fix number of cluster centers
      Optimal, but computationally hard.
   - The more cluster centers, the better the approximation

   <u>Algorithm:</u>   3 stages: Clustering (e.g. with K-Means) , PCA, classification
   1. Cluster data T into subcluster T_i and compute the respective center S_i
   2. Transform all points to zero mean (in respect to their cluster centre)
   3. Compute L first principal components $p_{T_i}$of all datasets T_i
   4. Project training samples onto associated principal component
   5. Form training pairs consisting of input data vector and output vector (after LPCA)
   6. Train a classifier for each set

Final step in PCA is classification (ML approach). Results may be anything: binary value, natural/real number
Most famous classifier: Nearest-Neighbour-Classifier, Gaussian Classifier, Support Vector Machine,

Appearance based recognition: Recognition systems (classifier) are trained by sample image

- No explicit design of models
- Properties (light source, surface, illumination) are automatically implemented by sampleimages
- That can also be seen as disadvantage: Classifier skills depend on training scene only!

Conclusion: Technique is suitable for highly differing object types , situations and poses

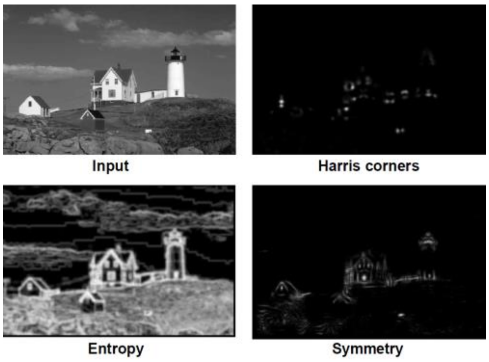
12. Local Features

Object recognition is very hard in real-life applications: Occlusion, illumination, rotation, scaling, …
→ Use local features of target objects:

- Local Features are well-suited for object detection and yield robustness against various operations
- Search for discriminative features (=salient image patches) in objects is motivated by human perception. It's easy for us to recognize images by just perceiving tiny but discriminative parts

How to extract Local Features – Guidance:

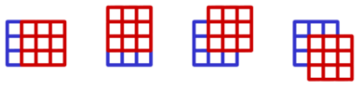
1. **Interest Point Detection (IP):**
    - First step in Local Feature analysis. Has to work well, otherwise early game over.
    - Detect interesting regions in input image and describe them appropriately. Search for them in target
    - IP should be robust against all kinds of disruptions
    - Context free: image independent
        - Maxima of a predefined saliency measurement
        - Rare within common images
    - Context dependent:
        - Creating a model
        - Global preanalysis. E.g. white spot is salient in target images, but not generally
        - Rare within a particular image (or a particular type of images)
    - Generally: Detected by simple features (local maxima in the scale space)


Input
Harris corners
Entropy
Symmetry

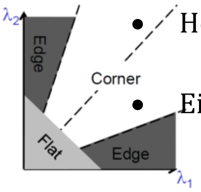
2. **Point → Region** 3 possibilities:
    - Associate region of predefined size (naïve)
    - Scale adaptive (Perhaps: region size depends on scaling in which IP was found)
    - Blob (Binary Large OBject): Binary image: Search connected region around IP
3. **Region Descriptor:** describes set of features of Interest Region (IR)
    - One descriptor for each IP
        - Easiest example: Complete local patch → Template Matching
    - Should be: unique, robust, invariant, fast
4. **Similarity Measurement**:
    - IP/IR can be compared by comparing the Euclidean distance of their unique descriptor vectors

Implementations of IP-Detectors: Detect regions that are unique in their surroundings

1. **Moravec-IP-Operator:** Not only interest pixels, work also for interest regions
    - Searches for large intensity variations in *all* directions → corner detector (by accident!)
    - Idea: Self comparison of values in target window/pixel to values in window/pixel (of same size) shifted in 4 directions: Shifted window is red: 4 directions clockwise from top-shift
        - Compute mean squared difference in 4 directions
    - Output is minimum of 4 results (→ guarantees that variance is high in *all* directions)
    - Easy to implement, less computational effort
    - Not robust against rotations, direction dependent (anisotropic), "hard" window, fixed scale



2. **Harris-Corner-Detector:** Same idea, but many improvements of Moravec
    - Gaussian "sliding" window: $w(u,v) = \exp(-\frac{u^2+v^2}{2\sigma^2})$
    - Use Sobel Filter to get partial derivatives g_x (horizontal) and g_y (vertical)
        - $S := \sum_{x,y} w_{x,y} \begin{pmatrix} g_x^2 & g_x g_y \\ g_y g_x & g_y^2 \end{pmatrix}$ → approx. Laplace-Filter

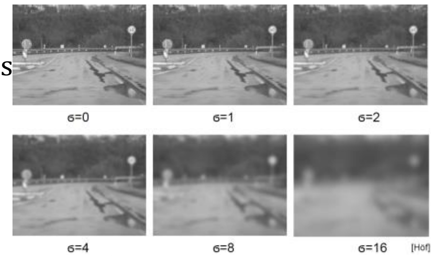


    - How to recognize if edge, corner or nothing was found? →
        - Eigenvalues of S: Both small → homogeneous Both large → corner Large&Small → Corner
    - Eigenvalues are hard to compute here; better: S' = Hesse Matrix (like in S)
        - For each point (x,y) compute the matrix $A(x,y) = \sum_{i,j\in\Omega_{x,y}} S'$
            - For all 4 matrices in S' compute one value (i.e. the sum of gradients in environment Ω)
        - $R := \det(A) - k \cdot Spur(A)^2$ with $k := 0{,}04$

        - No need to know real eigenvalues, just their ratio counts
        - Btw: $\det(A) = \lambda_1 \cdot \ldots \cdot \lambda_n$ $\quad Spur(A) = \lambda_1 + \cdots + \lambda_n$
    - R = corner strength of point (x, y)
    - Local maximum in $4/8$ neighbourhood in R are corners
- Direction independent (isotropic), no hard window, rotations
- Fixed scale

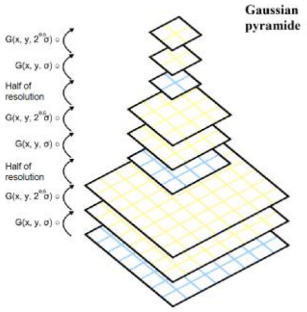
Scale-Invariant-Feature-Transform (SIFT): Detect & describes local features

- Invariant against: rotation, scaling, translation (size), noise, illumination, perspective, partial occlusion
- Not invariant against affine transformation (extendable)
- Breakthrough in CV! (1999)
- Finds distinctive IPs in scale space , regions about IP are described by descriptors Similarity measure by comparing descriptor vectors of other images (Eucl. dist)
- Applications: Robot can detect his own position, stitching (panoramas)



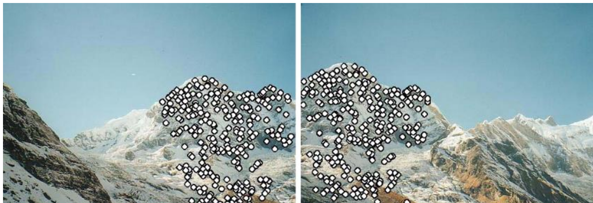
- <u>Idea</u>: Compute several octaves (i.e. scalings of the image). One octave is e.g. halving the size of the image by *imresize* o.ä. (Alternatively this can be reached by doubling the filter size). For each octave we compute several scale levels (i.e. we use a Gaussian kernel of increasing size *and* with increasing variance). We search maxima over kernel size and octaves and make sure that they appear in different octaves at the same position (different kernel size allowed)
- How scaling is implemented:
    - Convolute (again) with Gaussian kernels of increasing size & variance. It would be nice to apply a Laplacian Filter (edge detection). But: Computationally hard → approximate by the Difference of Gaussian (DoG) between the different scalings. Then continue with maxima detection (maxima are, where the one filter has a high effect and the other a low one → e.g. corners)

1. **Detector**: Scale adaptive IP
    - The increasing kernel size blurs the image. Stable keypoint should be stable against various intensities of blurring in various image scalings
    - Each IP consists of 4 features: position (x, y), scaling (σ) and orientation
    - IPs are searched within several scalings (increasing scale → deletes details → reducing resolution)
        - This space (see left) is called Gaussian scaling space
        - Scale space obtained by convolution: $M(x, y, \sigma) = G(x, y, \sigma) * g(x, y)$
        - Scaling reached by repeated smoothings of image by means of a Gaussian

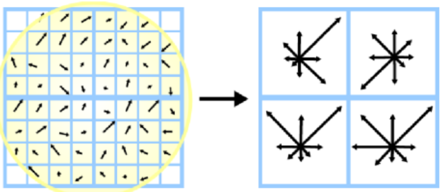


    - <u>Maxima Detection</u> (e.g. Non-Maximum-Suppression, below)
    - Search over all octaves over all scalings:
    - If a maxima (vary x, y, σ) is found, save the coordinates x, y, σ → candidates
        - Maxima includes also adjacent scales: 26 neighbors (8+9+9)
        - Candidate extraction reduces computational effort
    - Still too many candidates. Unstable keypoints (edges, low contrast) are going to be removed
    - Vary σ check whether maxima is found at "same" (correctly shifted) x, y, σ position
        - Discards low contrast regions and edge points, but holds corners
        - If yes, than we found an IP! "Stable keypoint"
        - Can also be reached by approximating Hessian Matrix by considering neighbouring IP

    → yields scaling-invariance (most distinctive feature of SIFT-approach)
    - Each object has a characteristic scale (maximum in scale space)
        - Different techniques to find: E.g. extrema or zero-crossings of 2nd derivative,
    - To get rotation invariance we need (isotropic) rotation-invariant kernel.
        - E.g. 2D-Gaussian, or normalized LoG-Filter
        - Save the orientation as feature (see above)

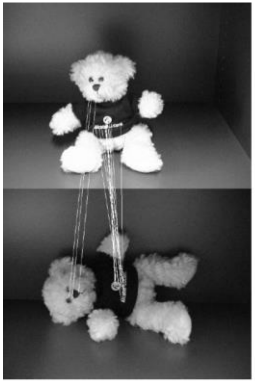


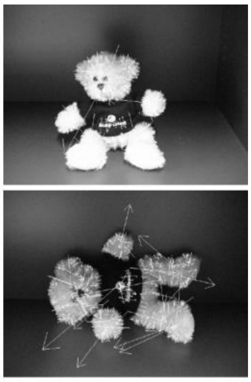
- **Descriptor**: Histogram of oriented gradients (HoG)
    - Each keypoint has its own descriptor (i.e. a vector with 128 values)
    - 3 features of vector are: position, orientation, scale
    - Intuitively: 3D-histogram of gradient over position and orientation of keypoint
    - Computation: Look at the 4×4 patch around the keypoint
        - Create 16 histograms of orientations based on x, y, σ,
        - 8 bins each (for orientation angle $0° - 360°$) $16 \cdot 8 = 128$
        - Gradient orientation in 8 directions
    - Gaussian weighting of values according to distance to keypoint
    - Finally vector is normalized to make comparisons easier and improves robustness against illumination



-
    - Compares images by comparing descriptors
        - Euclidean distance of vectors
    - How to decide if descriptor is accepted or rejected?
        - Search for 2 most similar vectors in database
        - Divide them: d_1/d_2
        - If too large (vectors are too similar) → reject
            - Reason: not discriminative, undecidable

- SIFT can be extended in various ways:
    - Descriptors in colour space → better discrimination
    - SURF (Speeded Up Robust Features)
    - MSER (Maximally Stable Extremal Regions)


Matching features

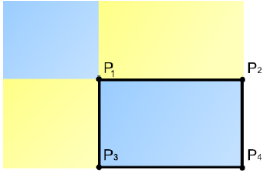

Found keypoints

Rectangle Features – Viola & Jones

- Can be implemented for *real-time* object recognition (→ efficient computation)
- Various approaches for implementation
    - Can approximate convolution
    - Using Integral Image
    - Classifiers (AdaBoost, Degenerate decision tree)

Integral Image: Facilitates efficient computation of rectangular features

- In Integral Image I of image g each pixel x, y denotes the sum of pixel-values in the rectangle spanned by the origin and (x, y) $I(x, y) := \sum_{x'<x} \sum_{y'<y} g(x', y')$ → Computationally hard
- Recursively implementable: $I(x, y) := g(x, y) + I(x-1, y) + I(x, y-1) - I(x-1, y-1)$
    - Advantage: One trial over image to compute complete integral image
- Once I is computed and given arbitrary pixels P_1 and P_4 (spanning the diagonal) of a rectangle, we can define the integral image (i.e. the pixel sum) of the rectangle immediately (access only 4 values of I)
    - $\int \blacksquare(P_1, P_4) := I(P_4) - I(P_3) - I(P_2) + I(P_1)$
- Allows testing with different window sizes → yields scaling invariance



Boosting: Algorithm to combine several weak classifier to obtain a strong one

- Finite set of M weak classifier k_i that classify with an error $< 50\%$ (hardly a restriction…)
    - Weak classifiers often just consider one feature (bad), But: quickly evaluable
- Trained by finite set of labelled vectors x_i with binary output
- Result: Strong classifier $C := sgn\left(\sum_j^M \alpha_j k_j(x_i)\right)$ $\alpha :=$ linear components, $sgn :=$ VZF
- Each weak classifier represents exactly one feature
- M adaptation steps:
    - Select best remaining classifier, add it to C by weight α_i reflecting performance

  - Performance includes weighting of samples: Initially all samples have same weighting
    - In each trial weighting is adapted: Incorrectly classified samples become higher weights
      - exponential error measurement
    - Classifiers with errors $e > 0{,}5$ (worse than chance): Change sign → weak classifier

AdaBoost: (Adaptive Boosting) Idea: Classifier Cascade to recognize rectangular features

- AdaBoost is one way to implement Boosting;
- <u>Procedure</u>: Train all weak classifiers, add classifier with lowest (adapted) error to the final (strong) classifier by a particular weighting and adjust the threshold for the remained classifiers. → Train again
- Decision of classifiers can be generally divided into:

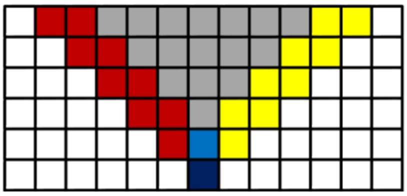
True Positive (correctly recognized)	**False Positive** (incorrectly recognized)
False Negative (incorrectly rejected)	**True Negative** (correctly rejected)
$\sum = 1$ (sum of targets)	$\sum = 1$ (sum of trash)

- If our cascade includes $M = 30$ classifiers, we need an average TP rate of 99,3% to obtain an overall detection rate (TP) of 80% $(0.993^{30} \approx 0.8)$
- To reach that, we increase the TP rate. That obviously increases FP rate, but average FP of 80% results in 0.12 FP in final decision. $(0.80^{30} \approx 0.12)$
- It is useful to use classifier cascades: 10 classifiers with 20 features each are 10 times faster than one classifier with 200 features
- Procedure: Compute all features that might indicate a rectangle at all positions in all scales, select feature with best separation between positive and negative and assign threshold θ_j

<u>Direct Feature Selection:</u> Similar to AdaBoost but classifier are trained just once. Then some classifier help to maximize feature detection, the others help to minimize false positive rate.

Cascade Classification of rectangular features:

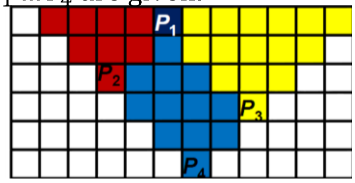
1. Compute integral image E
2. Evaluate search window (rectangular features) for all positions
3. Enlarge search window (commonly by 1.5), goto step 2.



How to find rotated rectangles? → Diagonal integral image (45° rotation)

- not the same I as for computing the integral image (x, y) =(row, col)
- $y \geq x$ otherwise deformed rectangle that meets left vertical border of image
  - Iterative: $I_D(x,y) \coloneqq \sum_{x'=1}^{x} \sum_{y'=y-(x-x')}^{y+(x-x')} g(x',y')$ Computationally hard
  - Recursiv: $\boldsymbol{I_D(x,y) = g(x,y) + g(x,y-1) + I_D(x-1,y-1) + I_D(x-1,y+1) - I_D(x-2,y)}$
    - Advantage: One trial to compute values for all pixels
- All rectangles rotated by 45° computable by looking up 5 values, iff coordinates $P_1 \dots P_4$ are given.
  - $\int \blacksquare(P_1, P_4) \coloneqq I(P_4) - I(P_3) - I(P_2) + I(P_1)$
- Adding this improves performance of AdaBoost can be improved by 10%

Viola-Jones recognizes pedestrians in varieties of shapes, before different backgrounds and under different illuminations, viewpoints etc.

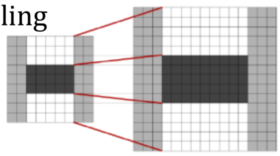


SURF – Speeded Up Robust Features Combination of SIFT and Rectangular Features

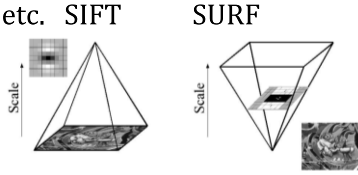
- Same quality in results as SIFT, but faster by changing some concepts:
- Replace Gaussian Filter of SIFT by Mean-Filter that is computable in constant time by the integral image
- The lengthy convolution process is approximated by the integral image which functions as a look-up-table for the values for a convolution
- The approximation chain is now: We are interested in the 2nd derivative, which is approximated by the Laplace Filter. This approximation is noise sensitive, so we use the Laplace-Of-Gaussian instead. Since

this one is hard to compute we approximate by the Difference-Of-Gaussian. This convolutional step is finally approximated by the integral image

- IP are detected by use of Hesse-Matrix $H(x,y,\sigma) := \begin{pmatrix} L_{XX} & L_{XY} \\ L_{YX} & L_{YY} \end{pmatrix}$ $Det(H) = L_{XX}L_{YY} - L_{XY}^2$
    - L_{XY} is the second derivative of g convoluted with different Gaussians G for scaling
    - H is symmetric (Satz von Schwarz)
    - IPs are maxima of normalized (according to scaling) determinant
    - Computationally very hard. Discretization of Gaussian is unavoidable

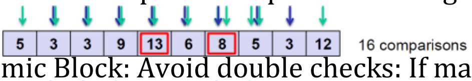


- Better: Approximate Hesse-Matrix by 9×9 boxlets (rectangle features). Called D_{XX}etc.
    - Only some scalings can be computed: Filter needs central pixel
    - Filter needs to be increased by 6 pixels ($15 \times 15, 21 \times 21$ ...) to keep shape
- Scaling is reached by up-scaling the filter, not by down-scaling the image
    - Possible since computation of integral image is independent from filter size

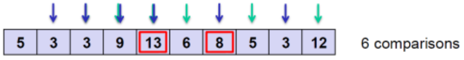


Non-Maximum-Suppression (NMS): Maxima detection to obtain thin edges

1. Naïve Idea: Compare each pixel to its neighbors: → 1.5 comparisons/pixel

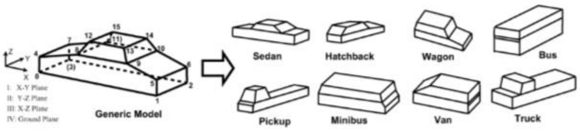


2. Dynamic Block: Avoid double checks: If maximum was found, skip next pixel: → 0.8 c/p



E.g.: First arrow: Compare to the right, no maximum possible, don't look left to 5…

- SURF-Descriptors:
    - Feature vector with 64 dimensions
    - Normalized (reason: different scaling). → Contrast invariance
    - More robust against noise than SIFT



13. Wire Frame Models

- PCA, SIFT, Viola & Jones, SURF are all *Bottom Up Processes* (i.e. data driven) Advantage: Generic
- Wire-Frame-Models are obviously models → *Top-Down-Process*

Idea: Objects are represented exclusively by their edges: Used in computergraphics e.g. for games

- Predefined deformable 3D-wire frame models with 12 flexible parameter to fit underlying object shape
- Advantage: Computationally practicable
- Disadvantage: Ambiguities: Same model might represent >1 real objects (not bijective)

How to create a wire-frame model?

- Define corners of the model and connect them by edges
- Introduce n parameters that restrict the distance between meaningful tupels of edges to a certain range
    - Possible restrictions: absolute (natural numbers), or relative (comparing with another distance)
- n parameters make up $\mathbb{R}^n$ which is restricted to a small subspace by the predefined ranges
- Vehicle models: $n = 12$, $+3$ parameters for position (x, y, θ)

Histogram of oriented gradients (HoG): Feature descriptors for object recognition

- Idea: Local object appearance can be described by distribution of intensity gradients (or edge direction)
- Used in Human detection and Wire Frame Models

Procedure:

1. Compute the gradient magnitude and orientation of the image
2. Divide image into small connected regions called cells (e.g. 4×4)
3. Define blocks (e.g. 2×2 cells) that slide convolution-like (but overlapping!) over image
4. For each cell: Define bins (e.g. 8 to cover area of 0° to 360°)
5. Look at each pixel: Increase bin of orientation *gradient* weighted by value in *magnitude*
    a. Count occurrences of orientation gradients of all pixels of cell and display as histogram
    b. Normalization of blocks (blocks are groups of spatially connected cells) (optional)

6. Concatenate histograms to get the HOG descriptor. Length: $\#block \times 2 \times 2 \times 8$
    a. For each block, there is for each cell a vector of 8 values (bins)
7. HOG Descriptor is vector of concatenated components of normalized histograms of all block regions

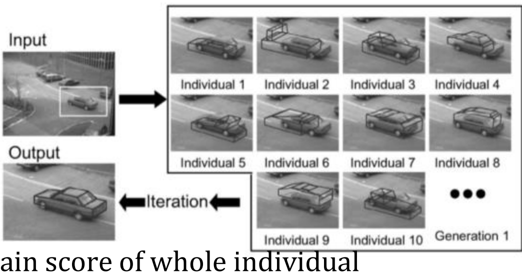
The whole is a preprocessing step, to get nice initial positions for the matching process. Idea is, that the basic edges of the model are easily found in the image.

Matching of WFM: Adapt parameters such that the wire-frame corners match gray value edges
→ Maximize fitness score (degree of matching) **Optimization problem**

- Initial position of wire frame model is computed by the maximum of the HoG (see above)
- Match the WFM by iteratively collecting evidence for the lines of the model
- Compute rectangle around possible position of line and compute gradient of rectangle
    - If gradients are perpendicular to line we have a good match
- Matching is implemented by an evolutionary computation approach

Evolutionary Computation (Darwin's principles) → Optimize 15 parameters

- Individuals are rectangles that make up the target WFM
- The individuals are "randomly" assigned (in frame of preprocessing)
- Compute gradient value ∀ pixels on a line and compare with the belonging part of the image → Fitness score of line. Combine them to obtain score of whole individual



- Each individual is projected into image plane (is a point in $\mathbb{R}^{15}$), individuals with high fitness score compute individuals for next generation
- Generations are somehow the weighted superposition of individuals
- Iterating over generations improve average score
- Saturation after ~15 generations
- Advantage of "random" initial assignment → unlikely to stuck in local maxima

14. Cosine & Wavelet Transform

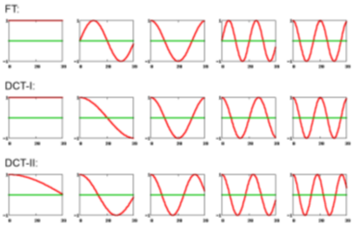
Orthogonal transforms are used to extract features (Fourier) or enhance special parts

Discrete Cosine Transfom (DCT): Discrete, real, linear, orthogonal transform

- Similar to Fourier Transformation (cosine as basis functions, computation of coefficients identical), but no complex basis functions and no sine basis functions
    - Same number of basis functions in DCT I and FT needed
- Disadvantage: Cosine is an <u>even function</u> (i.e. symmetric to Y-axis)
- Solution: Allow <u>odd functions</u> (point symmetric to origin) as Sine as basis function
    - Implementation in DCT: Allow Cosine basis functions with half integer frequencies
- The DCT can easily be computed from the FT (which is well implemented in most software → advantage for DCT) and shares the property of being separable
- Used in compression (e.g. JPG) by leaving out basis functions with small amplitude
- DCT is usually applied to squares of an image
- DCT I differs from DCT II by being even at certain positions of their interval
    - DCT-I: no phase shift, DCT-II: with phase shift

Frequencies of different orthogonal transformations: always $\cdot\ 2\pi$ sign is alternating ∀ 3

1. Fourier: $0, 1, 2, \ldots, n$
2. DCT I : $0, \frac{1}{2}, 1, \frac{3}{2} \ldots, \frac{n}{2}$
3. DCT II : $\frac{1}{4}, \frac{3}{4}, \frac{5}{4}, \ldots \frac{n}{2}$



amount of periods

Wavelet Transform

- FT & DCT analyse the wave structure of a complete image, BUT partition the image artificially.
    - Most dominant image structures (sharp borders with high wave frequency) are only locally.

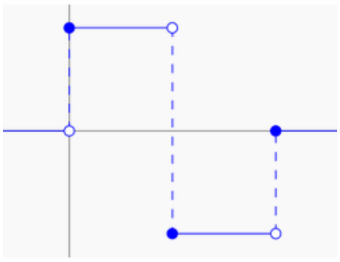
    - In FT that implies a need of many low frequency waves to balance out the strong ones
- Advantage: Overcome this by a wavelet transform, which is a *local FT*.
    - Meaning the functions are $\neq 0$ only in the area in which they are needed to model image
- Disadvantage: The basis functions are not necessarily orthogonal
    - does not imply information loss but complicates treatment
- Wavelets basis functions differ by location d (distance to origin), frequency, scaling s
    - The computation is recursively based on the mother wavelet $\Psi_{0,0}$
    - Any basis function can be computed by combining scaling and shifting property
- Disadvantage 2: Computational effort
    - Compensated by using scaling & shifting (convolve frequencies)

Gabor Wavelet Transform:

- Special case of Short-Time FT (STFT)
- Can be used to compress functions (tighter frequency) in comparison with Wavelet Transform

Haar-Wavelet: (Alfred Haar)

- First and most famous wavelet → Easy to implement, computationally fast
- Obtained by combining two rectangular functions
- Disadvantage: not continuous (→ not differentiable)



15. Compression

<u>Target</u>: Reduce redundancies within an image, e.g. by JPG compression

Allows to store more data → High financial interest

1. Lossless compression: Complete reproduction possible
2. Lossy compression: Loss of information is allowed, but make sure that they do not cause problems

Types of redundancies:

1. Coding redundancy (reduce the number of bits that code for a gray value → lossless)
2. Interpixel redundancy (Many pixels predict colour of neighbouring pixel)
    a. Especially over very short distances. In principle also possible for chessboards etc
3. Pyschovisual redundancy (information required for cognition)

Compression ratio: $C = \frac{n_1}{n_2}$ measures quality of compression (higher =better, below 1 → worse than original)

Maximal possible compression factor: $\frac{\#bits}{entropy}$

Example:

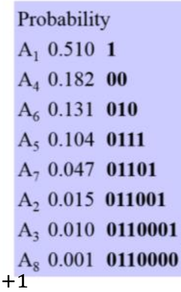
Greyscale image : Needs 8 bits to represent all $2^8 = 256$ possibilities → Max. entropy = 8

Binary image : 2 values: Need 1 bit to represent all 2^1 possibilities → Max. entropy = 1

Hufman Coding **Lossless** Coding redundancy

So far: Same codelength for each value (8 bits)

<u>Idea</u>: Examine histogram to detect that some colours occur frequently. Choose different code such that the frequent values get short codes, while assigning longer ones to the less frequent values



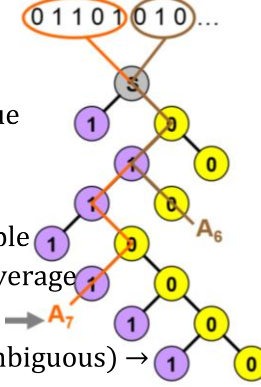
Probability		
A_1	0.510	**1**
A_4	0.182	**00**
A_6	0.131	**010**
A_5	0.104	**0111**
A_7	0.047	**01101**
A_2	0.015	**011001**
A_3	0.010	**0110001**
A_8	0.001	**0110000**

<u>Procedure:</u>

1. Order values according to probability → $p_1 \dots p_n$
2. Generate a binary tree by merging the iteratively the two lowest probabilities to a parent node p_{n+1}
   Write values smaller than the parent node left from it, larger ones right to it
   Repeat until only 2 values remain (p_{n+1} etc. is treated as a normal probability)
3. Recursively assign binary codes: Assign 0 for taking right path, 1 for left one. Always append number

<u>Result:</u>

- The complete image is transformed to one binary string. For decoding we need the tree:
    - Start at top, read next number til leaf is reached. The path through the tree codes a



special string. We look-up the value in the probability table to get the real pixel value For the next pixel start again at top

- Most probable value gets smallest code (1), then length stepwise increases (see image)
- A normal binary assignment of the 8 numbers would need 3 bits. The entropy of this example is 2.04 and the required memory by Hufman-Coding is $\sum_i p(A_i) \cdot \#code(A_i) = 2.09$, so in average 2.09 bits are necessary to encode the information
- Reason not to take increasing binary code 0,1,10,11,100,101: Code is not prefix free (i.e. ambiguous) → Saving where the gaps are needs additional space (Morse has 3 symbols: Short, Long, GAP)
    - Prefix free code: No code word is part prefix of another code word → guarantees uniqueness
    - Nice thing: Just concatenate all codes in one binary string.
    - As far as the end of search tree is reached, jump to top → no gaps needed. Code for each leave symbol and save.
    - Computation time : $O(n)$

Run Length Encoding (interpixel redundancy) **Lossless** – Naïve approach – Used in Windows Bitmap

- Idea: Remove redundancy by replacing n identical neighbouring (mostly row-wise) pixels of gray-value g with a tupel $< n, g >$. Compression rate $C = \frac{r}{2}$, $r := \emptyset$ run length
- Worst case ($r = 1$): amount of data is *doubled* (possible in fine greyscale images like clouds)
- Row handling is artificial, preprocessing with region labelling possible

Bit planes: Divide uncompressed 8-bit-image into 8 binary images (one for each bit position). Low planes are noisy, but high ones show overall structure (numbering according to slides, e.g. 7 indicates the highest plane with a contribution of $2^7 = 128$ to the overall image

Gray Code: (Frank Gray) Lossless

- Idea: 2 successive numbers in a code differ by at most one bit
    - E.g. 3 bits: 000-100-110-010-011-111-101-001
    - Code is not overlapping and unique
    - But: Different codes can represent same structure: $\#GrayCodes = \#Hamilton\ circle$ in hyper-cube

Psychovisual redundancy: **Lossy** Orthogonal transformations separating relevant parts

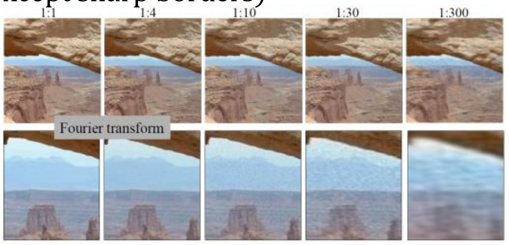
- If an image is not redundant: Impossible to remove anything without loss of information
    - But: 256 grey values represent more colours that are distinguishable by humans (30-100)
    - → Most images contain redundant information
- One idea: Half the amount of possible grey values (e.g. 8 bit image → 4 bit image)
    - Disadvantage: Mach bands

        - Visual phenomenon to perceive at the border of a uniquely coloured area a line that distinguishes strongly from neighbouring area (artificial contrast enhancement by our visual system)
    - Reducible by adding noise after compressing

Using Transformations to compress images

Reduction of high frequencies (Quantization)

- Most image information are contained in low frequencies (everything except sharp borders)
- Transform an image using e.g. FT,DCT, Wavelets,
    - Sort the coefficients by their magnitude
    - Map onto natural numbers (lossy method)
    - Use inverse FT to code back to image space
- Zooming makes blurring visible (stronger the more we precompress)
- Coefficient number is higher the more and the stronger the edges are
- Part of the JPG-Compression:



- Decompose image into subimages
- Size of sub images must consider ∅ number of frequencies containing relevant information
- Transform subimages with FT
- Apply Run Length Encoding on the mean values of all subimages
- Sort the remaining values of all subimages along the diagonal

Quantization by using Wavelets:

- Wavelet decomposition makes artificial division into subimages unnecessary (local FT)
- Yields much better results than normal FT or DCT compression
- Used in JPG-2000

16. Motion Analysis

Sparse motion analysis: continuous e.g. movement tracking like in eye tracking → not further handled

Dense motion analyis: Motion analysis by means of vector fields.

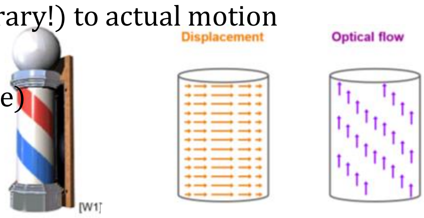
2 basic methods: Horn-Schunck-method vs. Lucas-Kanade-method

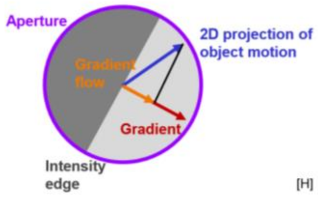
Optical Flow: A vector that describes for every pixel the detected motion (→ Discrete)

- Optical flow is the motion *visible* in the image plane (see below)
- Local optimal flow is estimated by the pixel's predefined environment
- Is assumed to be constant over short distances (see below)

3D motion → 2D image: **Aperture Problem**

- If a real-world-object moves in 3D, its new position is again mapped onto the 2D image surface
  - Possibly ambiguous: The *visible* displacement may be different (contrary!) to actual motion
  - Barber's pole illusion etc. →
- Reconstruction of real physical movement is hardly possible (limited aperture)
- Target: Since true 2D displacement is not detectable, detect the **flow of the gradient**
  - $\frac{\partial g}{\partial \vec{x}} = (\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y})$
  - Better approximation for 2D motion → use it to detect optical flow





Horn-Schank-Method Basic algorithm to detect optical flow

- Intensity constant assumption: Intensity of moving objects keep constant (→ ignore illumination etc.)
  - $g(x, y, t) = g(x + \Delta x, y + \Delta y, t + \Delta t)$ Δ indicating the change
- Make Taylor-expansion of this equation to obtain the **optical flow velocity:**
  - $\vec{v} := (v_x, v_y) = (\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t})$ (visible change)
- Leads to **optical flow equation**: $\frac{\partial g}{\partial \vec{x}} \cdot \vec{v} + \frac{\partial g}{\partial t} = 0 = v_x \cdot g_x + v_y \cdot g_y + g_t$ Horn-Schunck-constraint
  - Image is discrete (there is no "real" solution) → Compute g_x, g_y by a Sobel Filter
  - 2 variables v_x and v_y in one equation → underdetermined
- This problem exists for all pixels! → $g \in \mathbb{N}^{M \times N}$ → MN equations with $2MN$ unknowns.
- Solutions are not strongly mathematically connected but somehow influence/determine each other (nice!)
  → Finding optical flow equation is an optimization problem

New target: Assign optical flow vector such that optical flow equation is fulfilled as good as possible

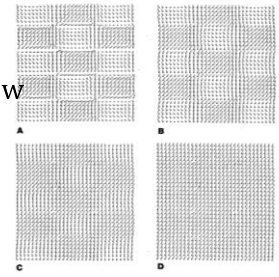
- Minimize **data error**: $E_{Data} = \sum_{pixel} (v_x \cdot g_x + v_y \cdot g_y + g_t)^2$
- Additional constraint: **Smoothness error:**
  - Adjacent pixels have identical optical flow (see above) → 1st derivative is small
  - Violated only for areas where non-horizontal movement takes place

  → Minimize $E = E_{Data} + \lambda E_{Smoothness}$

- To find the unknows $v_x, v_y,$ we plug it into the Euler-Langrange-Differential-Equation
- And solve the result iteratively using the Jacobi-Method
  - Iterative procedure as alternative to Gaussian
  - n linear equations with n variables each → ∃! Solution
  - Iteration s to $s+1$: $x_i^{s+1} := {}^{1}/_{a_{ii}}(b_i - \sum_{j \neq i} a_{ij} x_j^s)$
  - Choose suitable start values, i.e. $v_x^0(x, y) = 0$ (expect no flow at all)
  - Compute g_x, g_y, g_t better by a $2 \times 2 \times 2$ spatiotemporal mask (coherent for x, y, t)
  - Why so small mask, why no Gaussian filtering etc: Similar effect obtained by replacing derivation with linear equation (see above)

Result:

- $v_x \cdot g_x + v_y \cdot g_y + g_t = 0$ defines a straight line that is perpendicular to the gradient flow
- Optical flow vector is *somewhere* on line (and perpendicular)
  - Reason for imprecision: Aperture problem
- One gets one line for each pixel



Lucas-Kanade-Method

- Assumption: Optical flow is identical for neighbouring pixels (as in HSM)
- Idea: Lines of corresponding pixels cross at one point → Point of "true motion" → no strict solution/intersec.
- Plug optical flow equation into 2 close pixels and *assume:* $\vec{v}(x_1, y_1) \approx \vec{v}(x_2, y_2)$ → 2 equations, 2 unknowns
  - Easy solution for $n = 2$ neighbours $\begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} g_{x1} & g_{y1} \\ g_{x2} & g_{y2} \end{pmatrix}^{-1} \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$ b is the result
  - Overdetermined for $n > 2$ → Least square solution: $\vec{v} = (G^T G)^{-1} G^T \vec{b}$
- Procedure: Choose n neighbouring pixels, estimate g_x, g_y, g_t (sobel-like) , make overdetermined system, solve using least square solution