Visual Computing

by dcamenisch

1 Introduction

This document is a summary of the 2022 edition of the lecture *Visual Computing* at ETH Zurich. I do not guarantee correctness or completeness, nor is this document endorsed by the lecturers. If you spot any mistakes or find other improvements, feel free to open a pull request at `https://github.com/DannyCamenisch/vc-summary`. This work is published as CC BY-NC-SA.

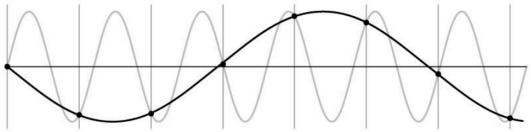
2 The Digital Image

An image is simply a continuous function over 2 or 3 variables (XY-coordinates and possibly time). Usually we use brightness as the value of the function, but other physical values can also be used. For a computer this is just a collection of numbers, but instead of continuous values we have discrete. Note that in real life images are never completely random and almost always contain some structure. It is important to know that **pixels are not little squares**, they are point measurements.
When taking a picture with a digital camera, we can encounter various problems, e.g.:

- Transmission Interference
- Compression Artefacts
- Spilling
- Sensor Noise

2.1 Sampling

When taking an image, we are sampling such a continuous. When trying to reconstruct the original function, we can encounter undersampling, i.e. when we loose information due to a too low amount of sampling points.

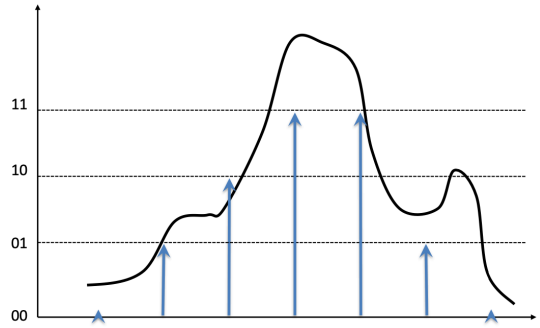


Due to undersampling, the result can not be distinguished from a lower or a higher frequency wave. Signal disguised as other frequencies is also called **aliasing**.

Nyquist-Shannon Sampling Theorem
For sine waves we have to sample at half the wave length. This corresponds to double the frequency, we also call this the **Nyquist Frequency**.

2.2 Quantization

Another problem we have to deal with is **quantization**, since the real valued function will get digital (integer) values, it is lossy. Compared to sampling which lets us reconstruct the original function. Simple quantization uses equally spaced levels with k intervals.



2.3 Image Properties

Image resolution is divided into two parts:

- Geometric Resolution: How many pixels per area
- Radiometric Resolution: How many bits per pixel

2.4 Noise

When taking pictures we can almost always encounter some noise. A common way to model this is additive gaussian noise:

$$I(x,y) = f(x,y) + c, \qquad c \sim \mathcal{N}(0,\sigma^2)$$

The signal to noise ratio (SNR) is an index of image quality:

$$\text{SNR} = \frac{F}{\sigma}, \qquad F = \frac{1}{XY}\sum_{x=1}^{X}\sum_{y=1}^{Y} f(x,y)$$

The usefulness for this metric can vary drastically depending on the type of image (dark images will have a higher SNR compared to bright images). Therefore we introduce peak SNR:

$$\text{PSNR} = \frac{F_{\max}}{\sigma}$$

3 Segmentation

Image segmentation is often viewed as the ultimate classification problem, once solved, computer vision is solved. A complete segmentation of an image is a finite set of disjunct regions $R_1, ..., R_n$, such that $I = \bigcup R_i$.

3.1 Thresholding

Thresholding is a simple segmentation process, that produces a binary image by labelling each pixel in or out of the region of interest. We do this by comparison of the grey level with a threshold value T. Another, better approach can be chromakeying. Hereby we measure the distance from a defined color g:

$$I_\alpha = |I - g| > T$$

One limit of thresholding is that it does not consider image context.

3.2 Segmentation Performance

If we want to choose the best performing segmentation algorithm or determine a good value for T, we need a performance metric. To use automatic analysis, one needs to know the true classification of each test, for this the test images have to be segmented by hand.
One performance metric is the ROC curve. This curve characterizes the error trade-off in binary classification tasks, by plotting the true positive fraction against the false positive fraction. We often choose the operating point on the ROC curve, by assigning cost and values to each outcome:

- V_{TN} - value of true negative
- V_{TP} - value of true positive

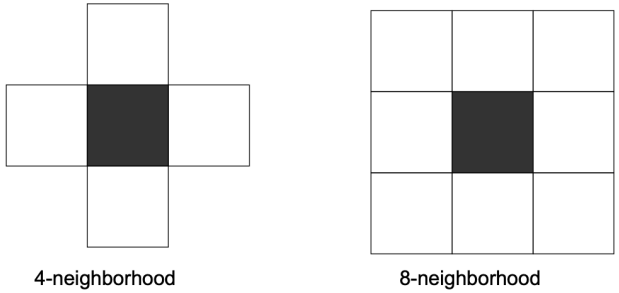
- C_{FN} - cost of false negative
- C_{FP} - cost of false positive

We then choose the point on the ROC curve with the gradient:

$$\beta = \frac{N}{P} \cdot \frac{V_{\text{TN}} + C_{\text{FP}}}{V_{\text{TP}} + C_{\text{FN}}}$$

3.3 Pixel Connectivity

We try to define which pixels are neighbors.



A 4 (or 8) connected path between p_1, p_n is a set of pixels such that every p_i is a 4 (or 8) neighbor of p_{i+1}. Now we can defined a region as 4 (or 8) connected if it contains a 4 (or 8) connected path between any two of its pixels.
With this we can introduce **region growing**. We start from a seed point or region and add neighboring pixels that satisfy the criteria defining a region until we can include no more pixels. There are different approaches to selecting the seed and we could also use multiple seeds. For the inclusion criteria we could choose thresholding or a distribution model.
Another criteria is a snake (active contour). While each point along the contour moves away from the seed, it always has to have some smoothness constraints (minimizing energy function).

3.4 Distance Measures

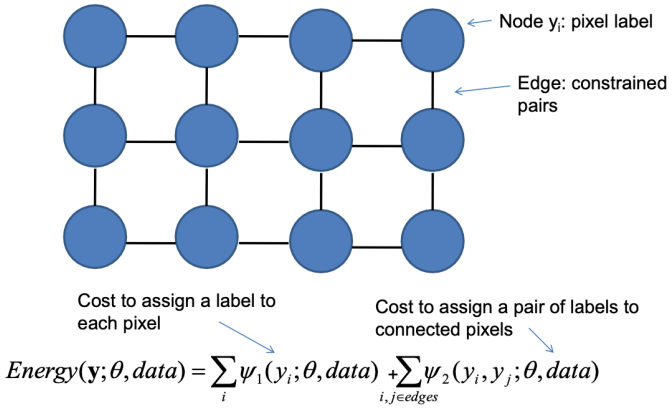
Plain background substraction $I_\alpha = |I - I_{bg}| > T$, where I_{bg} is the background image, we get this by fitting a Gaussian (Mixture) model per pixel. Even better would be:

$$I_\alpha = \sqrt{(I - I_{bg})^\top \Sigma (I - I_{bg})} > T$$

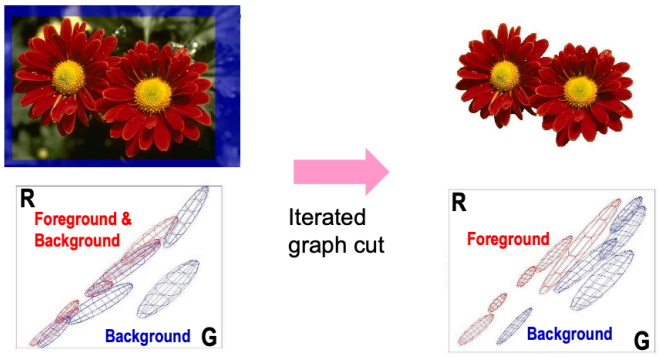
Where Σ is the background pixel appearance covariance matrix.

3.5 Markov Random Fields

We can add spatial relations with Markov Random Fields (2D Markov Chains).



Using a graph cut algorithm we can determine the optimal segementation. We can further optimize this by using iterated graph cut and k-means for learning the colour distribution (GMM) of the image.



4 Convolution and Filtering

Convolution and filtering are some of the most basic operations of image processing.

4.1 Filtering

Image filtering is the process of modifying pixels in an image base on some function of a local neighborhood of the pixel.

4.1.1 Linear Shift-Invariant Filtering

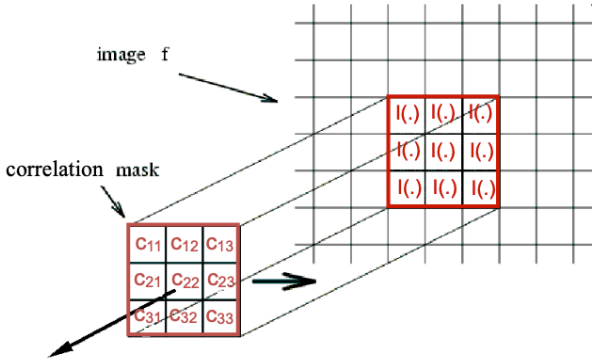
Linear lhift-invariant filtering means using linear combinations of neighbors and doing the same for each pixel (shift-invariant). These filters are often used for low-level image processing, smoothing / noise reduction, sharpening and feature detection. Linear operations can be written as:

$$I'(x,y) = \sum_{(i,j)\in N(x,y)} K(x,y;i,j) I(x,y)$$

Here I is the input image, I' the output image, K is the kernel and N is the neighborhood. Operations are shift-invariant if K does not depend on (x,y).

4.2 Correlation

Correlation, e.g. template matching:

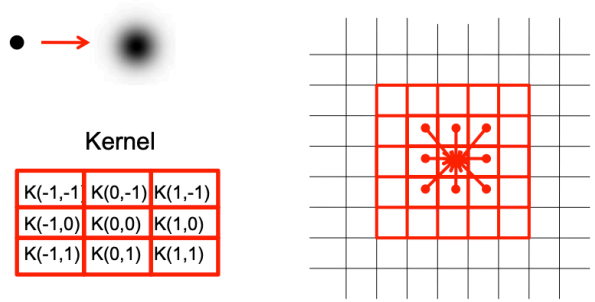


$$I' = K \circ I, \quad I'(x,y) = \sum_{(i,j)\in N(x,y)} K(i,j) I(x+i, y+j)$$

Correlation takes an input image and a weight mask, then each pixel gets "replaced" by the weighted sum of its neighborhood. This can be described as taking multiple input location and writing one output location.

4.3 Convolution

Convolution, e.g. point spread function:



$$I' = K * I, \quad I'(x,y) = \sum_{(i,j)\in N(x,y)} K(i,j)I(x-i,y-j)$$

This is similar to the correlation, but **the kernel is reversed**. This can be described as taking one input location and writing multiple output location, the opposite of the correlation. By default we use convolution for filtering. An example for a kernel would be:

$$K = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

This kernel is used for sharpening by accentuating differences with the local average. Another example would be:

$$K = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

This kernel looks for differences in the horizontal direction, this corresponds to finding vertical edges.

4.3.1 What about the Edges?

If we apply our filters to images, we need to **deal with the edges separately**. This is due to our window falling off the edge of the image. There are different techniques to deal with this problem:

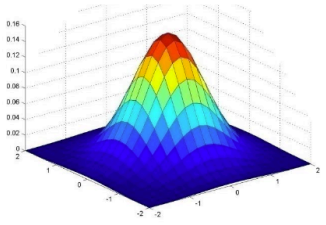
- Extend the image with black border
- Wrap the kernel around the edges
- Copy out the edge
- Mirror the image at the edge
- Vary filter near the edge

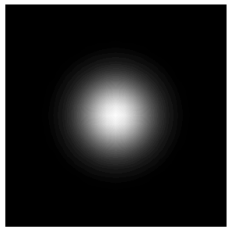
4.3.2 Separable Kernels

A kernel is separable, if it can be written as $K(m,n) = f(m)g(n)$. This means that the kernel can be separated into a function for the first coordinate and another for the second coordinate. If this is the case we can apply the separated functions individually to the image.

4.3.3 Gaussian Kernel

The idea of the **Gaussian Kernel** is to weight the contributions of neighboring pixels by nearness.





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

We can use the Gaussian Kernel for image smoothing, the best part being that the kernel is separable. The actual amount of smoothing depends on σ and the window size.

If we repeatedly apply the Gaussian filter, we produce the scale space of an image.

4.3.4 High-Pass Filters

High-pass filters are used to detect areas of the image where a lot is happening (high frequencies). Examples for these are the Laplacian operator K or the high-pass filter K':

$$K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad K' = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

High-pass filters can be used to perform image sharpening $I' = I + \alpha|K * I|$.

5 Image Features

Image features are about detecting the location of patterns in images, e.g. edge detection or facial landmarks.

5.1 Template Matching

Given an template t, e.g. template describing an eye, we want to locate an area, in an image s, that best fits this template. Alternatively we could also look for areas that match this template by a certain threshold.

To search for the best match, we try to minimize the mean squared error. This is the same as maximizing the area correlation:

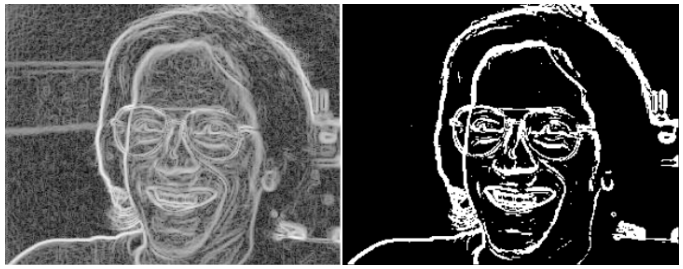
$$r(p,q) = \sum_{x=-\infty}^{\infty}\sum_{y=-\infty}^{\infty} s(x,y)\cdot t(x-p,y-q) = s(p,q) * t(-p,-q)$$

5.2 Edge Detection

We have previously seen kernels that can detect horizontal edges. To expand on this, we differentiate the following kernels:

$$\text{Prewitt}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \text{Sobel}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

We can also transpose these kernels to detect horizontal edges. From the resulting images (horizontal / vertical edges) we take the log sum squared and use different thresholds to achieve the final result.

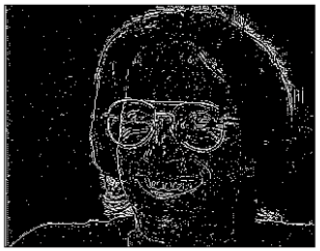


5.2.1 Laplacian Operator

The idea behind the Laplacian operator is to detect discontinuities in the second derivative. This corresponds to detecting zero-crossings. The operator is isotropic (rotationally invariant) and can be implemented with one of the following kernels:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

This operator is very sensitive to fine details and noise, therefore we might need to blur the image first. Additionally it will respond equally to weak and strong edges, so we want to suppress edges with low gradient magnitude. Blurring and applying the Laplacian operator can be combined into a convolution with Laplacian of Gaussian (LoG). Combining LoG with gradient based threshold delivers the best result.

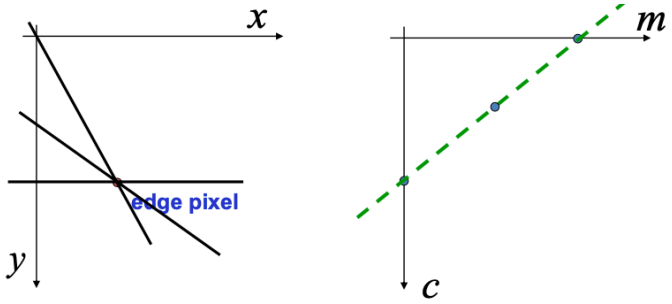



5.2.2 Canny Edge Detector

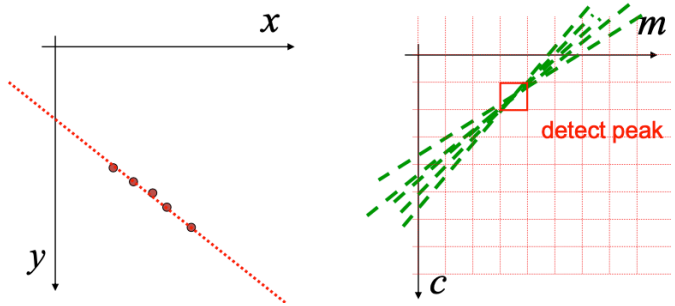
The Canny edge detector works by first smoothing the image with a Gaussian filter. Then we compute the gradient magnitude (Sobel, Prewitt, ...) and the angle of the gradient. After this we want to apply non-maxima suppression tot he gradient magnitude image. Combining this with double thresholding, to detect strong and weak edge pixels, and rejecting weak edge pixels not connected with strong edge pixels, results in the Canny edge detector.


5.3 Hough Transform

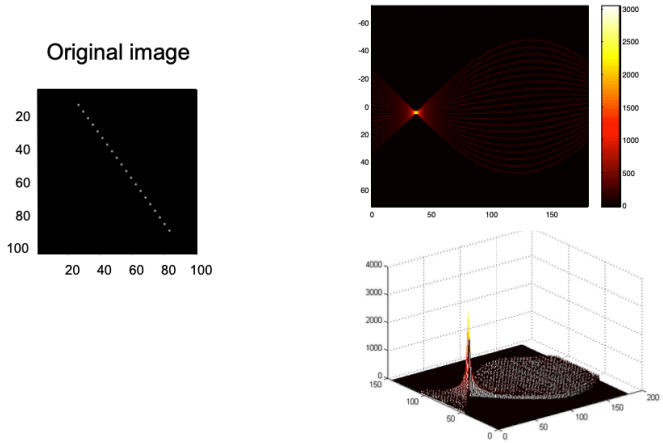
Hough transform can be used to find higher order entities in an image, e.g. lines or circles. The Hough transform is a generalized template matching technique. Considering detection of straight lines (y = mx + c), for each edge pixel there are infinitely many possible lines. We plot these possible lines in the 2D space formed by its parameters, we end up with a single line.



If we do this for multiple edge points and subdivide the parameter space into discrete bins, we can find the bin with the most possible lines. This gives us the detected line.



There is a problem with this approach, the parameter space is infinite. To avoid this problem we choose an alternative parametrization, in this case we represent a line as an angle and the distance from the origin. Now the representations in parameter space are not lines but sine waves. Again we find the maxima to find our lines.



To find multiple lines we do non-maxima suppression and keep every strong peek. To expand this concept to circle detection we simply change the parameter space.

5.4 Keypoint Detection

We might want to only find corners and not edges. We want this corner localization to be accurate, invariant and robust. We define the following:

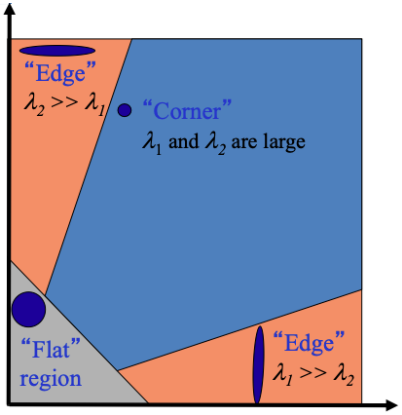
$$\mathbf{M} = \left(\sum_{(x,y)\in \text{window}} \begin{bmatrix} f_x^2(x,y) & f_x(x,y)f_y(x,y) \\ f_x(x,y)f_y(x,y) & f_y^2(x,y) \end{bmatrix} \right)$$

$$S(\Delta x, \Delta y) = (\Delta x, \Delta y)\ \mathbf{M}\ (\Delta x, \Delta y)^\top$$

Hereby f_x is the horizontal gradient and f_y the vertical gradient. $\mathbf{M}$ is called the structure matrix or nomal matrix. To detect feature points we know try to find points for which $\min \Delta^\top \mathbf{M} \Delta, ||\Delta|| = 1$ is large. This is the same as maximizing the eigenvalues of $\mathbf{M}$. The eigenvalue allow us to define a measure of "cornerness" (smaller k means more strict):

$$C(x,y) = \det \mathbf{M} - k \cdot (\text{trace } \mathbf{M})^2 = \lambda_1 \lambda_2 - k \cdot (\lambda_1 + \lambda_2)^2$$

If we plot the values for the eigenvalues, we can divide the space as follows:


"Edge"
$\lambda_2 >> \lambda_1$
"Corner"
λ_1 and λ_2 are large
"Flat"
region
"Edge"
$\lambda_1 >> \lambda_2$