

Computergrafik 2: Übung 8

Corner-Detektoren, Bildsegmentierung

Organisation

**KLAUSURANMELDUNG
(UNIWORX) NICHT
VERGESSEN!**

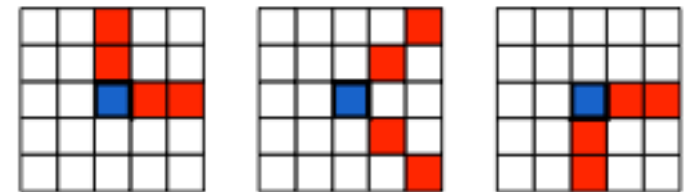
Besprechung Übung 7

- Anmerkungen?

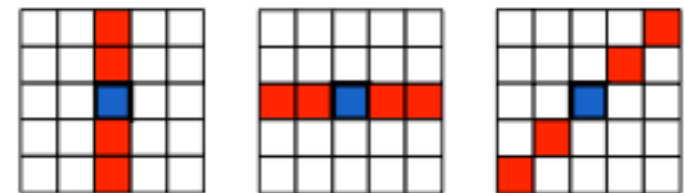
Quiz

- Was ist Non-Maximum Suppression und wofür braucht man das?
- Simple Corner Detektor: auf welchem Prinzip beruht er?
- Shading?
 - Wie kann man Shading-Probleme beheben?
- Was macht der Algorithmus von Otter? [?]
- Segmentierung?
 - Eigenschaften?
 - Arten?

Ecken:

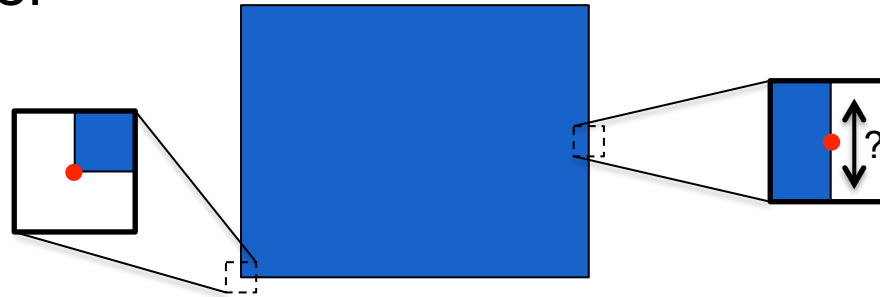


Keine Ecken:



Erkennung von Ecken

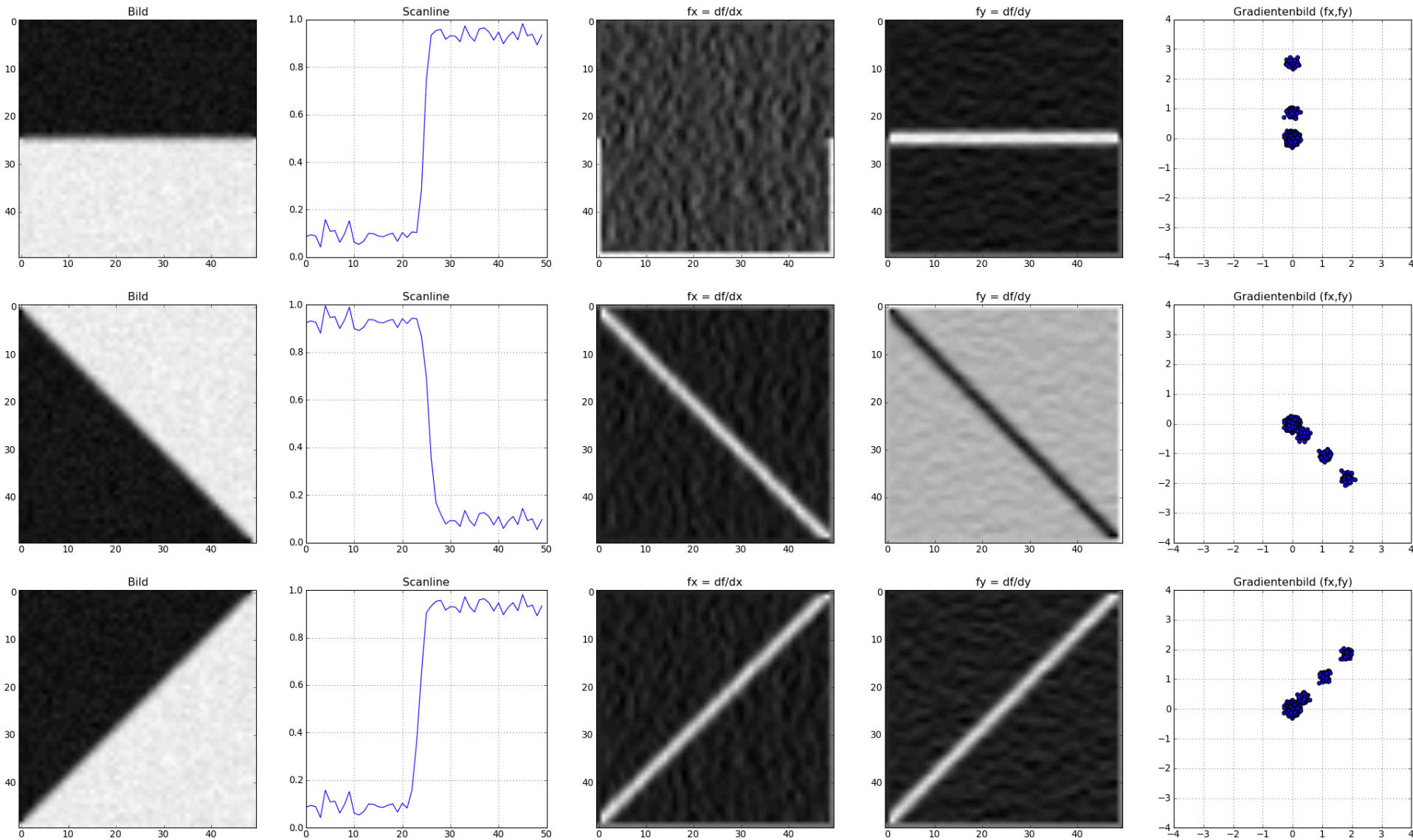
- Viele Bilderkennungsalgorithmen benötigen Merkmale, die eine stabile Position in (x,y) haben
- Kanten sind nur in einer Richtung lokalisiert
→ Ecken in zwei



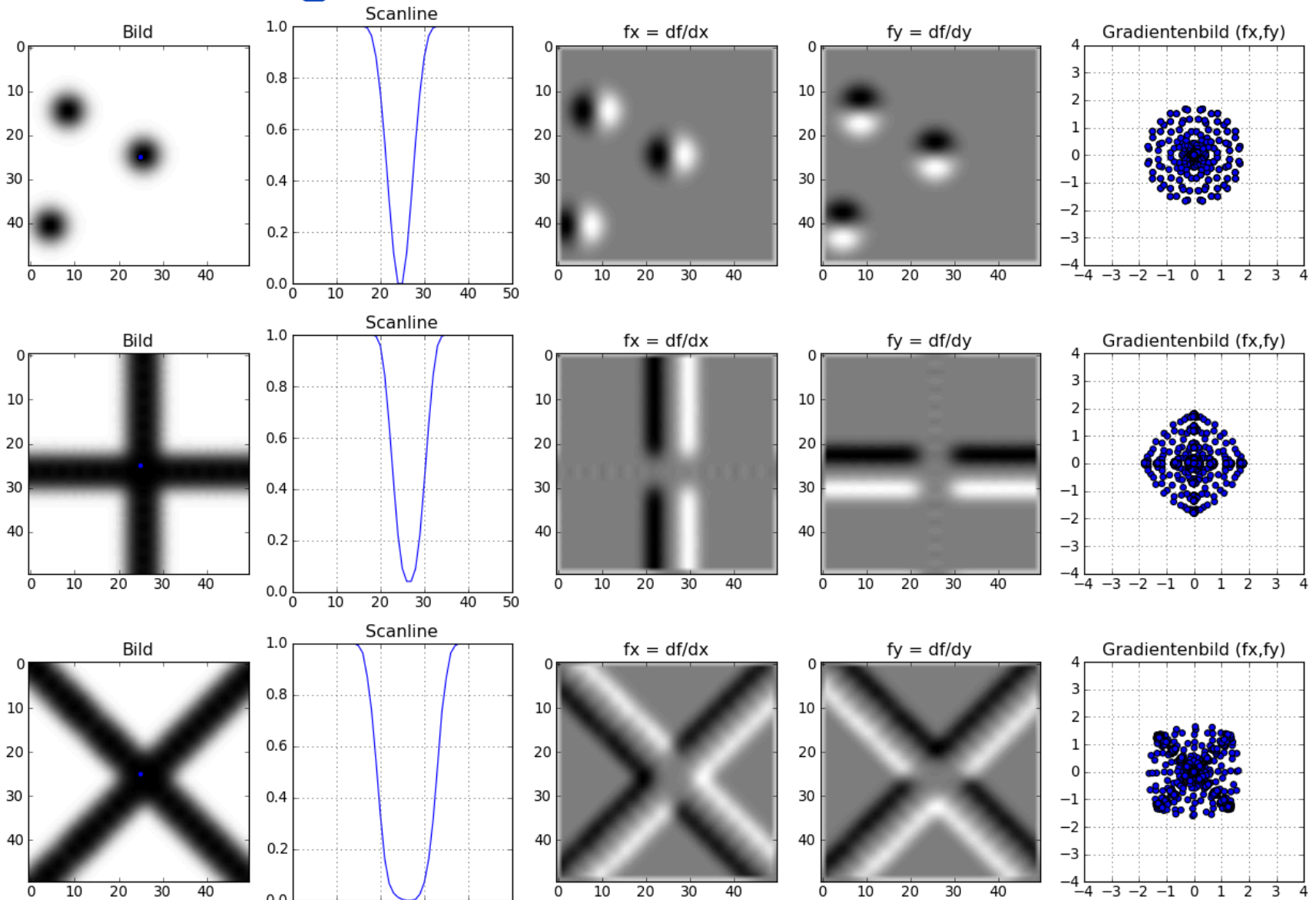
- Gewünschte Eigenschaften von Merkmalen (in verschiedenen Bildern vom gleichem Objekt / der gleichen Szene)
 - Genaue Lokalisierbarkeit
 - Invarianz gegenüber Rotation, Skalierung, Helligkeitsänderung
 - Robust gegenüber Rauschen

Slide and illustration adapted from Bernd Girod, Digital Image Processing

Verteilung der Gradienten



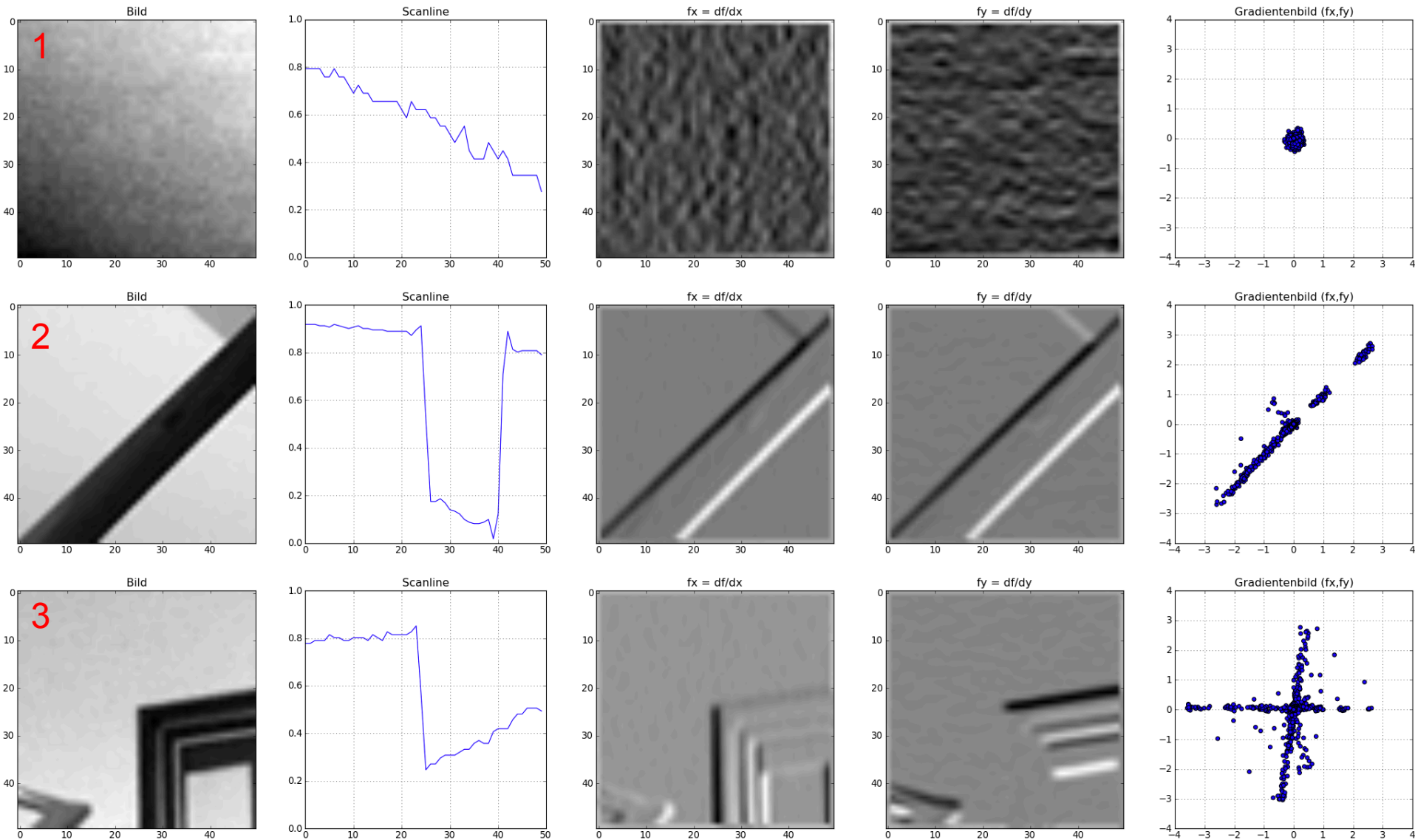
Verteilung der Gradienten



Verteilung der Gradienten in realem Bild



Verteilung der Gradienten in Bild



Gewichtete Kovarianzmatrix

$$M = \sum_{(x,y) \in \text{window}} w(x,y) * \begin{pmatrix} f_x^2(x,y) & f_x(x,y)f_y(x,y) \\ f_x(x,y)f_y(x,y) & f_y^2(x,y) \end{pmatrix}$$

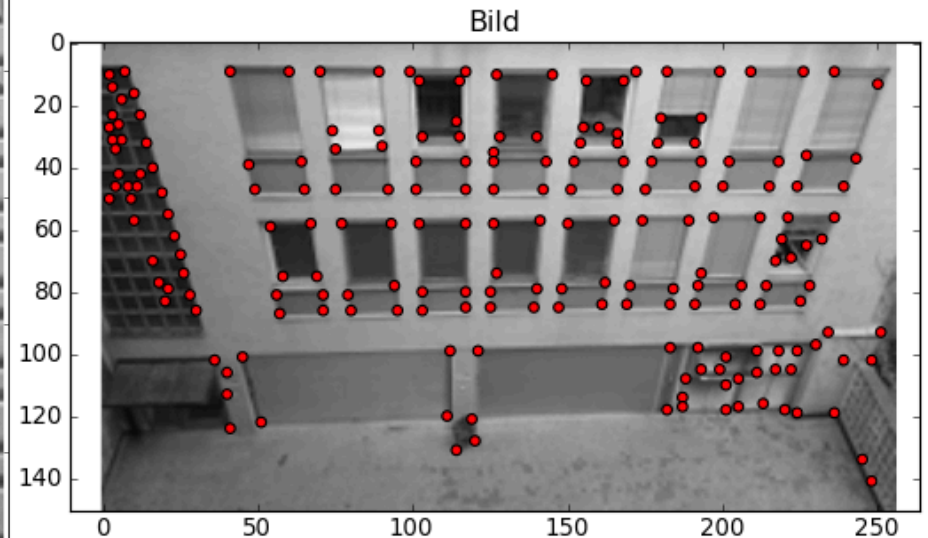
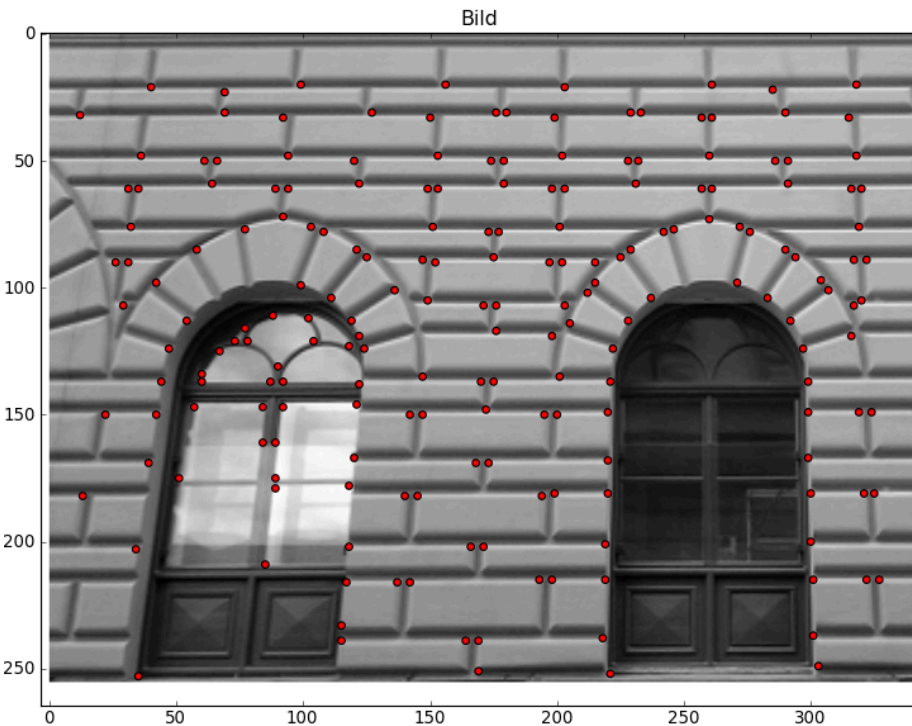
$w(x,y)$: Gewichtungsfunktion, z.B. Gauß-Funktion

- Maß für die Stärke der Ecken

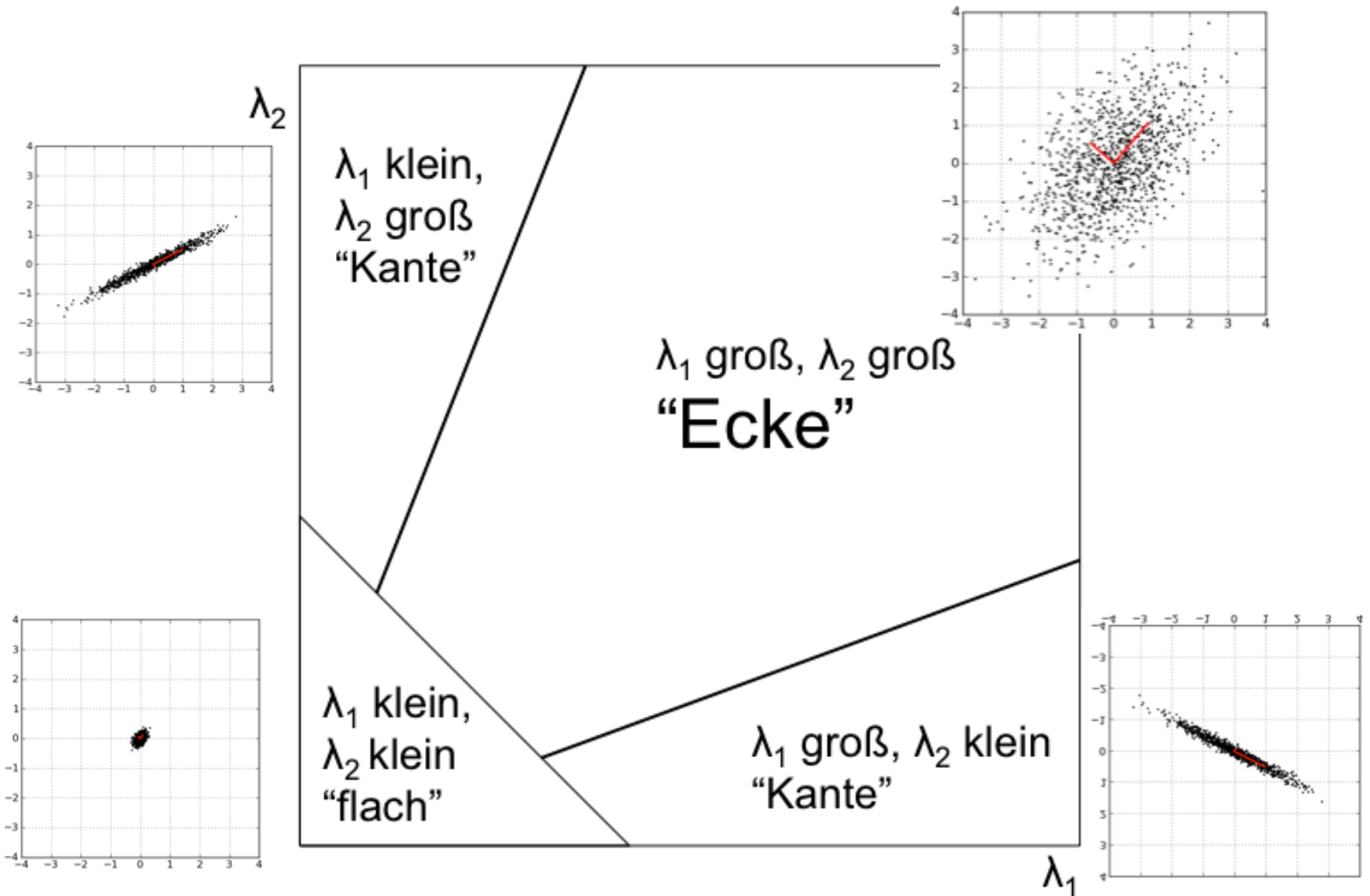
$$C(x,y) = \det(M) - k(\text{trace}(M))^2 = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

$$k = 0.04..0.06$$

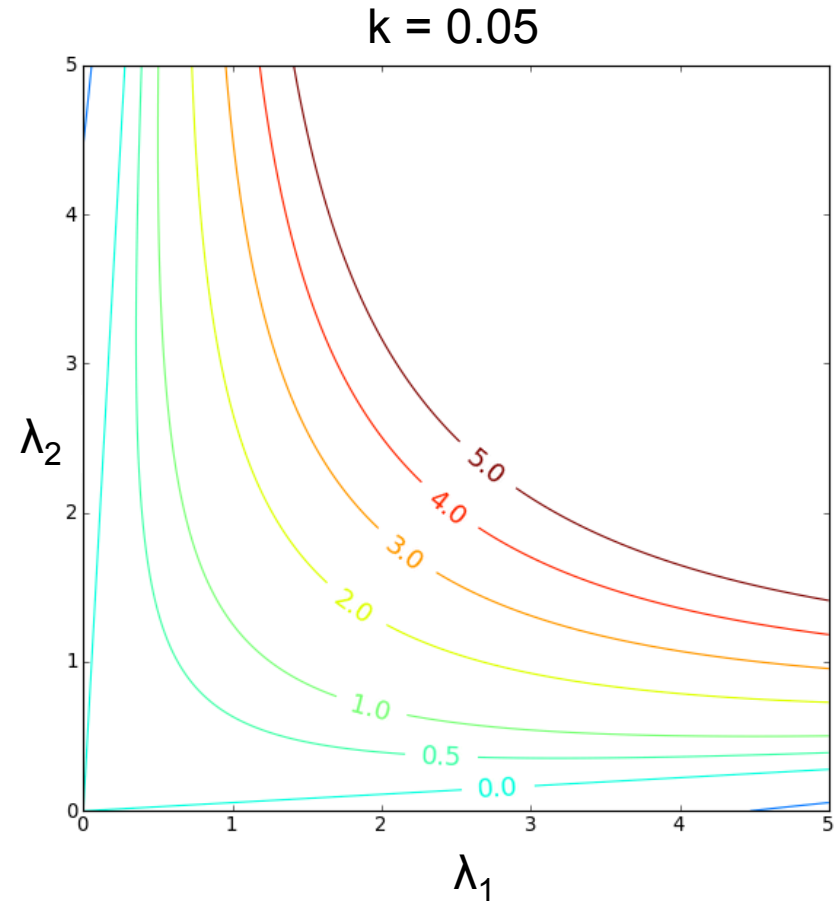
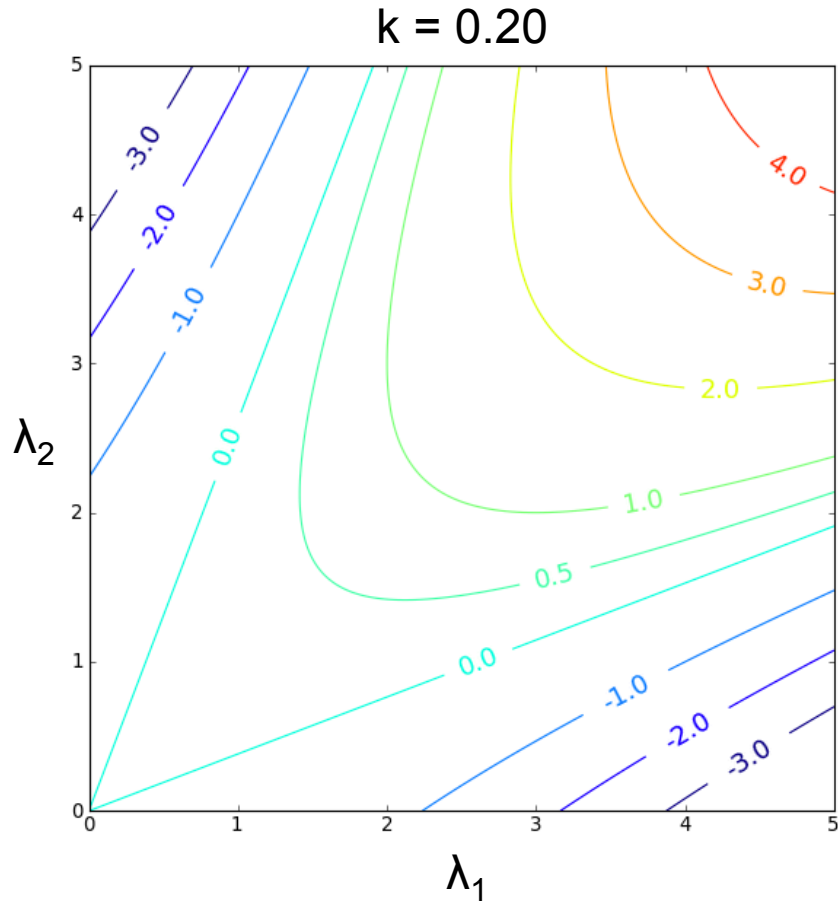
Harris Corner-Detektor: Ergebnis



Klassifikation mittels Eigenvektoren

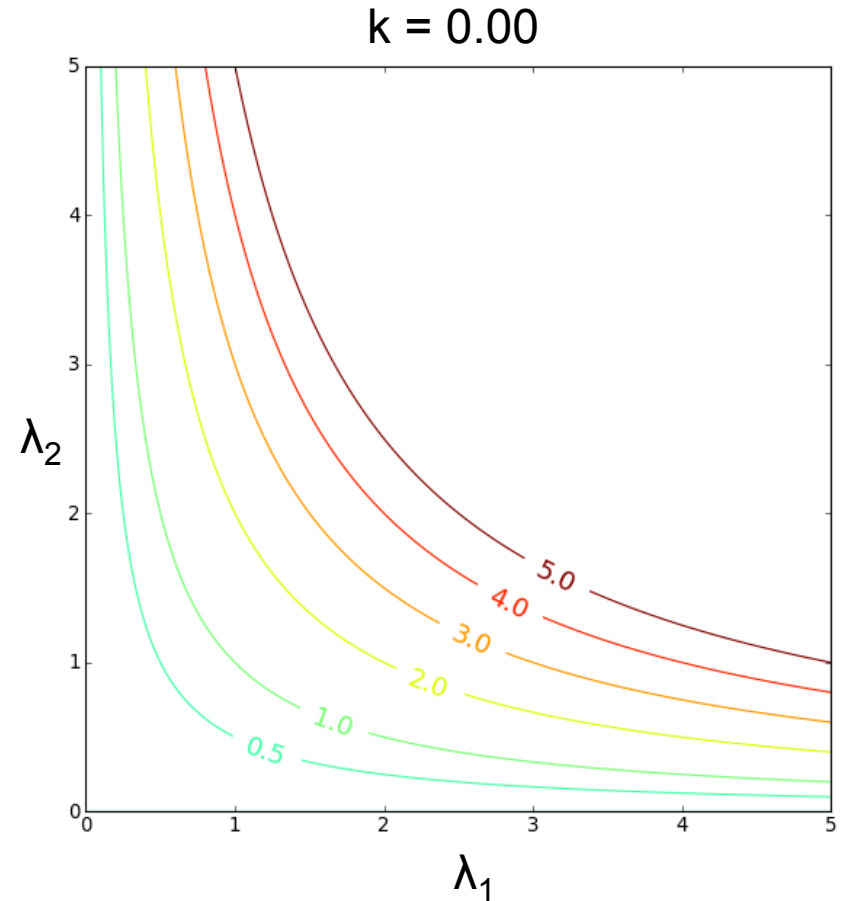
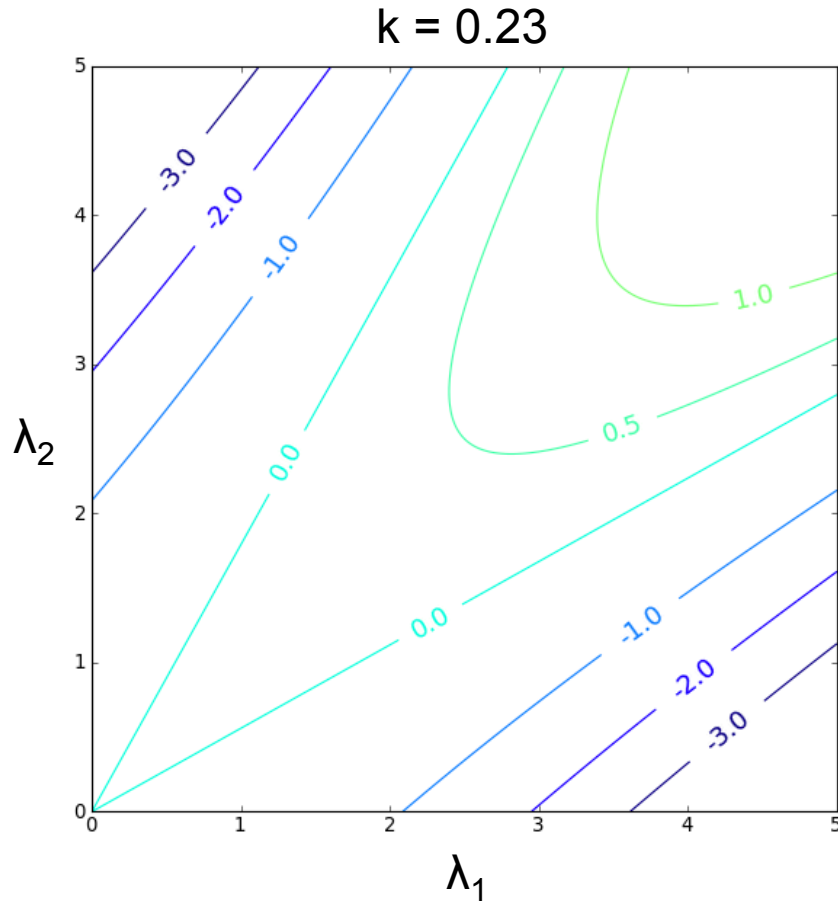


Harris Corner Detektor: Parameter “k”



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

Harris Corner Detektor: Parameter “k”



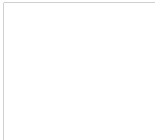
- $C = \det(M) - k \text{trace}(M)^2 = \lambda_1 \lambda_2 - k (\lambda_1 + \lambda_2)$
- “Sensitivity Parameter”

Harris Corner-Detektor: Algorithmus

1. Ableitungen f_x und f_y berechnen
2. Elementweise Produkte der Ableitungen berechnen:

$$f_x^2 = f_x f_x, \quad f_y^2 = f_y f_y, \quad f_{xy} = f_x f_y$$

3. Konvolution von f_x^2 , f_y^2 , f_{xy} mit


$$w = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

4. Für jedes Pixel (x,y) definiere:

$$M(x, y) = \begin{pmatrix} f_{xxSum}(x, y) & f_{xySum}(x, y) \\ f_{xySum}(x, y) & f_{yySum}(x, y) \end{pmatrix}$$

Harris Corner-Detektor: Algorithmus

5. Maß der Eckenstärke berechnen

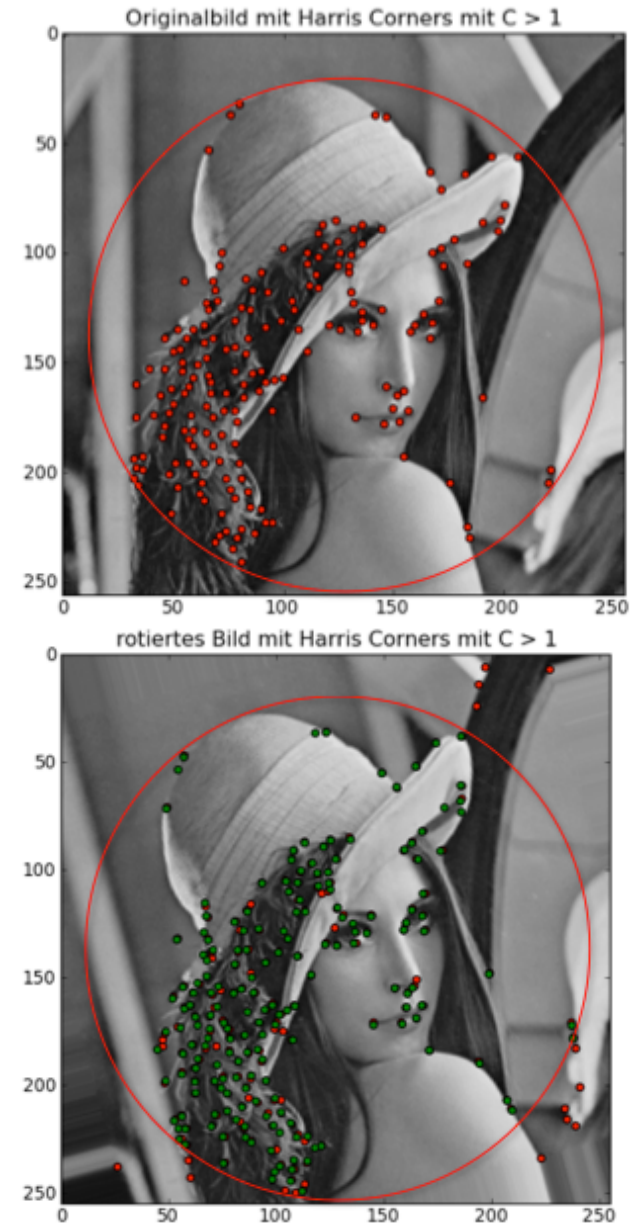
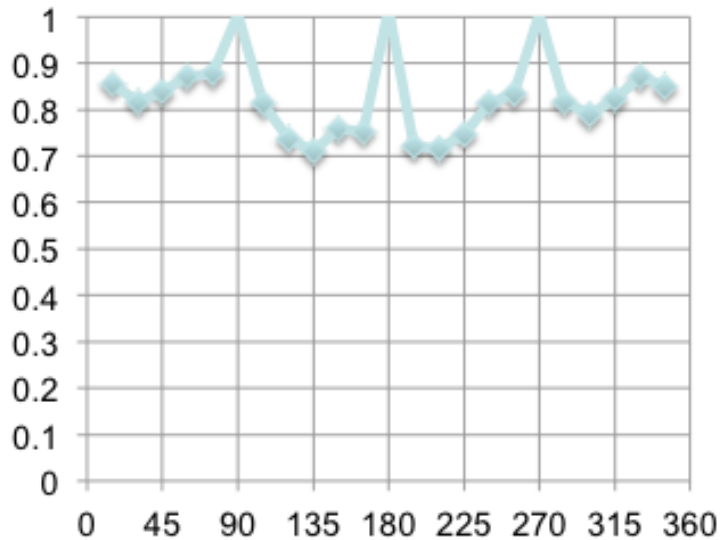
$$C(x, y) = \det(M) - k(\text{trace}(M))^2 = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

– $k = 0.04..0.06$

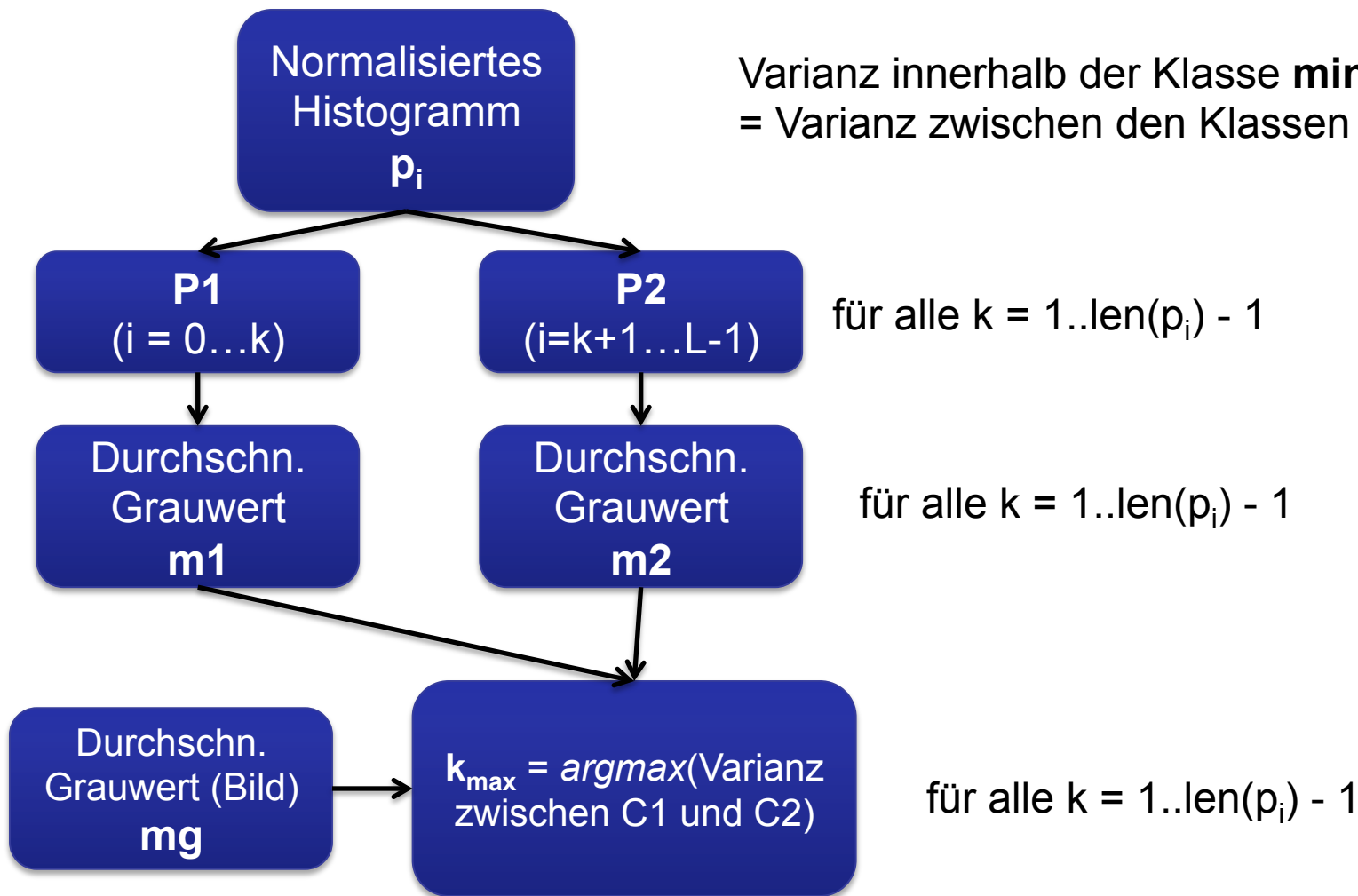
6. Non-Maximum Suppression (R=2) anwenden

Repeatability

- Invarianz gegenüber
 - Rotation
 - Translation
 - Skalierung
 - [Perspektive]
 - [Beleuchtungsverhältnisse]



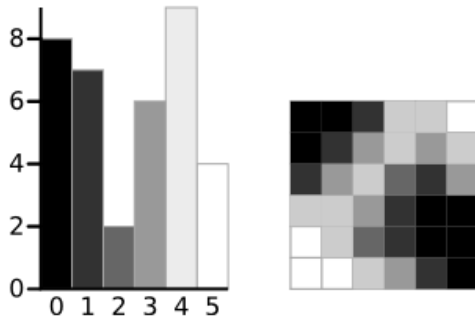
Algorithmus von Otsu



Varianz innerhalb der Klasse **minimieren**
= Varianz zwischen den Klassen **maximieren**

$$\sigma_B^2(k) = P_1(k) \cdot (m_1(k) - m_G)^2 + P_2(k) \cdot (m_2(k) - m_G)^2$$

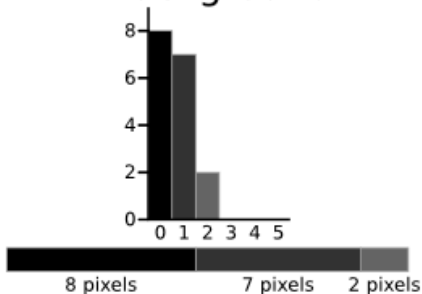
Algorithmus von Otsu



A 6-level grayscale image and its histogram

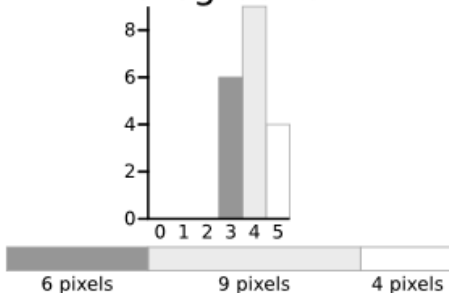
$$\begin{aligned} \text{Within Class Variance } \sigma_W^2 &= W_b \sigma_b^2 + W_f \sigma_f^2 = 0.4722 * 0.4637 + 0.5278 * 0.5152 \\ &= 0.4909 \end{aligned}$$

Background



$$\begin{aligned} \text{Weight } W_b &= \frac{8 + 7 + 2}{36} = 0.4722 \\ \text{Mean } \mu_b &= \frac{(0 \times 8) + (1 \times 7) + (2 \times 2)}{17} = 0.6471 \\ \text{Variance } \sigma_b^2 &= \frac{((0 - 0.6471)^2 \times 8) + ((1 - 0.6471)^2 \times 7) + ((2 - 0.6471)^2 \times 2)}{17} \\ &= \frac{(0.4187 \times 8) + (0.1246 \times 7) + (1.8304 \times 2)}{17} \\ &= 0.4637 \end{aligned}$$

Foreground



$$\begin{aligned} \text{Weight } W_f &= \frac{6 + 9 + 4}{36} = 0.5278 \\ \text{Mean } \mu_f &= \frac{(3 \times 6) + (4 \times 9) + (5 \times 4)}{19} = 3.8947 \\ \text{Variance } \sigma_f^2 &= \frac{((3 - 3.8947)^2 \times 6) + ((4 - 3.8947)^2 \times 9) + ((5 - 3.8947)^2 \times 4)}{19} \\ &= \frac{(4.8033 \times 6) + (0.0997 \times 9) + (4.8864 \times 4)}{19} \\ &= 0.5152 \end{aligned}$$

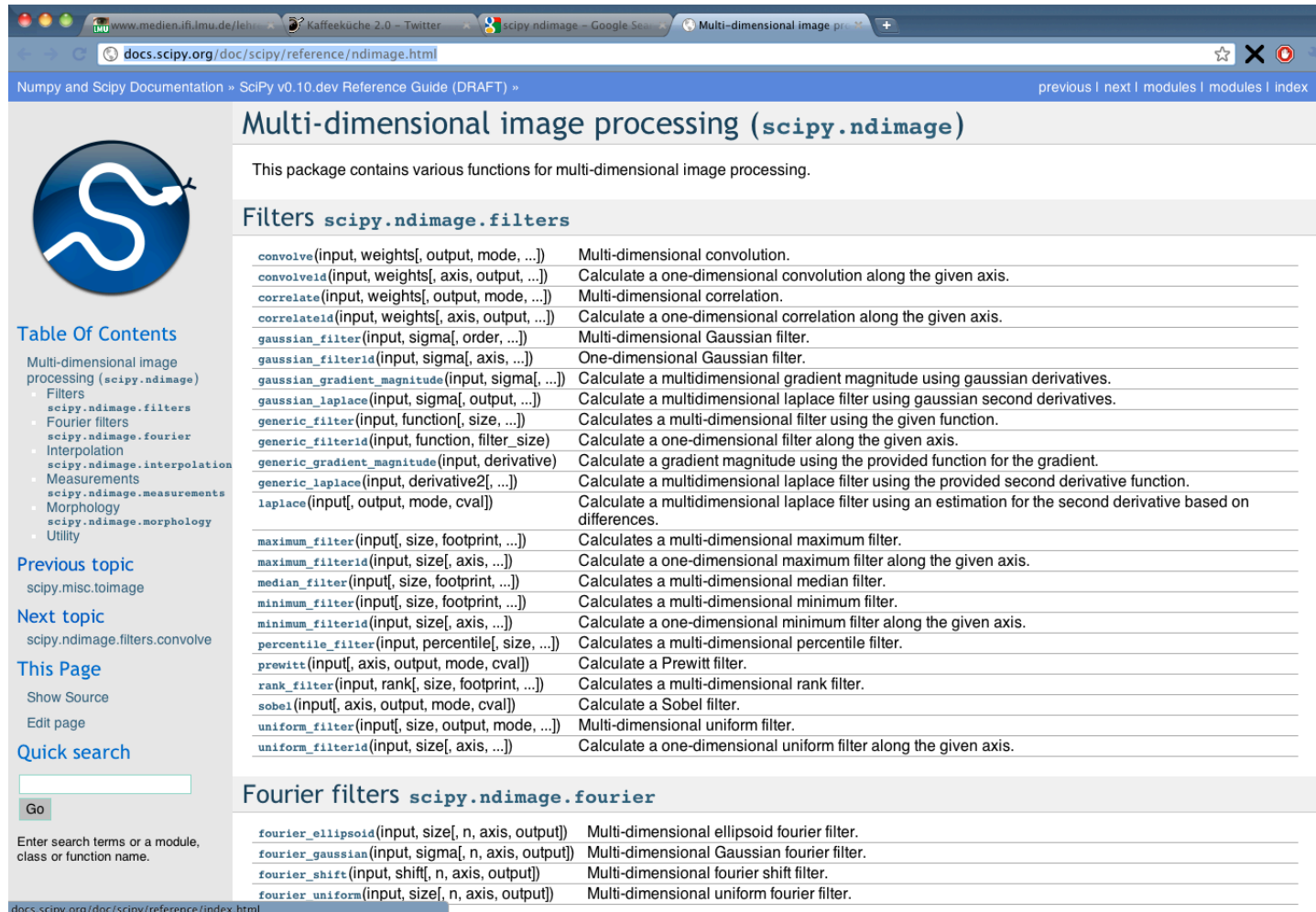
[<http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>]

Algorithmus von Otsu

Threshold	T=0	T=1	T=2	T=3	T=4	T=5
Weight, Background	$W_b =$					0.8889
Mean, Background	$M_b =$					2.0313
Variance, Background	$\sigma_b^2 = 0$	$\sigma_b^2 = 0$	$\sigma_b^2 = 0.2489$	$\sigma_b^2 = 0.4637$	$\sigma_b^2 = 1.4102$	$\sigma_b^2 = 2.5303$
Weight, Foreground	$W_f = 1$	$W_f = 0.7778$	$W_f = 0.5833$	$W_f = 0.5278$	$W_f = 0.3611$	$W_f = 0.1111$
Mean, Foreground	$M_f = 2.3611$	$M_f = 3.0357$	$M_f = 3.7143$	$M_f = 3.8947$	$M_f = 4.3077$	$M_f = 5.000$
Variance, Foreground	$\sigma_f^2 = 3.1196$	$\sigma_f^2 = 1.9639$	$\sigma_f^2 = 0.7755$	$\sigma_f^2 = 0.5152$	$\sigma_f^2 = 0.2130$	$\sigma_f^2 = 0$
Within Class Variance	$\sigma_W^2 = 3.1196$	$\sigma_W^2 = 1.5268$	$\sigma_W^2 = 0.5561$	$\sigma_W^2 = 0.4909$	$\sigma_W^2 = 0.9779$	$\sigma_W^2 = 2.2491$

Numpy: ndImage Library

- <http://docs.scipy.org/doc/scipy/reference/ndimage.html>



The screenshot shows a web browser displaying the SciPy documentation for the `ndimage` module. The page title is "Multi-dimensional image processing (`scipy.ndimage`)". The main content area lists various functions under the heading "Filters `scipy.ndimage.filters`".

Filters `scipy.ndimage.filters`

<code>convolve</code> (input, weights[, output, mode, ...])	Multi-dimensional convolution.
<code>convolve1d</code> (input, weights[, axis, output, ...])	Calculate a one-dimensional convolution along the given axis.
<code>correlate</code> (input, weights[, output, mode, ...])	Multi-dimensional correlation.
<code>correlate1d</code> (input, weights[, axis, output, ...])	Calculate a one-dimensional correlation along the given axis.
<code>gaussian_filter</code> (input, sigma[, order, ...])	Multi-dimensional Gaussian filter.
<code>gaussian_filter1d</code> (input, sigma[, axis, ...])	One-dimensional Gaussian filter.
<code>gaussian_gradient_magnitude</code> (input, sigma[, ...])	Calculate a multidimensional gradient magnitude using gaussian derivatives.
<code>gaussian_laplace</code> (input, sigma[, output, ...])	Calculate a multidimensional laplace filter using gaussian second derivatives.
<code>generic_filter</code> (input, function[, size, ...])	Calculates a multi-dimensional filter using the given function.
<code>generic_filter1d</code> (input, function, filter_size)	Calculate a one-dimensional filter along the given axis.
<code>generic_gradient_magnitude</code> (input, derivative)	Calculate a gradient magnitude using the provided function for the gradient.
<code>generic_laplace</code> (input, derivative2[, ...])	Calculate a multidimensional laplace filter using the provided second derivative function.
<code>laplace</code> (input[, output, mode, cval])	Calculate a multidimensional laplace filter using an estimation for the second derivative based on differences.
<code>maximum_filter</code> (input[, size, footprint, ...])	Calculates a multi-dimensional maximum filter.
<code>maximum_filter1d</code> (input, size[, axis, ...])	Calculate a one-dimensional maximum filter along the given axis.
<code>median_filter</code> (input[, size, footprint, ...])	Calculates a multi-dimensional median filter.
<code>minimum_filter</code> (input[, size, footprint, ...])	Calculates a multi-dimensional minimum filter.
<code>minimum_filter1d</code> (input, size[, axis, ...])	Calculate a one-dimensional minimum filter along the given axis.
<code>percentile_filter</code> (input, percentile[, size, ...])	Calculates a multi-dimensional percentile filter.
<code>prewitt</code> (input[, axis, output, mode, cval])	Calculate a Prewitt filter.
<code>rank_filter</code> (input, rank[, size, footprint, ...])	Calculates a multi-dimensional rank filter.
<code>sobel</code> (input[, axis, output, mode, cval])	Calculate a Sobel filter.
<code>uniform_filter</code> (input[, size, output, mode, ...])	Multi-dimensional uniform filter.
<code>uniform_filter1d</code> (input, size[, axis, ...])	Calculate a one-dimensional uniform filter along the given axis.

Fourier filters `scipy.ndimage.fourier`

<code>fourier_ellipsoid</code> (input, size[, n, axis, output])	Multi-dimensional ellipsoid fourier filter.
<code>fourier_gaussian</code> (input, sigma[, n, axis, output])	Multi-dimensional Gaussian fourier filter.
<code>fourier_shift</code> (input, shift[, n, axis, output])	Multi-dimensional fourier shift filter.
<code>fourier_uniform</code> (input, size[, n, axis, output])	Multi-dimensional uniform fourier filter.

The left sidebar contains a "Table Of Contents" with links to various sections like "Filters", "Fourier filters", "Interpolation", etc. There is also a search bar at the bottom left.

Numpy: Array-Sortierung

- Mit index-Arrays lässt sich der Sortierschlüssel bestimmen

```
A.shape = (<n>,3)
```

```
A[A[:,2].argsort(),:]
```

→ Sortiert alle Einträge aus a nach dem 3. Eintrag

- Alternativ: Python `itemgetter` und `sorted()` verwenden

```
from operator import itemgetter
```

```
sorted_items = sorted(A, key=itemgetter(2), reverse =  
True)
```

Numpy: Nützliche Funktionen

- Rotation

```
f = scipy.ndimage.rotate(img, alpha, reshape=False,  
mode='nearest')
```

mode : Beschreibt, wie Randpunkte "aufgefüllt" werden

- Skalierung

```
f = scipy.ndimage.zoom(img, scale, mode='nearest')
```

- Text

```
plt.text(x,y, str(label), color='red')
```