

# Computergrafik 2: Übung 7

Hough Transformation

**Organisation**

**KLAUSURANMELDUNG  
(UNIWORX) NICHT  
VERGESSEN!**

# Quiz

- Berechnung der „ersten Ableitung“ eines Bildes?
- Berechnung der „zweiten Ableitung“ eines Bildes?
- Was ist ein Gradient?
- Wozu dient die Laplace-Funktion?
- Was ist ein LoG-Filter? Was ist ein DoG-Filter?
- Canny Edge Detection? Algorithmus?
- Hough-Transform: Prinzip?

# Besprechung Übung 6

- Anmerkungen?

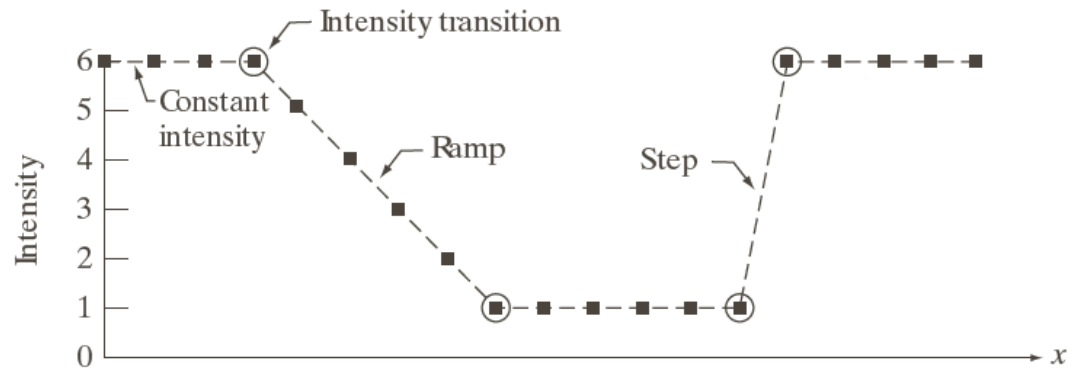
# Erste und zweite Ableitung von Bildern

- Erste Ableitung:

$$\frac{\partial f}{\partial x}(x) = f(x+1) - f(x)$$

- Zweite Ableitung:

$$\begin{aligned} \frac{\partial^2 f}{\partial^2 x}(x) &= \frac{\partial f}{\partial x}(x+1) - \frac{\partial f}{\partial x}(x) \\ &= (f(x+1) - f(x)) \\ &\quad - (f(x) - f(x-1)) \\ &= f(x+1) + f(x-1) - 2f(x) \end{aligned}$$



Scan line	6	6	6	6	5	4	3	2	1	1	1	1	1	1	6	6	6	6	6
1st derivative	0	0	-1	-1	-1	-1	-1	0	0	0	0	0	0	5	0	0	0	0	0
2nd derivative	0	0	-1	0	0	0	0	1	0	0	0	0	0	5	-5	0	0	0	0

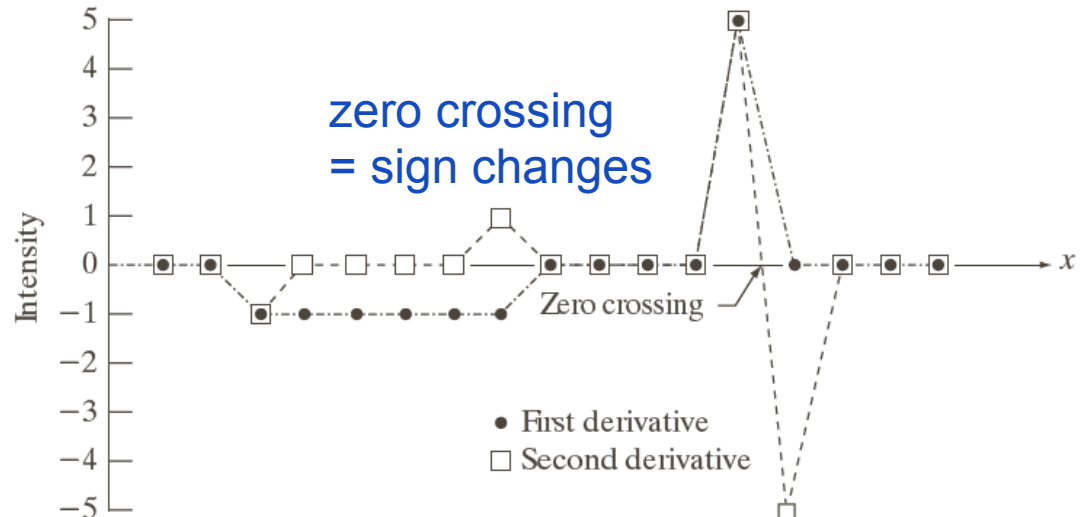


Abbildung: © R. C. Gonzalez & R. E. Woods, Digital Image Processing

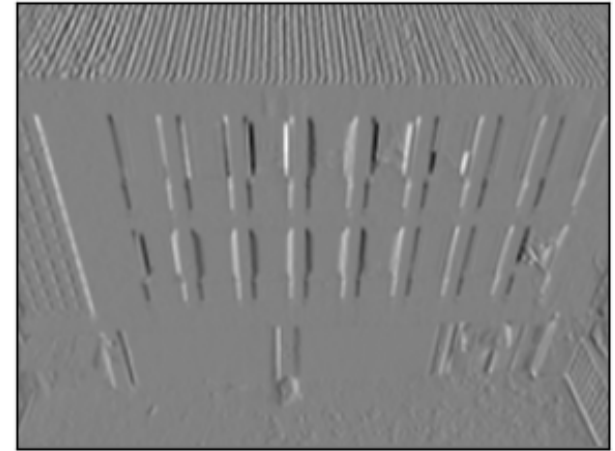
# Gradienten finden: Konvolution mit dem Sobel-Operator



\*

-1	0	1
-2	0	2
-1	0	1

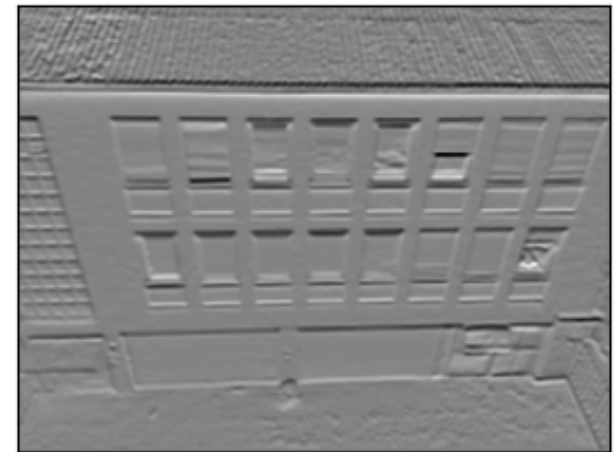
=



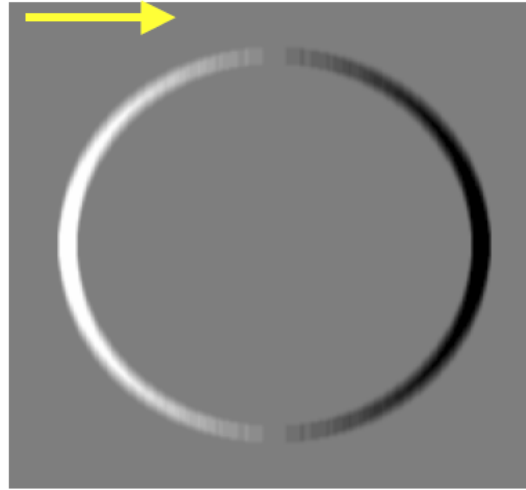
\*

-1	-2	-1
0	0	0
1	2	1

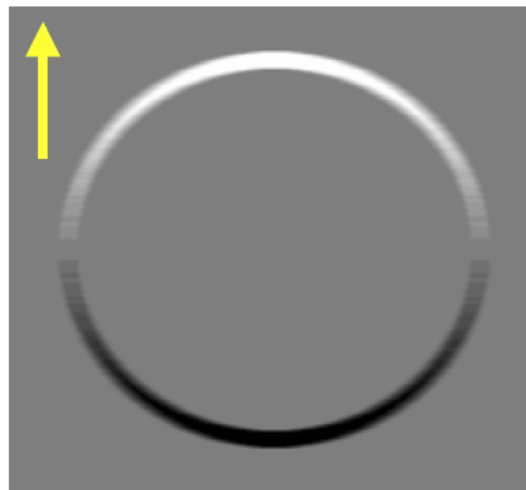
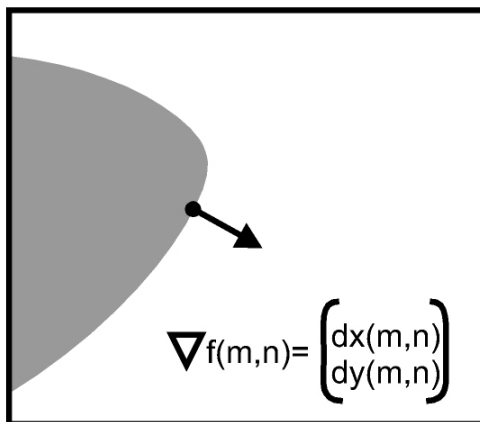
=



# Elemente des Gradienten



Betrag:  $\sqrt{G_x^2 + G_y^2}$   
Richtung:  $\tan^{-1}(G_y / G_x)$



# Laplace-Funktion

- zweite Ableitung in x-Richtung

$$\begin{aligned}\frac{\partial^2 f}{\partial^2 x}(x) &= \frac{\partial f}{\partial x}(x+1) - \frac{\partial f}{\partial x}(x) \\ &= (f(x+1) - f(x)) - (f(x) - f(x-1)) \\ &= f(x-1) - 2f(x) + f(x+1) \quad \Longrightarrow \quad \begin{pmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{pmatrix}\end{aligned}$$

- zweite Ableitung in y-Richtung

$$\begin{aligned}\frac{\partial^2 f}{\partial^2 y}(y) &= \frac{\partial f}{\partial y}(y+1) - \frac{\partial f}{\partial y}(y) \\ &= (f(y+1) - f(y)) - (f(y) - f(y-1)) \\ &= f(y-1) - 2f(y) + f(y+1) \quad \Longrightarrow \quad \begin{pmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{pmatrix}\end{aligned}$$



# Laplace-Funktion

- Summe der partiellen zweiten Ableitungen

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

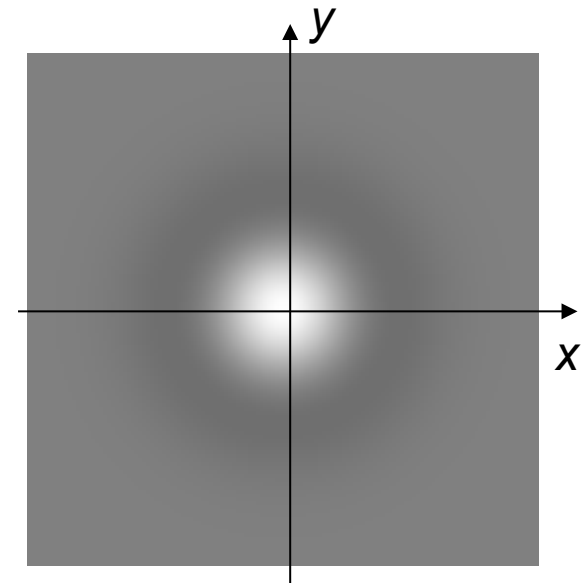
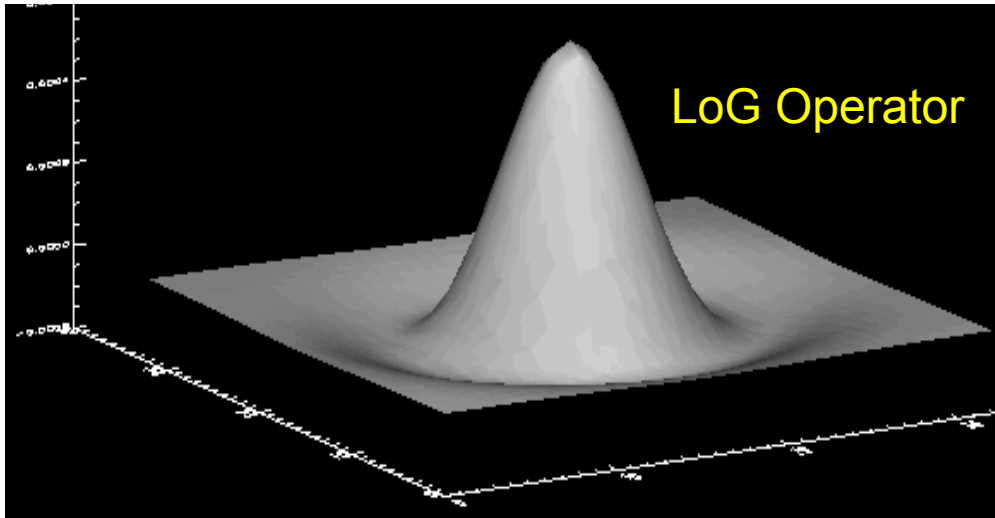
- Summe aller partiellen Ableitungen, um rotationsinvarianten Operator zu erhalten

$$\nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2} + \frac{\partial^2 f(x, y)}{\partial x \partial y} + \frac{\partial^2 f(x, y)}{\partial y \partial x} : \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

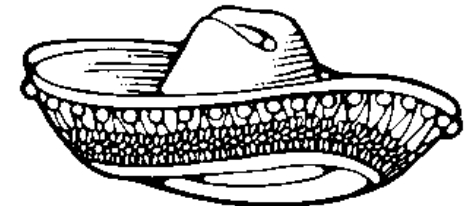
$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 1 \\ 0 & -2 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

# Marr-Hildreth-Filter = LoG-Filter

LoG-Filter: Laplace-Operator auf Gaußfunktion angewandt  
d.h. der **Faltung mit dem Laplacefilter** geht eine  
**Glättung** mit einer **Gaußfunktion** voraus



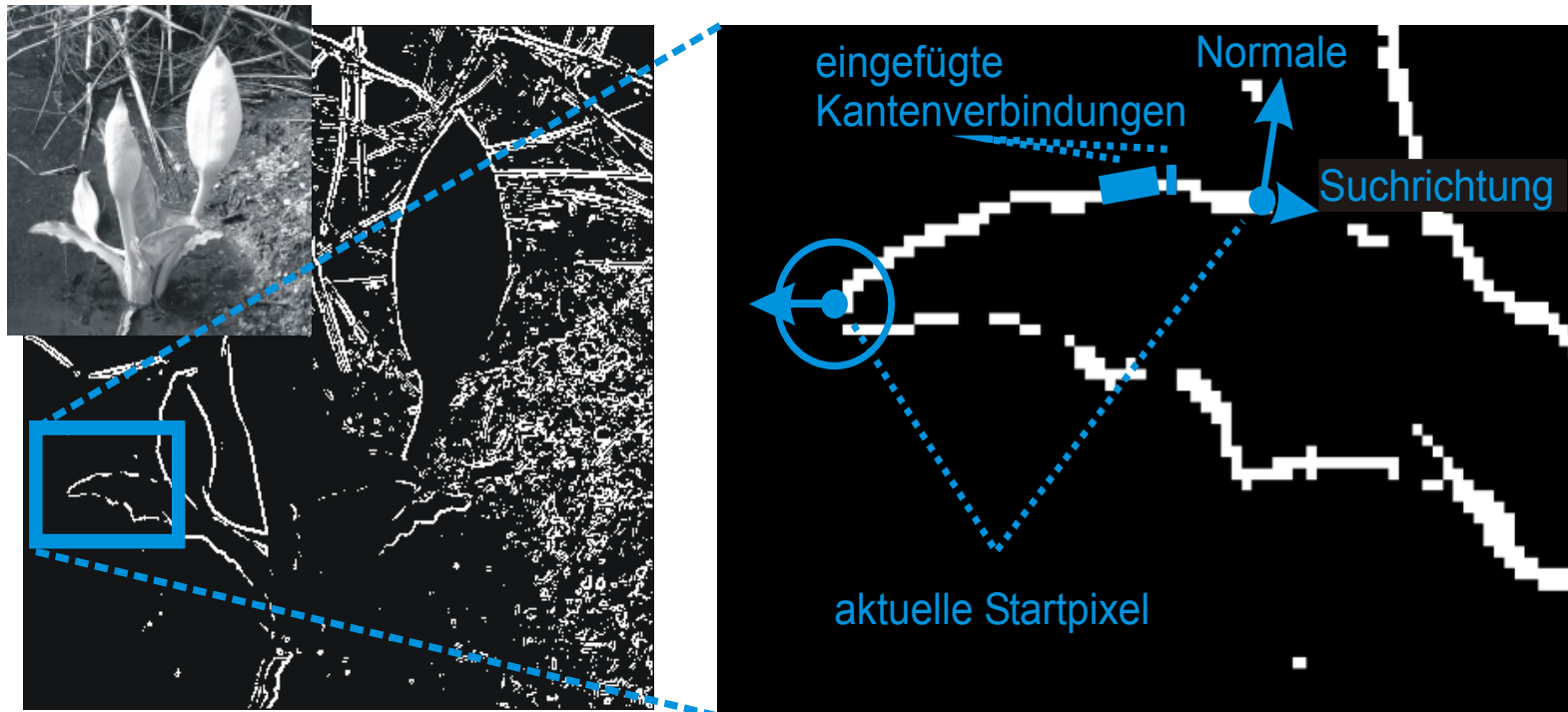
$$LoG_{\sigma}(x, y) = -\frac{1}{\pi\sigma^4} \left( 1 - \frac{x^2 + y^2}{2\sigma^2} \right) \exp\left( -\frac{x^2 + y^2}{2\sigma^2} \right)$$



Auch genannt: „Mexican hat“ filter

# Edge Linking verbindet Kantenpixel zu Kantenzügen

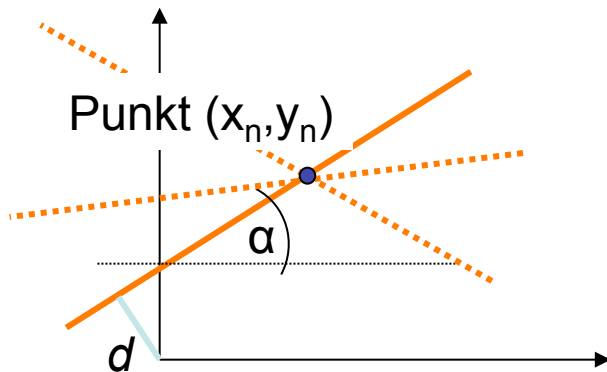
- Starte Kantenzug mit starker Kante
- Betrachte Nachbarpixel orthogonal zur Gradientenrichtung
- Setze Kantenzug fort auch bei schwacher Kante fort



# Hough Transformation für Geraden

Suche von Geraden in einem Binärbild

Geradenrepräsentation:  $x \cos(\alpha) + y \sin(\alpha) - d(\alpha) = 0$

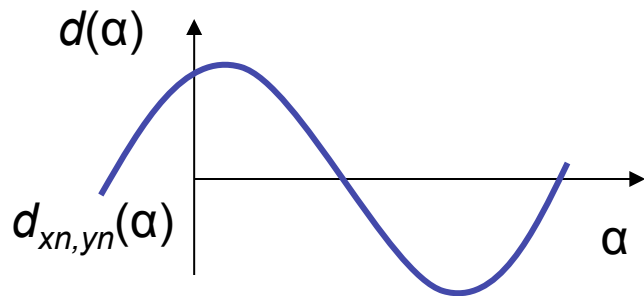


Hough Transformation:

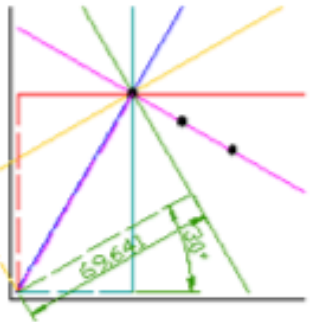
Suche alle Parameter  $(\alpha, d)$  für Geraden, die durch einen Punkt  $(x_n, y_n)$  gehen

$$d(\alpha) = x_n \cos(\alpha) + y_n \sin(\alpha)$$

Der Raum, der durch  $(\alpha, d)$  aufgespannt wird, heißt **Hough-Raum**



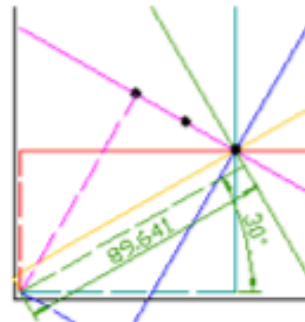
# Hough-Transformation



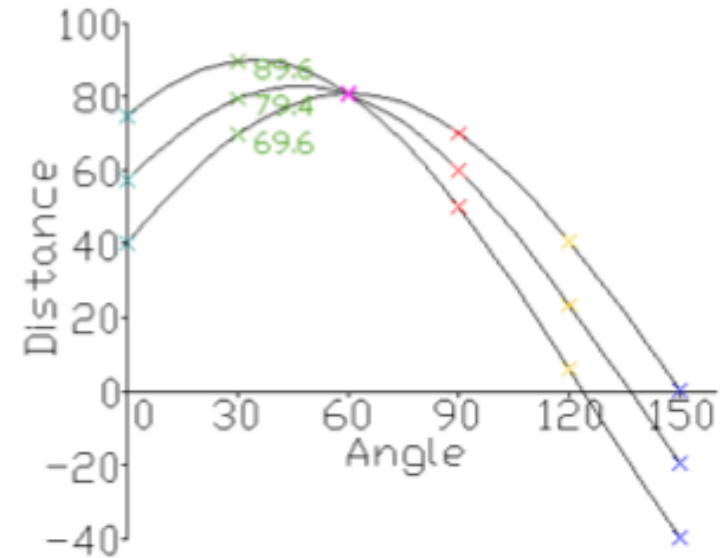
Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4



Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	-19.5



Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	-39.6

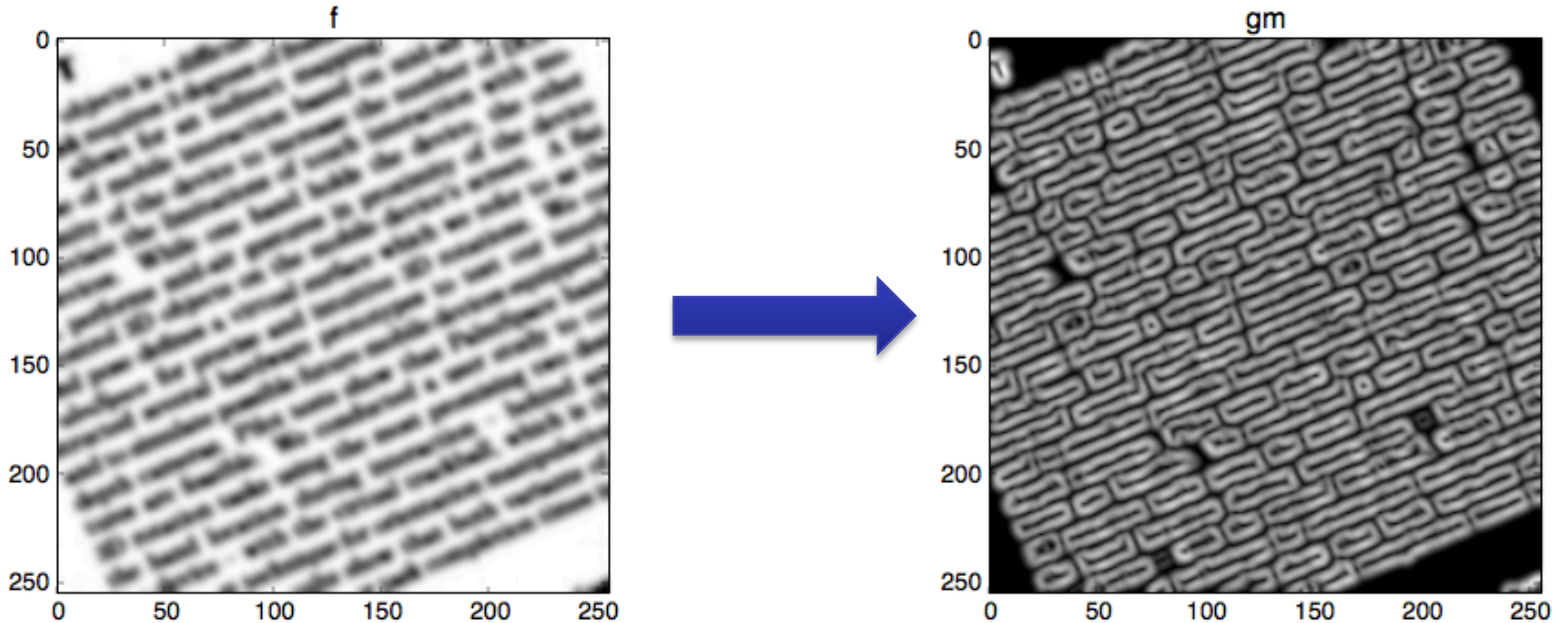


Geradendarstellung:

$$d(\alpha) = x_n \cos(\alpha) + y_m \sin(\alpha)$$

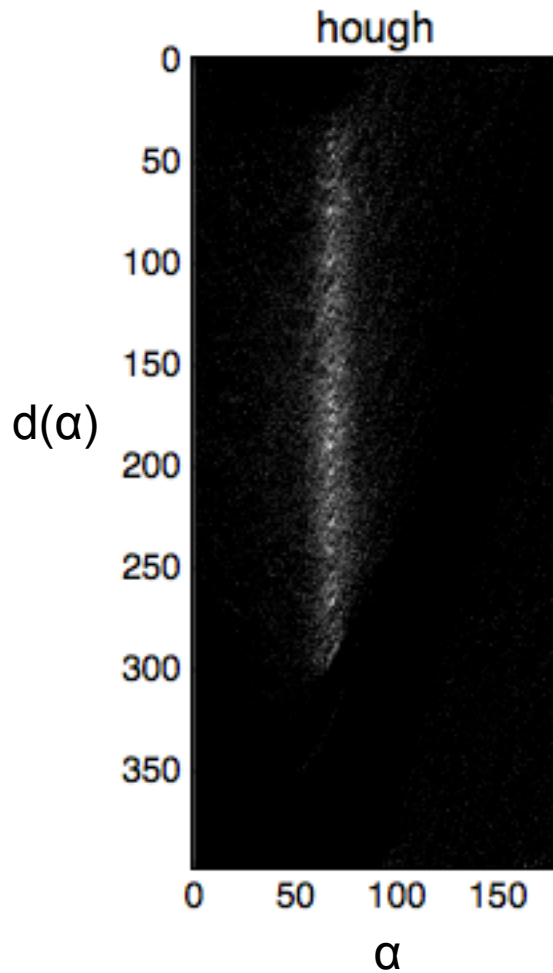
# Hough-Transformation: Tipps

- Starkes Gaußfilter anwenden:



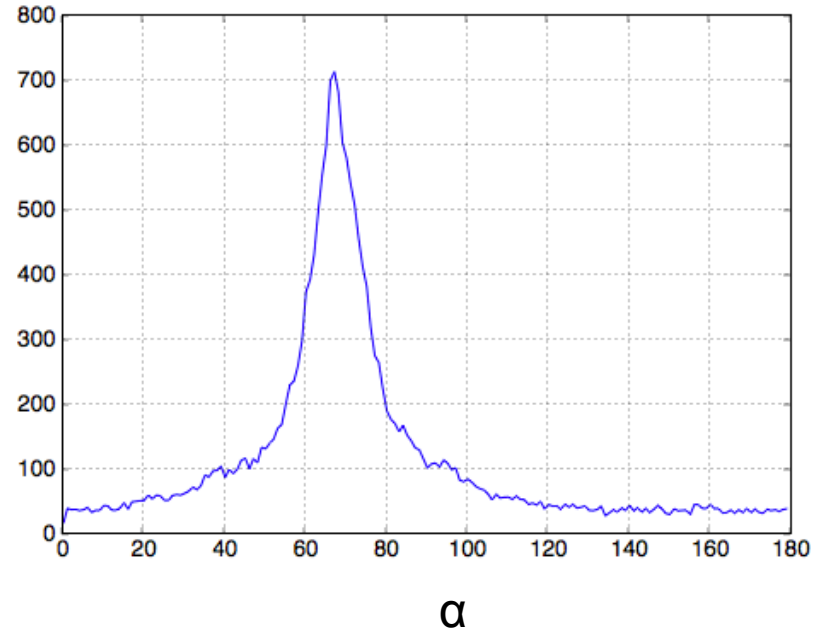
- Statt Inkrementierung Gradientenlänge verwenden
- Gradientenorientierung berücksichtigen

# Hough-Transformation: Tipps

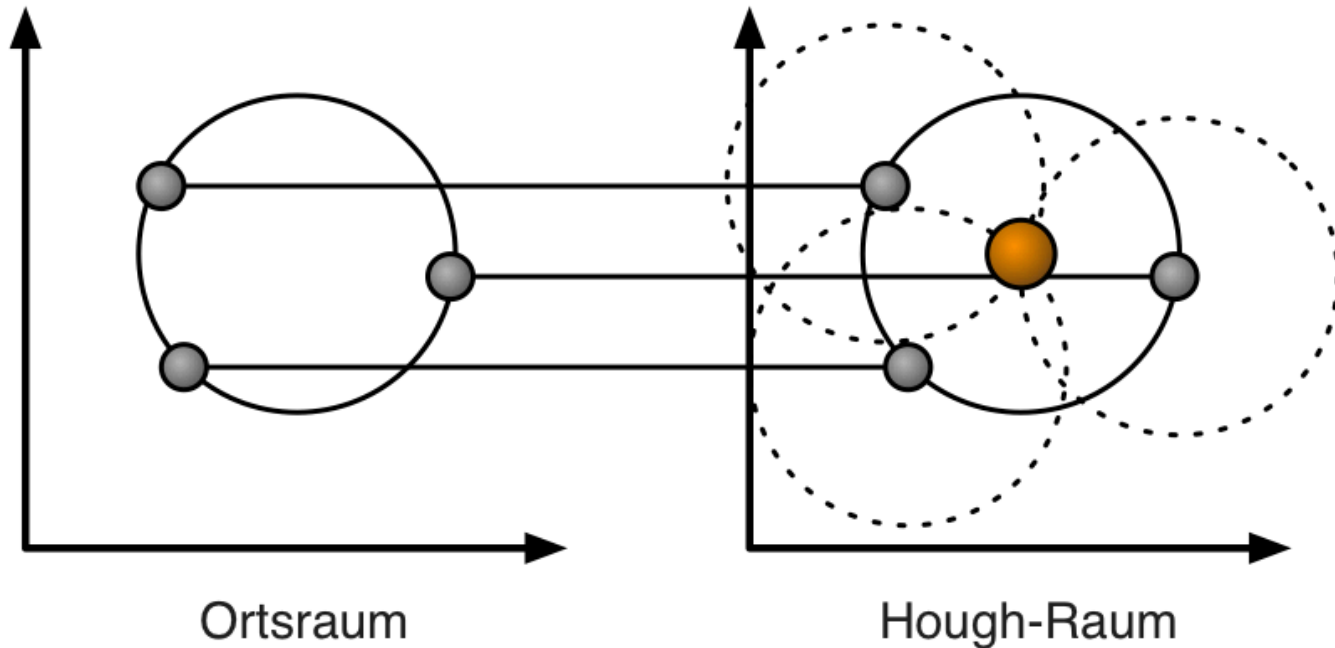


Spaltensumme  
für  $d(\alpha)$

Spaltensummen  
addieren



# Hough-Transformation für Kreise



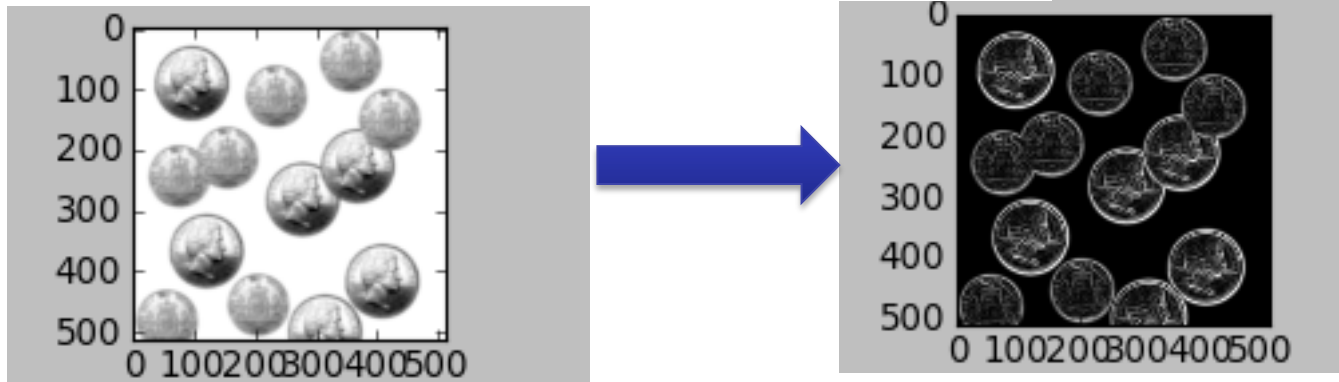
$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_c + r \cos(\alpha) \\ y_c + r \sin(\alpha) \end{pmatrix}$$



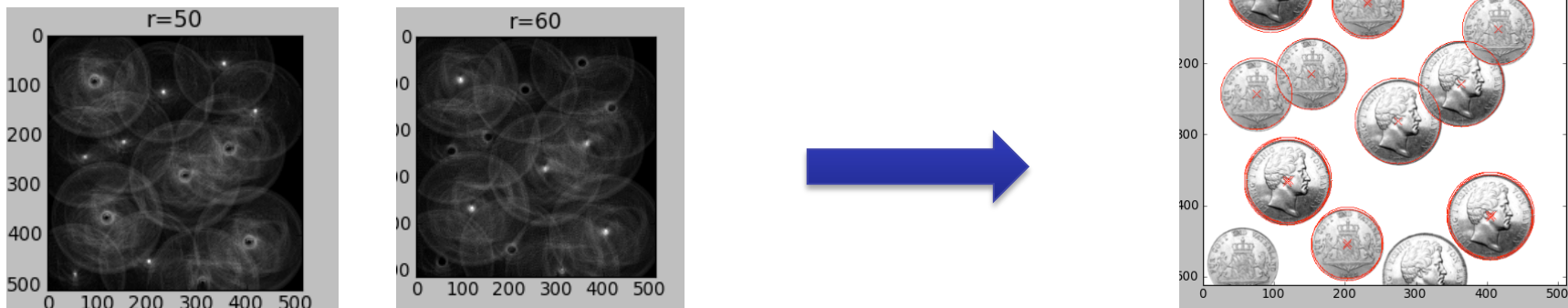
# Hough Transform für Kreise: Tipps

- Gradientenlänge verwenden, um Kreise zu finden

$$M(x_i, y_j) = \sqrt{G(x_i)^2 + G(y_j)^2}$$

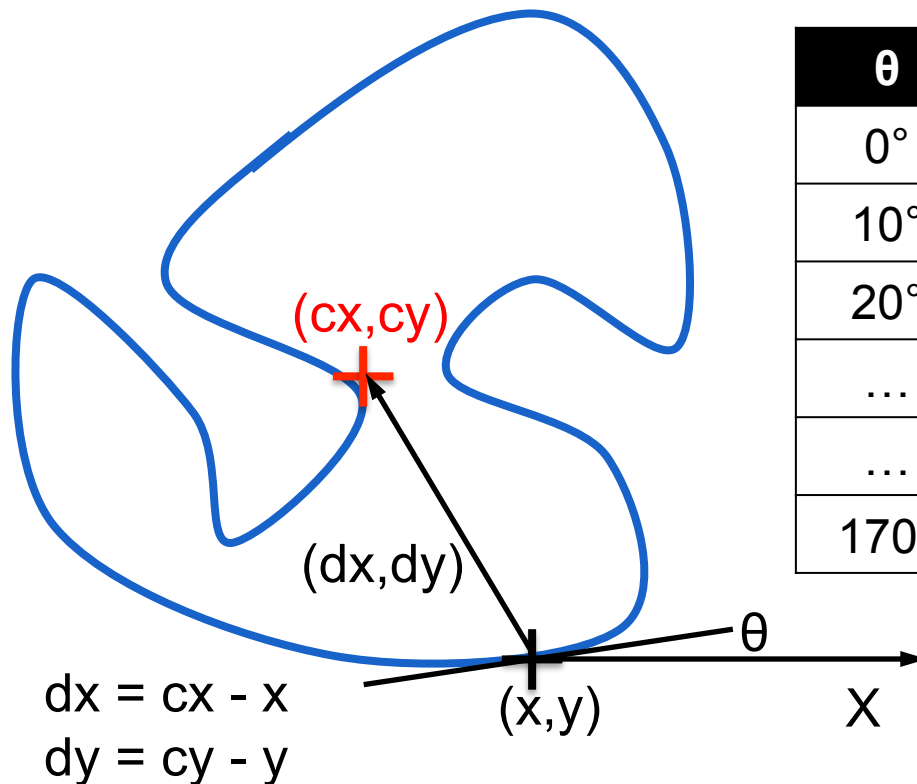


- Zwei Durchläufe für die verschiedenen Radii:



# Generalisierte Hough Transformation

- Hough Transformation für beliebige Formen

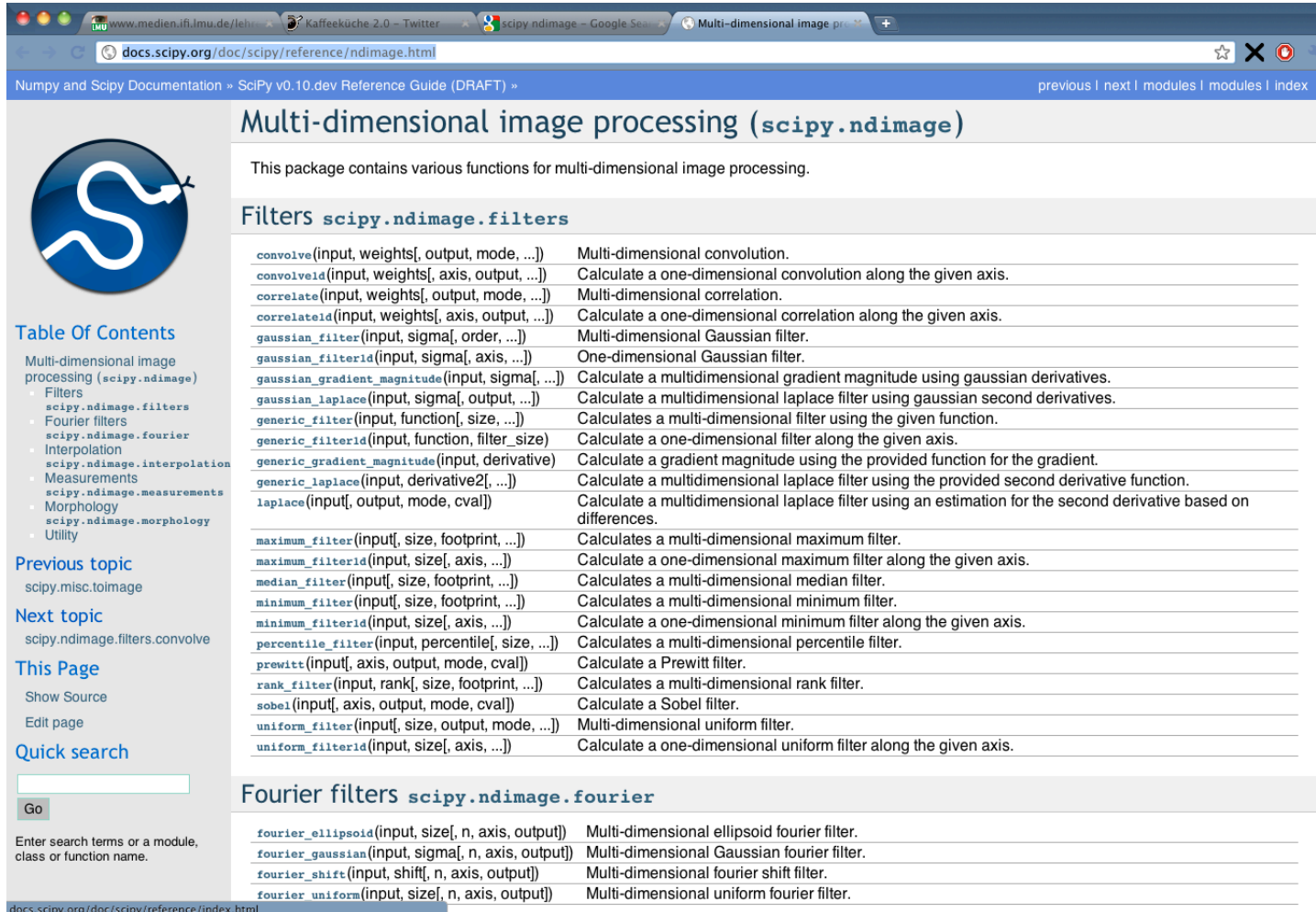


$\theta$	R
$0^\circ$	$(dx_1, dy_1)$
$10^\circ$	$(dx_2, dy_2), (dx_3, dy_3), (dx_5, dy_6)$
$20^\circ$	
...	...
...	...
$170^\circ$	$(dx_7, dy_7), (dx_8, dy_8)$

D.H. Ballard: Generalizing the Hough Transform to Detect Arbitrary Shapes.  
Pattern Recognition, Vol. 13, No. 2, pp. 111-122, 1981

# 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" for the `scipy.ndimage` module, listing sub-modules like `filters`, `fourier`, `interpolation`, `measurements`, `morphology`, and `utility`. It also includes "Previous topic" (`scipy.misc.toimage`), "Next topic" (`scipy.ndimage.filters.convolve`), and "This Page" (with links for "Show Source" and "Edit page"). A "Quick search" box is located at the bottom of the sidebar.

# Numpy: Array-Sortierung

- Mit index-Arrays lässt sich der Sortierschlüssel bestimmen

`A.shape` sei `(<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)`