



# Image Analysis and Object Recognition

## Exercise 4

Image filtering in frequency domain  
&  
Shape recognition using Fourier descriptors

SS 2014

(Course notes for internal use only!)

# Comments Ex 3: Algorithm Outline

- **Input:** binary edge image (from GoG-filtering)
- **Initialize** voting array  $H$

- $\rho_{ind} = [-\rho_{max}/2, \dots, \rho_{max}/2]$ ,  $\rho_{max} = 2 \cdot \sqrt{n_{rows}^2 + n_{columns}^2}$
- $\theta_{ind} = [-90, \dots, 89]$

- **for** each edge point  $(x, y)$  in the image

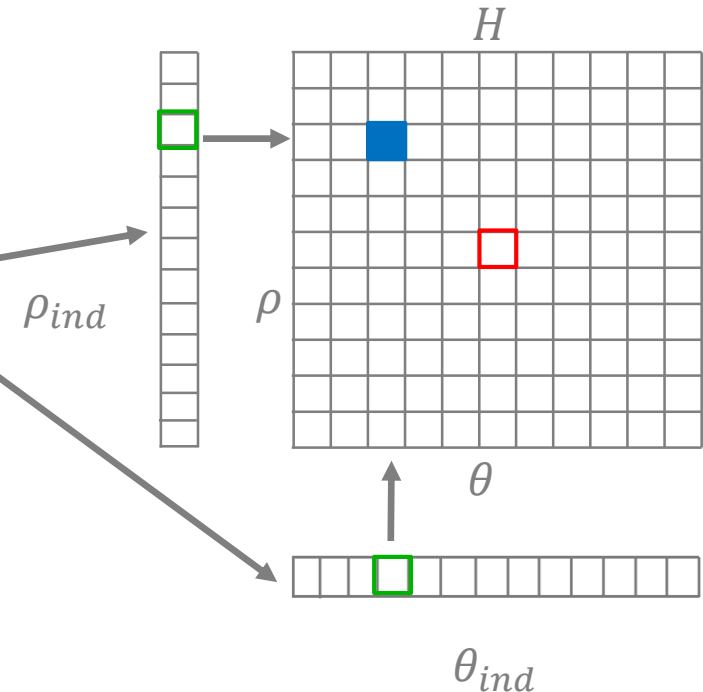
$\theta$  = gradient orientation at  $(x, y)$

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

$H(\theta, \rho) = H(\theta, \rho) + 1$

**end**

- Find the local maxima of  $H$



# Comments Ex 3: Algorithm extension

- Use the **gradient direction** of detected edges
  - GoG-filtering  $\rightarrow$  first derivatives in x- and y-direction:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}$

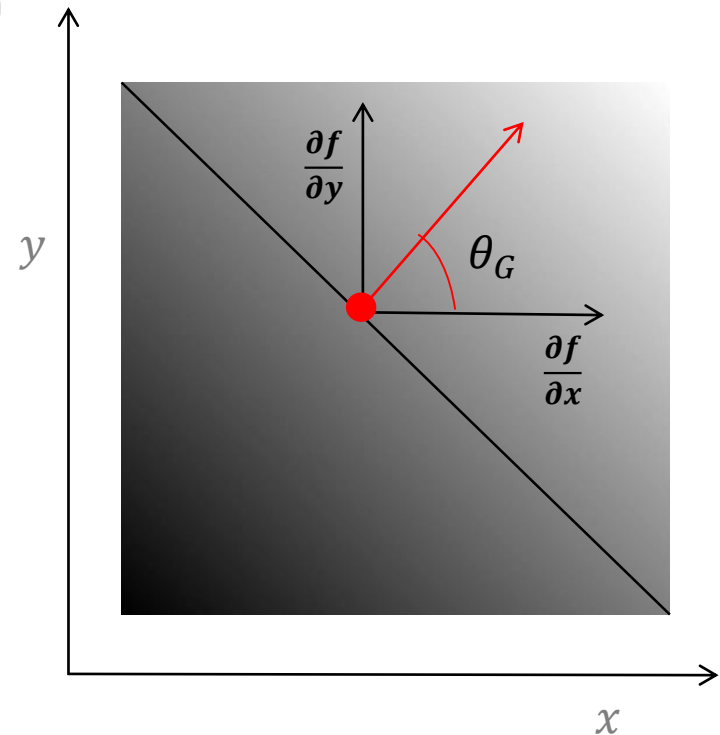
- Gradient direction:  $\theta_G = \tan^{-1} \left( \frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}} \right)$

- Attention using coordinates

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

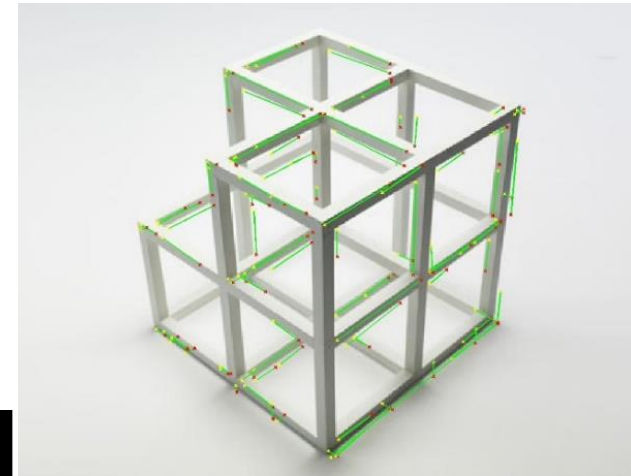
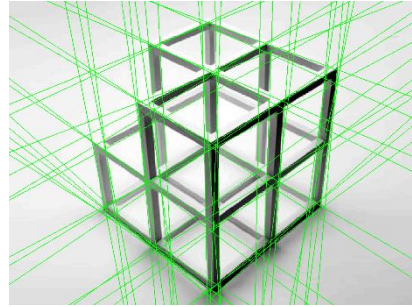
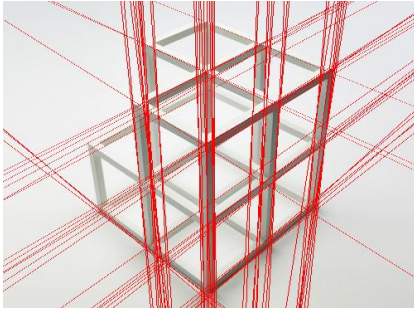
$\rightarrow x$  related to columns in image

$\rightarrow y$  related to rows in image

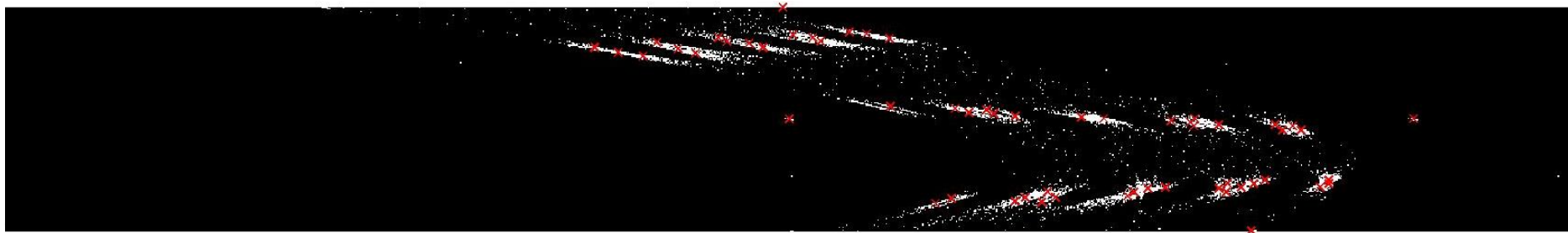
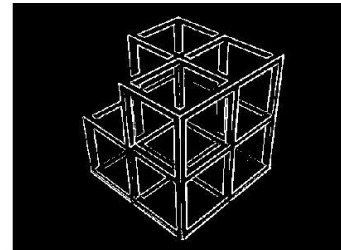
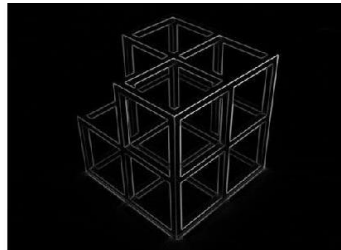
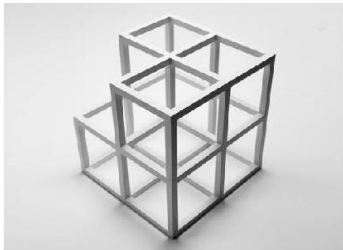


# Exercise 3: Results

- Octave: function *houghlines* not available



- Example results



# Submission Status

- Ex 1: Accomplished
- Ex 2: Submission of 2 groups missing
- Ex 3: Submission of 1 group missing



## Exercise 4

**A:** Image filtering in frequency domain  
&

**B:** Shape recognition using Fourier descriptors

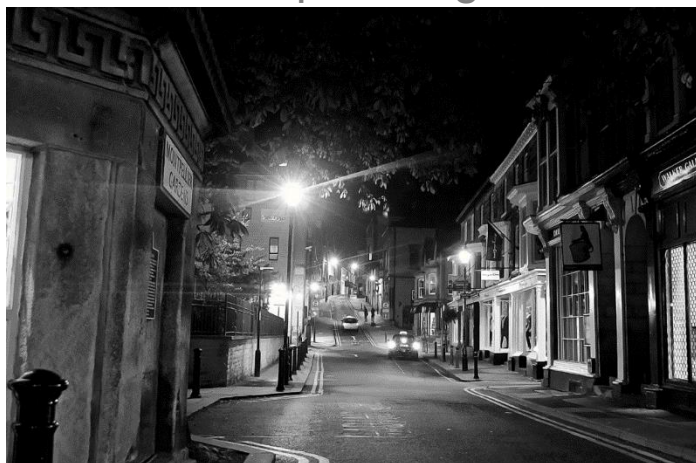


# Exercise 4

**A:** Image filtering in frequency domain

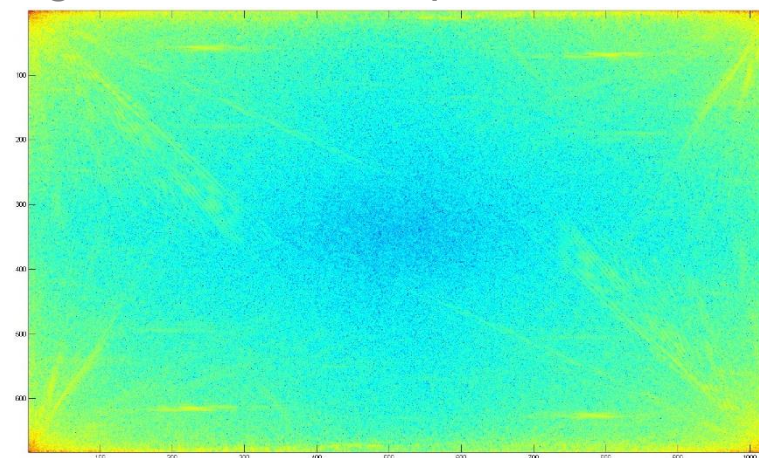
# Fast Fourier Transform

Input image



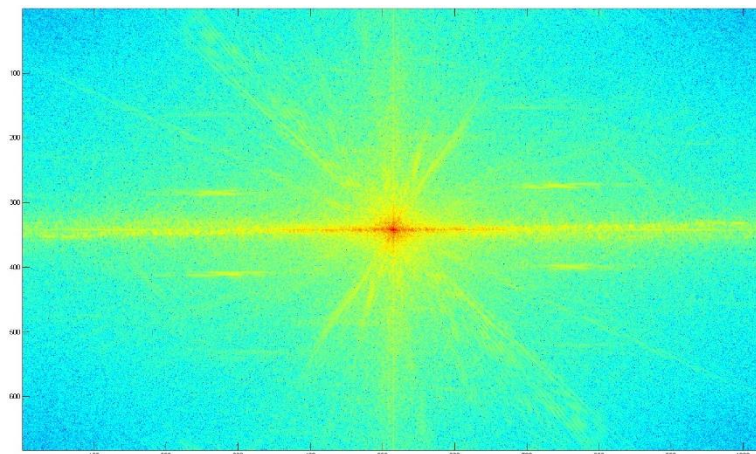
→  
*fft2*

Logarithmic scaled spectrum  $|F(u, v)|$



*imagesc(log(abs(fft\_image)))*

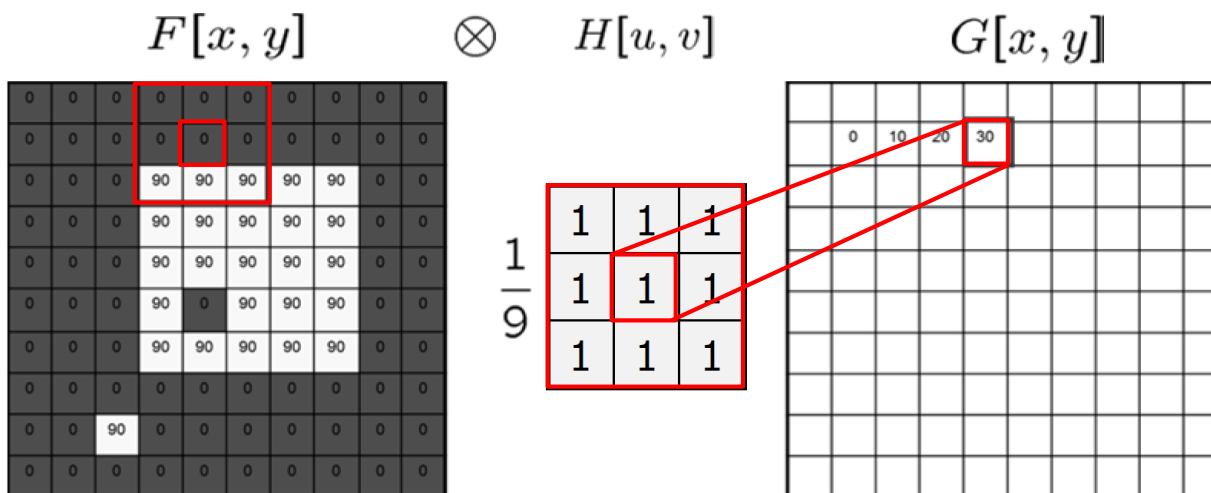
Logarithmic scaled centered spectrum  $|F(u, v)|$



*imagesc(log(abs(fftshift(fft\_image))))*



# Task A: Image Filtering



- Cross-correlation:**

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i + u, j + v]$$

$$G = H \otimes F$$

- Convolution:**

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v]$$

$$G = H \star F$$

- Can easily be done in frequency-domain

- Symmetric filter kernel  $\rightarrow$  Correlation = Convolution**

# The Convolution Theorem

$$f(x, y) \star h(x, y) \Leftrightarrow F(u, v)H(u, v)$$

$$f(x, y)h(x, y) \Leftrightarrow F(u, v) \star H(u, v)$$

where „ $\Leftrightarrow$ ” indicates a Fourier transform pair

- **Convolution** in spacial domain is equivalent to **multiplication** in frequency domain
- Efficient, when filter mask is large

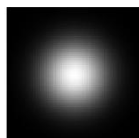
# Smoothing in Spatial Domain

$f(x, y)$



★

$h(x, y)$

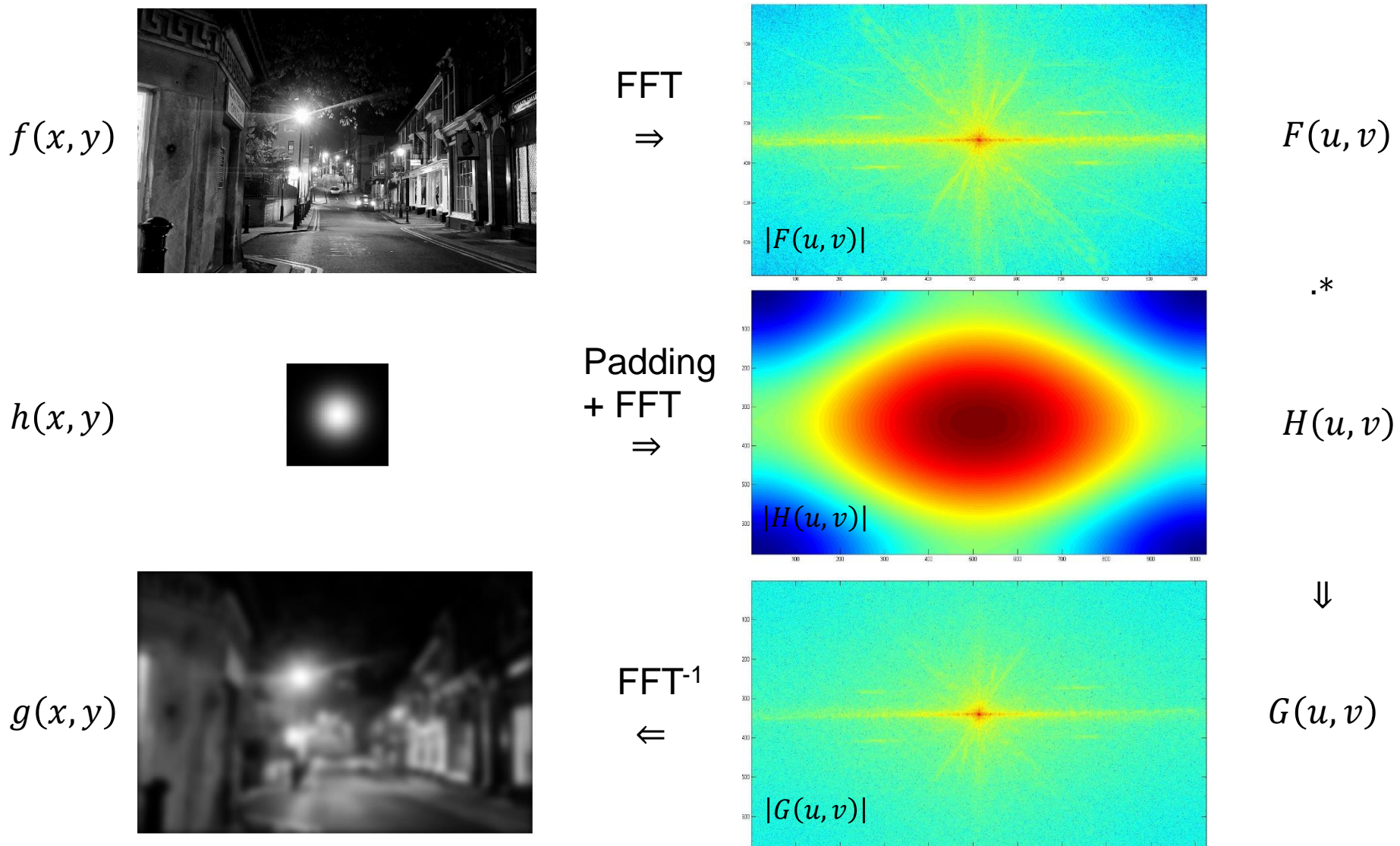


⇓

$g(x, y)$



# Smoothing in Frequency Domain

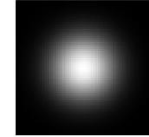


# Smoothing in Frequency Domain

- Inputs: image

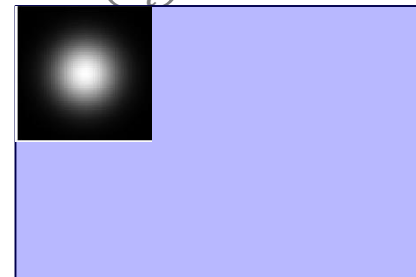
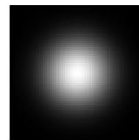


- and filter kernel

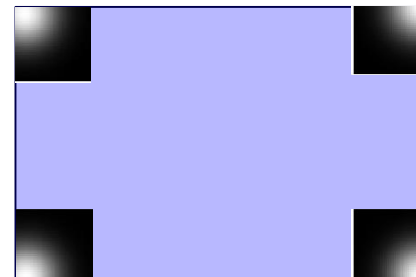
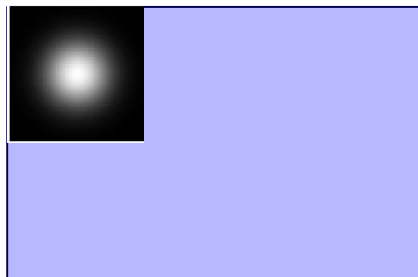


1) Padding of filter  $\rightarrow$  enlarge filter kernel to size  $s_i$  of image

- Copy filter kernel into matrix  $zeros(s_i)$

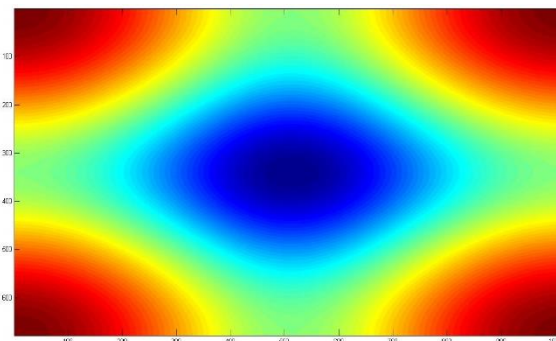
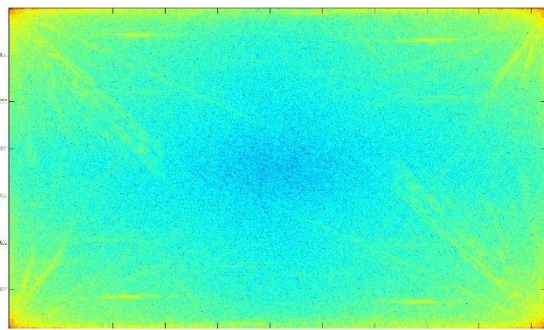


2) Center the filter using function *circshift*



# Smoothing in Frequency Domain

- 3) Transform image and filter kernel to frequency-domain using the function *fft2* (centering not necessary)



- 4) Multiply these arrays (complex values!) element-wise and transform result back to spatial domain using function *ifft2*





# Task A

- a. Read the input image *taskA.png* and convert it to a grayscale image from data type double with values between 0.0 and 1.0



- b. Add Gaussian noise to the image (function *imnoise*, parameters e.g.  $M=0$ ,  $V=0.01$ ) and plot the result
- c. Filter the noisy image with a self-made 2d Gaussian filter (e.g.  $\sigma=0.5$  and  $V=0.25$ ) in the frequency-domain and plot the result → noise removed?
- d. Plot the logarithmic centered image spectra of the original image, the noisy image, the Gaussian filter (padding) and the filtered image



# Exercise 4

**B:** Shape recognition using Fourier descriptors



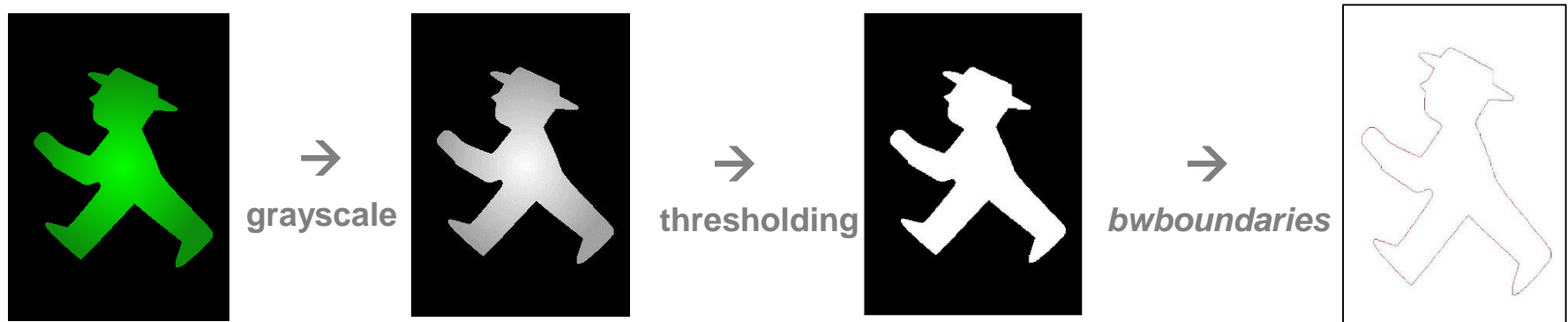
# Fourier Descriptors

- **Given:** Image which represents a shape of interest
- **Task:** Find this shape in other images automatically

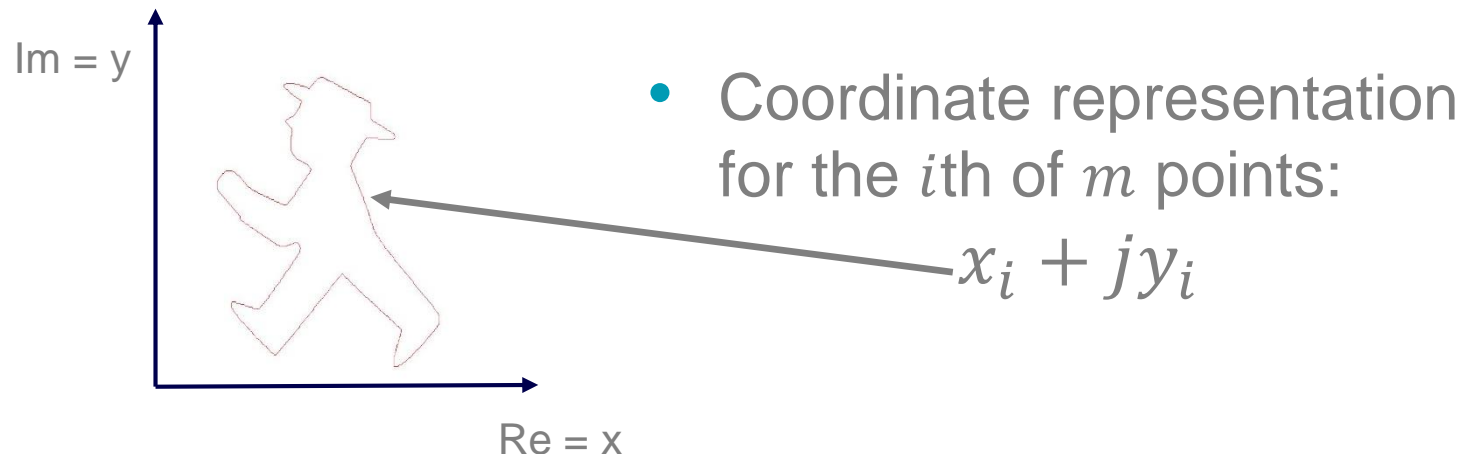


# Fourier Descriptors

- **Given:**  $m$  points representing the boundary of a **closed** region in the image



- Interpret the boundary coordinates as complex numbers



# Fourier Descriptors

- **Hint: Building the complex vector in Matlab**
- Interpret the boundary coordinates  $(x, y)$  as complex numbers in a vector  $b$

- $b = \begin{bmatrix} (y_1, x_1) \\ \vdots \\ (y_m, x_m) \end{bmatrix}$  ( $m \times 2$  array: output of *bwboundaries*)

- Building the complex vector  $D$ :

$$D = b(:, 2) + j * b(:, 1);$$

- Don't use  $j$  as variable in your code!

# Fourier Descriptors

- Result: Vector  $D$  with  $m$  complex-valued elements

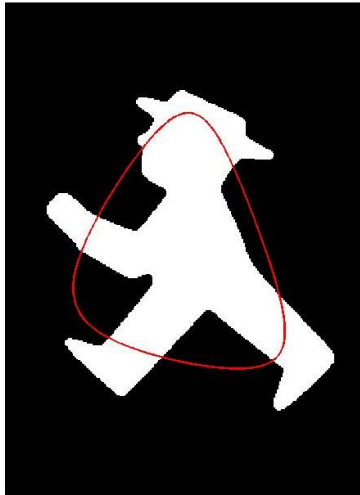
$$D = [(x_1 + jy_1), \dots, (x_m + jy_m)]^T$$

- DFT of  $D$  using function *fft* → **Fourier descriptor  $D_f$**
  - Simple manipulations of  $D_f$  in frequency-domain allow...
    - ...the representation of a generalized shape
    - ...the elimination of dependency of  $D$  from **position**, **scale** and **orientation**!
- **Crucial** for comparison of shapes!

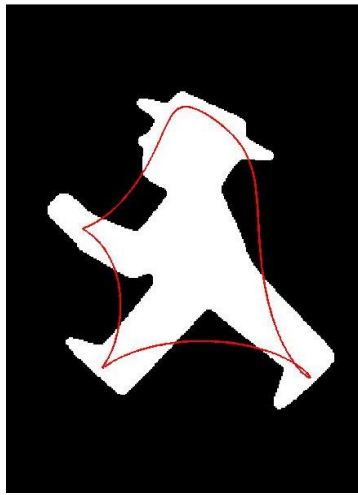
# Fourier Descriptor Manipulation

- Number of elements  $n$  in  $D_f \rightarrow$  generalization

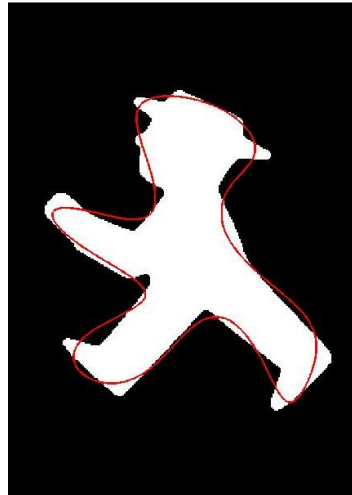
$n = 2$



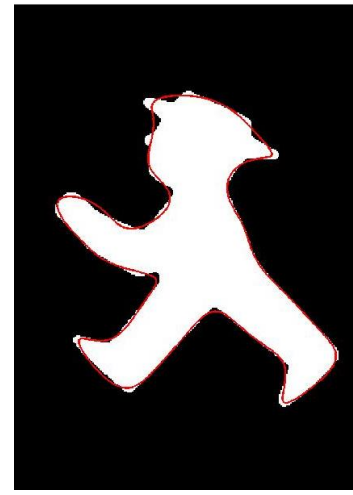
$n = 4$



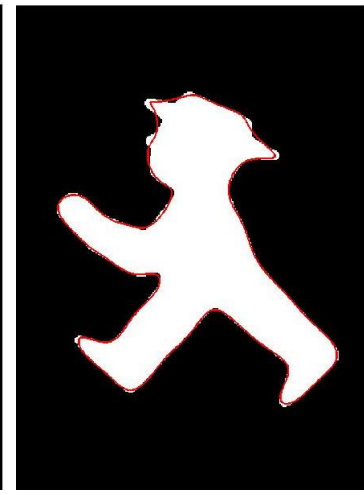
$n = 8$



$n = 16$



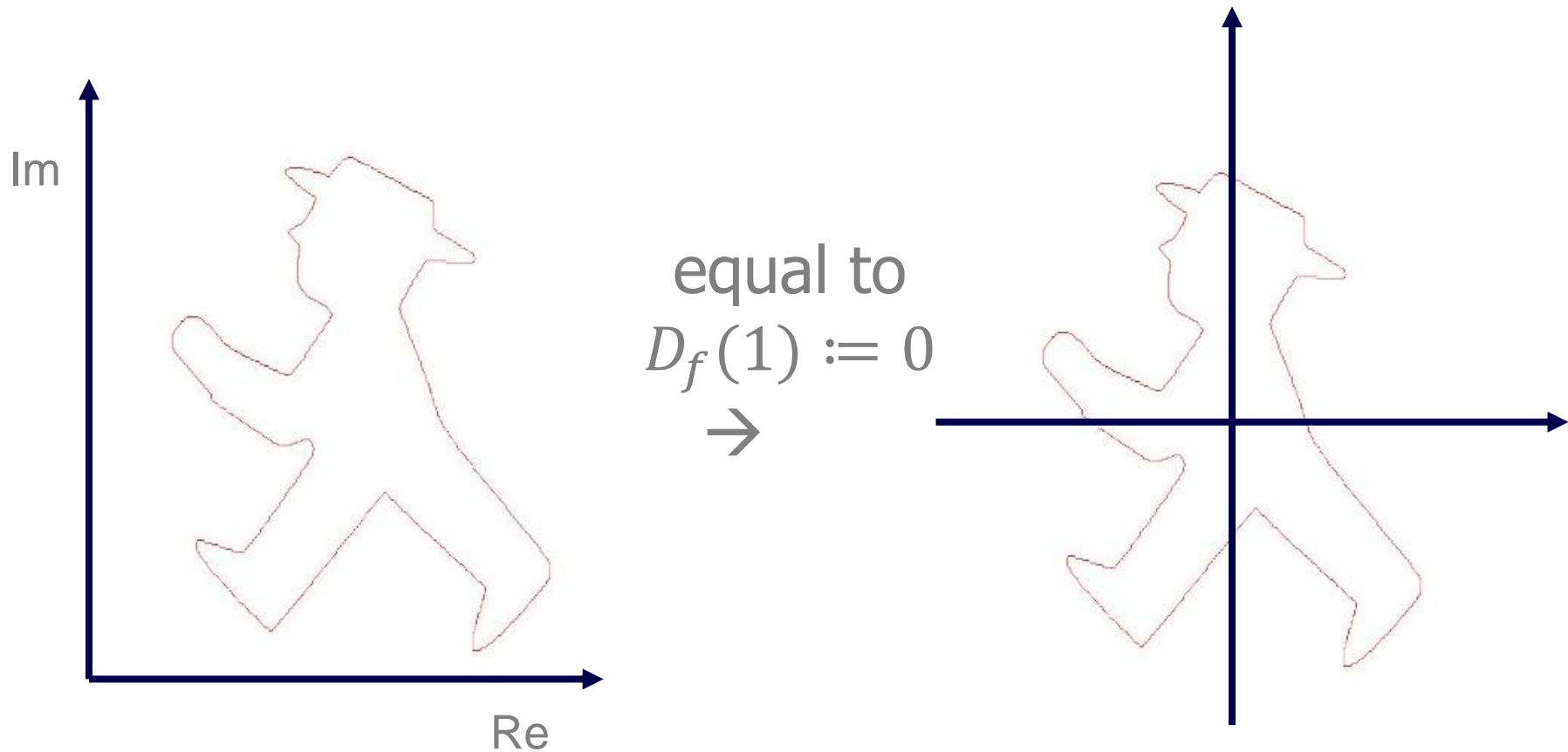
$n = 24$



- Reducing elements of  $D_f$ : Extract the first  $n$  elements (low frequency values) of  $D_f$  and forget the rest

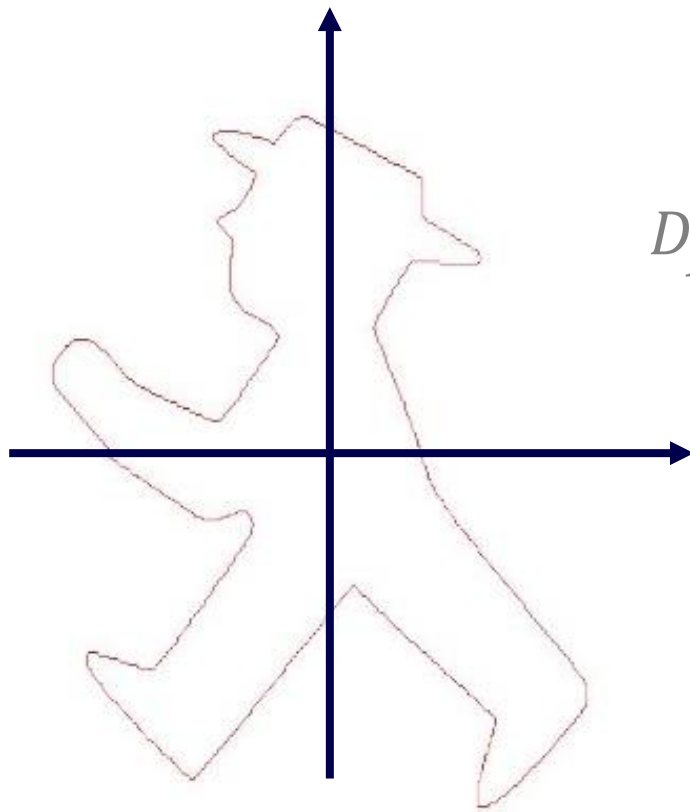
# Translation Invariance

- First Fourier component in  $D_f$  = centroid
- Throw away the first element by  $D_f = D_f(2:(n + 1))$



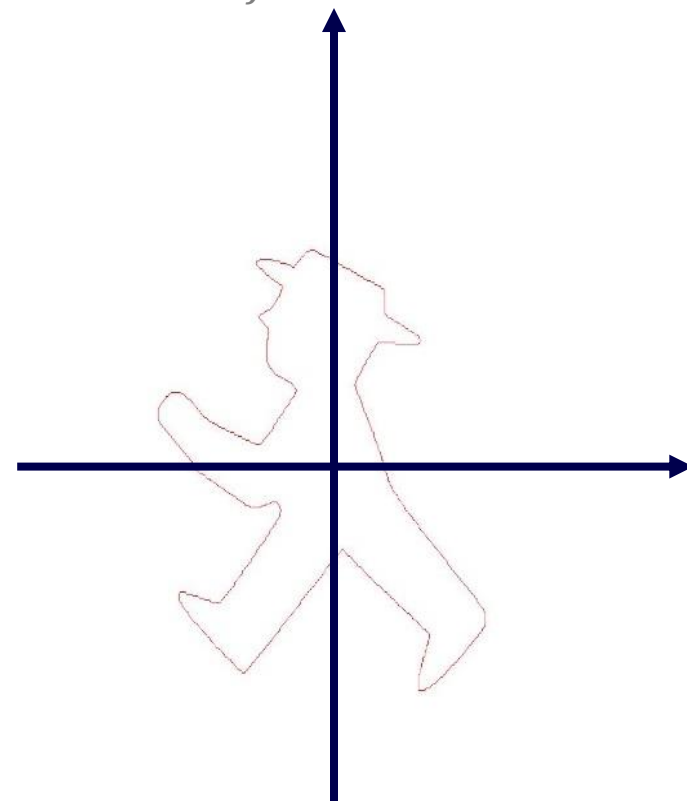
# Scale Invariance

- The second component  $D_f(2)$  corresponds to the radius (Corresponds to  $D_f(1)$  if we have a translation invariant  $D_f$ !!)
- Set this component to 1 by normalization of  $D_f$



$$D_f := \frac{D_f}{|D_f(2)|}$$

→



# Orientation Invariance

- Orientation is encoded in the phase information of  $D_f$
- Remove phase information by computing the absolute values of  $D_f$ :

$$D_f := |D_f|$$

- Function *abs* in Matlab
- The **amplitude spectrum** remains as final descriptor



# Comparison of Descriptors

- Comparison of two normalized Descriptors  $D_{f,1}$  and  $D_{f,2}$
- Euclidean distance  $d$

$$d = \sqrt{\sum_i \left( D_{f,1}(i) - D_{f,2}(i) \right)^2}$$

- Matlab:  $d = \text{norm}(D_{f,1} - D_{f,2})$ ;
- $D_{f,1}$  and  $D_{f,2}$  represent the same shape, if  $d < t$
- e.g.  $t = 0.06$

## Task B 1/4

- a. Read the image *trainingB.png* and convert it to a grayscale image from data type double with values between 0.0 and 1.0



- b. Derive a binary mask of the image where 1 represents the object of interest and 0 is background (functions *graythresh* and *im2bw*)

## Task B 2/4

- c. Build a Fourier-descriptor based on the binary image of b.
  - i. Extraction of boundaries of the binary mask: *bwboundaries*
  - ii. Use  $n = 24$  elements for the descriptor
  - iii. Make it invariant against translation, orientation and scale

### → Results:

- The final descriptor  $D_{f,train}$  is a  $1 \times n$  vector where the first element is 1.0
- A  $1 \times 1$  cell (matlab data type) containing an  $m \times 2$  array which represent the  $m$  corresponding border pixel coordinates of the found shape (output of *bwboundaries*)

# Task B 3/4

- d. Apply steps a.-c. on images *test1B.jpg* and *test2B.jpg* in order to identify all potential objects



→ Results for each image:

- Descriptors:  $k \times n$  array, where  $k$  is the number of identified boundaries
- Boundaries:  $k \times 1$  cell containing  $k$  ( $m \times 2$ ) arrays which represent the corresponding border pixel coordinates of the  $k$  found shapes

- e. Identify the searched object by comparison of Fourier-descriptor  $D_{f,train}$  (result of c) with all identified descriptors of the two test images  $D_{f,test}$  (result of d). Use the Euclidean distance of the element-wise differences, e.g. if

$$norm(D_{f,train} - D_{f,test}) < 0.06$$

→  $D_{f,test}$  represents the searched object

# Task B 4/4

- f. Plot the identified boundaries on the masks of the test images in order to validate the results (*imshow*, *hold on*, *plot*).



- Use the pixel coordinates of the shapes for plotting (result of *bwboundaries*)

# Matlab Cells

- Output of *bwboundaries*: ( $k \times 1$ ) Cell, where  $k$  is the number of identified closed boundaries

```
My_Cell =
```

```
[682x2 double]
[686x2 double]
[654x2 double]
[685x2 double]
[154x2 double]
[168x2 double]
[328x2 double]
[335x2 double]
[377x2 double]
[332x2 double]
[ 52x2 double]
[333x2 double]
[350x2 double]
[288x2 double]
[ 98x2 double]
[196x2 double]
[ 57x2 double]
[ 41x2 double]
[ 44x2 double]
[189x2 double]
[458x2 double]
[326x2 double]
[253x2 double]
[ 84x2 double]
[ 74x2 double]
[244x2 double]
[289x2 double]
[209x2 double]
[239x2 double]
[ 87x2 double]
[238x2 double]
[ 84x2 double]
[ 58x2 double]
[ 12x2 double]
[  3x2 double]
[216x2 double]
```

- Access the 34th array of boundary coordinates:

```
K>> boundary_points = My_Cell{34}
```

```
boundary_points =
```

```
51    886
50    887
49    888
50    888
51    888
52    888
53    888
54    888
54    887
53    887
52    887
51    886
```



**Thank you!**