Bauhaus-
Universität
Weimar

Image Analysis and Object Recognition

Exercise 5

Summer Semester 2024

(Course materials for internal use only!)

Computer Vision in Engineering – Prof. Dr. Rodehorst
M.Sc. Mariya Kaisheva
mariya.kaisheva@uni-weimar.de

Agenda

Topics:

Assignment 1.	Image enhancement, Binarization, Morphological operators
Assignment 2.	Gradient of Gaussian filtering, Förstner interest operator
Assignment 3.	Shape detection based on Hough-voting
Assignment 4.	Frequency domain filtering, Shape recognition via Fourier descriptors
Assignment 5.	**Clustering and Region Growing for Image Segmentation**
Assignment 6.	Convolutional neural networks for image classification
Final Project.	**-** *Will be announced during the last exercise class* **-**

Agenda

Start date and submission deadlines:

Assignment 1.	~~18.04-24 – 01.05.24~~
Assignment 2.	~~02.05.24 – 15.05.24~~
Assignment 3.	~~16.05.24 – 29.05.24~~
Assignment 4.	~~30.05.24 – 12.06.24~~
Assignment 5.	**20.06.24 – 26.06.24**
Assignment 6.	27.06.24 – 10.07.24
Final Project.	11.07.24 – 22.09.24

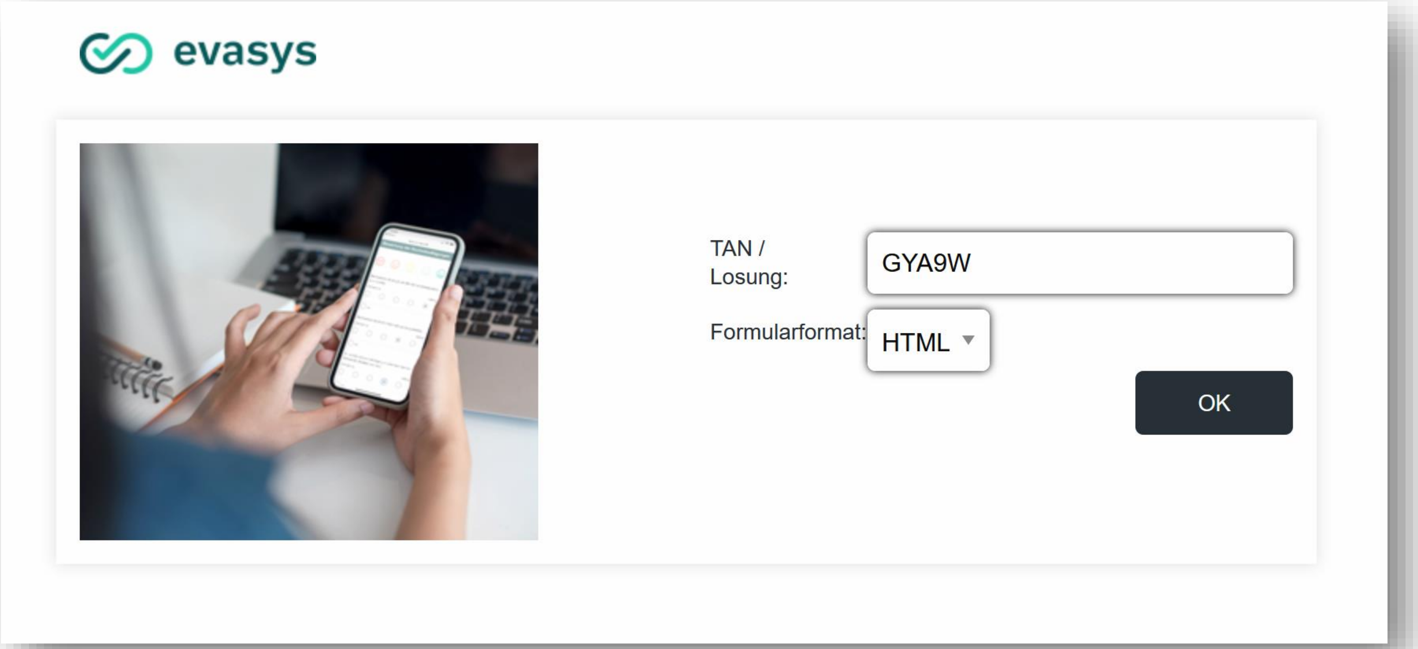
Wednesday by 23:00
(Central European Time)

Online Course Evaluation

Teaching Evaluation:

URL: https://evasys.uni-weimar.de/evasys/online/

Code: GYA9W



Assignment 4:
Sample Solution

Assignment 4: Overview

Topics:

- Filtering in frequency domain
- Shape recognition using Fourier descriptors

Goal:

- Practice noise removal in the frequency domain (Task A)
- Practice automatic shape detection using Fourier descriptors (Task B)

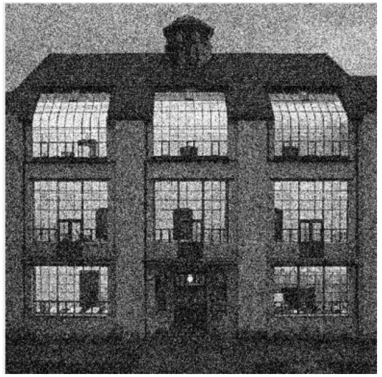
Input:

- All images provided for this assignment can be found on Moodle course page

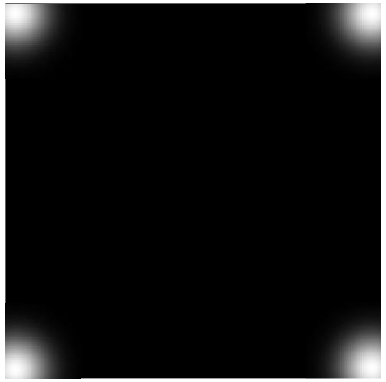
Task A

Note:
All Fourier spectra are on this slide have been logarithmically **scaled** and **shifted** for better **visualization only.**

$f(x,y)$



$h(x,y)$



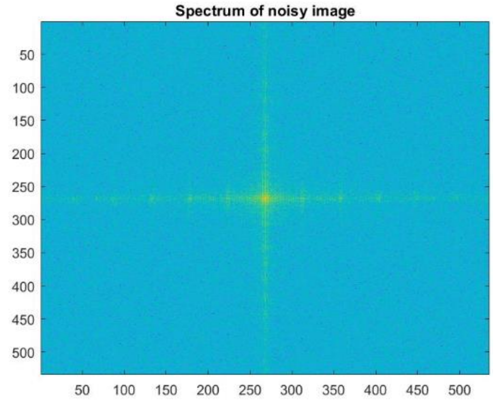
$g(x,y)$

FFT $\Rightarrow$

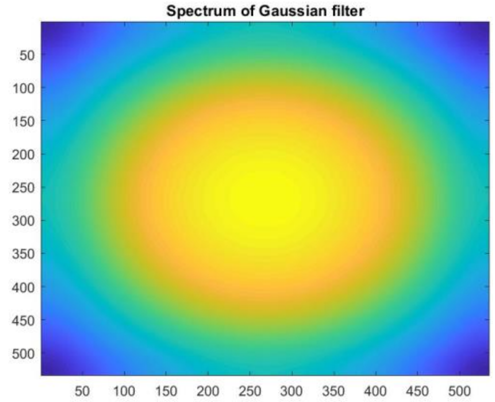
1

FFT $\Rightarrow$

FFT^{-1} $\Leftarrow$ 3



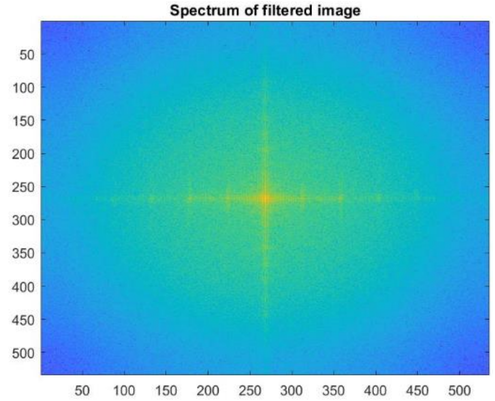
$F(u,v)$



$.*$ 2

$H(u,v)$

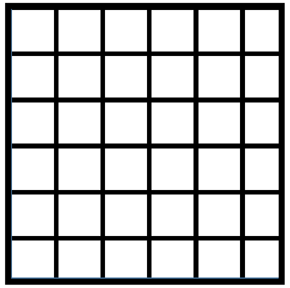
$\Downarrow$



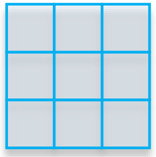
$G(u,v)$

Bauhaus-Universität Weimar

Task A

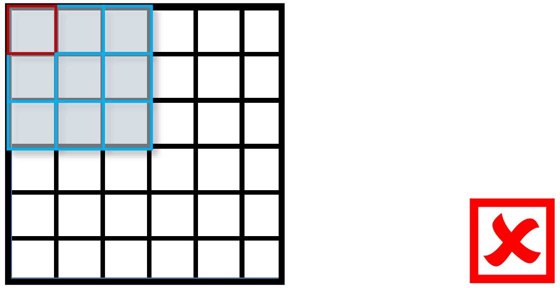


image

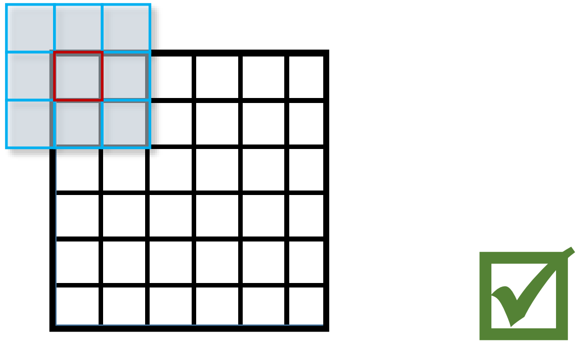


filter mask

No Filter Centering

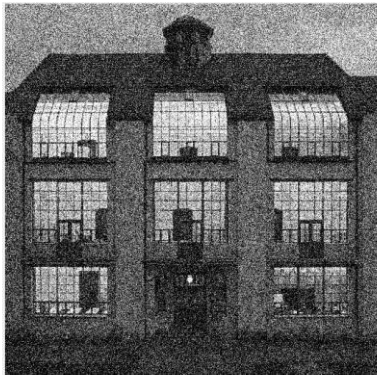
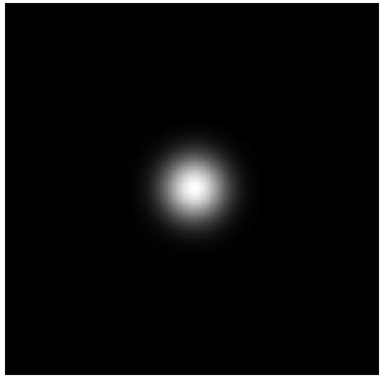


With Filter Centering



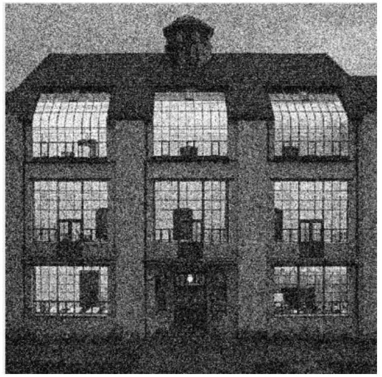
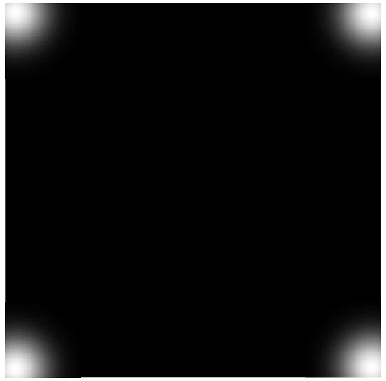
Task A

No Filter Centering in spatial domain

 $\star$  $\Rightarrow$ 

$f(x,y)$ $h(x,y)$ $g(x,y)$

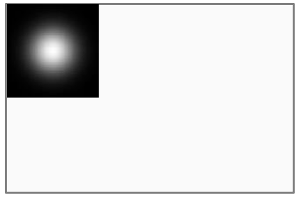
With Filter Centering in spatial domain

 $\star$  $\Rightarrow$ 

$\star$ - filtering in the frequency domain **without any spectrum shifting**

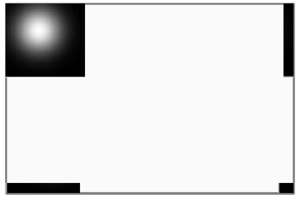
Task A

```
function A_fft_filtering
%        ===============
sigma = 1.4;                                          % standard deviation
Img = double(mean(imread('taskA.png'), 3)) / 255;       % read input image
Nsy = imnoise(Img, 'gaussian', 0, 0.01);                       % add noise
NsyFFT = fft2(Nsy);                                    % Fourier transform

kernel = Gauss1d(sigma)' * Gauss1d(sigma);            % 2D Gaussian kernel
filter = zeros(size(Nsy));
filter(1:size(kernel, 1), 1:size(kernel, 2)) = kernel;     % Filter padding
filter = circshift(filter, -floor(size(kernel)/2));         % Center filter
FilFFT = fft2(filter);                                 % Fourier transform

MulFFT = NsyFFT .* FilFFT;                    % Multiply image with filter
Res = ifft2(MulFFT);                           % Inverse Fourier transform

figure, imshow(Img), title('Original image');
figure, imshow(Nsy), title('Noisy image');
figure, imagesc(log(abs(fftshift(NsyFFT)))), title('Spectrum of noisy image');
figure, imagesc(log(abs(fftshift(FilFFT)))), title('Spectrum of Gaussian filter');
figure, imagesc(log(abs(fftshift(MulFFT)))), title('Spectrum of filtered image');
figure, imshow(Res), title('Filtered image');


function g = Gauss1d(sigma)
%            ==============
r = round(3*sigma); i = -r:r;
g = exp(-i.^2 / (2*sigma^2)) / (sigma*sqrt(2*pi));
```



Task A

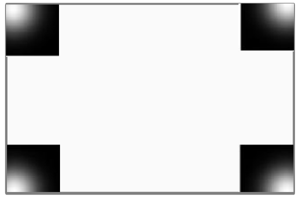
```
function A_fft_filtering
%        ===============
sigma = 1.4;                                              % standard deviation
Img = double(mean(imread('taskA.png'), 3)) / 255;          % read input image
Nsy = imnoise(Img, 'gaussian', 0, 0.01);                          % add noise
NsyFFT = fft2(Nsy);                                       % Fourier transform

kernel = Gauss1d(sigma)' * Gauss1d(sigma);               % 2D Gaussian kernel
filter = zeros(size(Nsy));
filter(1:size(kernel, 1), 1:size(kernel, 2)) = kernel;      % Filter padding
filter = circshift(filter, -floor(size(kernel)/2));          % Center filter
FilFFT = fft2(filter);                                    % Fourier transform

MulFFT = NsyFFT .* FilFFT;                        % Multiply image with filter
Res = ifft2(MulFFT);                               % Inverse Fourier transform

figure, imshow(Img), title('Original image');
figure, imshow(Nsy), title('Noisy image');
figure, imagesc(log(abs(fftshift(NsyFFT)))), title('Spectrum of noisy image');
figure, imagesc(log(abs(fftshift(FilFFT)))), title('Spectrum of Gaussian filter');
figure, imagesc(log(abs(fftshift(MulFFT)))), title('Spectrum of filtered image');
figure, imshow(Res), title('Filtered image');


function g = Gauss1d(sigma)
%            ==============
r = round(3*sigma); i = -r:r;
g = exp(-i.^2 / (2*sigma^2)) / (sigma*sqrt(2*pi));
```



```
circshift(Filter, [-1 -1]));
```

Task A

```
function A_fft_filtering
%        ===============
sigma = 1.4;                                                  % standard deviation
Img = double(mean(imread('taskA.png'), 3)) / 255;              % read input image
Nsy = imnoise(Img, 'gaussian', 0, 0.01);                            % add noise
NsyFFT = fft2(Nsy);                                            % Fourier transform

kernel = Gauss1d(sigma)' * Gauss1d(sigma);                    % 2D Gaussian kernel
filter = zeros(size(Nsy));
filter(1:size(kernel, 1), 1:size(kernel, 2)) = kernel;           % Filter padding
filter = circshift(filter, -floor(size(kernel)/2));               % Center filter
FilFFT = fft2(filter);                                         % Fourier transform

MulFFT = NsyFFT .* FilFFT;                              % Multiply image with filter
Res = ifft2(MulFFT);                                    % Inverse Fourier transform

figure, imshow(Img), title('Original image');
figure, imshow(Nsy), title('Noisy image');
figure, imagesc(log(abs(fftshift(NsyFFT)))), title('Spectrum of noisy image');
figure, imagesc(log(abs(fftshift(FilFFT)))), title('Spectrum of Gaussian filter');
figure, imagesc(log(abs(fftshift(MulFFT)))), title('Spectrum of filtered image');
figure, imshow(Res), title('Filtered image');


function g = Gauss1d(sigma)
%            ==============
r = round(3*sigma); i = -r:r;
g = exp(-i.^2 / (2*sigma^2)) / (sigma*sqrt(2*pi));
```



```
circshift(Filter, [-r -r]));
```

Input data

Task B

training image

test image 1

test image 2

test image 3

Task B

```matlab
function B_fourier_descr
%        ================
train_image = double(mean(imread('trainB.png'), 3)) / 255;      % read train image
[D_t, ~] = fourier_descr(Thresholding(train_image));      % build model descriptor

for k = 1:3
    name = strcat('test', num2str(k), 'B.jpg');       % name of current test image
    image = double(mean(imread(name), 3)) / 255;             % read test image
    mask = Thresholding(image);                              % binary mask
    [D, B] = fourier_descr(mask);                      % build all descriptors
    figure, imshow(mask), hold on;
    for i = 1 : size(D, 1)                                 % for each descriptor
        dist = norm(D_t - D(i, :));          % if descriptor is similar to model
        if dist < 0.07
            plot(B{i}(:, 2), B{i}(:, 1), 'r', 'LineWidth', 1);     % plot boundary
        end
    end
end

function [fd, B] = fourier_descr(Img)
%                  ==================
n = 25;                                        % number of descriptor elements
B = bwboundaries(Img);                                % extract all boundaries
fd = zeros(length(B), n-1);
for i = 1 : length(B)                                      % for each boundary
    if length(B{i}) > n
        desc = fft(B{i}(:,2) + j*B{i}(:,1));        % points as imaginary numbers
        fd(i, :) = abs(desc(2:n) / desc(2));              % normalize descriptor
    end
end

function mask = Thresholding(Image)                        % image thresholding
%              ===================
mask = im2bw(Image, graythresh(Image));
```

Output of `bwboundaries`:
$(\mathbf{k} \times \mathbf{1})$ 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]
```

Bauhaus-Universität Weimar

Task B

```matlab
function B_fourier_descr
%         ================
train_image = double(mean(imread('trainB.png'), 3)) / 255;     % read train image
[D_t, ~] = fourier_descr(Thresholding(train_image));      % build model descriptor

for k = 1:3
    name = strcat('test', num2str(k), 'B.jpg');       % name of current test image
    image = double(mean(imread(name), 3)) / 255;                 % read test image
    mask = Thresholding(image);                                      % binary mask
    [D, B] = fourier_descr(mask);                          % build all descriptors
    figure, imshow(mask), hold on;
    for i = 1 : size(D, 1)                                   % for each descriptor
        dist = norm(D_t - D(i, :));            % if descriptor is similar to model
        if dist < 0.07
            plot(B{i}(:, 2), B{i}(:, 1), 'r', 'LineWidth', 1);    % plot boundary
        end
    end
end

function [fd, B] = fourier_descr(Img)
%                  ==================
n = 25;                                         % number of descriptor elements
B = bwboundaries(Img);                               % extract all boundaries
fd = zeros(length(B), n-1);
for i = 1 : length(B)                                      % for each boundary
    if length(B{i}) > n
        desc = fft(B{i}(:,2) + j*B{i}(:,1));     % points as imaginary numbers
        fd(i, :) = abs(desc(2:n) / desc(2));            % normalize descriptor
    end
end

function mask = Thresholding(Image)                        % image thresholding
%              ===================
mask = im2bw(Image, graythresh(Image));
```

Translation $D_f(1) := 0$

Scale $D_f := \dfrac{D_f}{|D_f(2)|}$

Orientation $D_f := |D_f|$

Output of `bwboundaries`:
$(\mathbf{k} \times \mathbf{1})$ cell,

where k is the number of identified closed boundaries

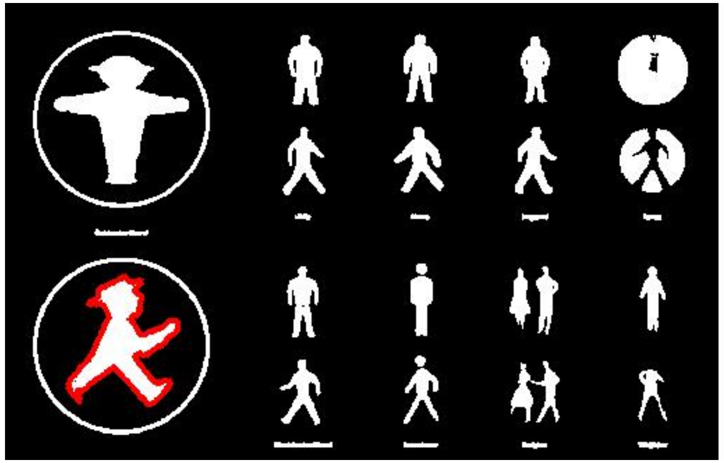
```
My_Cell =

    [682x2 double]
    [686x2 double]
    [654x2 double]
    [685x2 double]
    [154x2 double]
    [168x2 double]
    [328x2 double]
    [335x2 double]
    [377x2 double]
    [332x2 double]
    [ 52x2 double]
```

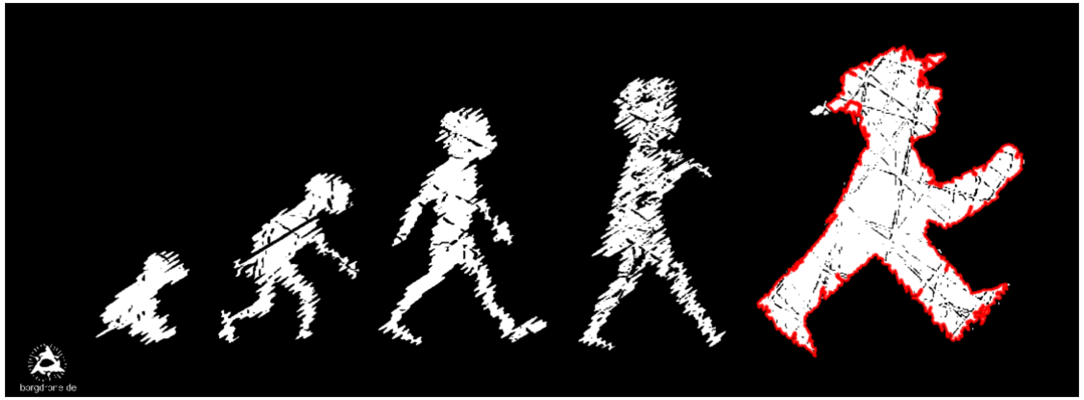
Bauhaus-Universität Weimar

Expected results

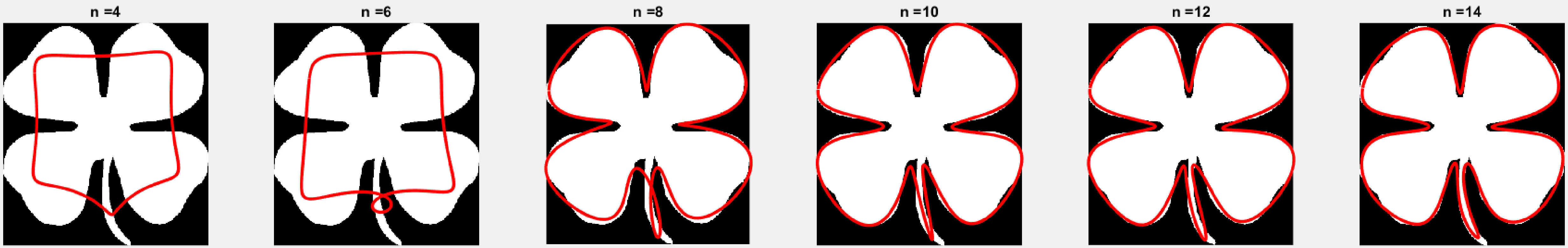
Task B

training image

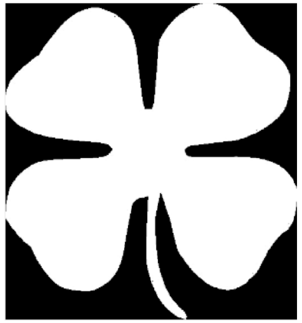




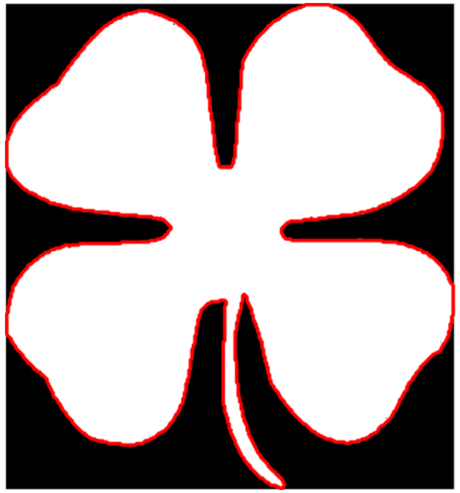
Discussion: Visualization of the simplified shape boundary



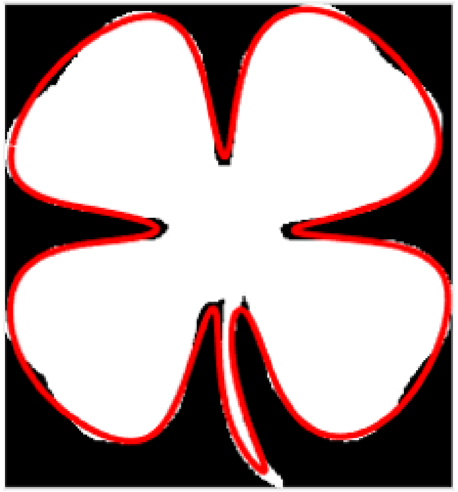
Discussion: Visualization of the simplified shape boundary



Binary Input Image

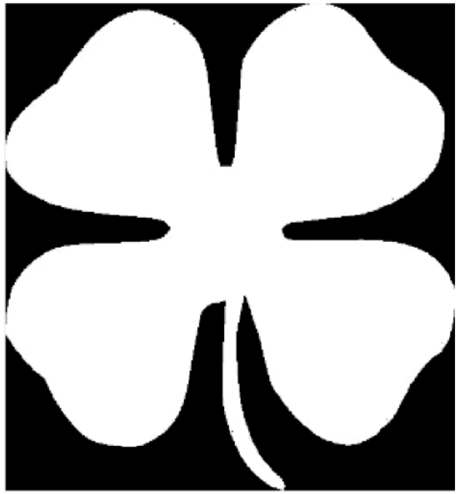


Binary Input Image
+
Complete Boundary Overlay



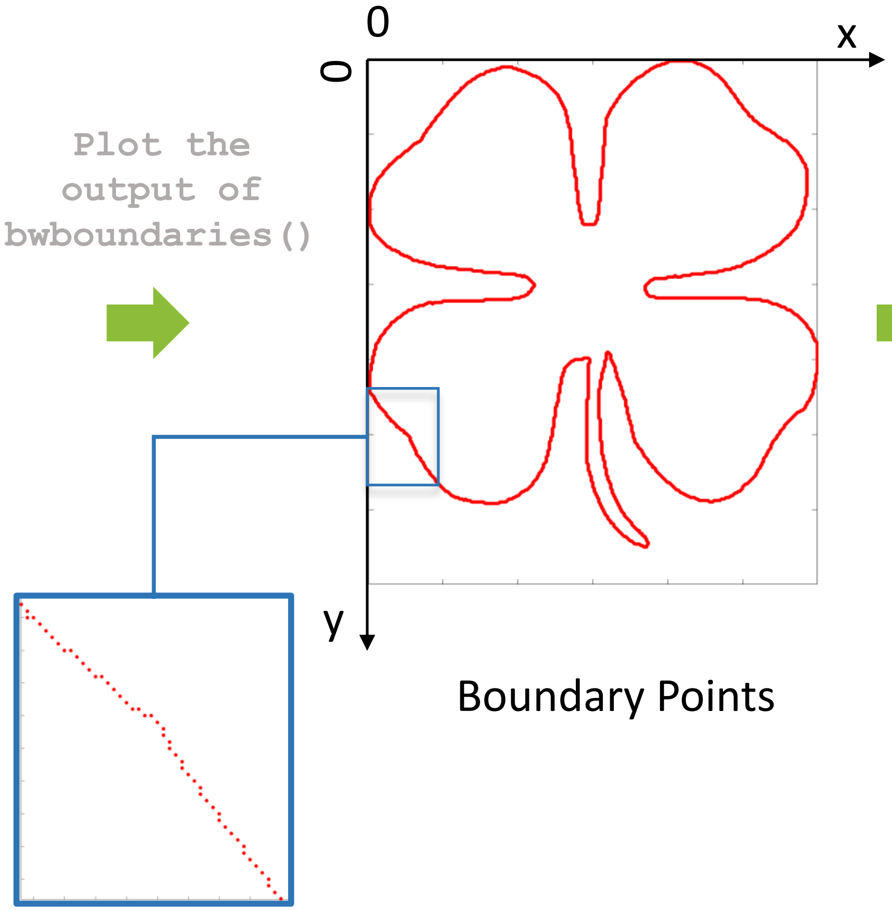
Binary Input Image
+
Simplified Boundary Overlay

Discussion: Visualization of the simplified shape boundary



Binary Input Image

Plot the output of bwboundaries()



Boundary Points

Close-Up of a Boundary Segment

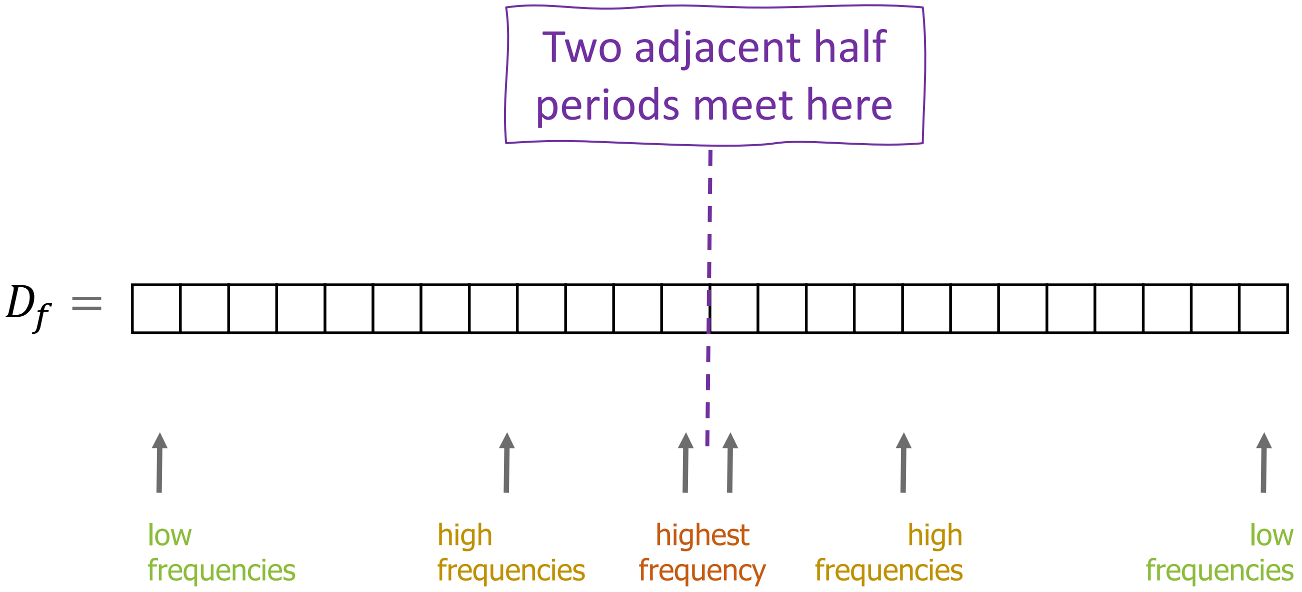
$$D = x + j * y$$

Complex-valued vector D

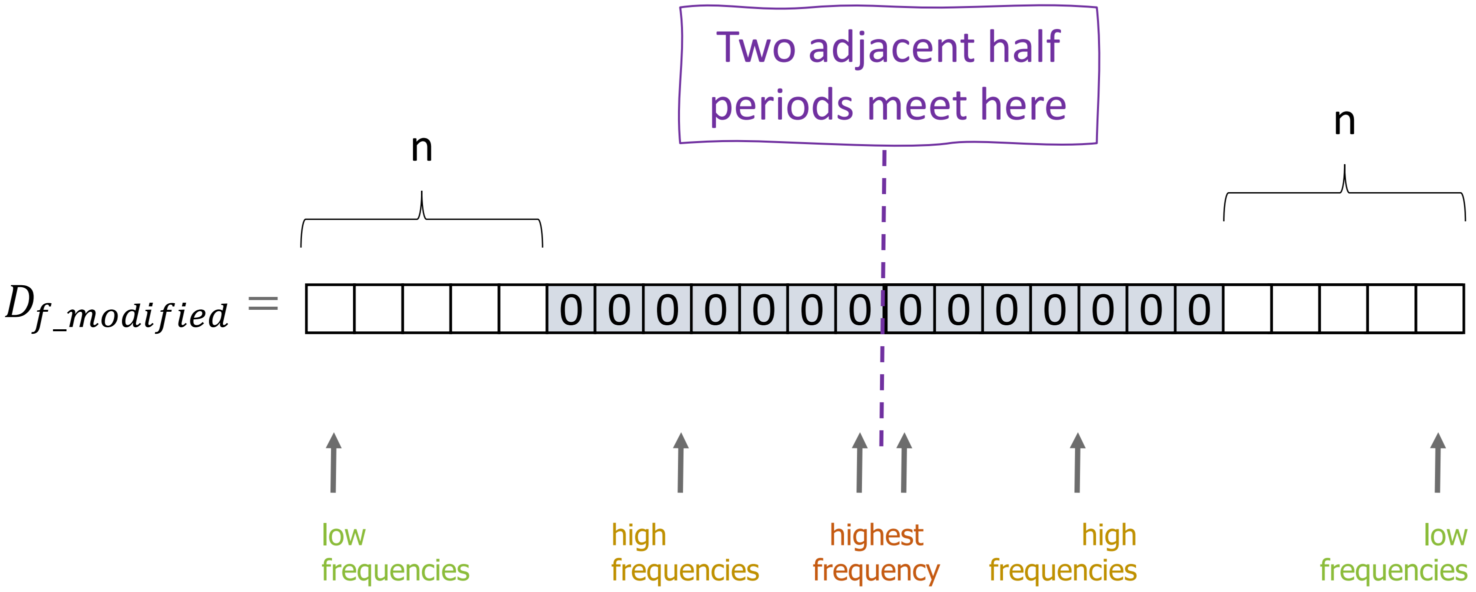
$$D_f = FFT(D)$$

Fourier Descriptor D_f

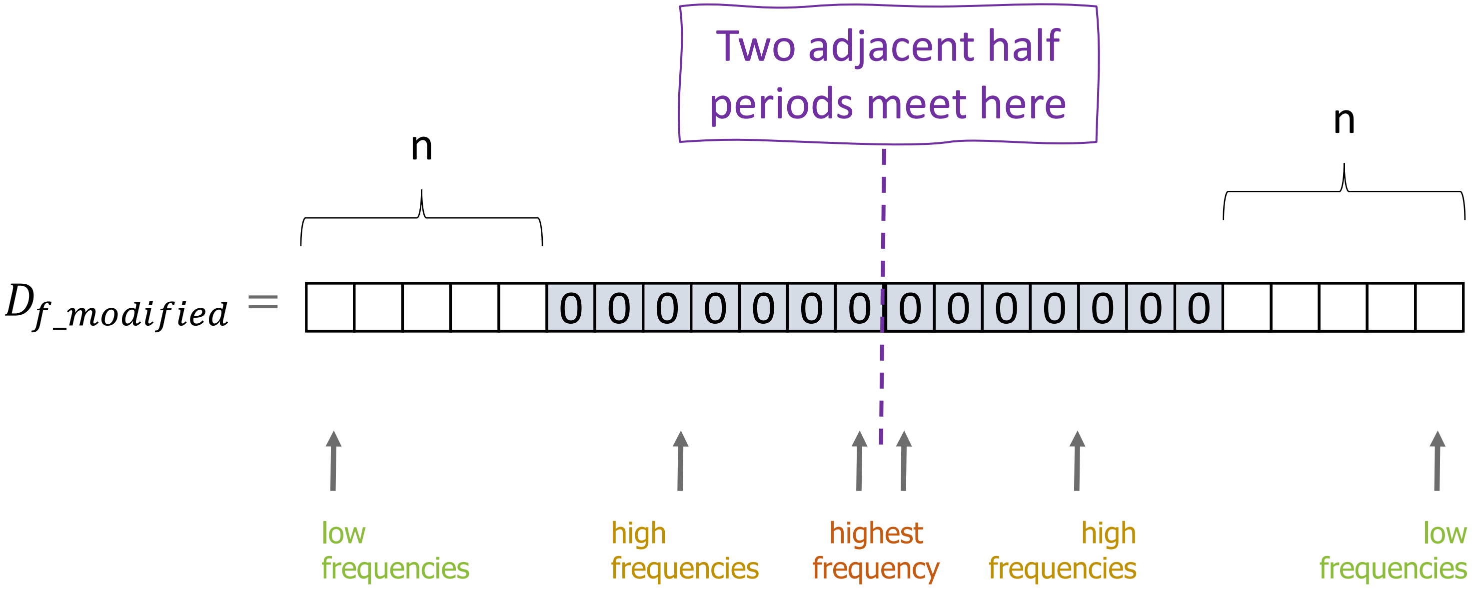
Discussion: Visualization of the simplified shape boundary



Discussion: Visualization of the simplified shape boundary



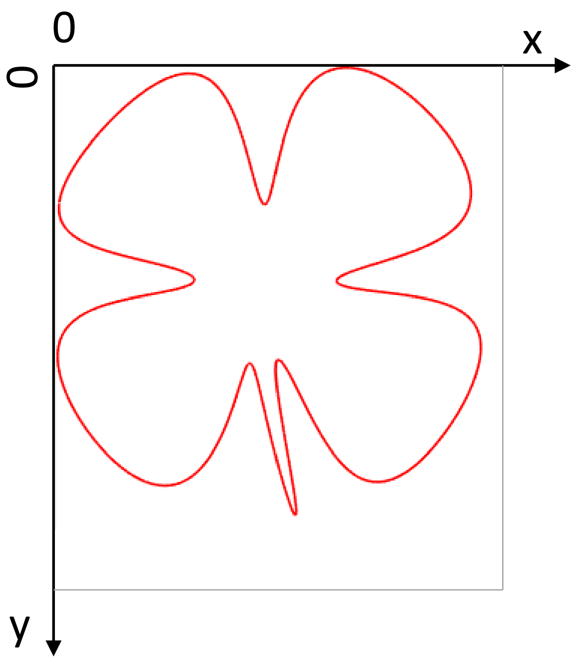
Discussion: Visualization of the simplified shape boundary



Implementation Tip:

1) Modify only the values in the first half period
2) Mirror the modified segment to complete the descriptor

Discussion: Visualization of the simplified shape boundary



Boundary Points

$$x = real(D_{simplified})$$

$$y = imag(D_{simplified})$$

$D_{simplified}$ ← Inverse FFT ← $D_{f\_modified}$

Bauhaus-Universität Weimar

Assignment 5

Assignment 5: Overview

Topics:

- *k-means* clustering
- Watershed segmentation

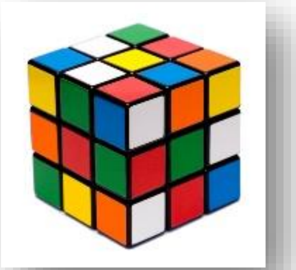
Due to the time schedule changes, **only one** of the two subtasks is compulsory.

Goal:

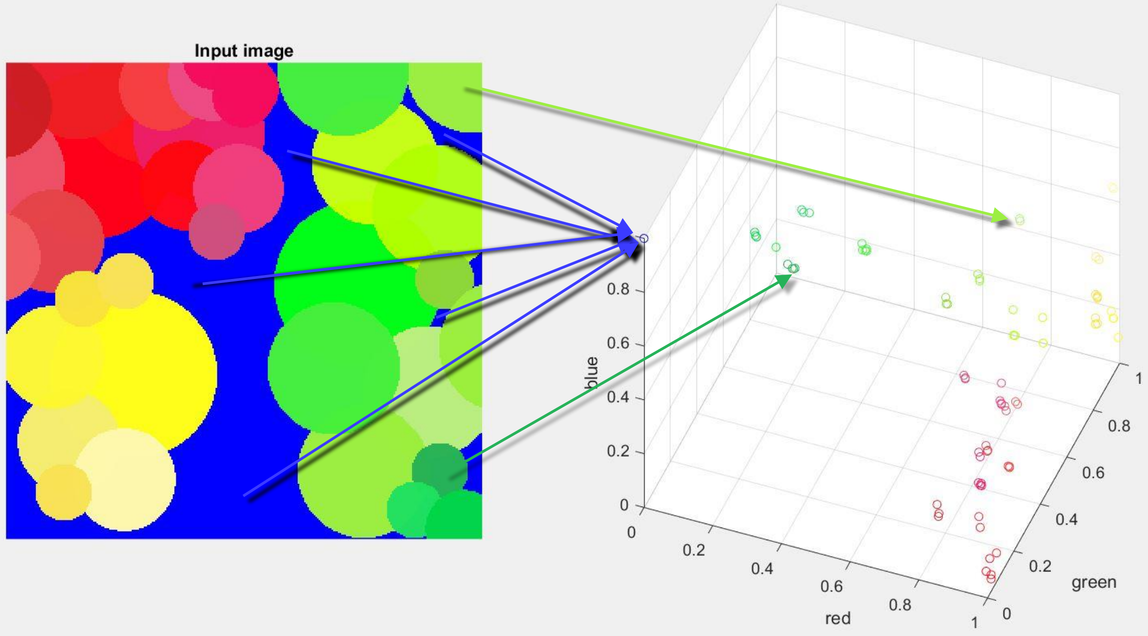
- Practice unsupervised image segmentation

Input:

- The required input images can be found on the Moodle course page



Assignment 5:Feature Space



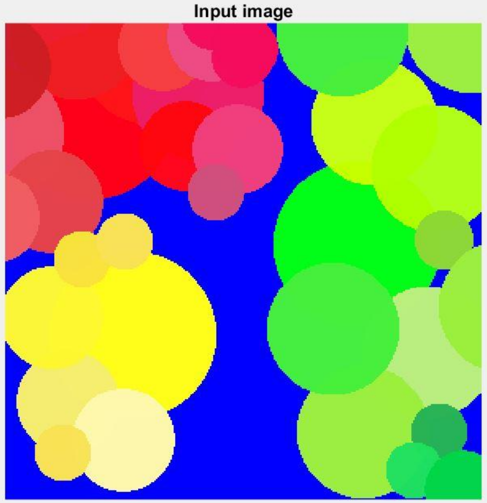
Given: 3-channel color image

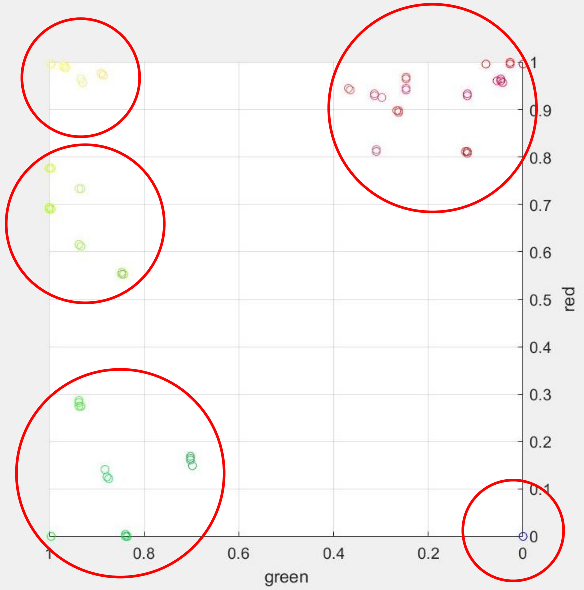
- Each channel (r, g, b) represents one dimension of a feature space
- Each pixel of the image maps to a point in that space
- Additional spatial support is given by the position (x, y) in the image

=> **5D feature space**

Artificially generated image

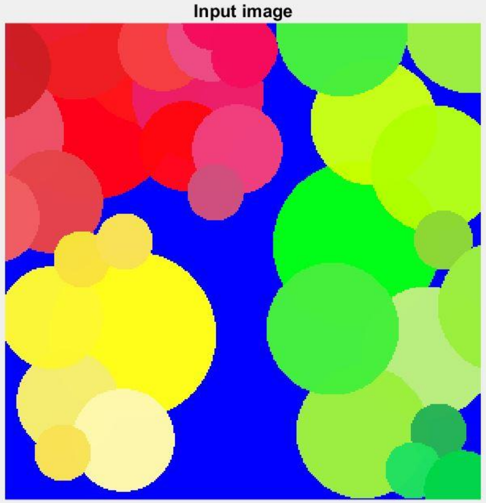
Assignment 5: Clustering

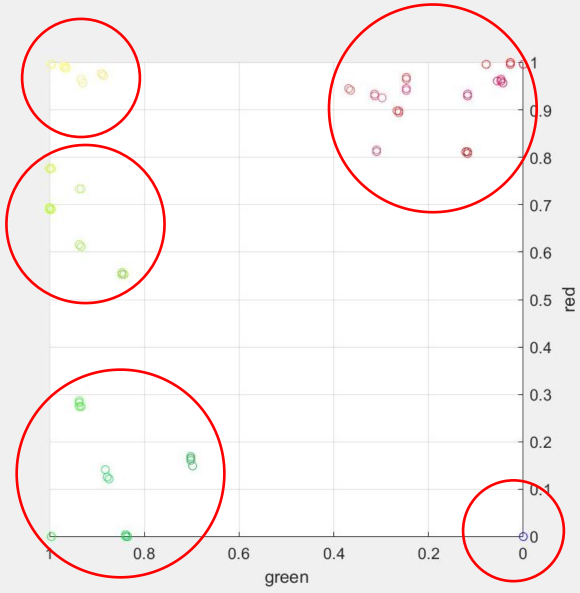


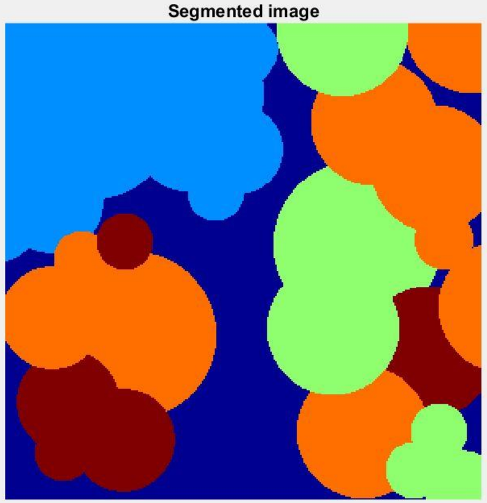


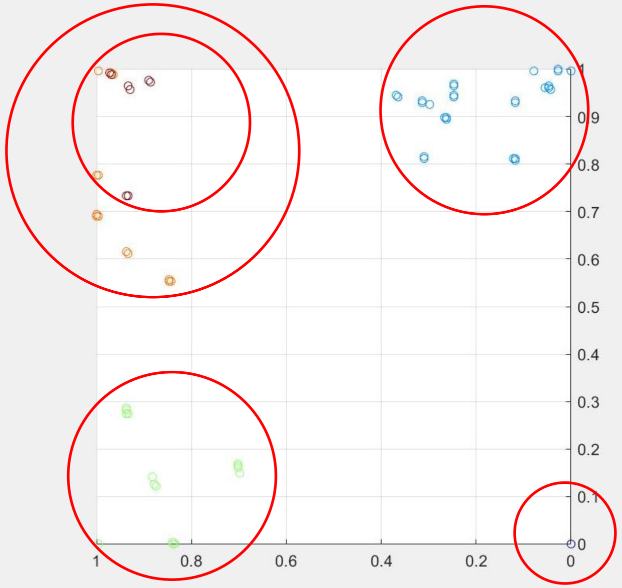
Artificially generated image

Assignment 5: Clustering Results



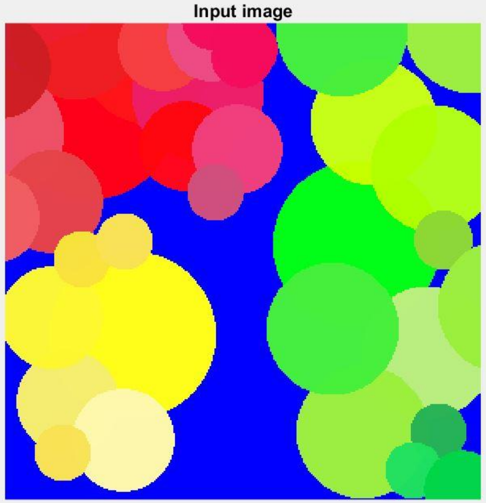


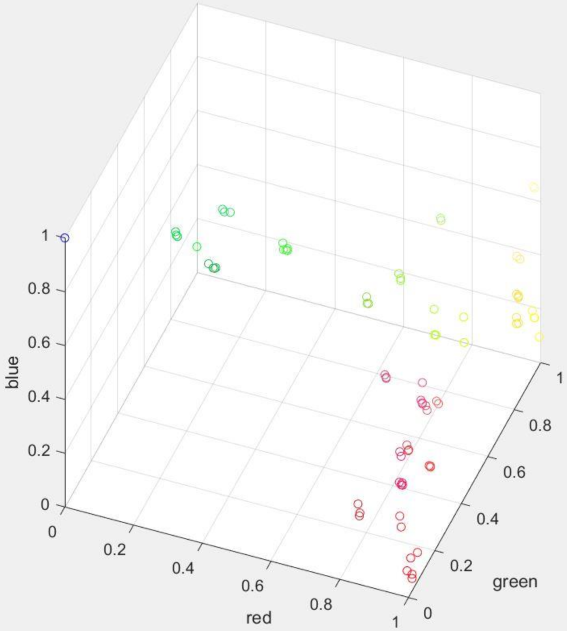


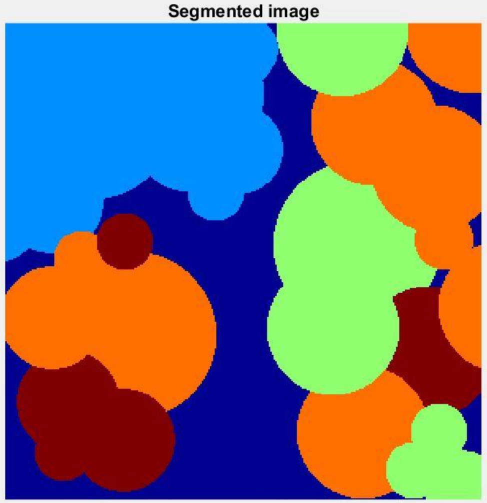


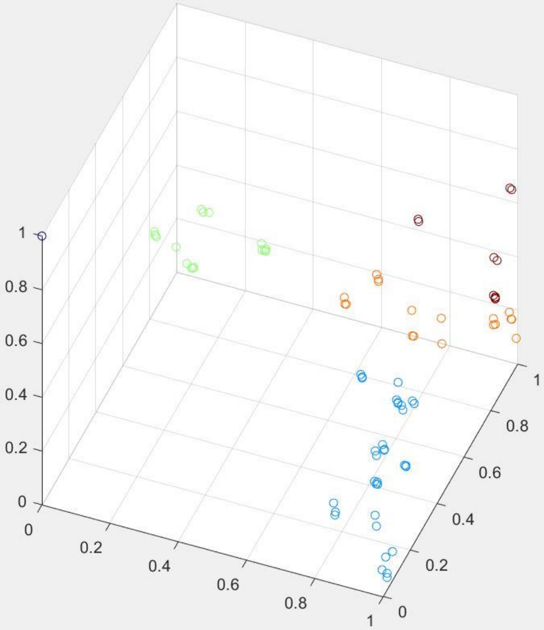
Artificially generated image

Assignment 5: Clustering Results



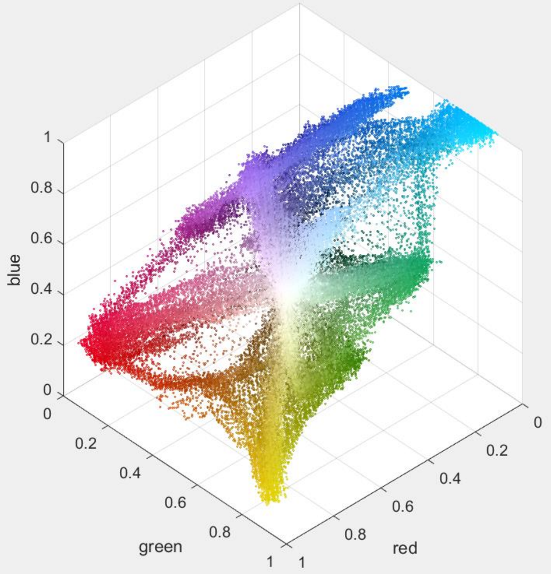


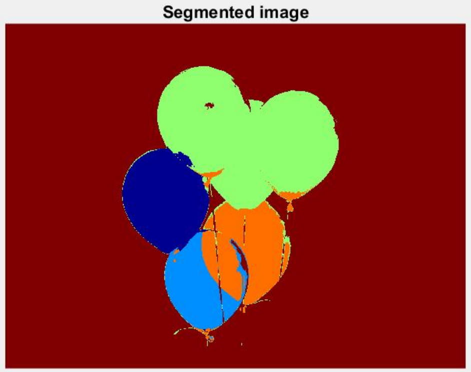


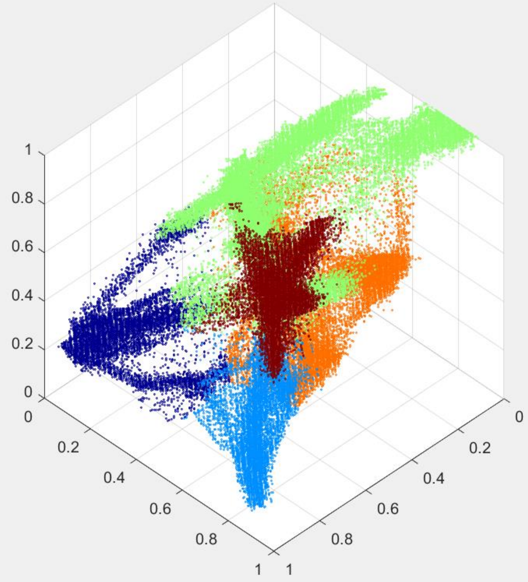


Artificially generated image

Assignment 5: Clustering Results







Real photo example

Assignment 5: Task A

Task A: *k-means* clustering

a. Read the exemplary color input image `inputEx5_1.jpg` and set up a **three-dimensional RGB feature space** (`reshape`).

b. Implement your **own** *k-means* clustering approach with random initialization (see lecture notes) to group the color features.

c. Select an appropriate number of clusters *k*, apply the algorithm and visualize the detected groups in feature and image space (e.g. with color coding: `colormap`).

d. Extend the three-dimensional feature space with **additional spatial support** using the pixel positions (*x*, *y*) and test your algorithm on the five-dimensional feature space. Are the results different or significantly better?

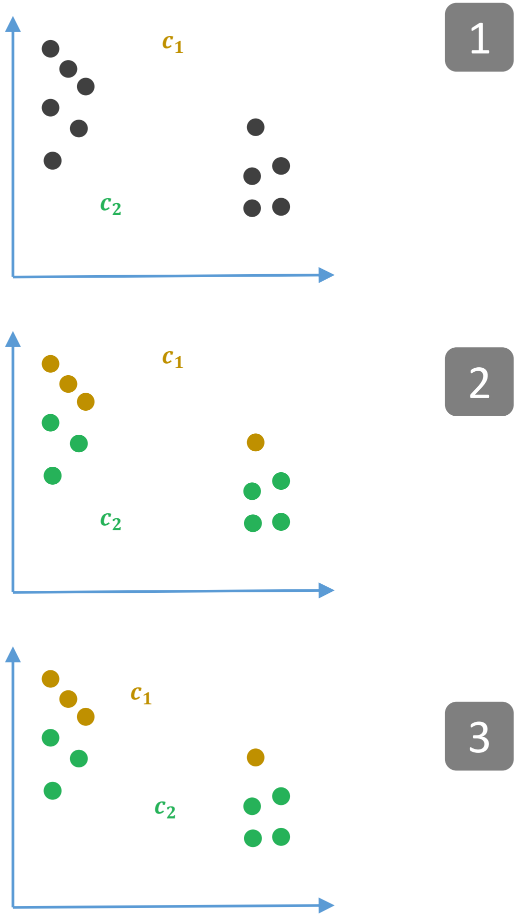
k-means: Overview

Algorithm description:

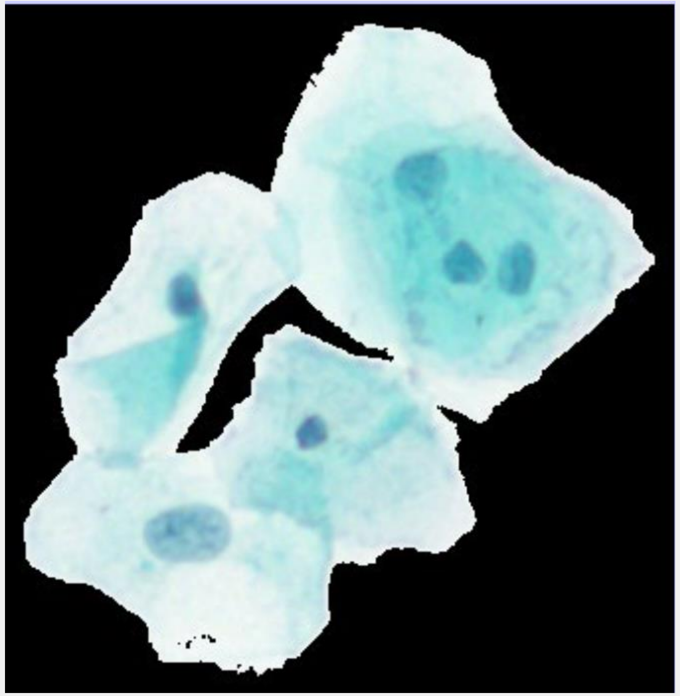
1. Randomly initialize *k* cluster centers
2. Assign each point to the closest center
3. Update cluster centers as the mean of the points
4. Repeat steps 2 and 3 until no data points are re-assigned

Free parameters:

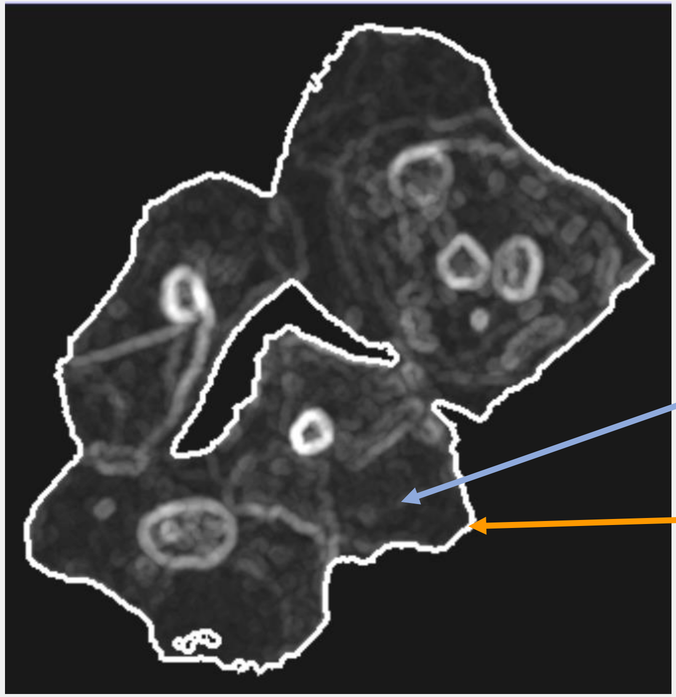
k – the number of clusters



Assignment 5: Watershed Segmentation



Input image



Gradient magnitude image

The **gradient magnitude image** can be interpreted as a **topographic surface** (3D relief), with valleys and mountains.

As **valley** we interpret larger regions with homogeneous intensity, whereas strong intensity changes can be seen as **mountains**.

Assignment 5: Task B

Task B: *Watershed Segmentation*

a. Load the provided image `inputEx4_2.jpg`, convert it to grayscale image and compute its **gradient magnitude.**

b. The starting flooding points, also known as *seeds* or *markers*, can be determined automatically or manually. To avoid oversegmentaion, you should either implement an interactive user selection for the **maker points** (`ginput`) or use the provided pre-selected points.

c. Implement the *watershed segmentation* method **by yourself**. Use the seeds selected in step **b**. as the starting points for region growing. It is recommended to apply a *4-neighbor topology* (introduced in lecture number 3).

d. Visualize the final segmentation result, as well as at least **two intermediate steps** during the region growing procedure. Apply an appropriate colormap to the segmented regions (`colormap`).

e. Shortly describe the benefits and drawbacks of the watershed segmentation method.

Watershed: Overview

Algorithm description:

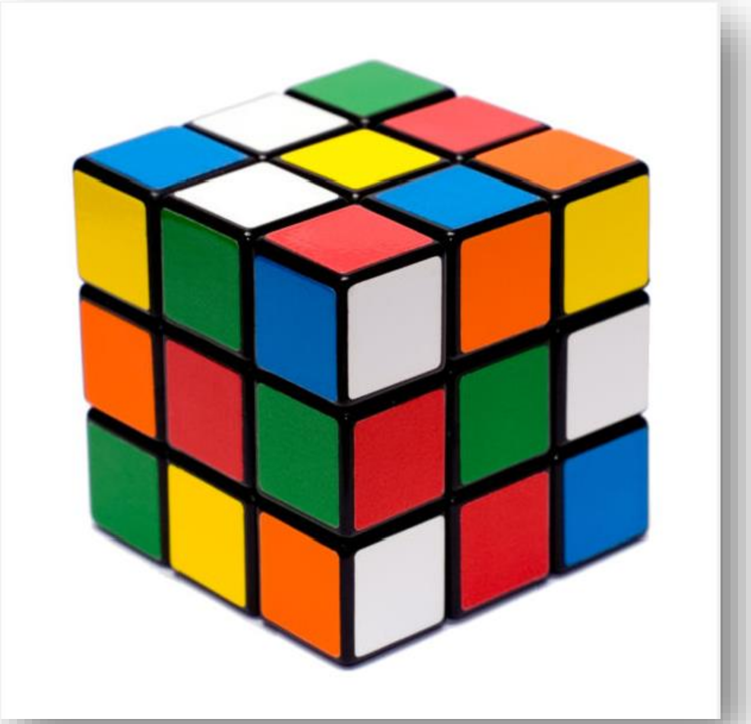
1. Select n seed points (each seed belongs to a different catchment basin)
2. Compute the gradient magnitude image G
3. Flood (grow) regions starting from every seed point

   until a valley ridge (mountain) is reached according to the

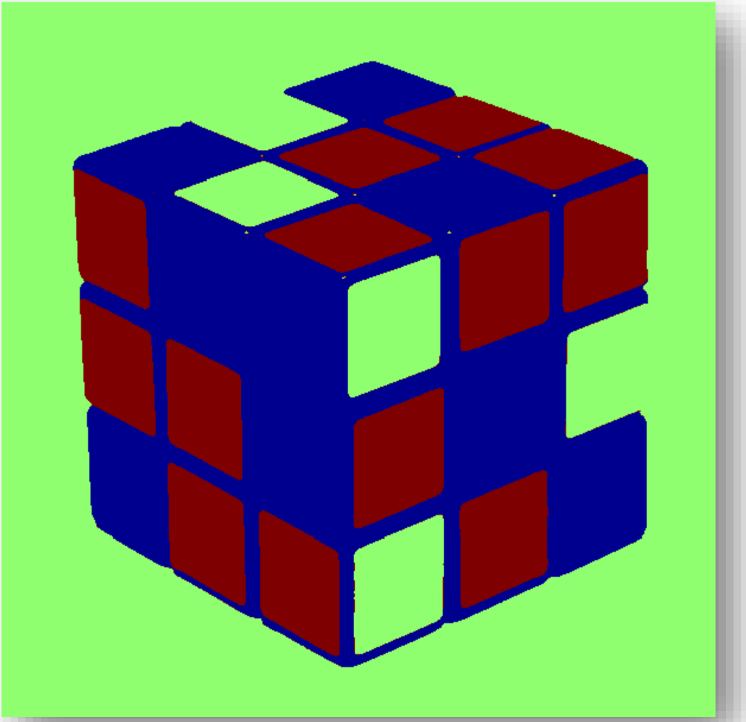
   gradient magnitude intensities of the neighboring pixels

Free parameters:
seed (marker) points – number and location

Assignment 5: Sample results - Task A



Input image



Segmented image:
3 clusters using k-means

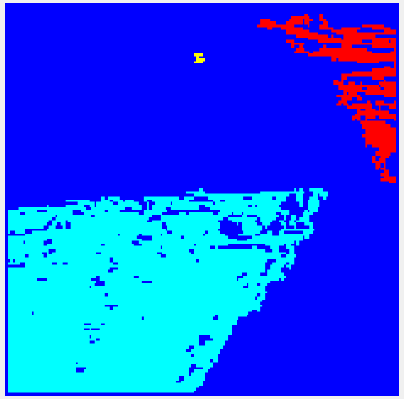
Assignment 5: Sample results - Task B

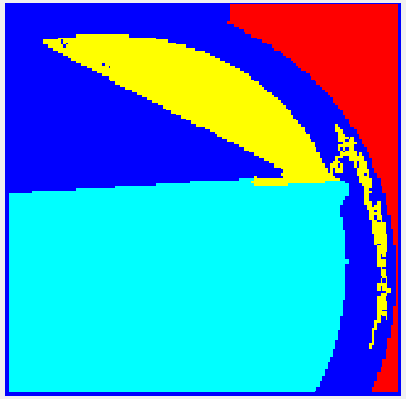
segmentation results at 33%

segmentation results at 67%

final segmentation

Input image with 3 markers





Input image with 6 markers