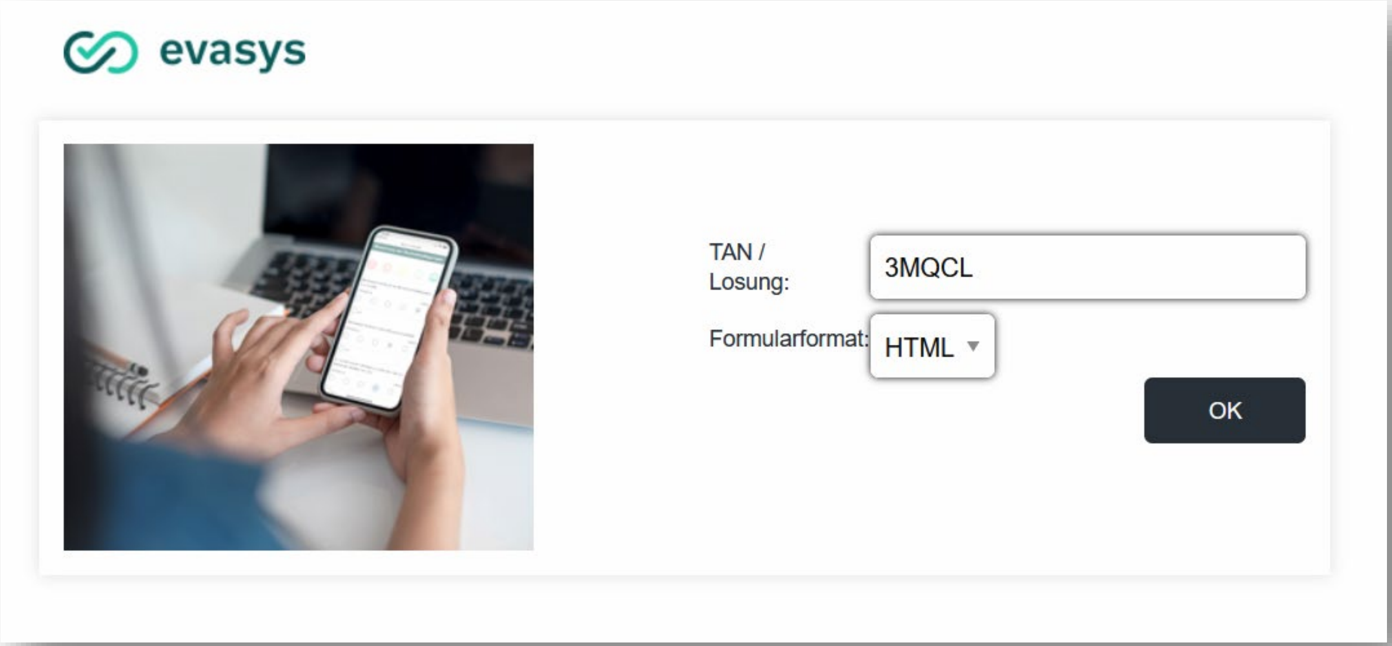
Online Course Evaluation

Teaching Evaluation:

*URL: **https://cloud14.evasys.de/uniweimar/online/***

*Code: **3MQCL***



Bauhaus-
Universität
Weimar

Image Analysis and Object Recognition

Exercise 5

Summer Semester 2025

(Course materials for internal use only!)

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

Agenda

	Topics:	Submission Dates:
Assignment 1.	Image enhancement, Binarization, Morphological operators	30.04.25
Assignment 2.	Gradient of Gaussian filtering, Förstner interest operator	21.05.25
Assignment 3.	Shape detection based on Hough-voting	04.06.25
Assignment 4.	Filtering in the frequency domain, Fourier descriptors	18.06.25
Assignment 5.	**Image segmentation and clustering**	**02.07.25**
Final Project.	**-** *Will be announced during the last exercise class* **-**	10.08.25

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)

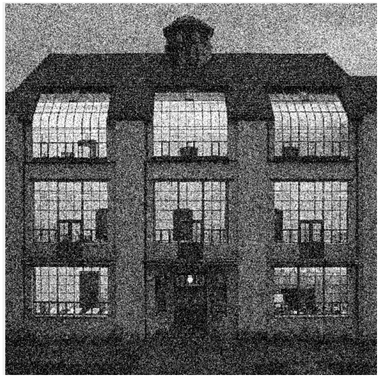
Input:

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

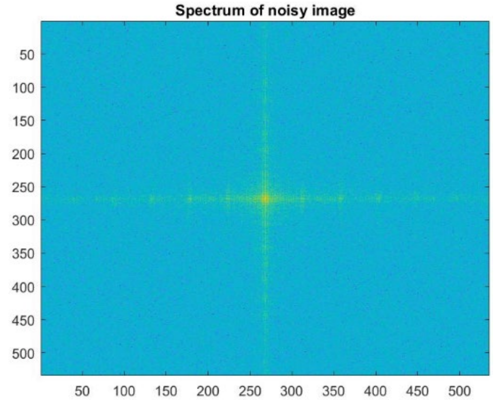
Task A

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

$f(x,y)$



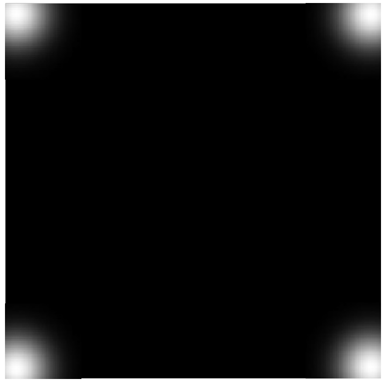
FFT $\Rightarrow$



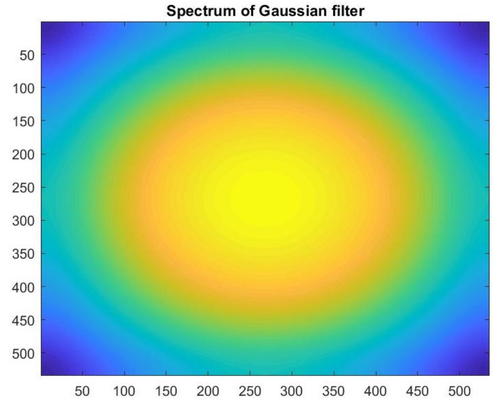
$F(u,v)$

1

$h(x,y)$



FFT $\Rightarrow$



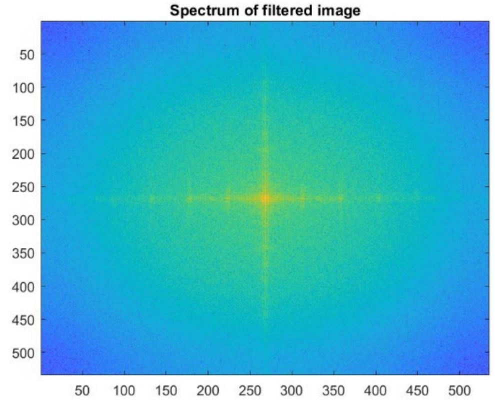
$.*$ 2

$H(u,v)$

$\Downarrow$

$g(x,y)$

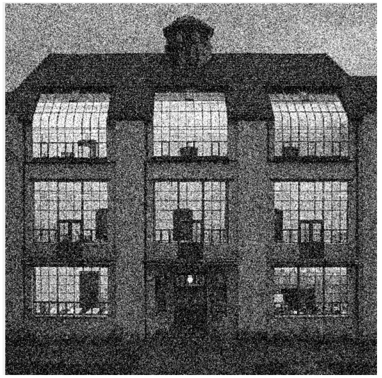
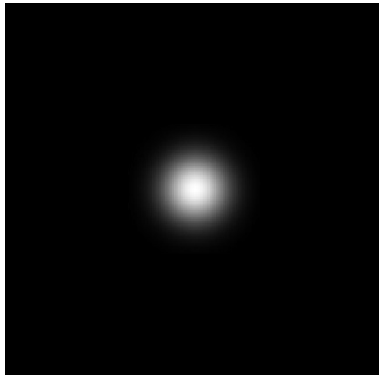
FFT^{-1} $\Leftarrow$ 3



$G(u,v)$

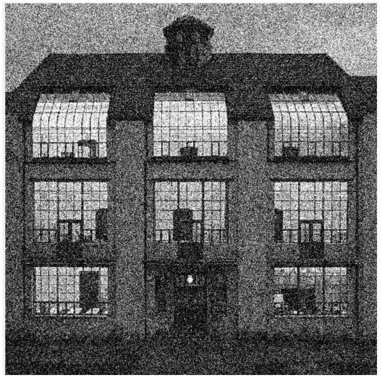
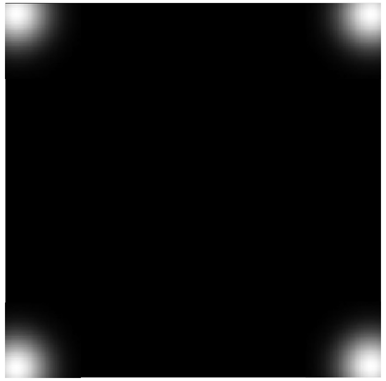
Task A

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**

Bauhaus-Universität Weimar

Task A

```python
def main():
    sigma = 1.4
    img = np.array(Image.open('taskA.png').convert('L')).astype(np.float64) / 255.0
    noisy = add_gaussian_noise(img, 0, 0.01) # Add Gaussian noise

    # Compute 2D Gaussian kernel
    kernel_1d = gauss1d(sigma)
    kernel_2d = np.outer(kernel_1d, kernel_1d)

    # Create padded Gaussian filter
    filter_padded = np.zeros(noisy.shape)
    k_height, k_width = kernel_2d.shape
    filter_padded[:k_height, :k_width] = kernel_2d
    shift_y, shift_x = -np.floor_divide(np.array(kernel_2d.shape), 2) # Center the filter
    filter_padded = np.roll(filter_padded, (int(shift_y), int(shift_x)),axis=(0, 1))

    # Filter in frequency domain
    noisy_fft = np.fft.fft2(noisy)
    filter_fft = np.fft.fft2(filter_padded)
    result_fft = noisy_fft * filter_fft
    filtered = np.real(np.fft.ifft2(result_fft))

    diplay_results(img, noisy, noisy_fft, filter_fft, result_fft, filtered)

def add_gaussian_noise(image, mean=0, var=0.01):
    """Add Gaussian noise to an image"""
    noise = np.random.normal(mean, np.sqrt(var), image.shape)
    noisy = image + noise
    noisy = np.clip(noisy, 0, 1)
    return noisy

def gauss1d(sigma):
    r = round(3 * sigma)
    x = np.arange(-r, r + 1)
    g = np.exp(-x**2 / (2 * sigma**2)) / (sigma * np.sqrt(2 * np.pi))
    return g
```

Bauhaus-
Universität
Weimar

Task A

```python
def diplay_results(img, noisy, noisy_fft, filter_fft, result_fft, filtered):
    plt.figure(figsize=(12, 8))

    plt.subplot(2, 3, 1)
    plt.imshow(img, cmap='gray')
    plt.title('Original image')
    plt.axis('off')

    plt.subplot(2, 3, 2)
    plt.imshow(noisy, cmap='gray')
    plt.title('Noisy image')
    plt.axis('off')

    plt.subplot(2, 3, 3)
    plt.imshow(np.log(np.abs(np.fft.fftshift(noisy_fft)) + 1), cmap='viridis')
    plt.title('Spectrum of noisy image')
    plt.axis('off')

    plt.subplot(2, 3, 4)
    plt.imshow(np.log(np.abs(np.fft.fftshift(filter_fft)) + 1), cmap='viridis')
    plt.title('Spectrum of Gaussian filter')
    plt.axis('off')

    plt.subplot(2, 3, 5)
    plt.imshow(np.log(np.abs(np.fft.fftshift(result_fft)) + 1), cmap='viridis')
    plt.title('Spectrum of filtered image')
    plt.axis('off')

    plt.subplot(2, 3, 6)
    plt.imshow(filtered, cmap='gray')
    plt.title('Filtered image')
    plt.axis('off')

    plt.tight_layout()
    plt.savefig('fft_filtering_results.png', dpi=300)
    plt.show()
```

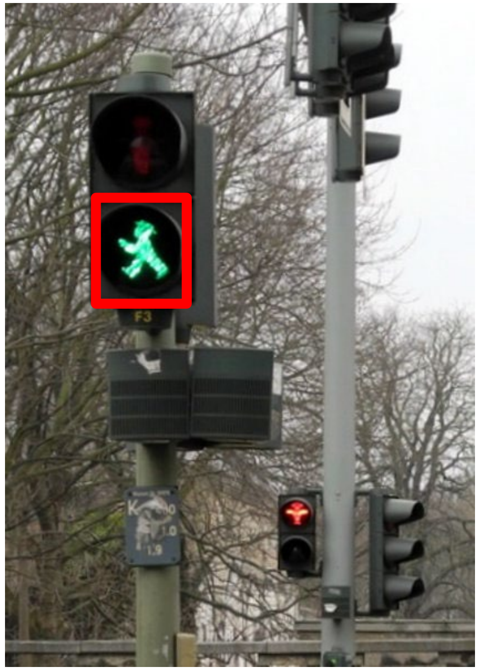
Input data

Task B

training image

test image 1

test image 2



test image 3

Bauhaus-Universität Weimar

Task B

```python
def main():

    train_image_path = 'trainB.png'
    if os.path.exists(train_image_path):
        train_image = np.array(Image.open(train_image_path).convert('L')).astype(np.float64) / 255.0
        train_mask = thresholding(train_image)
        train_descriptors, _ = fourier_descriptors(train_mask)                    # Compute training Fourier descriptor
        model_descriptor = train_descriptors[0]                                   # Expect sigle training descriptor
        test_images = ['test1B.jpg', 'test2B.jpg', 'test3B.jpg']

        for test_image_path in test_images:                                        # Process each test image
            if os.path.exists(test_image_path):
                # Read and preprocess test image
                test_image = np.array(Image.open(test_image_path).convert('L')).astype(np.float64) / 255.0
                test_mask = thresholding(test_image)

                # Compute Fourier descriptors for test image
                test_descriptors, test_boundaries = fourier_descriptors(test_mask)

                plt.figure(figsize=(10, 8)); plt.imshow(test_mask, cmap='gray')
                plt.title(f'Objects detected in {test_image_path}')

                # Compare descriptors and visualize matches
                for i, descriptor in enumerate(test_descriptors):
                    distance = np.linalg.norm(model_descriptor - descriptor)      # Distance between descriptors
                    if distance < 0.075:                                          # Threshold for matching
                        boundary = test_boundaries[i]                             # Extract boundary points
                        plt.plot(boundary[:, 1], boundary[:, 0], 'r', linewidth=2)# Plot boundary

                plt.axis('off'); plt.tight_layout(); plt.show()
            else:
                print(f"Test image {test_image_path} not found.")
    else:
        print(f"Training image {train_image_path} not found.")
```

Task B

```python
def thresholding(image):
    threshold = filters.threshold_otsu(image)  # Calculate threshold using Otsu's method
    mask = image > threshold                   # Apply threshold to create binary mask
    return mask


def fourier_descriptors(binary_mask, n=25):

    boundaries = measure.find_contours(binary_mask, 0.5)              # Find boundaries in the binary mask
    fd = np.zeros((len(boundaries), n-1))                               # Initialize descriptors array

    for i, boundary in enumerate(boundaries):                           # Iterate over each boundary
        if len(boundary) > n:
            complex_boundary = boundary[:, 1] + 1j * boundary[:, 0]     # Boundary points to complex numbers
            fourier_result = np.fft.fft(complex_boundary)               # Compute Fourier transform
            fd[i, :] = np.abs(fourier_result[1:n] / fourier_result[1])  # Normalize the descriptor

    return fd, boundaries
```

Translation $D_f(1) \coloneqq 0$

Scale $D_f \coloneqq \frac{D_f}{|D_f(2)|}$

Orientation $D_f \coloneqq |D_f|$

Task B

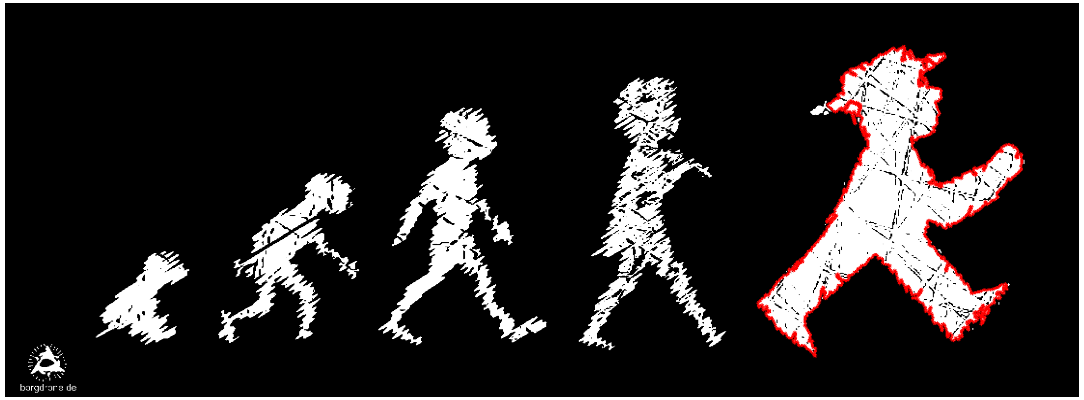
Closed Shape Boundaries

Bauhaus-Universität Weimar

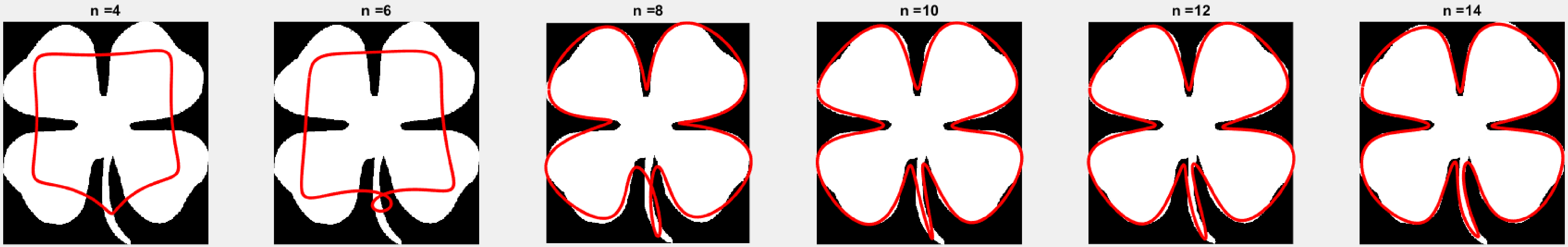
Expected results

Task B

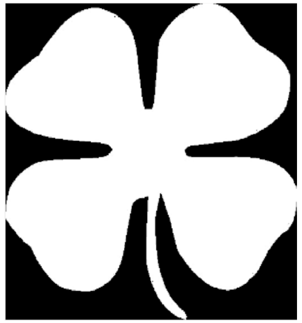
training image



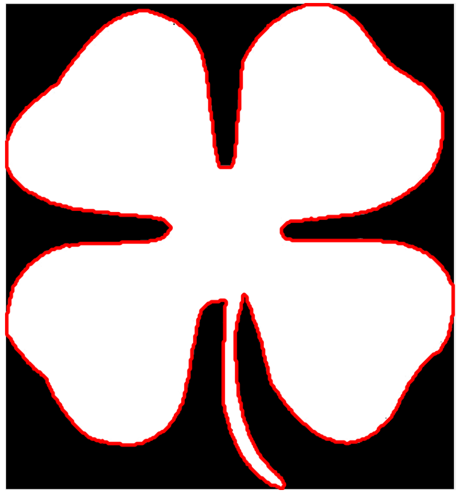
Discussion: Visualization of the simplified shape boundary



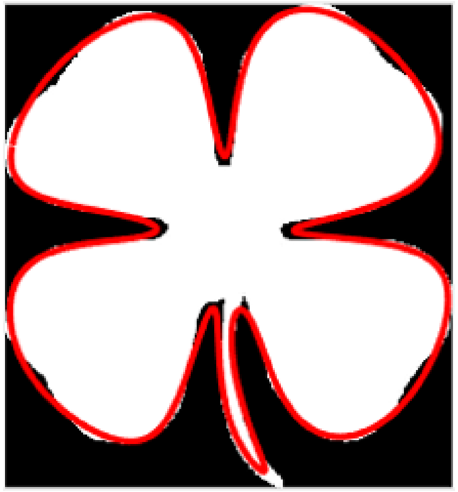
Discussion: Visualization of the simplified shape boundary



Binary Input Image

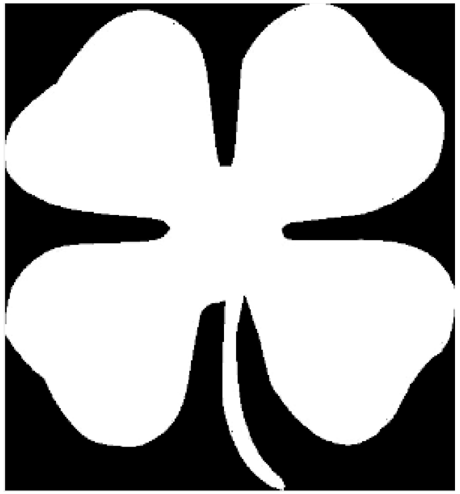


Binary Input Image
+
Complete Boundary Overlay



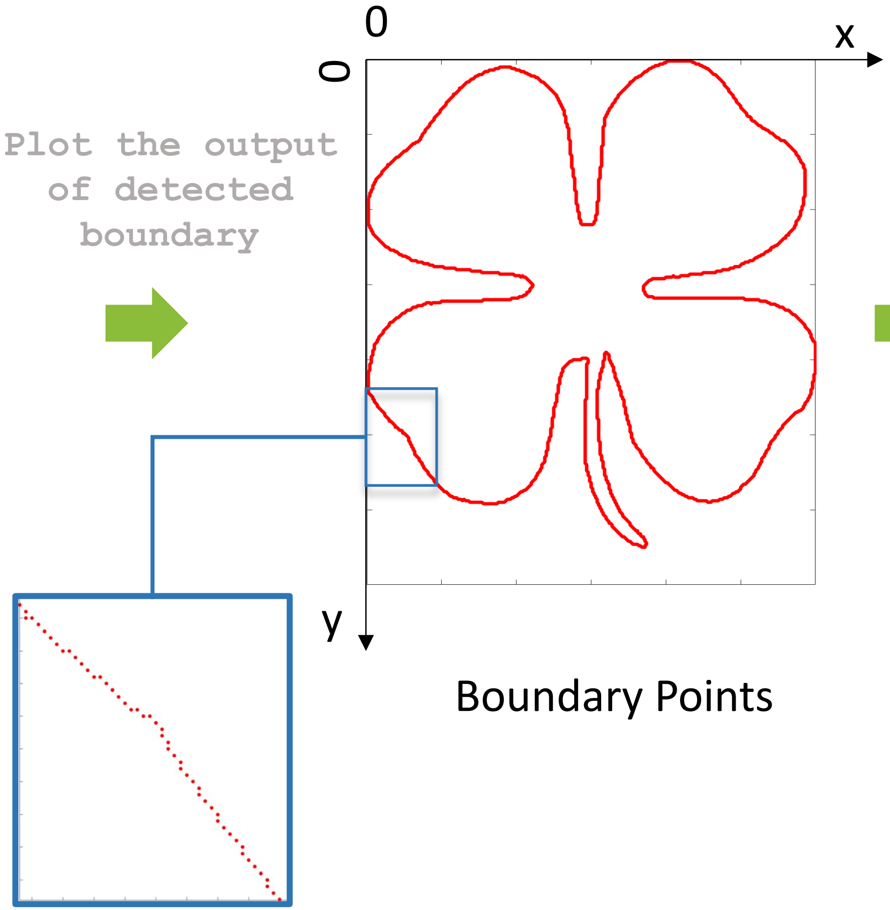
Binary Input Image
+
Simplified Boundary Overlay

Discussion: Visualization of the simplified shape boundary



Binary Input Image

Plot the output of detected boundary



Boundary Points

Close-Up of a Boundary Segment

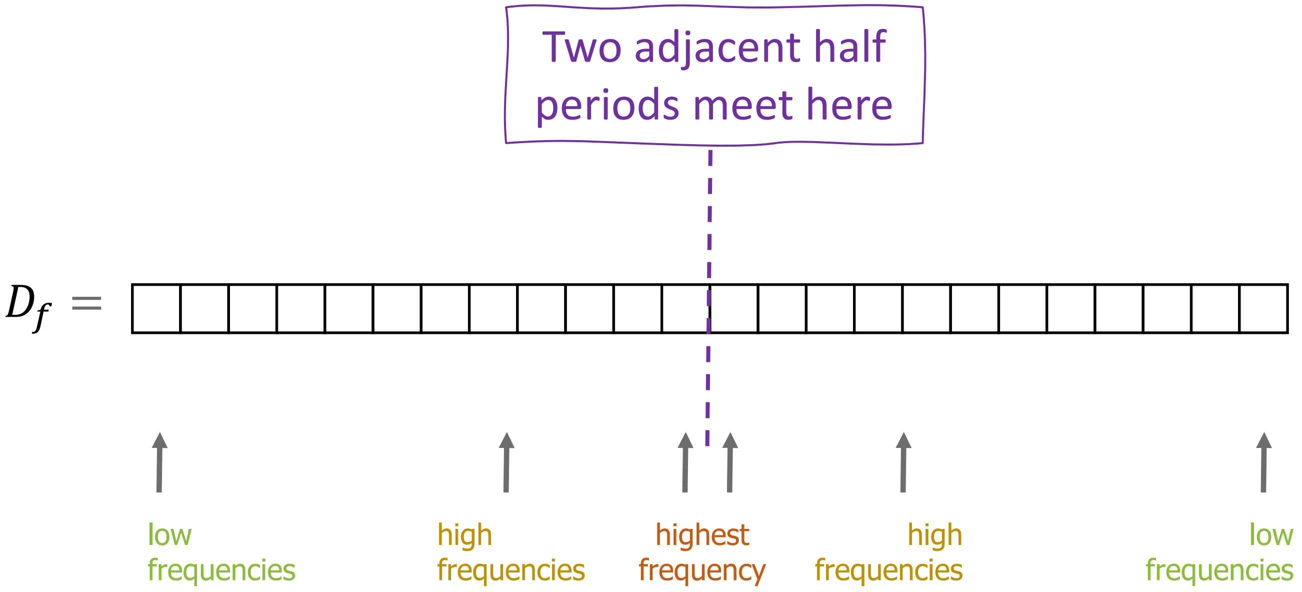
$$D = x + j * y$$

Complex-valued vector D

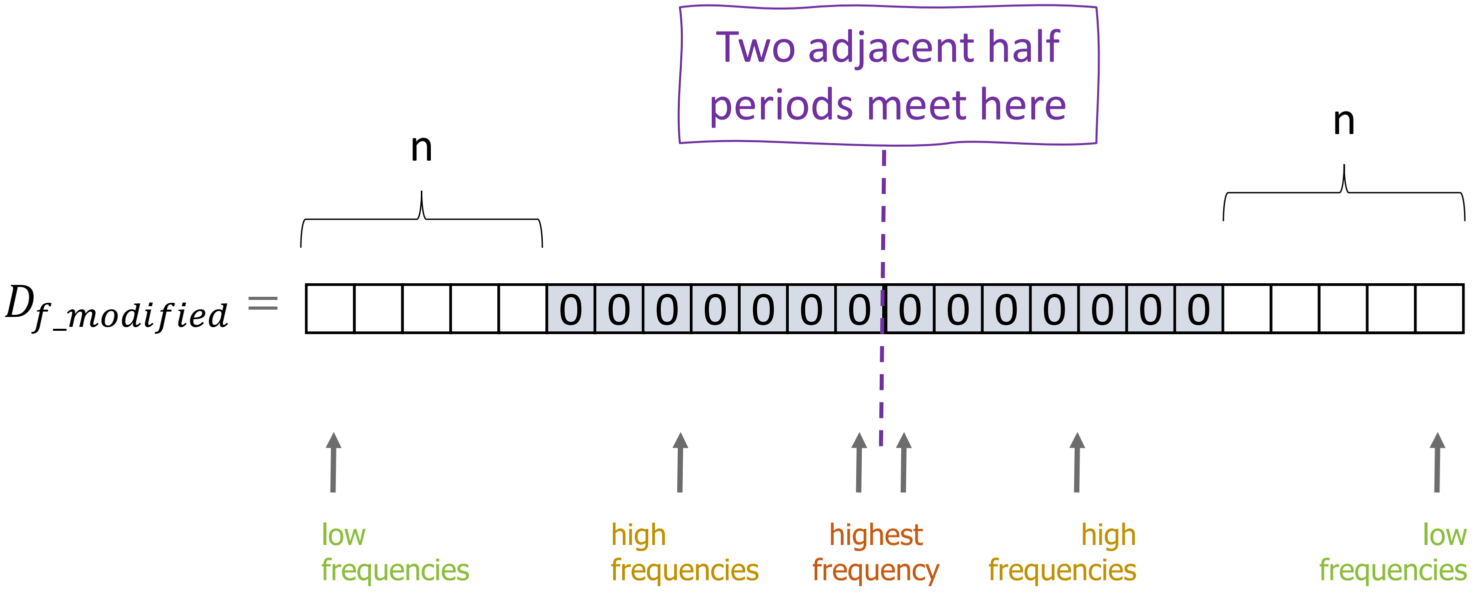
$$D_f = FFT(D)$$

Fourier Descriptor D_f

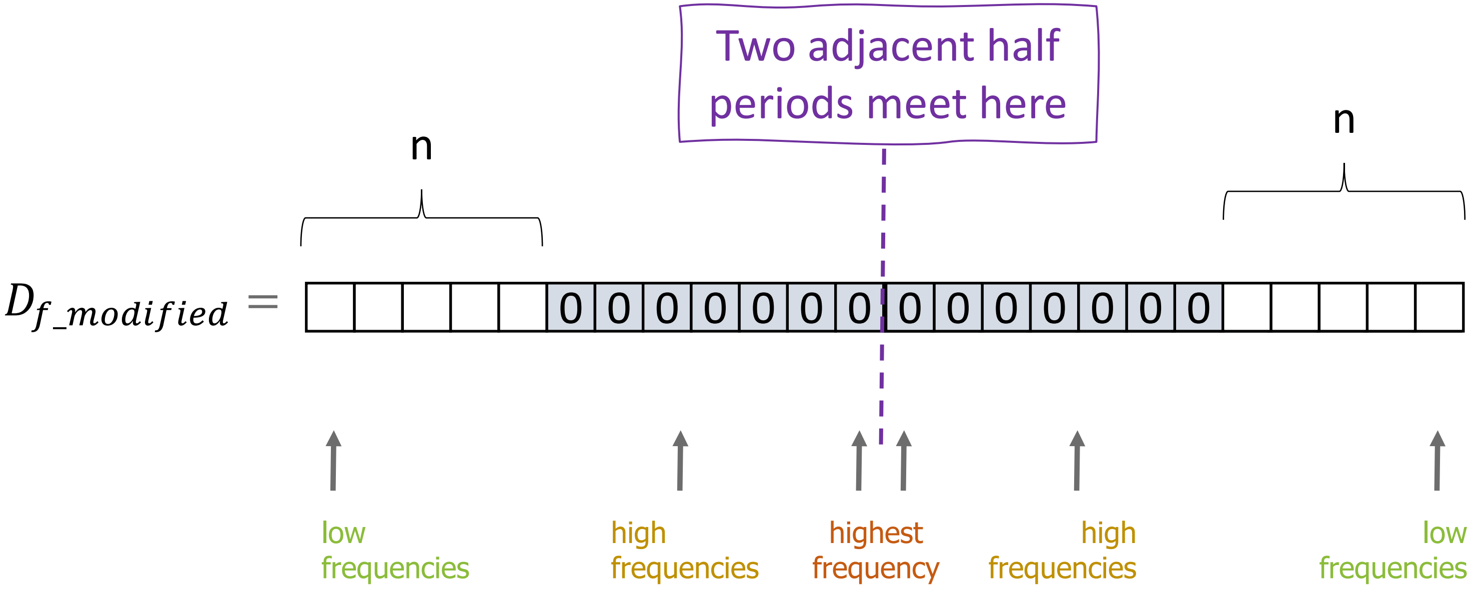
Discussion: Visualization of the simplified shape boundary



Discussion: Visualization of the simplified shape boundary



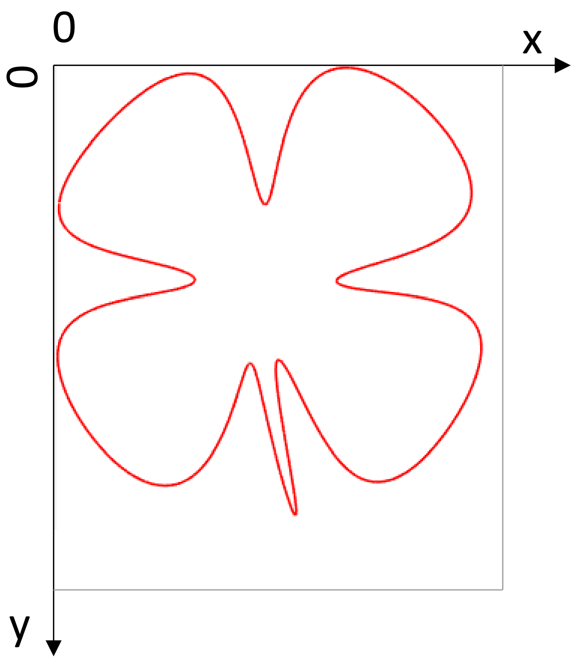
Discussion: Visualization of the simplified shape boundary



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

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

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

$\leftarrow$ $D_{simplified}$ $\xleftarrow{\text{Inverse FFT}}$ $D_{f\_modified}$

Assignment 5

Assignment 5: Overview

Topics:

- *k-means* clustering
- Watershed segmentation

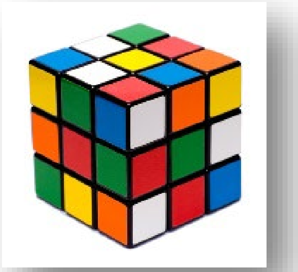
Only one of the two subtasks is compulsory. You may choose either of them.

Goal:

- Practice unsupervised image segmentation

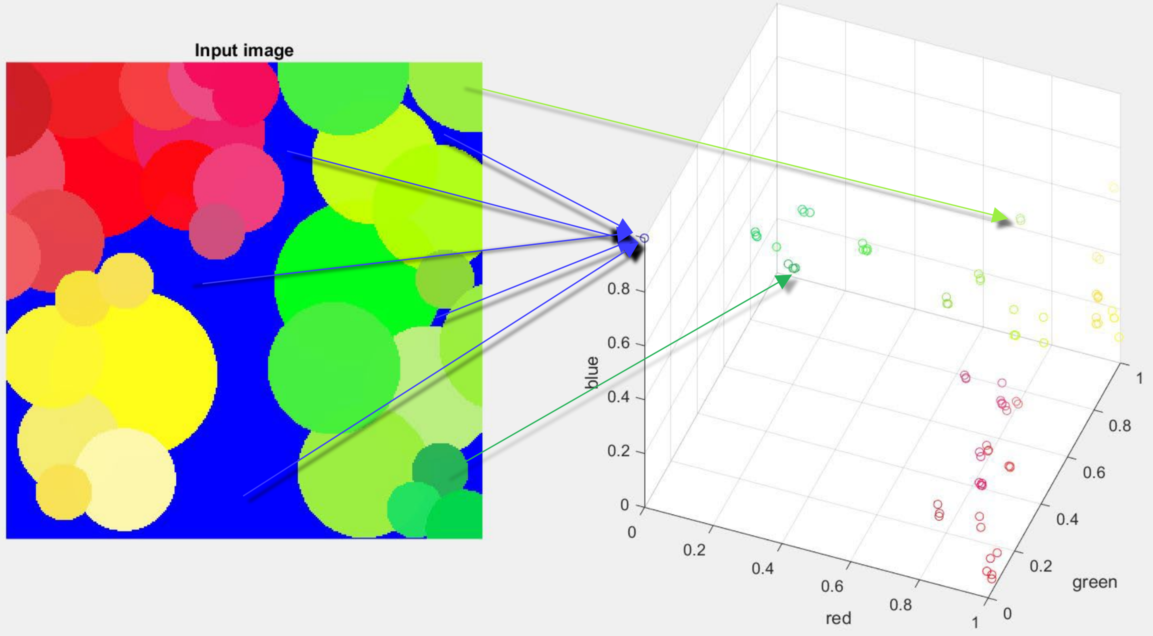
Input:

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



Bauhaus-Universität Weimar

Assignment 5:Feature Space



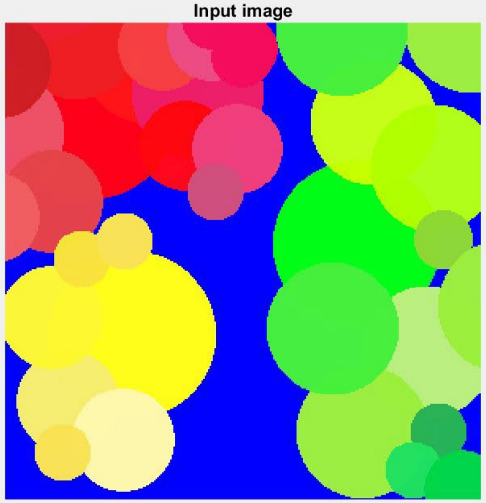
Artificially generated image

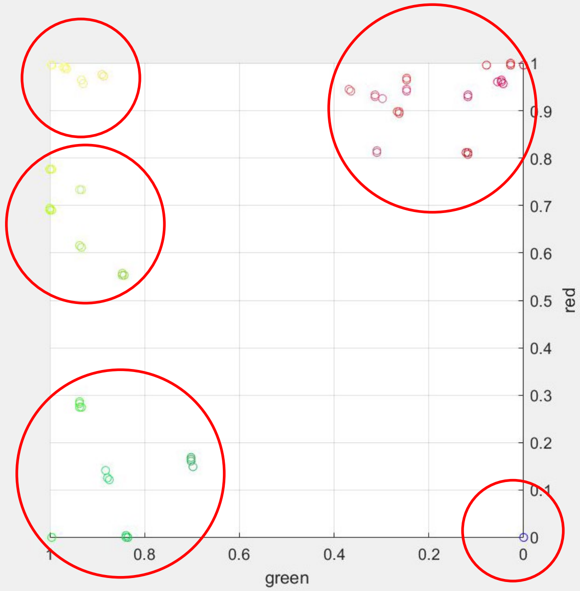
Given: 3-channel color image

- 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**

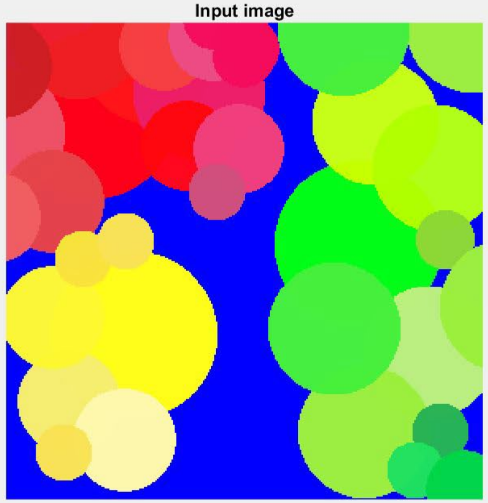
Assignment 5: Clustering

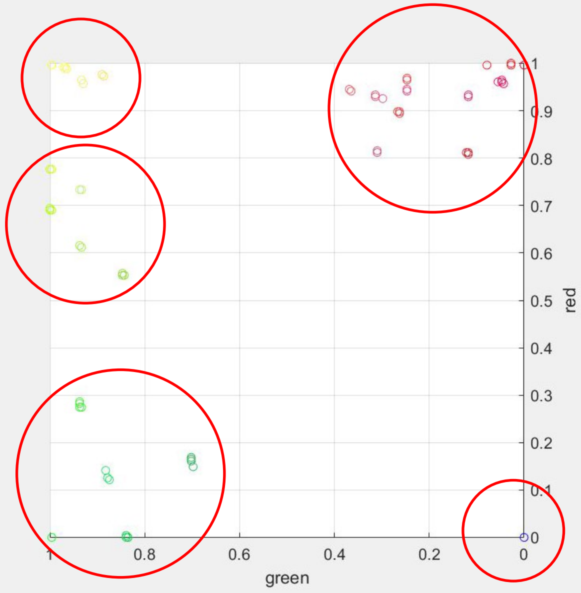


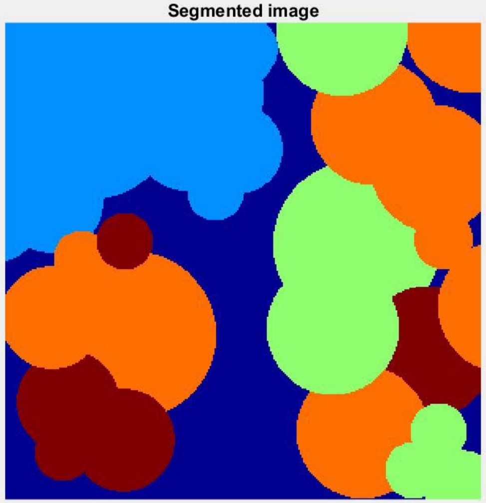


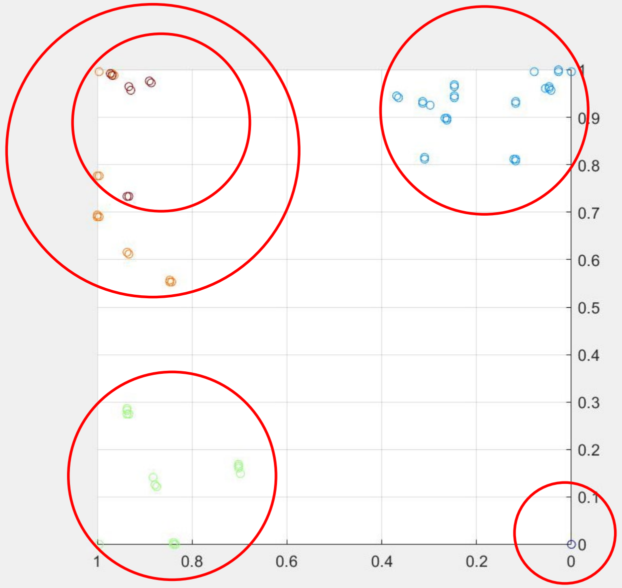
Artificially generated image

Assignment 5: Clustering Results



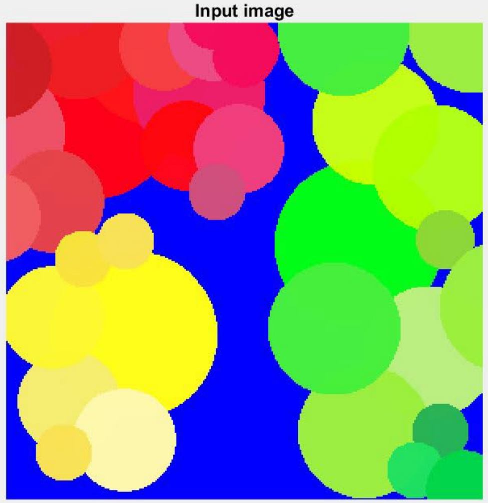


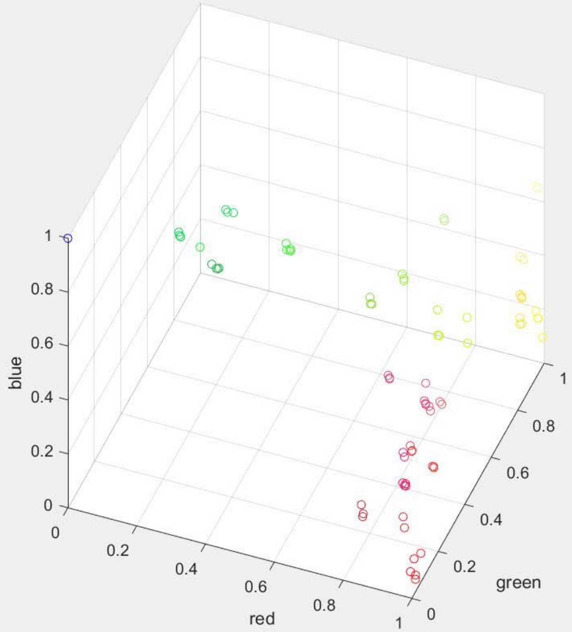


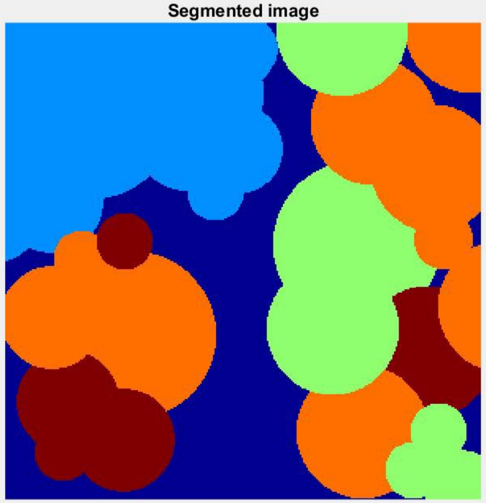


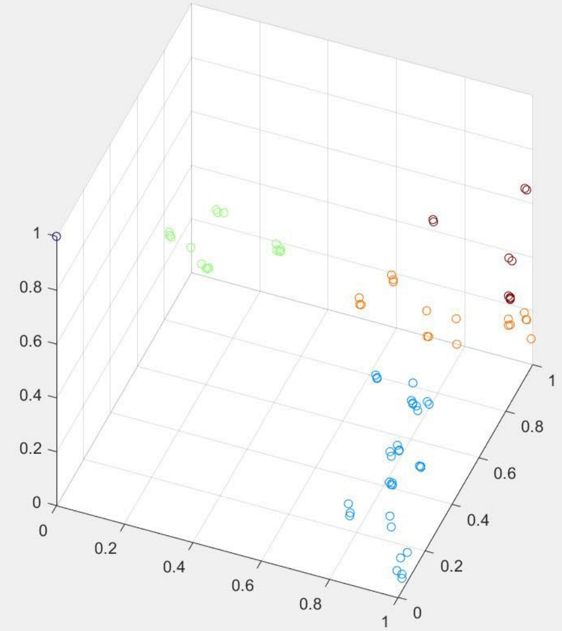
Artificially generated image

Assignment 5: Clustering Results





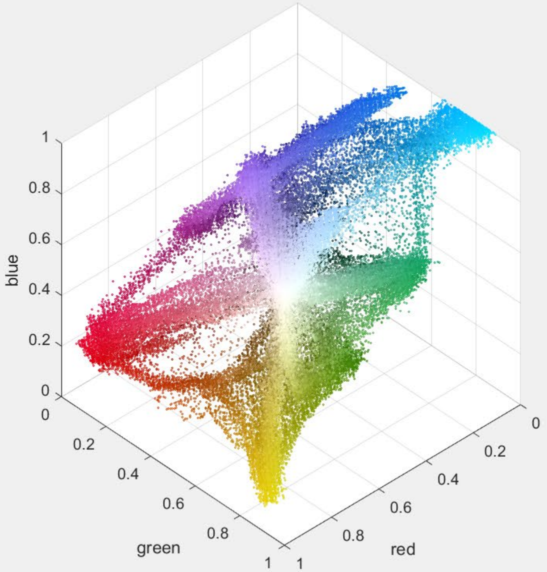




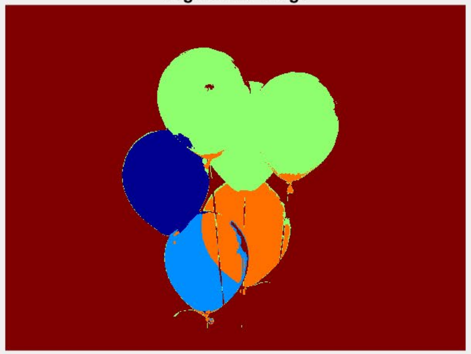
Artificially generated image

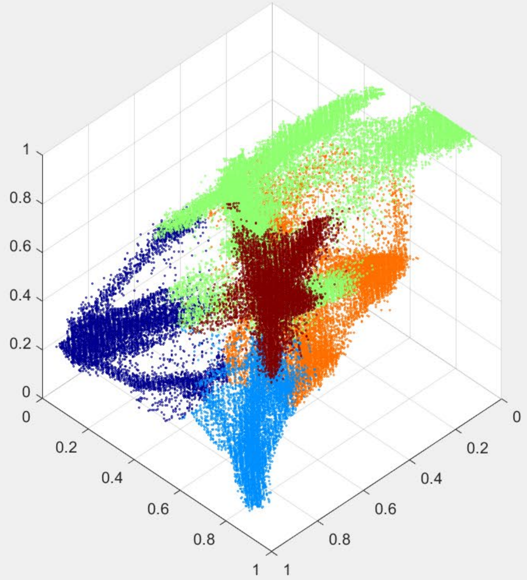
Assignment 5: Clustering Results

Input image



Segmented image





Real photo example

Input image reference: https://images.app.goo.gl/9QN27cztfuQ3pA2R9

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.

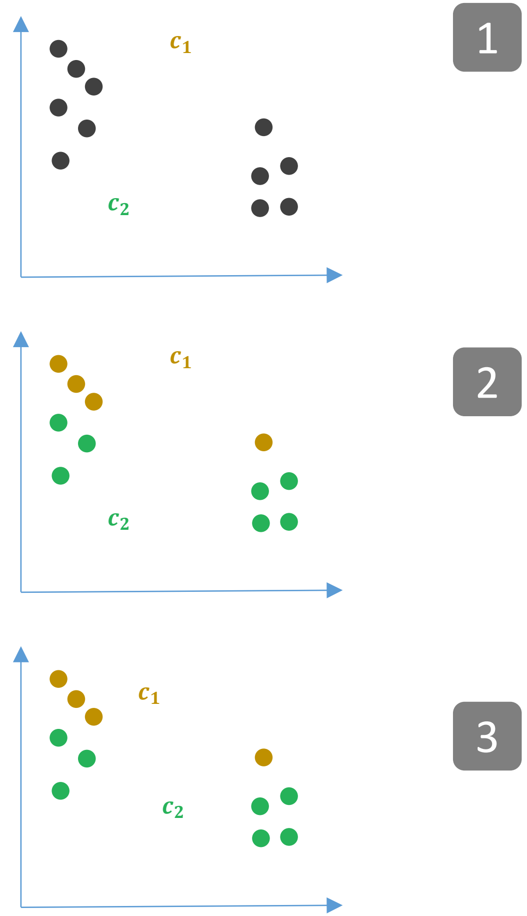
c. Select an appropriate number of clusters *k*, apply the algorithm and visualize the detected groups in feature and image space (e.g. with color coding: `colormap`).

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:

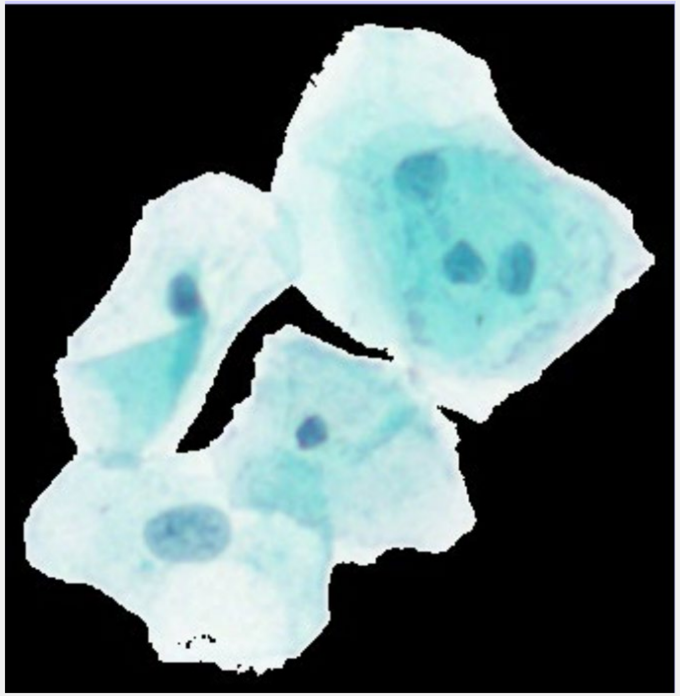
1. Randomly initialize *k* cluster centers
2. Assign each point to the closest center
3. Update cluster centers as the mean of the points
4. Repeat steps 2 and 3 until no data points are re-assigned



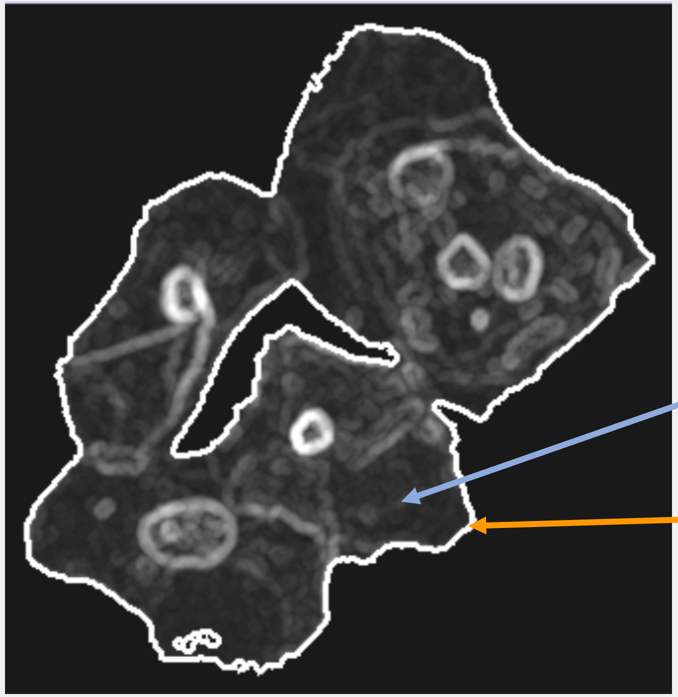
Free parameters:

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**.

Bauhaus-Universität Weimar

Assignment 5: Task B

Task B: *Watershed Segmentation*

a. Load the provided image `inputEx5_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).

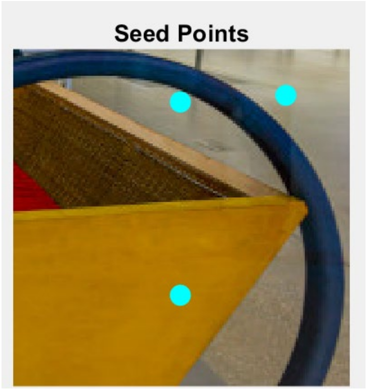
d. Visualize the final segmentation result, as well as at least **two intermediate steps** during the region growing procedure. Apply an appropriate colormap to the segmented regions (`colormap`).

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

Watershed: Overview

Algorithm description:

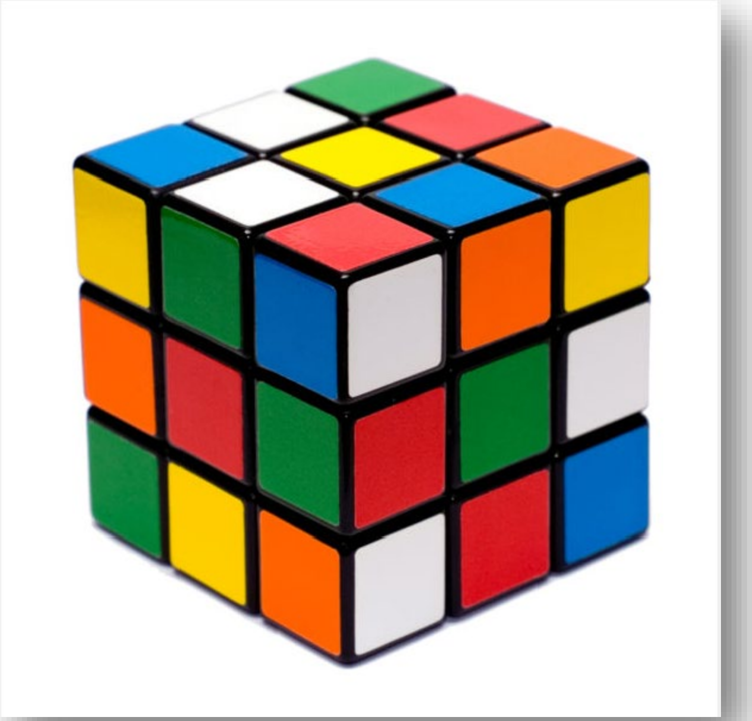
1. Select n seed points (each seed belongs to a different catchment basin)
2. Compute the gradient magnitude image G
3. Flood (grow) regions starting from every local minima found in the vicinity the seed points.
4. Build watersheds along the strongest gradient between neighboring catchment basins.



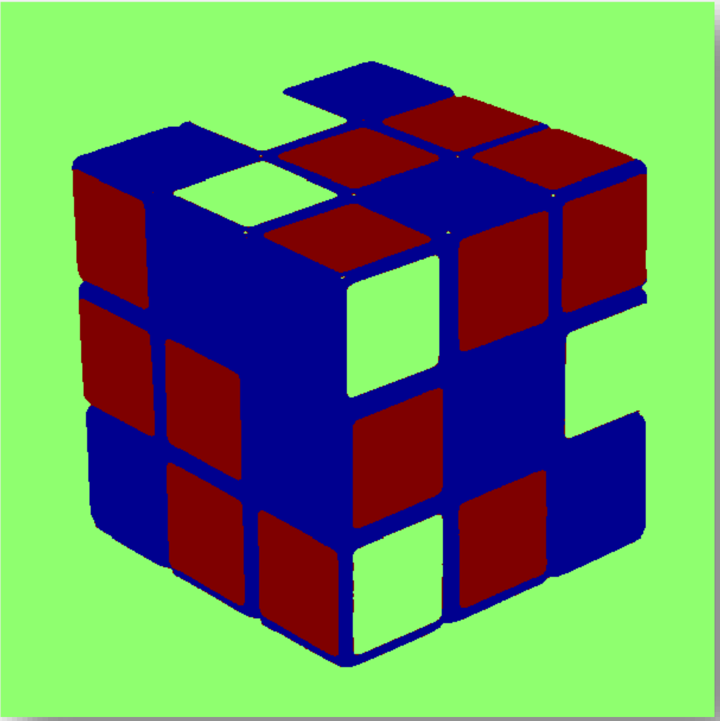
Free parameters:

seed (marker) points – number and location

Assignment 5: Sample results - Task A



Input image

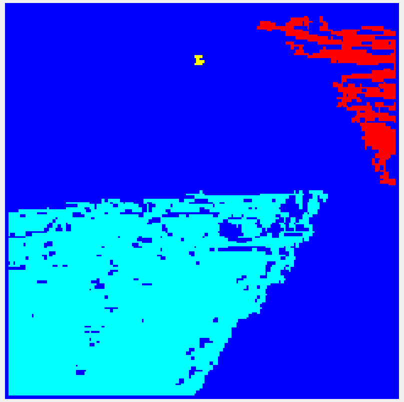
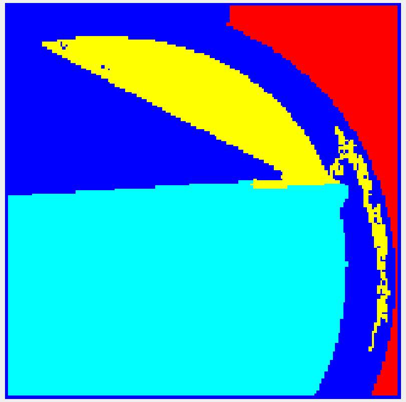


Segmented image:
3 clusters using k-means

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