DIT281 Project 1
(itp22104@hua.gr / Anastasios Kotronis)

Γενικές βοηθητικές συναρτήσεις

In [1]:
```
%matplotlib notebook

from pathlib import Path

import cv2
import math
import matplotlib.pyplot as plt
import numpy as np
```

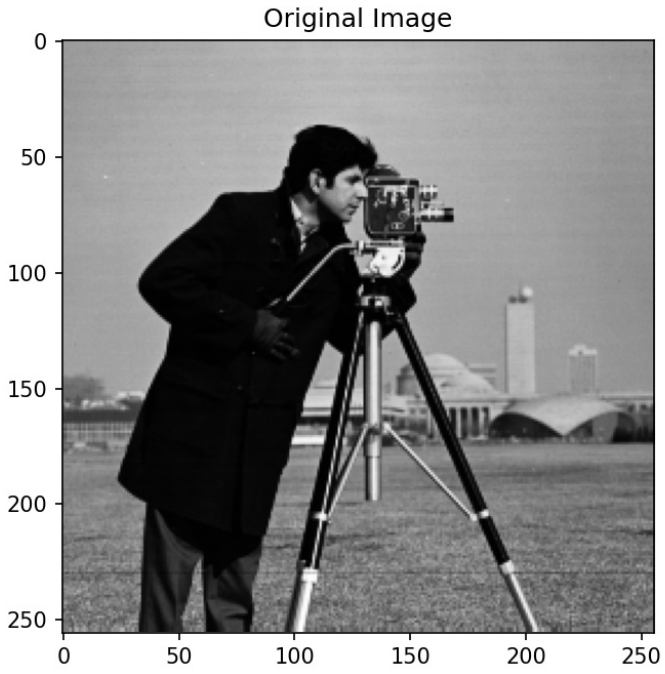
In [142…
```
def load_image_as_gray(image_name, resize_to=None):
    """
    `resize_to` can be a 2-tuple with the new resolution
    """
    image_path = Path.cwd().resolve() / 'images' / image_name
    image = cv2.imread(f'{image_path}', cv2.IMREAD_GRAYSCALE)
    if resize_to:
        image = cv2.resize(image, resize_to)
    return image


def filter_image(image, percentage=1., to_keep_per_dim=None):
    """
    Sample image
    - By a fixed percentage for each dimension or
    - By a tuple of fixed numbers for each dimension
    """
    if to_keep_per_dim is None:
        rows, cols = image.shape
        # based on percentage
        x, y = int(rows * percentage), int(cols * percentage)
        to_keep_per_dim = (x, y)
    # based on fixed number per dim
    x, y = [k // l for k,l in zip(image.shape, to_keep_per_dim)]

    # Create a mask to identify positions
    mask = np.zeros_like(image, dtype=bool)
    mask[::x, ::y] = True

    # Initialize output array with zeros
    output = np.zeros_like(image)

    # Assign the initial numbers at specified positions
    output[mask] = image[mask]
    return output


def calculate_mse(first_image, last_image):
    return np.mean((first_image - last_image) ** 2)
```

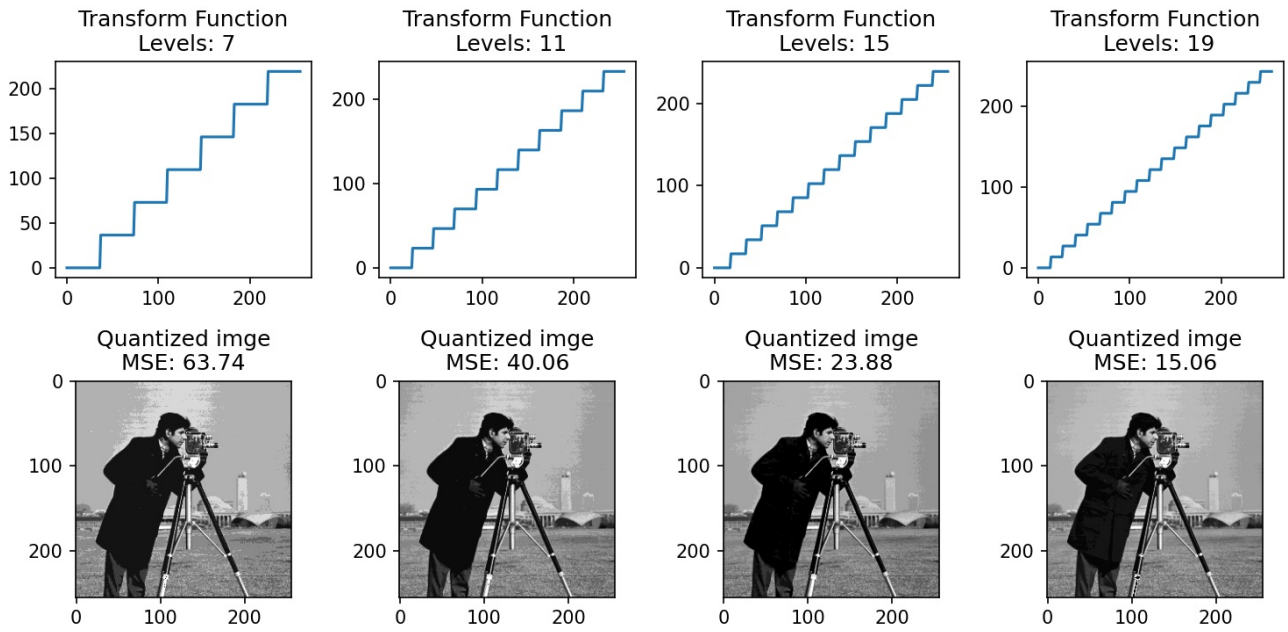
Άσκηση #1: Κβάντιση

In [93]:
```
def uniform_quantizer(input_image, levels):
    # Calculate step from original_image shape
    step = input_image.shape[-1] // levels

    # Create a placeholder image with zeros
    quantized_image = np.zeros_like(input_image)

    # Uniform quantization
    for i in range(levels):
        from_, to_ = i * step, (i + 1) * step
        mask = (input_image >= from_) & (input_image < to_)
        quantized_image[mask] = (from_ + to_) // 2
    return quantized_image


def quantizer_coords(levels):
    x = np.arange(0, 256)
    step_size = 256 / levels
    y = np.floor(x / step_size) * step_size
    return x, y
```

In [98]:
```
# Load grayscale image
```

```
original_image = load_image_as_gray('cameraman.bmp')

# Display original image
plt.figure(figsize=(5, 5))
plt.subplot(1, 1, 1)
plt.imshow(original_image, cmap='gray')
plt.title('Original Image')


# # A. Implement and apply uniform quantizers with 7, 11, 15, and 19 levels
quantizer_levels = [7, 11, 15, 19]
levels_cnt = len(quantizer_levels)

grid_x_size = 10
plt.figure(figsize=(grid_x_size, grid_x_size // 2))
for i, levels in enumerate(quantizer_levels):
    quantized_image = uniform_quantizer(original_image, levels)

    # B. Plot quantizer transform function
    x, y = quantizer_coords(levels)
    plt.subplot(2, levels_cnt, i + 1), plt.plot(x, y, label=f'Levels = {levels}')
    plt.title(f'Transform Function\n Levels: {levels}')

    mse = calculate_mse(original_image, quantized_image)

    # C. Display quantized images
    plt.subplot(2, levels_cnt, levels_cnt + i + 1), plt.imshow(quantized_image, cmap='gray')
    plt.title(f'Quantized imge\nMSE: {mse:.2f}')

# Display the plots
plt.tight_layout()
plt.show()
```





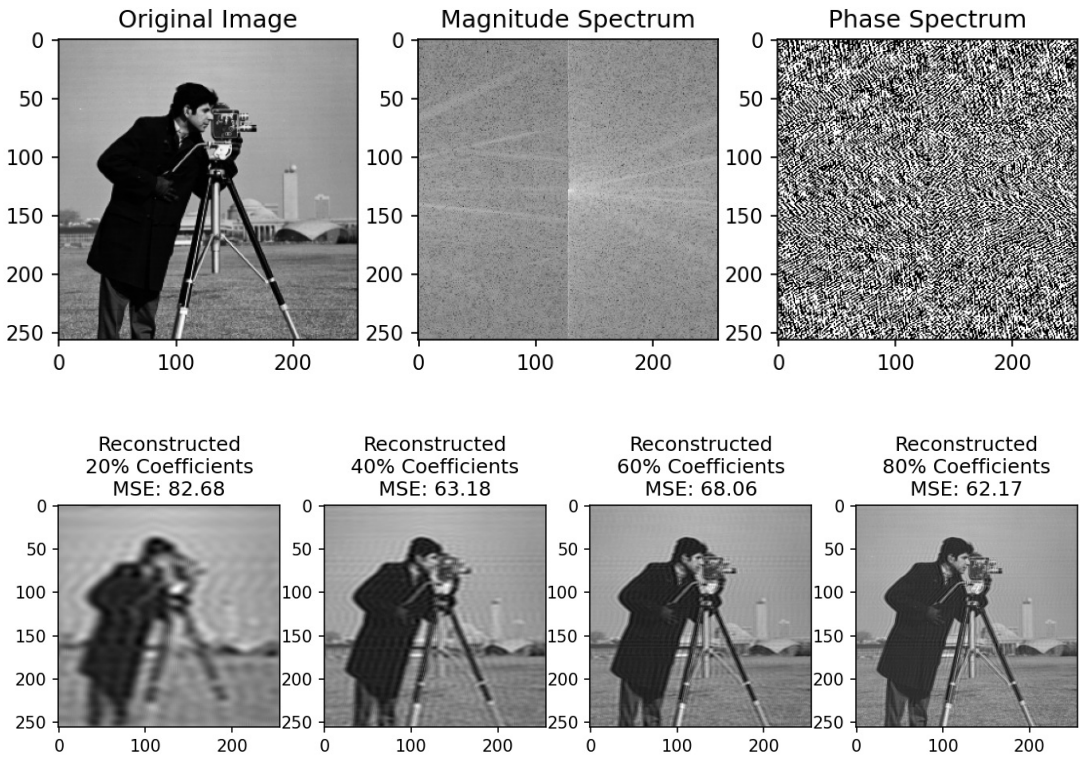
Παρατηρούμε ότι με την αύξηση των επιπέδων του κβαντιστή η ποιότητα της εικόνας βελτιώνεται και πλησιάζει την αρχική κάτι που συμβαδίζει και με τη μείωση του σφάλματος

Άσκηση #2: Μετασχηματισμός Fourier

In [ ]:
```
%matplotlib notebook
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [212…
```
# Load image
original_image = load_image_as_gray('cameraman.bmp')

# Perform 2D Fourier Transform (DFT)
dft = cv2.dft(np.float32(original_image))

# Shift zero frequency component (Direct Current - DC) from top left to the center
dft_shift = np.fft.fftshift(dft)

# Apply log function to highlighting details across
# wide magnitude ranges and make magnitude lighter
magnitude_spectrum = np.log(np.abs(dft_shift))

# Calculate phase spectrum in radians
phase_spectrum = np.angle(dft_shift)

# Display original image, magnitude spectrum, and phase spectrum
grid_x_size = 9
plt.figure(figsize=(grid_x_size, grid_x_size // 3))
plt.subplot(131), plt.imshow(original_image, cmap='gray')
plt.title('Original Image')
plt.subplot(132), plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('Magnitude Spectrum')
plt.subplot(133), plt.imshow(phase_spectrum, cmap='gray')
```

```
plt.title('Phase Spectrum')
plt.show()

# Reconstruct images with 20%, 40%, 60%, and 80% coefficients
percentages = [0.2, 0.4, 0.6, 0.8]
percentages_cnt = len(percentages)

grid_x_size = 11
plt.figure(figsize=(grid_x_size, grid_x_size // 4 + 1))

for i, percentage in enumerate(percentages):
    # Apply the mask to the shifted DFT
    dft_shift_filtered = filter_image(dft_shift, percentage)
    # Shift zero frequency component (DC) back fom the center to top left
    dft_shift_filtered_back = np.fft.ifftshift(dft_shift_filtered)
    # Perform the inverse Fourier Transform
    reconstructed_image = cv2.idft(dft_shift_filtered_back)

    # Normalize reconstructed image
    reconstructed_image = cv2.normalize(
        reconstructed_image, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U
    )

    mse = calculate_mse(original_image, reconstructed_image)

    plt.subplot(1, percentages_cnt, i + 1), plt.imshow(reconstructed_image, cmap='gray')
    plt.title(f'Reconstructed\n{int(percentage * 100)}% Coefficients\nMSE: {mse:.2f}')

plt.show()
```



Παρατηρούμε ότι με την αύξηση του ποσοστού των συντελεστών η ποιότητα της ανακατασκευασμένης εικόνας βελτιώνεται και πλησιάζει την αρχική κάτι που συμβαδίζει και με τη μείωση του σφάλματος

Άσκηση #3: Μετασχηματισμός συνημίτονου

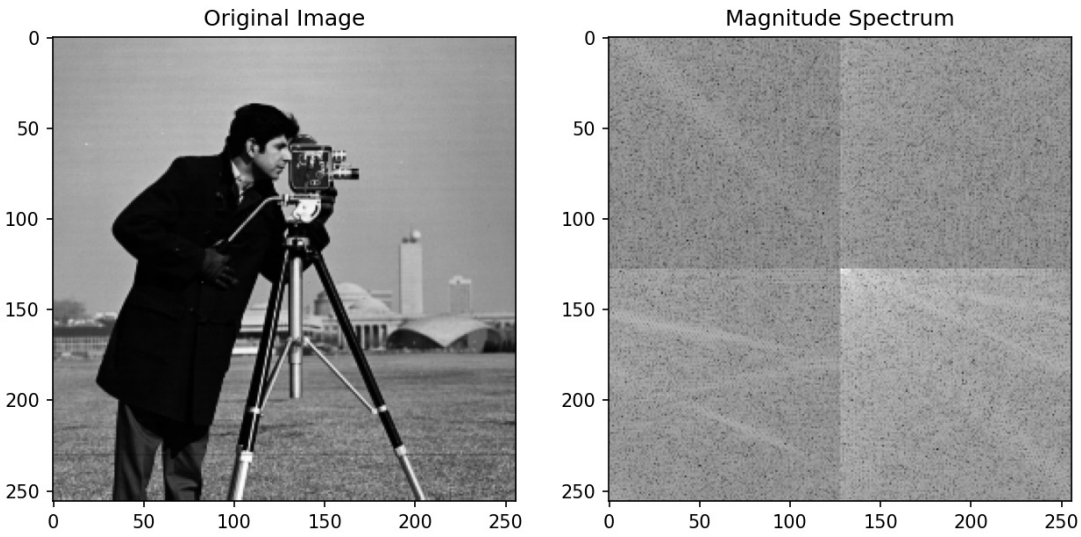
In [138…

```
def dct(image):
    """
    Discrete cosine transform
    """
    rows, cols = image.shape
    result = np.zeros((rows, cols))
    auf = lambda u: math.pow(rows, -0.5) if u == 0 else math.pow(2 / rows, -0.5)
    avf = lambda v: math.pow(cols, -0.5) if v == 0 else math.pow(2 / cols, -0.5)
    for u in range(rows):
        for v in range(cols):
            sum_value = 0
            for m in range(rows):
                for n in range(cols):
                    sum_value += (
                        image[m, n] *
```

```
                        np.cos(np.pi / rows * (m + 0.5) * u) *
                        np.cos(np.pi / cols * (m + 0.5) * v)
                    )
            result[u, v] = auf(u) * avf(v) * sum_value
    return result


def idct(image):
    """
    Inverse Discrete cosine transform
    """
    rows, cols = image.shape
    result = np.zeros((rows, cols))
    auf = lambda u: math.pow(rows, -0.5) if u == 0 else math.pow(2 / rows, -0.5)
    avf = lambda v: math.pow(cols, -0.5) if v == 0 else math.pow(2 / cols, -0.5)
    for m in range(rows):
        for n in range(cols):
            sum_value = 0
            for u in range(rows):
                for v in range(cols):
                    sum_value += (
                        auf(u) * avf(v) * image[u, v] *
                        np.cos(np.pi / rows * (m + 0.5) * u) *
                        np.cos(np.pi / cols * (m + 0.5) * v)
                    )
            result[u, v] = sum_value
    return result
```

In [148…

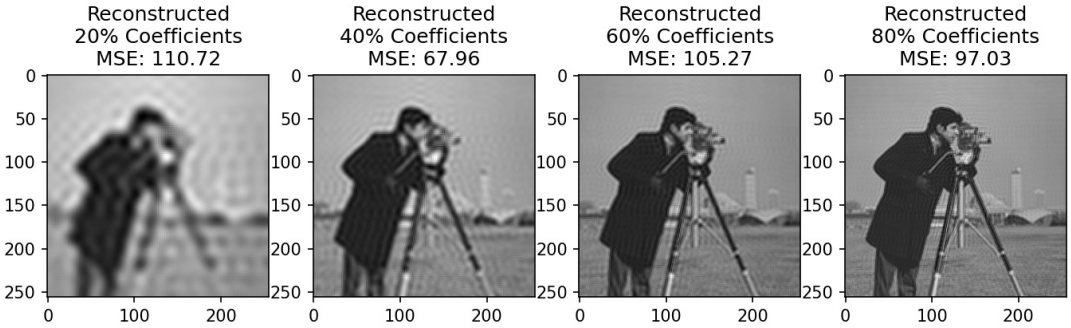
```
original_image = load_image_as_gray('cameraman.bmp'

# Perform 2D Cosine Transform (DCT)
dct_transformed = dct(np.float32(original_image))
dct_shift = np.fft.fftshift(dct_transformed)

# Apply log function to highlighting details across
# wide magnitude ranges and make magnitude lighter
magnitude_spectrum = np.log(np.abs(dct_shift))

# Display original image, magnitude spectrum, and phase spectrum
plt.figure(figsize=(10, 5))

plt.subplot(121), plt.imshow(original_image, cmap='gray')
plt.title('Original Image')

plt.subplot(122), plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('Magnitude Spectrum')

plt.show()
```



In [152…

```
# Reconstruct images with 20%, 40%, 60%, and 80% coefficients and plot
percentages = [0.2, 0.4, 0.6, 0.8]
percentages_cnt = len(percentages)

grid_x_size = 11
plt.figure(figsize=(grid_x_size, grid_x_size // 4 + 2))

for i, percentage in enumerate(percentages):
    # Apply the mask to the input_image image
    dct_shift_filtered = filter_image(dct_shift, percentage)
```

```
    dct_shift_filtered_back = np.fft.fftshift(dct_shift_filtered)

    # Perform the inverse Cosine Transform
    reconstructed_image = idct(dct_transformed_filtered)

    # Normalize reconstructed image
    reconstructed_image = cv2.normalize(
        reconstructed_image, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U
    )
    mse = calculate_mse(original_image, reconstructed_image)

    plt.subplot(1, percentages_cnt, i + 1), plt.imshow(reconstructed_image, cmap='gray')
    plt.title(f'Reconstructed\n{int(percentage*100)}% Coefficients\nMSE: {mse:.2f}')

plt.show()
```



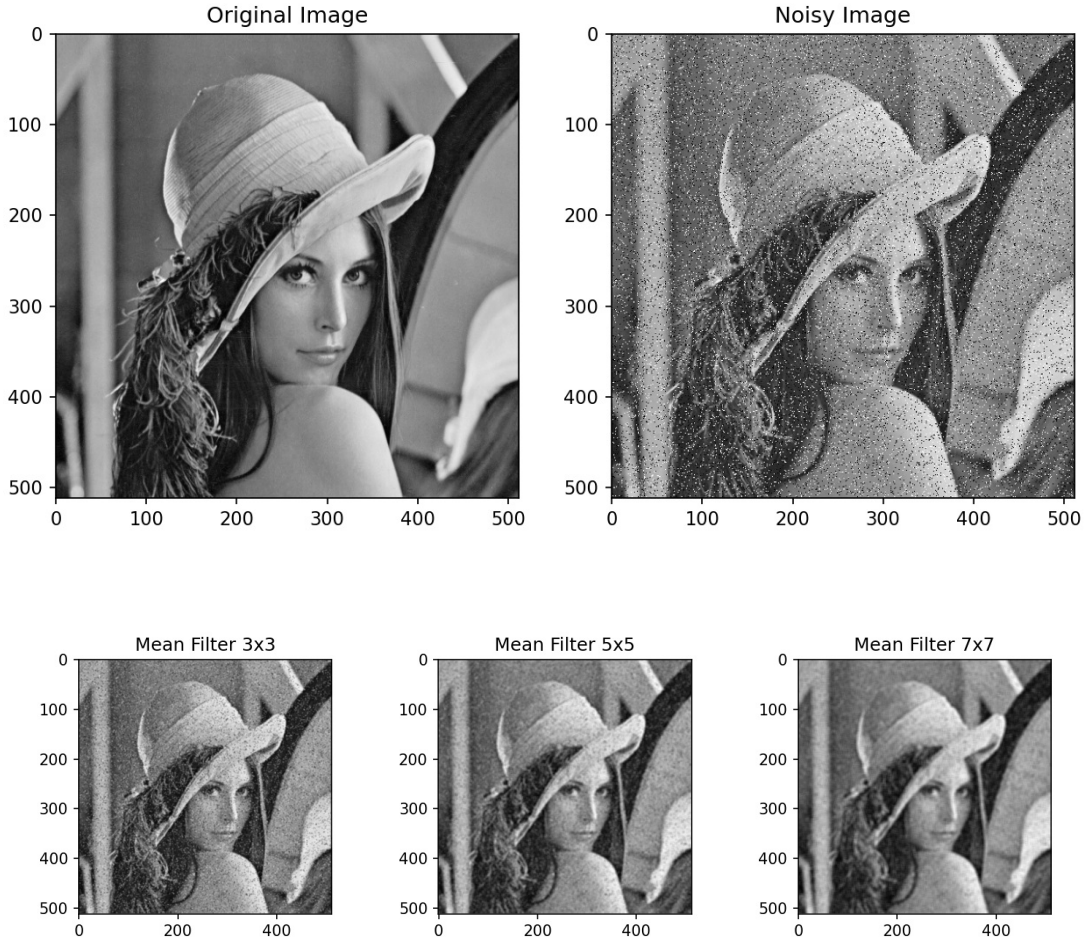
Παρατηρούμε ότι με την αύξηση του ποσοστού των συντελεστών η ποιότητα της ανακατασκευασμένης εικόνας βελτιώνεται και πλησιάζει την αρχική.

Άσκηση #4: Φιλτράρισμα στο πεδίο του χώρου

In [209…
```
def add_salt_and_pepper_noise(image, density):
    noisy_image = image.copy()
    total_pixels = image.size
    num_noisy_pixels = int(density * total_pixels)

    # Add white and black pixels to random indices from
    # the discrete uniform distribution per coordinate
    # Add 'salt' noise (white pixels)
    salt_coordinates = [np.random.randint(0, i - 1, num_noisy_pixels) for i in image.shape]
    noisy_image[tuple(salt_coordinates)] = 255

    # Add 'pepper' noise (black pixels)
    pepper_coordinates = [np.random.randint(0, i - 1, num_noisy_pixels) for i in image.shape]
    noisy_image[tuple(pepper_coordinates)] = 0
    return noisy_image


def apply_mean_filter(image, filter_size):
    # https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html
    return cv2.blur(image, (filter_size, filter_size))
```

In [211…
```
# Load image
original_image = load_image_as_gray('lenna.bmp')

# Add noise
noise_density = 0.05
noisy_image = add_salt_and_pepper_noise(original_image, noise_density)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(original_image, cmap='gray')
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Noisy Image')

plt.show()

# B. Apply successive mean filters (3x3, 5x5, and 7x7)
filter_sizes = [3, 5, 7]
filtered_images = [apply_mean_filter(noisy_image, size) for size in filter_sizes]
```

```
# C. Print the filtered images and compare with the original image
plt.figure(figsize=(12, 3))

for i, size in enumerate(filter_sizes):
    plt.subplot(1, len(filter_sizes), i + 1)
    plt.imshow(filtered_images[i], cmap='gray')
    plt.title(f'Mean Filter {size}x{size}')

plt.show()
```



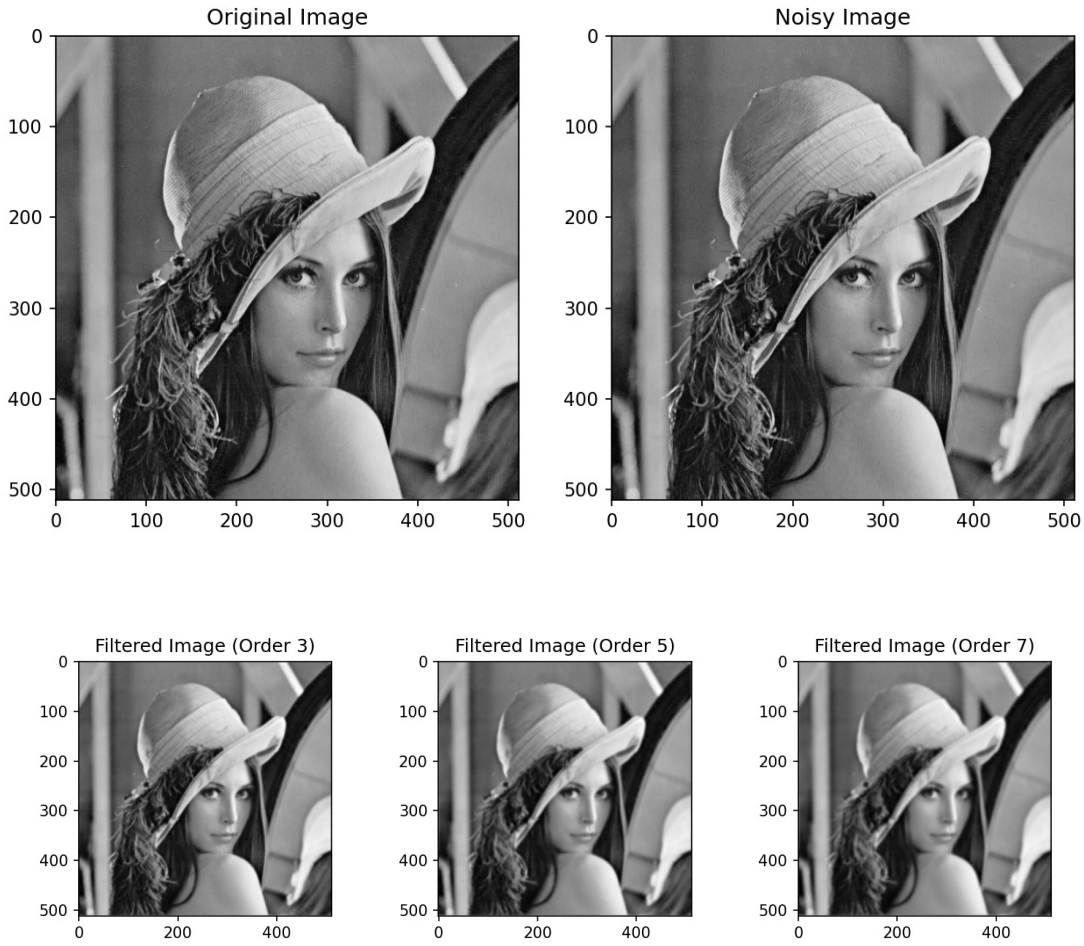
Παρατηρούμε ότι με την αύξηση των διαστάσεων στα φίλτρα ο θόρυβος αφαιρείται με μεγαλύτερη επιτυχία, εις βάρος ωστόσο της ευκρίνειας καθώς η εικόνα θολώνει

Άσκηση #5: Φιλτράρισμα στο πεδίο της συχνότητας

In [267…

```
# Load image
original_image = load_image_as_gray('lenna.bmp')

# Add Gaussian noise with mean 0 and variance 0.01
noise = np.random.normal(0, 0.01, original_image.shape)
noisy_image = original_image + noise

plt.figure(figsize=(10, 5))

# Display the original and noisy images
plt.subplot(1, 2, 1)
plt.imshow(original_image, cmap='gray')
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Noisy Image')
plt.show()

# Apply Butterworth filters
orders = [3, 5, 7]
plt.figure(figsize=(12, 3))

# Apply Fourier Transform
f_transform = np.fft.fft2(noisy_image)
rows, cols = noisy_image.shape
```

```
for i, order in enumerate(orders):
    # Create Butterworth filter
    center = (rows // 2, cols // 2)
    butterworth_filter = 1 / (1 +
                              ((np.fft.fftshift(np.mgrid[:rows, :cols][0]) - center[0]) ** 2 +
                               (np.fft.fftshift(np.mgrid[:rows, :cols][1]) - center[1]) ** 2) /
                               (center[0] * center[1] / 4)
                              ) ** order

    # Apply the filter
    f_transform_filtered = f_transform * butterworth_filter

    # Apply Inverse Fourier Transform
    filtered_image = np.fft.ifft2(f_transform_filtered).real

    plt.subplot(1, len(orders), i + 1)
    plt.imshow(filtered_image, cmap='gray')
    plt.title(f'Filtered Image (Order {order})')
plt.show()
```



Παρατηρούμε ότι με την αύξηση της τάξης στα φίλτρα ο θόρυβος αφαιρείται με μεγαλύτερη επιτυχία, εις βάρος ωστόσο της ευκρίνειας καθώς η εικόνα θολώνει

Άσκηση #6: Τμηματοποίηση εικόνας

In [150…

```
from __future__ import annotations
from typing import Union, Optional
import numpy as np


class MeanShift:

    def __init__(self, radius=20, radius_norm_step=50):
        self.radius = radius
        self.radius_norm_step = radius_norm_step

    def fit(self, data):
        centroids = dict(enumerate(data))
        weights = list(range(self.radius_norm_step)[::-1])
        while True:
            new_centroids = []
            for i in centroids:
```

```
                in_bandwidth = []
                centroid = centroids[i]

                for featureset in data:
                    distance = np.linalg.norm(featureset - centroid) or 0.00000000001
                    weight_index = min(int(distance / self.radius), self.radius_norm_step - 1)
                    to_add = (weights[weight_index] ** 2) * [featureset]
                    in_bandwidth += to_add

                new_centroid = np.average(in_bandwidth, axis=0)
                new_centroids.append(tuple(new_centroid))

            uniques = sorted(list(set(new_centroids)))
            to_pop = []
            for i in uniques:
                for ii in [i for i in uniques]:
                    if i == ii:
                        pass
                    elif np.linalg.norm(np.array(i) - np.array(ii)) <= self.radius:
                        to_pop.append(ii)
                        break

            for i in to_pop:
                try:
                    uniques.remove(i)
                except:
                    pass

            prev_centroids = dict(centroids)
            centroids = {}
            centroids = {i:np.array(v) for i,v in enumerate(uniques)}

            optimized = True

            for i in centroids:
                if not np.array_equal(centroids[i], prev_centroids[i]):
                    optimized = False

            if optimized:
                break

        self.centroids = centroids
        return self


    def predict(self, data):
        classifications = []
        for row in data:
            distances = [np.linalg.norm(row - self.centroids[centroid]) for centroid in self.centroids]
            classification = (distances.index(min(distances)))
            classifications.append(classification)
        return classifications
```

In [156…

```
import cv2
import numpy as np
import matplotlib.pyplot as plt


def mean_shift(file, max_iterations, eps, spatial_window, color_window):
    # Load the image
    image_path = f"{Path.cwd().resolve() / 'images' / file}"
    image = cv2.imread(image_path)

    # Convert BGR to RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # # Convert to CIELAB color space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)

    # Define Mean Shift parameters
    mean_shift_params = {
        'sp': spatial_window,
        'sr': color_window,
        'termcrit':(cv2.TERM_CRITERIA_MAX_ITER, max_iterations, eps)
    }

    # Apply Mean Shift segmentation
    mean_shift_result = cv2.pyrMeanShiftFiltering(image, **mean_shift_params)
    mean_shift_result_gray = cv2.cvtColor(mean_shift_result, cv2.COLOR_BGR2GRAY)

    num_shades_of_gray = len(np.unique(mean_shift_result_gray))
    print(f"Number of different shades of gray: {num_shades_of_gray}")

    # Display the results
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(image)
```

```
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(mean_shift_result_gray, cmap='gray')
    plt.title("Mean Shift Segmentation")
    plt.axis("off")

    plt.show()
```

Διαφορετικές τιμές για *r, d και T*

Αρχικές τιμές

In [157…
```
mean_shift(
    file='butterfly.jpg',
    max_iterations=50, # T
    eps=0.2,
    spatial_window=20, # r
    color_window=40    # d
)
```

```
Number of different shades of gray: 48
```

Χαμηλότερο *r* οδηγεί σε περισσότερες αποχρώσεις

In [160…
```
mean_shift(
    file='butterfly.jpg',
    max_iterations=50, # T
    eps=0.2,
    spatial_window=5, # r
    color_window=40   # d
)
```

```
Number of different shades of gray: 52
```

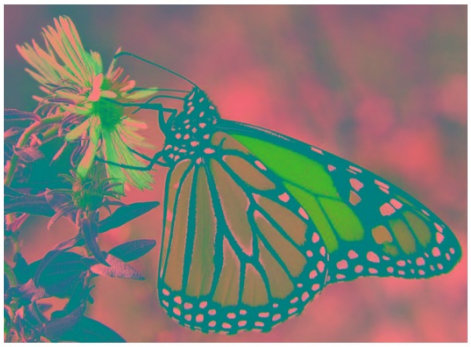
Χαμηλότερο *d* οδηγεί σε περισσότερες αποχρώσεις

In [161…

```
mean_shift(
    file='butterfly.jpg',
    max_iterations=50, # T
    eps=0.2,
    spatial_window=20, # r
    color_window=5     # d
)
```

```
Number of different shades of gray: 60
```

Original Image



Mean Shift Segmentation

Άσκηση #7: Ανίχνευση ακμών

In [3]:

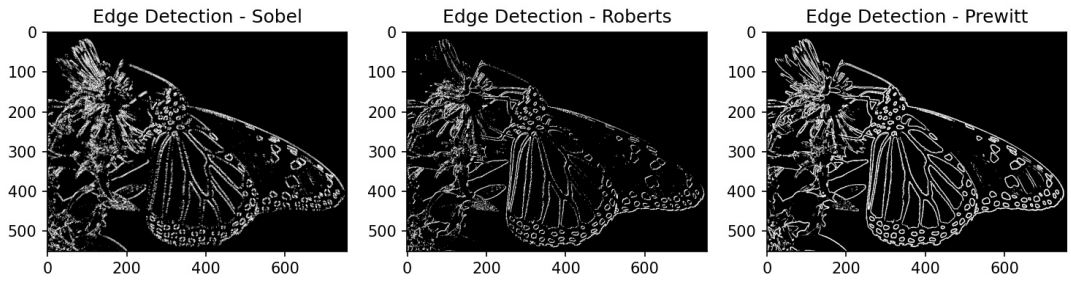
```
def prewitt(src):
    kernel_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]], dtype=np.float32)
    kernel_y = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]], dtype=np.float32)

    gradient_x = cv2.filter2D(src, cv2.CV_64F, kernel_x)
    gradient_y = cv2.filter2D(src, cv2.CV_64F, kernel_y)

    edges = np.sqrt(gradient_x**2 + gradient_y**2)
    # Normalize the edges to the range [0, 255]
    edges = ((edges / np.max(edges)) * 255)
    return edges

def LoG(src, ddepth, ksize, sigmaX):
    edges = cv2.GaussianBlur(src, ksize=(5, 5), sigmaX=variance)
    edges = cv2.Laplacian(edges, cv2.CV_64F)
    return edges


def apply_edge_detection(image, method, threshold=None, variance=None):
    # Apply the specified edge detection method
    dispatcher = {
        'sobel':{
            'function':cv2.Sobel, 'kwargs':{'src':image, 'ddepth':cv2.CV_64F, 'dx':1, 'dy':1, 'ksize':3}
        },
        'roberts':{
            'function':cv2.filter2D, 'kwargs':{
                'src':image, 'ddepth':cv2.CV_64F, 'kernel':np.array([[1, 0], [0, -1]])
            }
        },
        'prewitt':{'function':prewitt, 'kwargs':{'src':image}},
        'kirsch':{
            'function':cv2.filter2D,
```

```
            'kwargs':{
                'src':image, 'ddepth':cv2.CV_64F, 'kernel':np.array([[5, 5, 5], [-3, 0, -3], [-3, -3, -3]])
            }
        },
        'log':{
            'function':LoG,
            'kwargs':{'src':image, 'ddepth':cv2.CV_64F, 'ksize':(5, 5), 'sigmaX':variance}
        },
        'canny':{
            'function':cv2.Canny,
            'kwargs':{'image':image, 'threshold1':100, 'threshold2':200}
        }
    }
    function = dispatcher[method]['function']
    kwargs = dispatcher[method]['kwargs']
    edges = function(**kwargs)

    # Apply global thresholding using Otsu's method for 'sobel', 'roberts', 'prewitt' methods
    if method in ['sobel', 'roberts', 'prewitt']:
        _, edges = cv2.threshold(np.abs(edges).astype(np.uint8), 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    elif method in ['kirsch', 'log'] and threshold is not None:
        _, edges = cv2.threshold(np.abs(edges).astype(np.uint8), threshold, 255, cv2.THRESH_BINARY)
    return edges.astype(np.uint8)
```

In [5]:
```
# Load the images
butterfly_image = load_image_as_gray('butterfly_g.jpg')
cameraman_image = load_image_as_gray('cameraman.bmp')
lenna_image = load_image_as_gray('lenna.bmp')
```

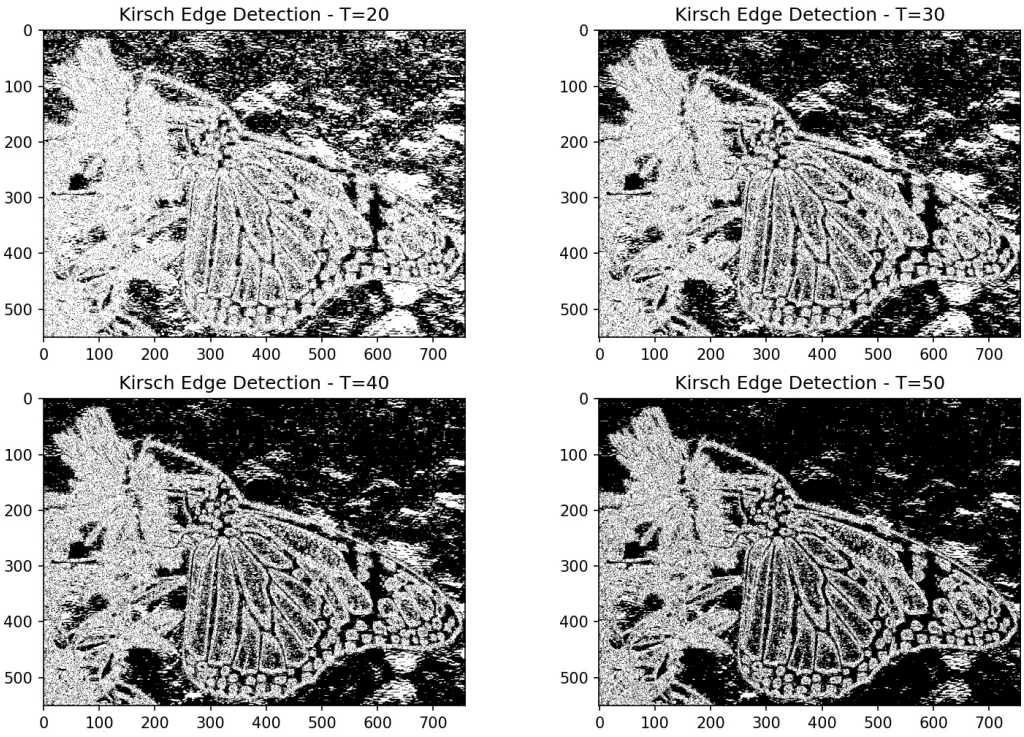
Υπολογισμός ακμών με χρήση μεθόδων ***Sobel, Roberts και Prewitt*** και καθολικής κατωφλίωσης του ***Otsu***

In [60]:
```
plt.figure(figsize=(12, 3))

# Edge detection using Sobel, Roberts, Prewitt, and Kirsch
edge_methods = ['sobel', 'roberts', 'prewitt']
for i, method in enumerate(edge_methods):
    edges = apply_edge_detection(butterfly_image, method)
    plt.subplot(1, len(edge_methods), i + 1)
    plt.imshow(edges, cmap='gray')
    plt.title(f'Edge Detection - {method.title()}')
    plt.show()
```



Υπολογισμός ακμών με χρήση μεθόδου του ***Kirsch*** και σύγκριση διαφορετικών τιμών κατωφλίου (*20, 30, 40, 50*)

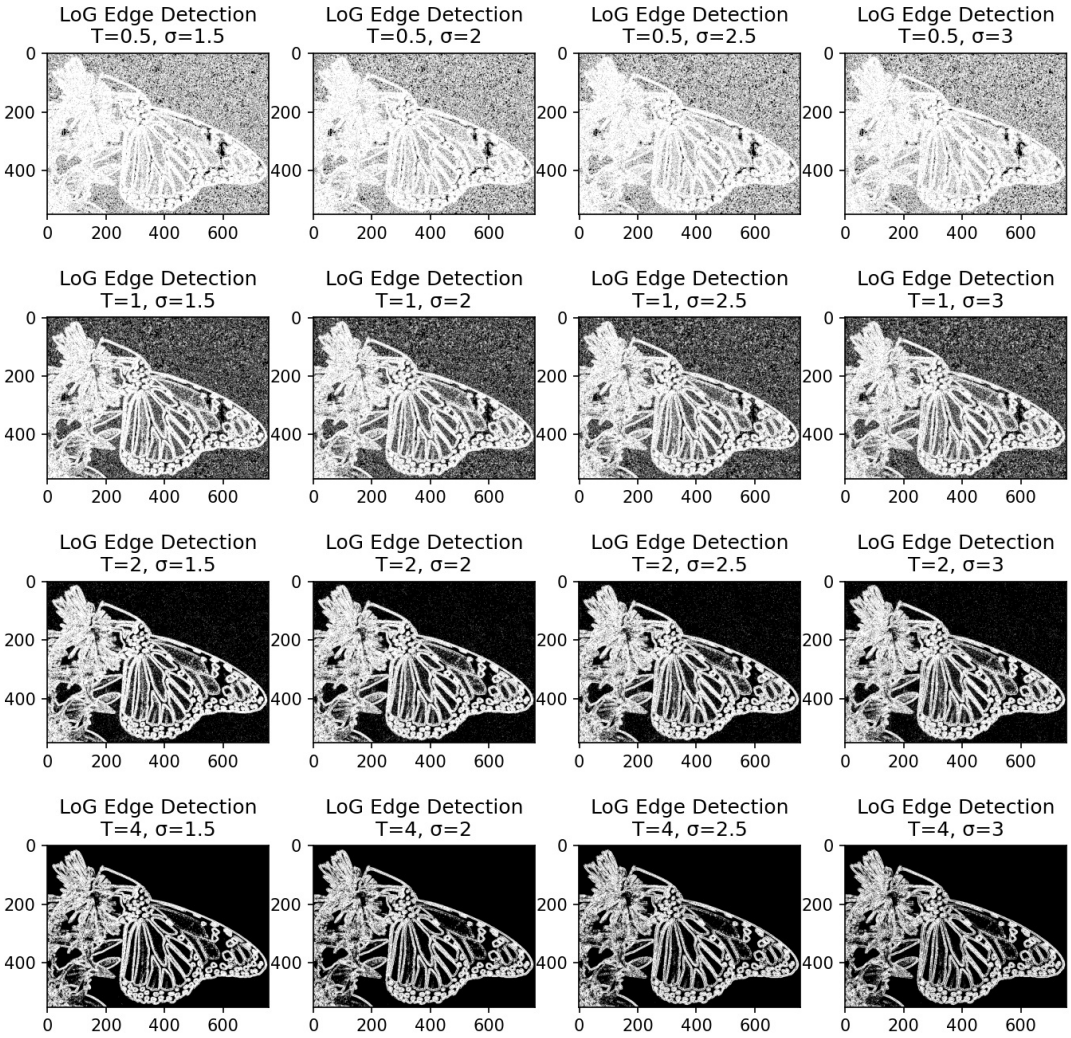
In [69]:
```
grid_x_size = 12
plt.figure(figsize=(grid_x_size, grid_x_size // 2 + 2))

# Edge detection using Sobel, Roberts, Prewitt, and Kirsch
thresholds = [20, 30, 40, 50]
num_thresholds = len(thresholds)
num_cols = num_thresholds // 2

for i, threshold in enumerate(thresholds):
    edges = apply_edge_detection(butterfly_image, 'kirsch', threshold)
    plt.subplot(2, num_cols, i + 1)
    plt.imshow(edges, cmap='gray')
    plt.title(f'Kirsch Edge Detection - T={threshold}')
    plt.show()
```



Μια "καλή" τιμή για τη μέθοδο του ***Kirsch*** δείχνει να είναι το `T=50` ενώ μεταξύ των 4ων μεθόδων η ακριβέστερη φαίνεται να είναι αυτή του ***Prewitt***

Υπολογισμός ακμών με χρήση φίλτρου ***LoG*** και σύγκριση διαφορετικών τιμών διακύμανσης και του κατωφλίου

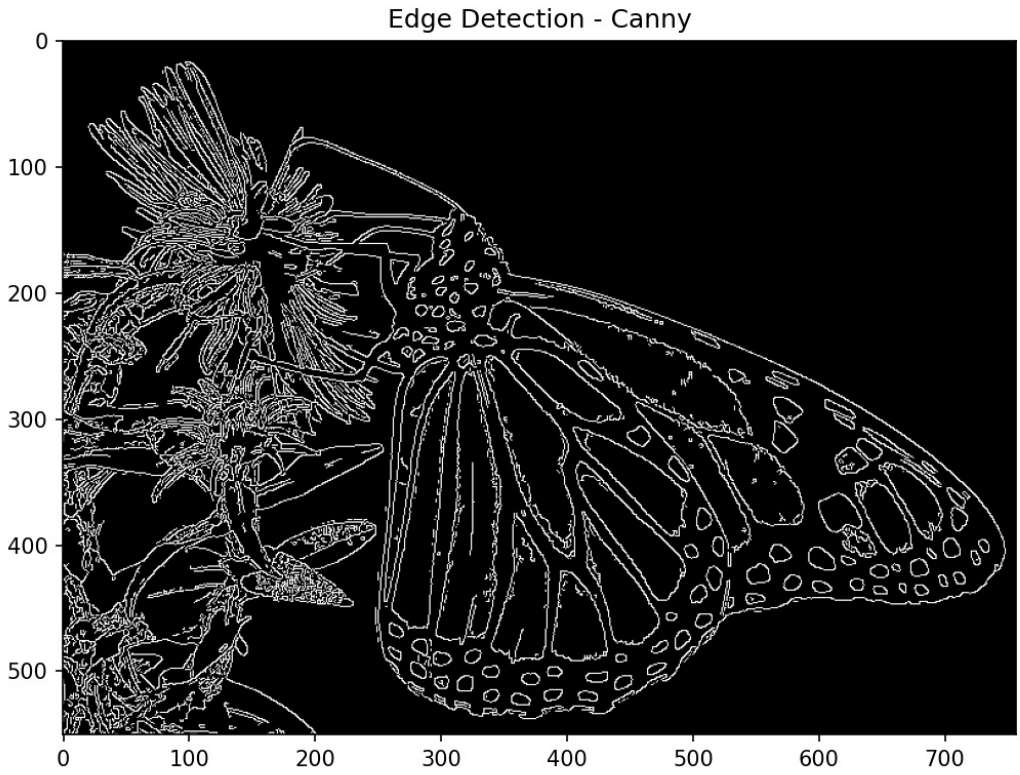
In [6]:
```
grid_x_size = 11
plt.figure(figsize=(grid_x_size, grid_x_size))

# Edge detection using Sobel, Roberts, Prewitt, and Kirsch
thresholds = [0.5, 1, 2, 4]
variances = [1.5, 2, 2.5, 3]
num_thresholds = len(thresholds)
num_rows = len(thresholds)

for i, threshold in enumerate(thresholds):
    for j, variance in enumerate(variances):
        position = i * len(variances) + j + 1
        edges = apply_edge_detection(butterfly_image, 'log', threshold, variance)
        plt.subplot(num_rows, len(variances), position)
        plt.imshow(edges, cmap='gray')
        plt.title(f'LoG Edge Detection\nT={threshold}, σ={variance}')
        plt.show()
```



"Καλές" τιμές για την ***LoG*** φαίνονται να είναι το `T=4` ενώ οι συνδυασμοί αυτής της τιμής με τις διαφορετικές τιμές του $\sigma = 1.5, 2, 2.5, 3$ δείχνουν να έχουν παρόμοια καλό αποτέλεσμα.

Υπολογισμός ακμών με χρήση της μεθόδου του ***Canny***

In [7]:
```
# Edge detection using Canny
butterfly_edges_canny = apply_edge_detection(butterfly_image, 'canny')
plt.figure(figsize=(8, 8))
plt.subplot(1, 1, 1)
plt.imshow(butterfly_edges_canny, cmap='gray')
plt.title('Edge Detection - Canny')
plt.show()
```

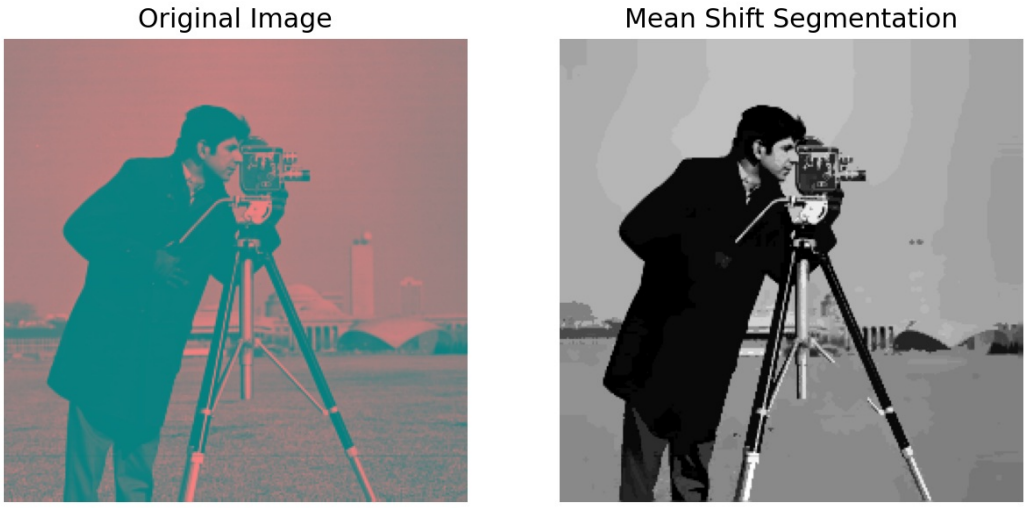


Η μέθοδος του ***Canny*** δείχνει να είναι η ακριβέστερη

Βέλτιστες παράμετροι για το mean shift

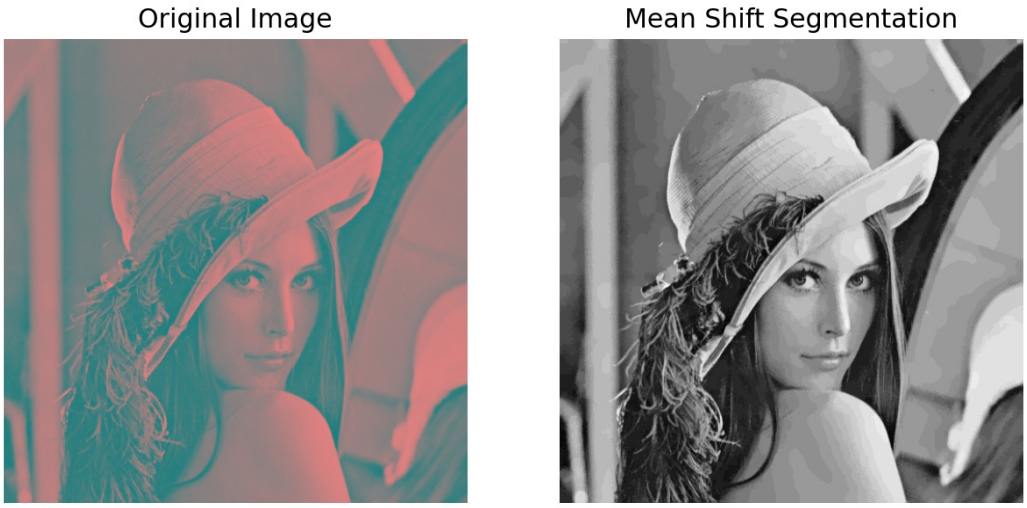
In [170…
```
mean_shift(
    file='cameraman.bmp',
    max_iterations=50, # T
    eps=0.2,
    spatial_window=15, # r
    color_window=20    # d
)
```

```
Number of different shades of gray: 29
```



In [179…
```
mean_shift(
    file='lenna.bmp',
    max_iterations=50, # T
    eps=0.2,
    spatial_window=5, # r
    color_window=5    # d
)
```

```
Number of different shades of gray: 26
```



Άσκηση #8: Εξαγωγή χαρακτηριστικών για ανάκτηση εικόνων

In [101
```
from pathlib import Path
```

```
from pathlib import Path
import random
import itertools
from skimage import feature
import numpy as np
```

In [ ]:
```
# Norm implementation

def manhattan_norm(x, y):
    return np.sum(abs(x - y))

def euclidean_norm(x, y):
    return np.sqrt(np.sum((x - y) ** 2))
```

In [287…
```
class ImageDatabase:
    char_names = ['a1', 'a2']
    metric_names = ['b1', 'b2']
    database_items = []

    def __init__(self, path):
        self.path = path
        self.name = self.path.name
        self.category = '_'.join(self.name.split('_')[:-1])
        self.image = load_image_as_gray(self.path)
        self.a1 = self.extract_features(self.image, self.char_names[0])
        self.a2 = self.extract_features(self.image, self.char_names[1])
        self.__class__.database_items.append(self)
        print(len(self.__class__.database_items), end=' ')

    @classmethod
    def get_item_by_path(cls, path):
        return next(iter([item for item in cls.database_items if item.path == path]), None)

    # Function to extract features from an image
    def extract_features(self, image, char_name):
        # Feature extraction - A1: Normalized brightness histogram
        if char_name == 'a1':
            hist_brightness, _ = np.histogram(image.flatten(), bins=256, range=[0,256], density=True)
            return hist_brightness
        elif char_name == 'a2':
            # Feature extraction - A2: Normalized LBP histogram
            lbp = feature.local_binary_pattern(image, P=8, R=5)
            hist_lbp, _ = np.histogram(lbp.flatten(), bins=256, range=[0,256], density=True)
            return hist_lbp

    # Function to compute similarity metrics
    def similarity_metric(self, x, y, metric_name):
        if metric_name == 'b1':
            return manhattan_norm(x, y)
        elif metric_name == 'b2':
            return euclidean_norm(x, y)

    # Function to perform image retrieval
    def image_retrieval(self, char_name, metric_name):
        results = []

        # Exclude self for similarity comparison
        to_compare_with = [item for item in self.database_items if item.path != self.path]

        for db_item in to_compare_with:
            similarity = self.similarity_metric(
                getattr(self, char_name),
                getattr(db_item, char_name),
                metric_name
            )
            results.append((db_item, similarity))

        # Sort results by similarity in descending order
        results.sort(key=lambda x: x[1], reverse=True)
        return results[:10]
```

In [283…
```
image_paths = list((Path.cwd().resolve() / 'flowers').iterdir())#[:2]
categories = set(['_'.join(x.name.split('_')[:-1]) for x in image_paths])
paths_per_category = [[p for p in image_paths if c in p.name] for c in categories]
```

In [312…
```
# Create a feature database for all images
for path in image_paths:
    ImageDatabase(path)
```

In [290…
```
feature_combinations = list(itertools.product(ImageDatabase.char_names, ImageDatabase.metric_names))
feature_combinations
```

Out[290]:
```
[('a1', 'b1'), ('a1', 'b2'), ('a2', 'b1'), ('a2', 'b2')]
```

In [ ]:
```
# Select random query images from different categories
query_image_paths = [random.choice(p) for p in paths_per_category]
```

In [331…

```python
for query_image_path in query_image_paths:
    query_image = ImageDatabase.get_item_by_path(query_image_path)

    print(f" Query Image: {query_image_path.name} ".center(80, '='))

    # Perform image retrieval for each combination of features and metrics
    for char_name, metric_name in feature_combinations:
        results = query_image.image_retrieval(char_name, metric_name)

        print(f"\nResults for Features {char_name} and Metric {metric_name}:")
        for i, (item, similarity) in enumerate(results):
            print(f"{i + 1}. Image {item.name} - Similarity: {similarity:.4f}")
        correct_category_retrieved = (
            100 * sum([x[0].category == query_image.category for x in results]) / len(results)
        )
        print(f"==> Correct categories: {correct_category_retrieved:.2f}%")
        print()
```

```
==================== Query Image: bougainvillea_00068.jpg ====================

Results for Features a1 and Metric b1:
1. Image bougainvillea_00004.jpg - Similarity: 1.4014
2. Image bougainvillea_00034.jpg - Similarity: 1.2577
3. Image bougainvillea_00047.jpg - Similarity: 1.2577
4. Image bougainvillea_00079.jpg - Similarity: 1.2314
5. Image bougainvillea_00036.jpg - Similarity: 1.1571
6. Image bougainvillea_00049.jpg - Similarity: 1.1571
7. Image bougainvillea_00002.jpg - Similarity: 1.1419
8. Image gardenias_00052.jpg - Similarity: 1.1402
9. Image bougainvillea_00074.jpg - Similarity: 1.1085
10. Image bougainvillea_00032.jpg - Similarity: 1.0686
==> Correct categories: 90.00%


Results for Features a1 and Metric b2:
1. Image tulip_00036.jpg - Similarity: 0.1564
2. Image orchids_00012.jpg - Similarity: 0.1555
3. Image bougainvillea_00066.jpg - Similarity: 0.1396
4. Image gardenias_00025.jpg - Similarity: 0.1370
5. Image bougainvillea_00076.jpg - Similarity: 0.1234
6. Image bougainvillea_00019.jpg - Similarity: 0.1218
7. Image bougainvillea_00006.jpg - Similarity: 0.1214
8. Image bougainvillea_00004.jpg - Similarity: 0.1183
9. Image bougainvillea_00024.jpg - Similarity: 0.1147
10. Image bougainvillea_00014.jpg - Similarity: 0.1105
==> Correct categories: 70.00%


Results for Features a2 and Metric b1:
1. Image garden_roses_00085.jpg - Similarity: 1.0296
2. Image daisies_00035.jpg - Similarity: 0.7430
3. Image tulip_00005.jpg - Similarity: 0.6567
4. Image lilies_00083.jpg - Similarity: 0.6566
5. Image bougainvillea_00019.jpg - Similarity: 0.6510
6. Image bougainvillea_00026.jpg - Similarity: 0.6412
7. Image bougainvillea_00006.jpg - Similarity: 0.6270
8. Image bougainvillea_00061.jpg - Similarity: 0.6230
9. Image bougainvillea_00022.jpg - Similarity: 0.5811
10. Image bougainvillea_00076.jpg - Similarity: 0.5772
==> Correct categories: 60.00%


Results for Features a2 and Metric b2:
1. Image bougainvillea_00006.jpg - Similarity: 0.1651
2. Image bougainvillea_00019.jpg - Similarity: 0.1538
3. Image bougainvillea_00026.jpg - Similarity: 0.1334
4. Image bougainvillea_00066.jpg - Similarity: 0.1334
5. Image bougainvillea_00076.jpg - Similarity: 0.1271
6. Image bougainvillea_00024.jpg - Similarity: 0.1141
7. Image bougainvillea_00014.jpg - Similarity: 0.1110
8. Image bougainvillea_00016.jpg - Similarity: 0.1044
9. Image bougainvillea_00063.jpg - Similarity: 0.1034
10. Image bougainvillea_00059.jpg - Similarity: 0.1033
==> Correct categories: 100.00%

======================== Query Image: lilies_00004.jpg ========================

Results for Features a1 and Metric b1:
1. Image garden_roses_00038.jpg - Similarity: 1.7136
2. Image garden_roses_00057.jpg - Similarity: 1.4898
3. Image bougainvillea_00004.jpg - Similarity: 1.4184
4. Image gardenias_00012.jpg - Similarity: 1.2976
5. Image lilies_00041.jpg - Similarity: 1.2363
6. Image gardenias_00023.jpg - Similarity: 1.1500
7. Image lilies_00032.jpg - Similarity: 1.1178
8. Image daisies_00024.jpg - Similarity: 1.1141
9. Image orchids_00055.jpg - Similarity: 1.1007
10. Image gardenias_00084.jpg - Similarity: 1.0971
==> Correct categories: 20.00%
```

```
Results for Features a1 and Metric b2:
1. Image garden_roses_00065.jpg - Similarity: 0.5984
2. Image orchids_00048.jpg - Similarity: 0.4285
3. Image hibiscus_00057.jpg - Similarity: 0.2328
4. Image lilies_00027.jpg - Similarity: 0.1796
5. Image gardenias_00027.jpg - Similarity: 0.1307
6. Image bougainvillea_00076.jpg - Similarity: 0.1213
7. Image lilies_00038.jpg - Similarity: 0.1155
8. Image lilies_00074.jpg - Similarity: 0.1144
9. Image daisies_00044.jpg - Similarity: 0.1134
10. Image gardenias_00014.jpg - Similarity: 0.1074
==> Correct categories: 30.00%


Results for Features a2 and Metric b1:
1. Image lilies_00083.jpg - Similarity: 0.6831
2. Image lilies_00067.jpg - Similarity: 0.6492
3. Image garden_roses_00073.jpg - Similarity: 0.5919
4. Image hydrangeas_00014.jpg - Similarity: 0.5362
5. Image peonies_00037.jpg - Similarity: 0.5138
6. Image tulip_00027.jpg - Similarity: 0.4664
7. Image peonies_00038.jpg - Similarity: 0.4656
8. Image daisies_00065.jpg - Similarity: 0.4516
9. Image bougainvillea_00042.jpg - Similarity: 0.4445
10. Image bougainvillea_00046.jpg - Similarity: 0.4419
==> Correct categories: 20.00%


Results for Features a2 and Metric b2:
1. Image lilies_00083.jpg - Similarity: 0.2949
2. Image orchids_00045.jpg - Similarity: 0.2541
3. Image hibiscus_00039.jpg - Similarity: 0.2277
4. Image lilies_00027.jpg - Similarity: 0.1645
5. Image hibiscus_00075.jpg - Similarity: 0.1491
6. Image lilies_00067.jpg - Similarity: 0.1385
7. Image bougainvillea_00066.jpg - Similarity: 0.1348
8. Image lilies_00038.jpg - Similarity: 0.1079
9. Image lilies_00074.jpg - Similarity: 0.1071
10. Image bougainvillea_00028.jpg - Similarity: 0.1008
==> Correct categories: 50.00%

====================== Query Image: hydrangeas_00016.jpg ======================

Results for Features a1 and Metric b1:
1. Image peonies_00018.jpg - Similarity: 1.2127
2. Image tulip_00058.jpg - Similarity: 1.0286
3. Image tulip_00004.jpg - Similarity: 0.9997
4. Image hydrangeas_00051.jpg - Similarity: 0.9943
5. Image gardenias_00057.jpg - Similarity: 0.9203
6. Image hydrangeas_00063.jpg - Similarity: 0.9091
7. Image daisies_00004.jpg - Similarity: 0.9021
8. Image lilies_00020.jpg - Similarity: 0.7938
9. Image hydrangeas_00043.jpg - Similarity: 0.7850
10. Image hydrangeas_00048.jpg - Similarity: 0.7669
==> Correct categories: 40.00%


Results for Features a1 and Metric b2:
1. Image orchids_00058.jpg - Similarity: 0.3903
2. Image orchids_00027.jpg - Similarity: 0.2785
3. Image hydrangeas_00051.jpg - Similarity: 0.1674
4. Image bougainvillea_00006.jpg - Similarity: 0.1179
5. Image hibiscus_00029.jpg - Similarity: 0.1104
6. Image hydrangeas_00048.jpg - Similarity: 0.0861
7. Image daisies_00018.jpg - Similarity: 0.0846
8. Image lilies_00086.jpg - Similarity: 0.0811
9. Image hydrangeas_00067.jpg - Similarity: 0.0798
10. Image daisies_00047.jpg - Similarity: 0.0777
==> Correct categories: 30.00%


Results for Features a2 and Metric b1:
1. Image tulip_00035.jpg - Similarity: 0.8916
2. Image tulip_00075.jpg - Similarity: 0.8536
3. Image tulip_00031.jpg - Similarity: 0.8434
4. Image orchids_00051.jpg - Similarity: 0.8398
5. Image daisies_00035.jpg - Similarity: 0.7927
6. Image daisies_00070.jpg - Similarity: 0.7148
7. Image tulip_00014.jpg - Similarity: 0.7120
8. Image tulip_00077.jpg - Similarity: 0.6598
9. Image garden_roses_00014.jpg - Similarity: 0.6525
10. Image hydrangeas_00043.jpg - Similarity: 0.6272
==> Correct categories: 10.00%


Results for Features a2 and Metric b2:
1. Image gardenias_00049.jpg - Similarity: 0.3052
```

```
2. Image orchids_00033.jpg - Similarity: 0.2143
3. Image tulip_00059.jpg - Similarity: 0.2067
4. Image garden_roses_00058.jpg - Similarity: 0.1378
5. Image hydrangeas_00051.jpg - Similarity: 0.1336
6. Image tulip_00014.jpg - Similarity: 0.1162
7. Image gardenias_00083.jpg - Similarity: 0.1151
8. Image gardenias_00014.jpg - Similarity: 0.1085
9. Image peonies_00027.jpg - Similarity: 0.1066
10. Image hydrangeas_00043.jpg - Similarity: 0.1025
==> Correct categories: 20.00%

======================== Query Image: daisies_00053.jpg ========================

Results for Features a1 and Metric b1:
1. Image daisies_00017.jpg - Similarity: 1.6526
2. Image daisies_00074.jpg - Similarity: 1.5846
3. Image daisies_00007.jpg - Similarity: 1.5819
4. Image daisies_00035.jpg - Similarity: 1.5507
5. Image garden_roses_00057.jpg - Similarity: 1.4984
6. Image orchids_00057.jpg - Similarity: 1.4664
7. Image daisies_00031.jpg - Similarity: 1.4217
8. Image daisies_00043.jpg - Similarity: 1.3981
9. Image daisies_00046.jpg - Similarity: 1.3981
10. Image tulip_00017.jpg - Similarity: 1.3677
==> Correct categories: 70.00%


Results for Features a1 and Metric b2:
1. Image daisies_00007.jpg - Similarity: 0.7110
2. Image daisies_00017.jpg - Similarity: 0.5494
3. Image daisies_00074.jpg - Similarity: 0.4789
4. Image daisies_00035.jpg - Similarity: 0.4294
5. Image orchids_00048.jpg - Similarity: 0.4078
6. Image daisies_00063.jpg - Similarity: 0.2943
7. Image daisies_00010.jpg - Similarity: 0.2832
8. Image hibiscus_00057.jpg - Similarity: 0.2382
9. Image peonies_00068.jpg - Similarity: 0.1820
10. Image daisies_00048.jpg - Similarity: 0.1491
==> Correct categories: 70.00%


Results for Features a2 and Metric b1:
1. Image daisies_00007.jpg - Similarity: 1.2809
2. Image daisies_00017.jpg - Similarity: 1.0752
3. Image garden_roses_00015.jpg - Similarity: 0.9179
4. Image daisies_00074.jpg - Similarity: 0.8465
5. Image bougainvillea_00026.jpg - Similarity: 0.7435
6. Image orchids_00048.jpg - Similarity: 0.6937
7. Image daisies_00035.jpg - Similarity: 0.6856
8. Image tulip_00030.jpg - Similarity: 0.6800
9. Image garden_roses_00028.jpg - Similarity: 0.6426
10. Image bougainvillea_00067.jpg - Similarity: 0.6347
==> Correct categories: 40.00%


Results for Features a2 and Metric b2:
1. Image daisies_00007.jpg - Similarity: 0.6461
2. Image daisies_00017.jpg - Similarity: 0.5409
3. Image daisies_00074.jpg - Similarity: 0.4254
4. Image daisies_00035.jpg - Similarity: 0.3330
5. Image daisies_00063.jpg - Similarity: 0.2574
6. Image daisies_00010.jpg - Similarity: 0.2321
7. Image orchids_00033.jpg - Similarity: 0.2300
8. Image garden_roses_00072.jpg - Similarity: 0.1216
9. Image tulip_00024.jpg - Similarity: 0.1171
10. Image daisies_00024.jpg - Similarity: 0.1155
==> Correct categories: 70.00%

======================== Query Image: hibiscus_00079.jpg ========================

Results for Features a1 and Metric b1:
1. Image gardenias_00081.jpg - Similarity: 1.3279
2. Image bougainvillea_00004.jpg - Similarity: 1.1501
3. Image orchids_00012.jpg - Similarity: 1.0567
4. Image hibiscus_00029.jpg - Similarity: 1.0553
5. Image hibiscus_00076.jpg - Similarity: 0.9731
6. Image hibiscus_00073.jpg - Similarity: 0.9647
7. Image hibiscus_00022.jpg - Similarity: 0.9641
8. Image hibiscus_00034.jpg - Similarity: 0.8990
9. Image hibiscus_00015.jpg - Similarity: 0.8951
10. Image daisies_00080.jpg - Similarity: 0.8903
==> Correct categories: 60.00%


Results for Features a1 and Metric b2:
1. Image daisies_00010.jpg - Similarity: 0.2802
2. Image orchids_00024.jpg - Similarity: 0.2692
3. Image hibiscus_00057.jpg - Similarity: 0.2304
4. Image hibiscus_00039.jpg - Similarity: 0.2258
```

```
5. Image hibiscus_00048.jpg - Similarity: 0.2258
6. Image hibiscus_00064.jpg - Similarity: 0.1628
7. Image hibiscus_00037.jpg - Similarity: 0.1622
8. Image hibiscus_00046.jpg - Similarity: 0.1622
9. Image hibiscus_00075.jpg - Similarity: 0.1539
10. Image hibiscus_00070.jpg - Similarity: 0.1517
==> Correct categories: 80.00%


Results for Features a2 and Metric b1:
1. Image hibiscus_00003.jpg - Similarity: 0.9421
2. Image hibiscus_00057.jpg - Similarity: 0.8168
3. Image gardenias_00023.jpg - Similarity: 0.8105
4. Image gardenias_00049.jpg - Similarity: 0.7526
5. Image daisies_00006.jpg - Similarity: 0.7478
6. Image hibiscus_00027.jpg - Similarity: 0.7117
7. Image hibiscus_00004.jpg - Similarity: 0.6176
8. Image daisies_00088.jpg - Similarity: 0.6057
9. Image hibiscus_00064.jpg - Similarity: 0.6044
10. Image hibiscus_00006.jpg - Similarity: 0.5655
==> Correct categories: 60.00%


Results for Features a2 and Metric b2:
1. Image hibiscus_00057.jpg - Similarity: 0.2137
2. Image hibiscus_00039.jpg - Similarity: 0.2080
3. Image hibiscus_00048.jpg - Similarity: 0.2080
4. Image hibiscus_00003.jpg - Similarity: 0.1506
5. Image hibiscus_00037.jpg - Similarity: 0.1462
6. Image hibiscus_00046.jpg - Similarity: 0.1462
7. Image hibiscus_00075.jpg - Similarity: 0.1301
8. Image hibiscus_00070.jpg - Similarity: 0.1244
9. Image hibiscus_00064.jpg - Similarity: 0.1242
10. Image orchids_00002.jpg - Similarity: 0.1173
==> Correct categories: 90.00%

======================== Query Image: orchids_00024.jpg ========================

Results for Features a1 and Metric b1:
1. Image daisies_00007.jpg - Similarity: 1.6270
2. Image orchids_00025.jpg - Similarity: 1.5945
3. Image gardenias_00080.jpg - Similarity: 1.5856
4. Image hibiscus_00034.jpg - Similarity: 1.5751
5. Image orchids_00026.jpg - Similarity: 1.5503
6. Image orchids_00022.jpg - Similarity: 1.5434
7. Image orchids_00032.jpg - Similarity: 1.5384
8. Image orchids_00039.jpg - Similarity: 1.5384
9. Image orchids_00052.jpg - Similarity: 1.5266
10. Image garden_roses_00065.jpg - Similarity: 1.5240
==> Correct categories: 60.00%


Results for Features a1 and Metric b2:
1. Image orchids_00057.jpg - Similarity: 0.7736
2. Image orchids_00022.jpg - Similarity: 0.5427
3. Image orchids_00048.jpg - Similarity: 0.5040
4. Image orchids_00021.jpg - Similarity: 0.4728
5. Image orchids_00058.jpg - Similarity: 0.4701
6. Image orchids_00027.jpg - Similarity: 0.3901
7. Image orchids_00013.jpg - Similarity: 0.3521
8. Image orchids_00025.jpg - Similarity: 0.3372
9. Image orchids_00005.jpg - Similarity: 0.3226
10. Image orchids_00055.jpg - Similarity: 0.2925
==> Correct categories: 100.00%


Results for Features a2 and Metric b1:
1. Image orchids_00051.jpg - Similarity: 1.0367
2. Image orchids_00019.jpg - Similarity: 1.0268
3. Image orchids_00054.jpg - Similarity: 1.0200
4. Image orchids_00002.jpg - Similarity: 1.0188
5. Image daisies_00058.jpg - Similarity: 0.9810
6. Image tulip_00054.jpg - Similarity: 0.9808
7. Image orchids_00060.jpg - Similarity: 0.9776
8. Image orchids_00062.jpg - Similarity: 0.9705
9. Image gardenias_00014.jpg - Similarity: 0.9630
10. Image orchids_00059.jpg - Similarity: 0.9598
==> Correct categories: 70.00%


Results for Features a2 and Metric b2:
1. Image orchids_00019.jpg - Similarity: 0.5035
2. Image daisies_00079.jpg - Similarity: 0.4999
3. Image tulip_00034.jpg - Similarity: 0.4978
4. Image orchids_00002.jpg - Similarity: 0.4955
5. Image orchids_00054.jpg - Similarity: 0.4954
6. Image orchids_00062.jpg - Similarity: 0.4810
7. Image tulip_00051.jpg - Similarity: 0.4782
8. Image orchids_00051.jpg - Similarity: 0.4776
```

```
9. Image hibiscus_00022.jpg - Similarity: 0.4770
10. Image orchids_00066.jpg - Similarity: 0.4748
==> Correct categories: 60.00%

==================== Query Image: garden_roses_00053.jpg ======================

Results for Features a1 and Metric b1:
1. Image garden_roses_00038.jpg - Similarity: 1.6810
2. Image garden_roses_00052.jpg - Similarity: 1.5889
3. Image garden_roses_00050.jpg - Similarity: 1.5535
4. Image garden_roses_00085.jpg - Similarity: 1.5397
5. Image garden_roses_00057.jpg - Similarity: 1.4763
6. Image garden_roses_00065.jpg - Similarity: 1.4499
7. Image garden_roses_00077.jpg - Similarity: 1.4214
8. Image garden_roses_00015.jpg - Similarity: 1.4171
9. Image gardenias_00012.jpg - Similarity: 1.4116
10. Image garden_roses_00084.jpg - Similarity: 1.2865
==> Correct categories: 90.00%


Results for Features a1 and Metric b2:
1. Image garden_roses_00052.jpg - Similarity: 0.7375
2. Image garden_roses_00038.jpg - Similarity: 0.7341
3. Image garden_roses_00050.jpg - Similarity: 0.7185
4. Image garden_roses_00057.jpg - Similarity: 0.7145
5. Image garden_roses_00085.jpg - Similarity: 0.6574
6. Image garden_roses_00015.jpg - Similarity: 0.6488
7. Image garden_roses_00065.jpg - Similarity: 0.5973
8. Image garden_roses_00084.jpg - Similarity: 0.5592
9. Image garden_roses_00071.jpg - Similarity: 0.5211
10. Image garden_roses_00077.jpg - Similarity: 0.5194
==> Correct categories: 100.00%


Results for Features a2 and Metric b1:
1. Image garden_roses_00057.jpg - Similarity: 1.1592
2. Image garden_roses_00052.jpg - Similarity: 1.1346
3. Image garden_roses_00038.jpg - Similarity: 1.1337
4. Image garden_roses_00050.jpg - Similarity: 1.1130
5. Image garden_roses_00085.jpg - Similarity: 1.0361
6. Image garden_roses_00015.jpg - Similarity: 1.0075
7. Image garden_roses_00065.jpg - Similarity: 0.9416
8. Image garden_roses_00084.jpg - Similarity: 0.9342
9. Image garden_roses_00077.jpg - Similarity: 0.9117
10. Image garden_roses_00071.jpg - Similarity: 0.8776
==> Correct categories: 100.00%


Results for Features a2 and Metric b2:
1. Image garden_roses_00057.jpg - Similarity: 0.5645
2. Image garden_roses_00052.jpg - Similarity: 0.5538
3. Image garden_roses_00038.jpg - Similarity: 0.5491
4. Image garden_roses_00050.jpg - Similarity: 0.5425
5. Image garden_roses_00015.jpg - Similarity: 0.4727
6. Image garden_roses_00065.jpg - Similarity: 0.4710
7. Image garden_roses_00084.jpg - Similarity: 0.4418
8. Image garden_roses_00077.jpg - Similarity: 0.4332
9. Image garden_roses_00085.jpg - Similarity: 0.4304
10. Image garden_roses_00071.jpg - Similarity: 0.4133
==> Correct categories: 100.00%

======================== Query Image: peonies_00015.jpg ========================

Results for Features a1 and Metric b1:
1. Image garden_roses_00065.jpg - Similarity: 1.3917
2. Image peonies_00018.jpg - Similarity: 1.2576
3. Image hibiscus_00029.jpg - Similarity: 1.1909
4. Image peonies_00072.jpg - Similarity: 1.0349
5. Image peonies_00028.jpg - Similarity: 1.0181
6. Image daisies_00053.jpg - Similarity: 0.9600
7. Image peonies_00036.jpg - Similarity: 0.9317
8. Image peonies_00044.jpg - Similarity: 0.9317
9. Image peonies_00040.jpg - Similarity: 0.8869
10. Image peonies_00048.jpg - Similarity: 0.8869
==> Correct categories: 70.00%


Results for Features a1 and Metric b2:
1. Image hibiscus_00048.jpg - Similarity: 0.2179
2. Image peonies_00068.jpg - Similarity: 0.1777
3. Image peonies_00058.jpg - Similarity: 0.1621
4. Image orchids_00055.jpg - Similarity: 0.1179
5. Image daisies_00041.jpg - Similarity: 0.1164
6. Image lilies_00083.jpg - Similarity: 0.1106
7. Image peonies_00018.jpg - Similarity: 0.1096
8. Image daisies_00084.jpg - Similarity: 0.0961
9. Image peonies_00072.jpg - Similarity: 0.0920
10. Image peonies_00080.jpg - Similarity: 0.0902
==> Correct categories: 50.00%
```

```
Results for Features a2 and Metric b1:
1. Image garden_roses_00085.jpg - Similarity: 0.9999
2. Image orchids_00020.jpg - Similarity: 0.7702
3. Image peonies_00037.jpg - Similarity: 0.6685
4. Image peonies_00045.jpg - Similarity: 0.6685
5. Image peonies_00038.jpg - Similarity: 0.6222
6. Image peonies_00046.jpg - Similarity: 0.6222
7. Image gardenias_00036.jpg - Similarity: 0.5907
8. Image peonies_00052.jpg - Similarity: 0.5827
9. Image hydrangeas_00022.jpg - Similarity: 0.5817
10. Image garden_roses_00067.jpg - Similarity: 0.5814
==> Correct categories: 50.00%


Results for Features a2 and Metric b2:
1. Image peonies_00058.jpg - Similarity: 0.1722
2. Image garden_roses_00058.jpg - Similarity: 0.1182
3. Image peonies_00068.jpg - Similarity: 0.1151
4. Image peonies_00037.jpg - Similarity: 0.1045
5. Image peonies_00045.jpg - Similarity: 0.1045
6. Image hibiscus_00003.jpg - Similarity: 0.1023
7. Image peonies_00038.jpg - Similarity: 0.0997
8. Image peonies_00046.jpg - Similarity: 0.0997
9. Image hydrangeas_00028.jpg - Similarity: 0.0996
10. Image garden_roses_00036.jpg - Similarity: 0.0909
==> Correct categories: 60.00%

======================== Query Image: tulip_00052.jpg ========================

Results for Features a1 and Metric b1:
1. Image tulip_00017.jpg - Similarity: 1.5083
2. Image tulip_00041.jpg - Similarity: 1.2922
3. Image tulip_00046.jpg - Similarity: 1.2922
4. Image tulip_00056.jpg - Similarity: 1.1726
5. Image peonies_00072.jpg - Similarity: 1.1494
6. Image tulip_00074.jpg - Similarity: 1.1375
7. Image tulip_00079.jpg - Similarity: 1.1330
8. Image tulip_00006.jpg - Similarity: 1.1161
9. Image gardenias_00006.jpg - Similarity: 1.1014
10. Image orchids_00052.jpg - Similarity: 1.0689
==> Correct categories: 70.00%


Results for Features a1 and Metric b2:
1. Image tulip_00056.jpg - Similarity: 0.5706
2. Image garden_roses_00077.jpg - Similarity: 0.5214
3. Image tulip_00033.jpg - Similarity: 0.3585
4. Image tulip_00079.jpg - Similarity: 0.3555
5. Image tulip_00059.jpg - Similarity: 0.2777
6. Image gardenias_00036.jpg - Similarity: 0.2530
7. Image tulip_00074.jpg - Similarity: 0.2223
8. Image tulip_00036.jpg - Similarity: 0.1628
9. Image orchids_00012.jpg - Similarity: 0.1509
10. Image tulip_00063.jpg - Similarity: 0.1305
==> Correct categories: 70.00%


Results for Features a2 and Metric b1:
1. Image orchids_00057.jpg - Similarity: 1.2738
2. Image tulip_00056.jpg - Similarity: 0.9938
3. Image tulip_00035.jpg - Similarity: 0.7959
4. Image tulip_00005.jpg - Similarity: 0.7903
5. Image orchids_00033.jpg - Similarity: 0.7861
6. Image tulip_00033.jpg - Similarity: 0.7443
7. Image tulip_00079.jpg - Similarity: 0.7347
8. Image tulip_00023.jpg - Similarity: 0.7316
9. Image hydrangeas_00033.jpg - Similarity: 0.6617
10. Image bougainvillea_00046.jpg - Similarity: 0.6372
==> Correct categories: 60.00%


Results for Features a2 and Metric b2:
1. Image tulip_00056.jpg - Similarity: 0.4967
2. Image tulip_00079.jpg - Similarity: 0.3266
3. Image orchids_00020.jpg - Similarity: 0.3176
4. Image tulip_00033.jpg - Similarity: 0.3074
5. Image orchids_00005.jpg - Similarity: 0.2889
6. Image tulip_00059.jpg - Similarity: 0.2208
7. Image tulip_00035.jpg - Similarity: 0.1825
8. Image tulip_00074.jpg - Similarity: 0.1524
9. Image gardenias_00077.jpg - Similarity: 0.1474
10. Image tulip_00005.jpg - Similarity: 0.1369
==> Correct categories: 70.00%

====================== Query Image: gardenias_00068.jpg ======================

Results for Features a1 and Metric b1:
```

```
1. Image tulip_00058.jpg - Similarity: 1.3573
2. Image gardenias_00081.jpg - Similarity: 1.2261
3. Image garden_roses_00058.jpg - Similarity: 1.1391
4. Image daisies_00062.jpg - Similarity: 1.1188
5. Image gardenias_00057.jpg - Similarity: 1.1134
6. Image gardenias_00011.jpg - Similarity: 1.0980
7. Image tulip_00054.jpg - Similarity: 1.0731
8. Image tulip_00075.jpg - Similarity: 1.0636
9. Image gardenias_00043.jpg - Similarity: 1.0562
10. Image gardenias_00049.jpg - Similarity: 1.0562
==> Correct categories: 50.00%


Results for Features a1 and Metric b2:
1. Image garden_roses_00085.jpg - Similarity: 0.6595
2. Image gardenias_00081.jpg - Similarity: 0.5555
3. Image gardenias_00043.jpg - Similarity: 0.3560
4. Image gardenias_00049.jpg - Similarity: 0.3560
5. Image orchids_00024.jpg - Similarity: 0.2743
6. Image gardenias_00036.jpg - Similarity: 0.2471
7. Image hibiscus_00057.jpg - Similarity: 0.2333
8. Image peonies_00068.jpg - Similarity: 0.1948
9. Image gardenias_00077.jpg - Similarity: 0.1704
10. Image gardenias_00011.jpg - Similarity: 0.1553
==> Correct categories: 60.00%


Results for Features a2 and Metric b1:
1. Image gardenias_00081.jpg - Similarity: 0.9111
2. Image gardenias_00036.jpg - Similarity: 0.7465
3. Image daisies_00063.jpg - Similarity: 0.7315
4. Image gardenias_00023.jpg - Similarity: 0.6760
5. Image gardenias_00043.jpg - Similarity: 0.6655
6. Image gardenias_00049.jpg - Similarity: 0.6655
7. Image tulip_00075.jpg - Similarity: 0.6367
8. Image gardenias_00054.jpg - Similarity: 0.6307
9. Image daisies_00079.jpg - Similarity: 0.5991
10. Image gardenias_00078.jpg - Similarity: 0.5850
==> Correct categories: 70.00%


Results for Features a2 and Metric b2:
1. Image gardenias_00081.jpg - Similarity: 0.4540
2. Image gardenias_00043.jpg - Similarity: 0.3067
3. Image gardenias_00049.jpg - Similarity: 0.3067
4. Image daisies_00063.jpg - Similarity: 0.2601
5. Image gardenias_00036.jpg - Similarity: 0.2531
6. Image gardenias_00011.jpg - Similarity: 0.1637
7. Image garden_roses_00058.jpg - Similarity: 0.1231
8. Image gardenias_00077.jpg - Similarity: 0.1183
9. Image hibiscus_00064.jpg - Similarity: 0.1079
10. Image gardenias_00023.jpg - Similarity: 0.1061
==> Correct categories: 70.00%
```

Παρατηρούμε ότι ανα κατηγορία, οι συνδυσμοί χαρακτηριστικού/μετρικής ***A|B*** κατα φθίνουσα σειρά επιτυχίας είναι:

1. **bougainvillea** → *A2|B2, A1|B1, A1|B2, A2|B1*
2. **lilies** → *A2|B2, A1|B2, A1|B1, A2|B1*
3. **hydrangeas** → *A1|B1, A1|B2, A2|B2, A2|B1*
4. **daisies** → *A1|B1, A1|B2, A2|B2, A2|B1*
5. **hibiscus** → *A2|B2, A1|B2, A1|B1, A2|B1*
6. **orchids** → *A1|B2, A2|B1, A1|B1, A2|B2*
7. **garden_roses** → *A1|B2, A2|B1, A2|B2, A1|B1*
8. **peonies** → *A1|B1, A2|B2, A1|B2, A2|B1*
9. **tulip** → *A1|B1, A1|B2, A2|B2, A2|B1*
10. **gardenias** → *A2|B1, A2|B2, A1|B2, A1|B1*

Συνολικά ο συνδυασμός ***A1|B1*** φαίνεται να έρχεται συχνότερα πρώτος