Worksheet 3: Morphological processing + Segmentation

Name: **Prerana Sanjay Kulkarni**
SRN: **PES1UG21CS451**
Unit 3
2024
***Submission format**:*
Make a copy of this doc -> type your answers/code + the output image after each question.

1. Using the x31_f18.tif image perform the following:

a. Use Erosion on the image and display.

```
img = imread('x31_f18.tif');
SE_erode = strel('disk',8);
eroded = imerode(img,SE_erode);
imshow(eroded);
```

b. Use Dilation on the image and display.

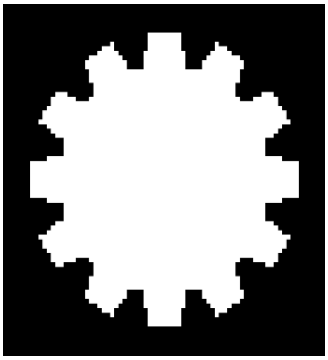
```
SE_dilate = strel('disk',5);
dilated = imdilate(img,SE_dilate);
imshow(dilated);
```

c. Are they inverses of each other? Justify your answer.

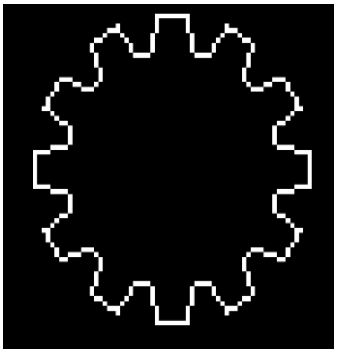
While erosion and dilation have opposing effects, they are not exactly inverses of each other. However, they are related by the principle of duality. We obtain the same result by eroding X or by dilating its complementary and taking the complementary of the resulting set. We say that Erosion and Dilatation are two dual operations according to the complementation.

Original Image:

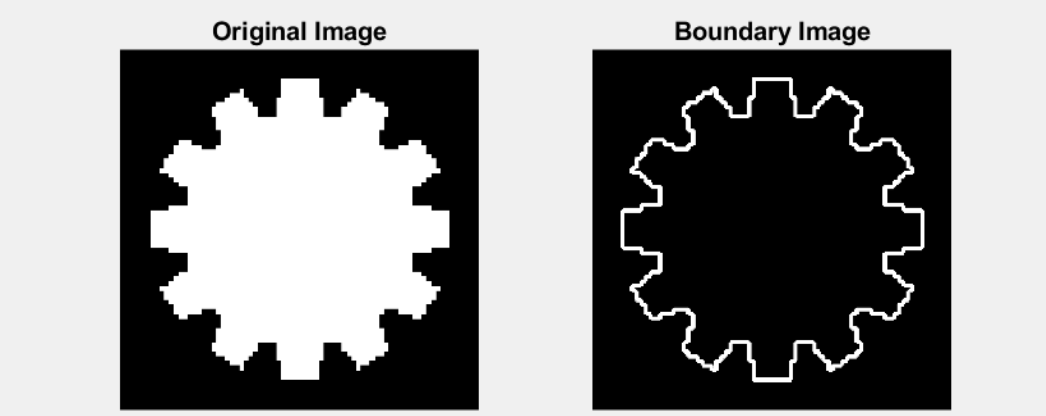
2. Extract the boundary from the FreemanCode.png



TO->

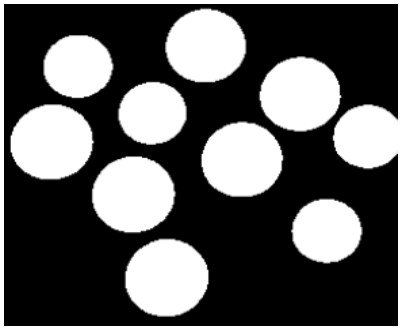


```
freemanCode  = imread('FreemanCode.png');
SE = strel('disk',2);
dilate = imdilate(freemanCode,SE);
erode = imerode(freemanCode,SE);
diff = dilate - erode;
figure;
subplot(1,2,1);
imshow(freemanCode);
title('Original Image');
subplot(1,2,2);
imshow(diff);
title('Boundary Image');
```

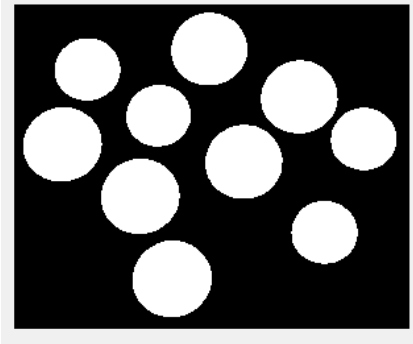


3. Using coins.png, binarize the image first and then fill the holes.

TO→



```
coins = imread('coins.png');
bin = imbinarize(coins);
fill = imfill(bin,'holes');
figure; imshow(fill);
```



4. Take the Japanese character in kawaii.png.

a. Thin the image to obtain the skeleton of the image

```
originalImage = imread('kawaii.png');
if size(originalImage, 3) == 3
    grayImage = rgb2gray(originalImage);
else
    grayImage = originalImage;
end

binaryImage = imbinarize(grayImage);
skeletonImage = bwmorph(binaryImage, 'thin', Inf);
imshow(skeletonImage);
title('Skeleton Image');
```

b. Upon thinning, have you noticed some parts are disconnected? Dilate the OG Kawaii image and then thin it.

```
dilatedImage = bwmorph(skeletonImage, 'dilate', 1);
thinnedImageAfterDilation = bwmorph(dilatedImage, 'thin', Inf);
imshow(thinnedImageAfterDilation);
title('thinned image');
```

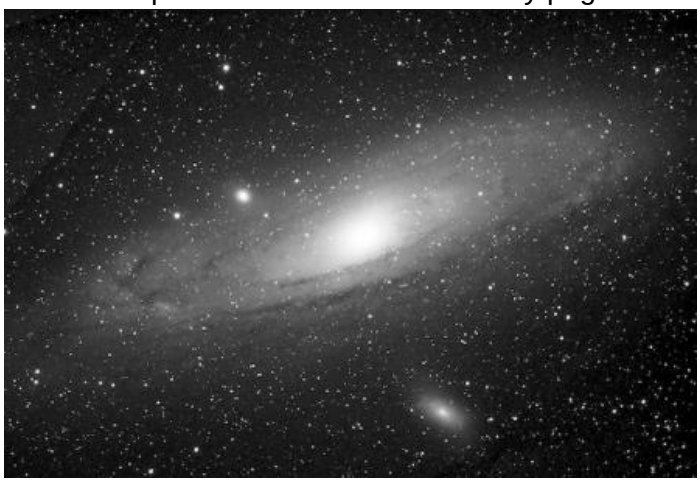
c. If you end up with spurs like the image below, use structural elements to fix:

```
cleanedImage = bwmorph(thinnedImageAfterDilation, 'spur', Inf);
imshow(cleanedImage);
title('Cleaned Image');
```

Try to obtain the image below using these morphological operators:

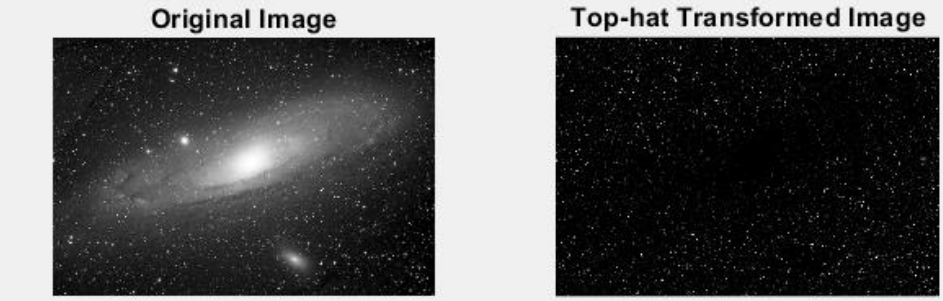
5. Given the following images:

a. Perform Top-hat transform on the Galaxy.png and observe the results.



```
image = imread('Galaxy.png');
if size(image, 3) == 3
    image = rgb2gray(image);
end
% Define the structuring element (SE)
% For example, a 3x3 square SE
SE = ones(3);
openedImage = imopen(image, SE);
topHatImage = image - openedImage;
figure;
subplot(1, 2, 1);
imshow(image);
```

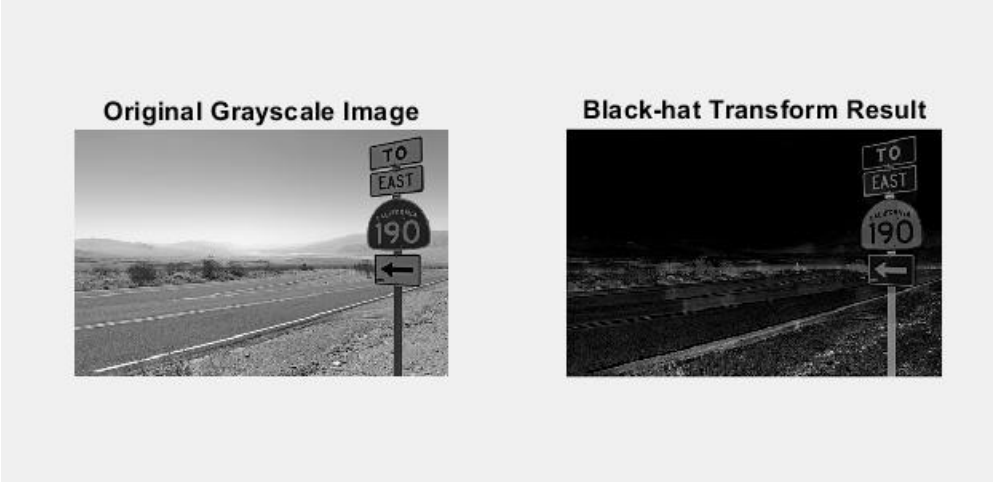
```
title('Original Image');
subplot(1, 2, 2);
imshow(topHatImage, [ ]);
title('Top-hat Transformed Image');
```



b. Perform Black-hat transform on the Cali.png and observe results.


```
originalImage = imread('Cali.png');
grayImage = rgb2gray(originalImage);
kernelSize = 15; % You can vary this to see different results
se = strel('rectangle', [kernelSize, kernelSize]);
blackhatImage = imbothat(grayImage, se);
subplot(1, 2, 1);
imshow(grayImage);
title('Original Grayscale Image');

subplot(1, 2, 2);
```

```
imshow(blackhatImage);
title('Black-hat Transform Result');
```



c. Looking at the result of these transforms, explain briefly why we use Top-hat and Black-hat transforms respectively (In context of the above images).

Top-hat Transform: The top-hat transform is employed to accentuate bright features within an image that are set against a darker background. For instance, when applied to an image of a galaxy, it illuminates the bright stars and finer details against the dark sky, facilitating tasks like star counting or distribution analysis without the interference of the galaxy's overall brightness.

Black-hat Transform: On the other hand, the black-hat transform is designed to spotlight dark elements within an image that are situated on a lighter background. When used on an image of a roadside with signs, it highlights the darker details against the bright areas, which is beneficial for enhancing the visibility of dark objects in bright scenes. This is crucial for image analysis in various contexts, including improving the legibility of signs in road scenes under different lighting conditions.

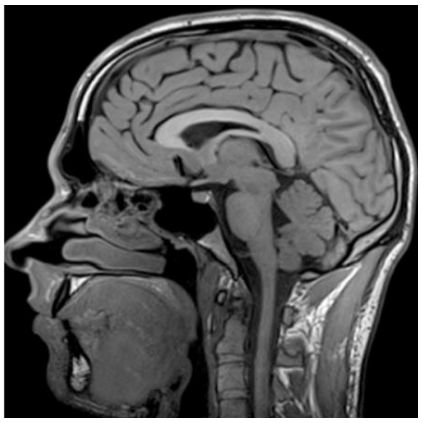
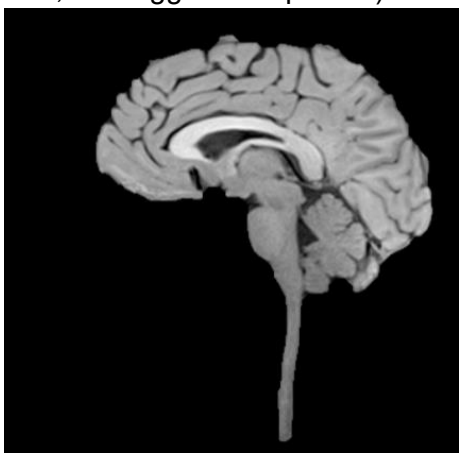
6. What is Image Segmentation?

Image segmentation is a technique in image processing and computer vision that divides an image into distinct segments or groups of pixels, each representing a different object or part of an object within the image. The primary objective of image segmentation is to transform or simplify the image representation into a form that is more meaningful and easier to analyze. This process is essential for numerous computer vision tasks, such as object recognition, image editing, and medical image analysis.

a. What are the different methods used?

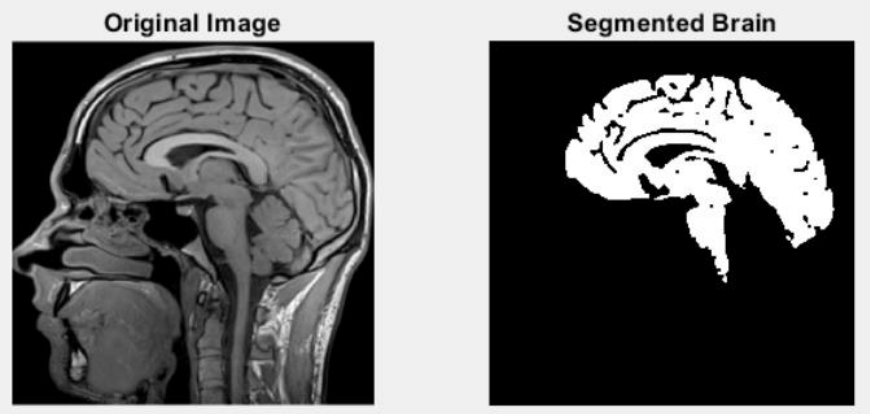
1. Thresholding: This technique divides an image into two segments based on a predetermined threshold value, making it suitable for images with distinct contrasts between the foreground and background.
2. Edge Detection: This method identifies the boundaries of objects within an image, serving as a foundational step in segmentation to isoslate objects before applying more sophisticated segmentation techniques.
3. Region Growing: Starting from a seed point, this technique expands a region by adding adjacent pixels that are similar to the seed point, proving effective for segmenting images with intricate shapes and textures.
4. Watershed Transform: This segmentation technique divides an image based on the relative topography of the image, particularly adept at separating objects that are interconnected by thin bridges.
5. Clustering: This approach groups pixels into clusters based on their similarity in color, intensity, or other characteristics, often used in conjunction with other segmentation methods to refine segmentation outcomes.

b. Using segmentation, extract only the brain from scan.png. (Hint: binerize the image, use operations like ‘open’ and ‘close’, find biggest component)


TO ->


```
image = imread('scan.png');
if size(image, 3) == 3
    image = rgb2gray(image);
end

% Binarize the image
% Assuming the brain is the brightest part of the image
threshold = 100;
binaryImage = image > threshold;
% Define the structuring element (SE) for morphological operations
% For example, a 3x3 square SE
SE = ones(3);
openedImage = imopen(binaryImage, SE);
closedImage = imclose(openedImage, SE);
```

```
% Use bwconncomp to find connected components
cc = bwconncomp(closedImage);

% Calculate the area of each connected component
areas = regionprops(cc, 'Area');

% Find the index of the largest component
[~, maxIndex] = max([areas.Area]);

% Extract the indices of the pixels belonging to the largest component
largestComponentIndices = cc.PixelIdxList{maxIndex};

% Create a binary mask for the largest component
largestComponentMask = false(size(binaryImage));
largestComponentMask(largestComponentIndices) = true;

% Display the original and the segmented brain
figure;
subplot(1, 2, 1);
imshow(image);
title('Original Image');

subplot(1, 2, 2);
imshow(largestComponentMask);
title('Segmented Brain');
```

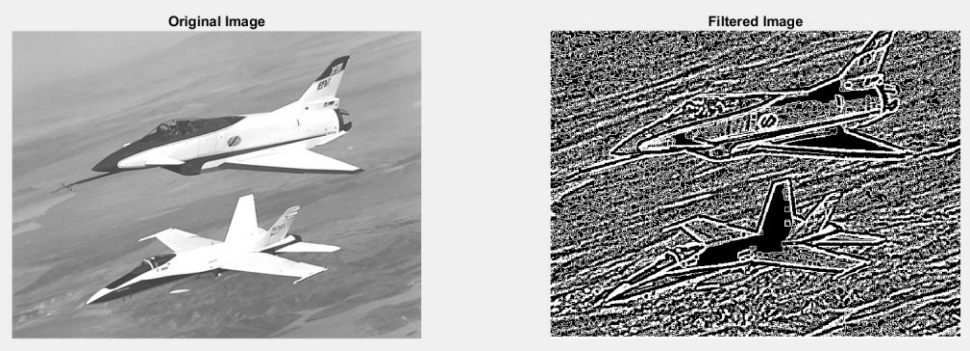


7. Edge Detection

a. Using Marr-hildreth edge detector, give the edge output of the x31_f18.tif image.

```
sigma = 2.5
Gaussian_filter = fspecial('gaussian', [5 5], sigma);
Gaussian_img = imfilter(img, Gaussian_filter, 'same');
Laplacian = [-1 -1 -1; -1 8 -1; -1 -1 -1];
Laplacian_img = conv2(Gaussian_img, Laplacian, 'same');
edges = Laplacian_img

figure;
```

```
subplot(1,2,1); imshow(img); title('Original Image');
subplot(1,2,2); imshow(Laplacian_img); title('Filtered Image');
```



b. Do the same with Canny edge detection.

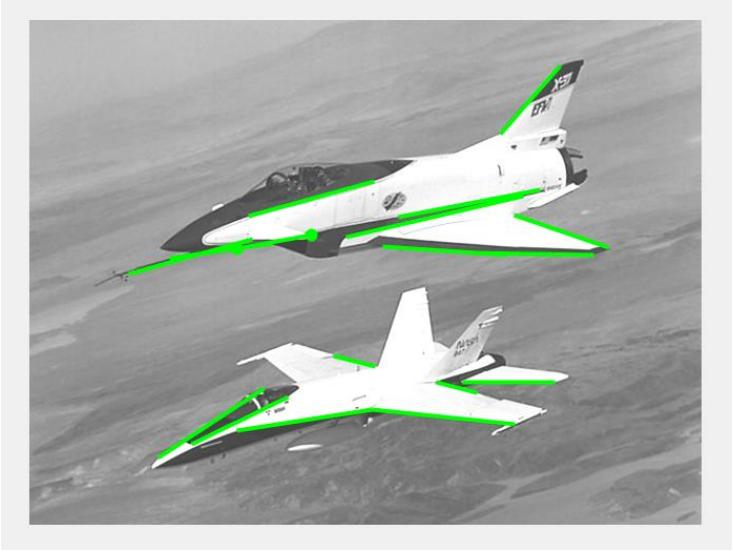
```
edges_canny = edge(img, 'canny', 0.5, 1.2);
imshow(edges_canny);
title('Canny Filter');
```

8. What is Hough Transform and why is it really good at filling up gaps?

The Hough Transform is designed to detect imperfect instances of objects within a certain class of shapes by using a voting procedure in a parameter space. This voting procedure helps in identifying object candidates as local maxima in an accumulator space, which is explicitly constructed by the algorithm for computing the Hough Transform.

a. Using hough Transform, find the edges of x31_f18.tif image.

```
% Perform the Hough Transform on the output of the Canny Edge Detection
% algorithm from the previous problem
[H,theta,rho] = hough(edges_canny);
P = houghpeaks(H, 50);

% Find lines corresponding to the peaks
lines = houghlines(edges_canny,theta,rho,P);

% Display
figure;
imshow(img);
hold on;
% Plot the edges
plot(lines(1).point1(1), lines(1).point1(2), 'g*', 'LineWidth', 2);
plot(lines(1).point2(1), lines(1).point2(2), 'g*', 'LineWidth', 2);


% Plot the lines
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1), xy(:,2), 'LineWidth', 2, 'Color', 'green');
end
hold off;
```



b. Use thresholding to refine the image if needed.

```
threshold = graythresh(img);
BW = imbinarize(img, threshold);
BW = edge(BW, 'canny', 0.5, 1.2);
[H,theta,rho] = hough(BW);
P = houghpeaks(H, 50);
lines = houghlines(BW,theta,rho,P);
figure;
imshow(img);
```

```
hold on;
% Plot the edges
plot(lines(1).point1(1), lines(1).point1(2), 'g*', 'LineWidth', 2);
plot(lines(1).point2(1), lines(1).point2(2), 'g*', 'LineWidth', 2);

% Plot the lines
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1), xy(:,2), 'LineWidth', 2, 'Color', 'green');
end
hold off;
```

9. Thresholding

a. Using Simple, global Thresholding, find output on cameraman.tif

```
cameraman = imread('cameraman.tif');

% Simple Global Thresholding
level = graythresh(cameraman);
bw_simple = imbinarize(cameraman, level);
```

b. Do the same with global Thresholding.

```
% Global Thresholding
threshold_value = 128;
bw_global = cameraman > threshold_value;
```

c. Do the same with Otsu's method.

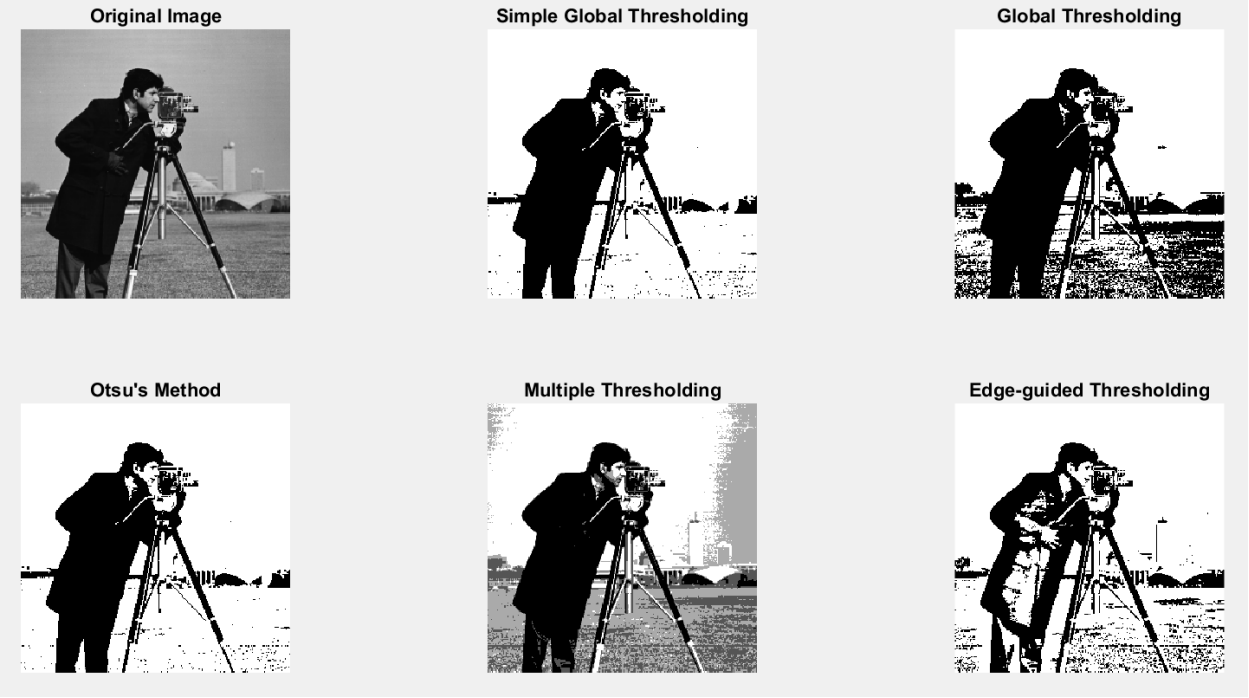
```
% Otsu's Method
bw_otsu = imbinarize(cameraman, level);
```

d. Do the same with multiple and variable methods.

```
% Multiple Thresholding
numThresholds = 3;
thresholds = multithresh(cameraman, numThresholds);
binaryImage = imquantize(cameraman, thresholds);
```

e. Explore Edge-guided Thresholding.

```
% Edge-guided Thresholding
edges = edge(cameraman, 'sobel');
bw_edge_guided = imbinarize(cameraman, 'adaptive', 'ForegroundPolarity',
'dark', 'Sensitivity', 0.5);

% Create a figure with subplots
figure;

subplot(2, 3, 1);
imshow(cameraman);
title('Original Image');

subplot(2, 3, 2);
imshow(bw_simple);
title('Simple Global Thresholding');

subplot(2, 3, 3);
imshow(bw_global);
title('Global Thresholding');

subplot(2, 3, 4);
imshow(bw_otsu);
title('Otsu''s Method');

subplot(2, 3, 5);
imshow(binaryImage, []);
title('Multiple Thresholding');

subplot(2, 3, 6);
imshow(bw_edge_guided);
title('Edge-guided Thresholding');
```

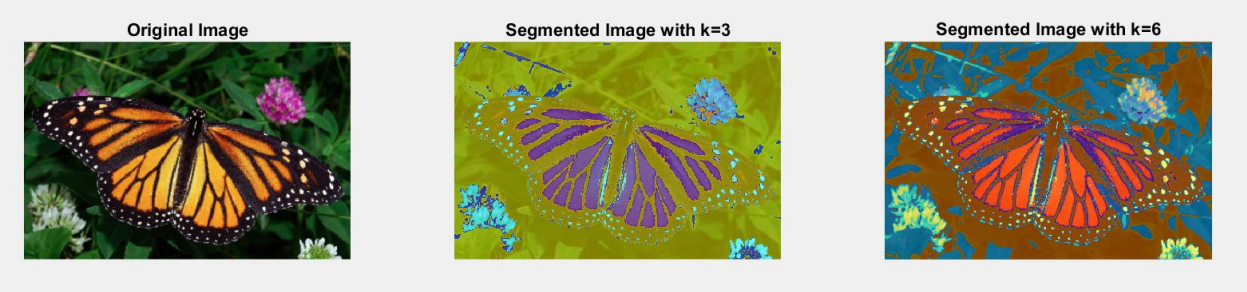


10. Perform K-means clustering segmentation on the butterfly.png.

a. Put the value of k=3.
b. Put the value of k=6.
c. Observe and report the differences.

```
% Read the original image
butterfly = imread('butterfly.png');

% Perform K-means clustering with k=3
[L_k3, Centers_k3] = imsegkmeans(butterfly, 3);

% Perform K-means clustering with k=6
[L_k6, Centers_k6] = imsegkmeans(butterfly, 6);

% Display the original image
subplot(1, 3, 1);
imshow(butterfly);
title('Original Image');

% Display the segmented image with k=3
subplot(1, 3, 2);
B_k3 = labeloverlay(butterfly, L_k3);
imshow(B_k3);
title('Segmented Image with k=3');

% Display the segmented image with k=6
subplot(1, 3, 3);
```

```
B_k6 = labeloverlay(butterfly, L_k6);
imshow(B_k6);
title('Segmented Image with k=6');
```

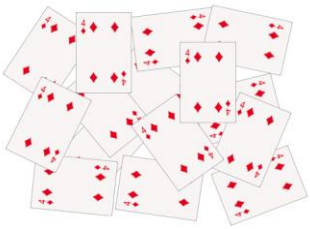
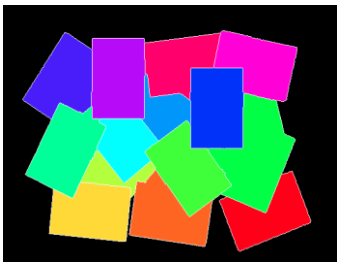


11. What is watershed segmentation and why is it used where regions generally touch?

Watershed segmentation is a technique used in image processing for separating touching objects in an image. It works by treating the image as a topographic surface where high intensity (light pixesls) represents peaks and hills, and low intensity (dark pixels) represents valleys. The algorithm starts by filling isolated valleys (local minima) with different colored water (labels). As the water rises, depending on the peaks (gradients) nearby, water from different valleys with different colors will start to merge. To prevent this merging, barriers are built in locations where water merges, which are the boundaries of the objects. This process continues until all peaks are under water, and the barriers created give the segmentation result.

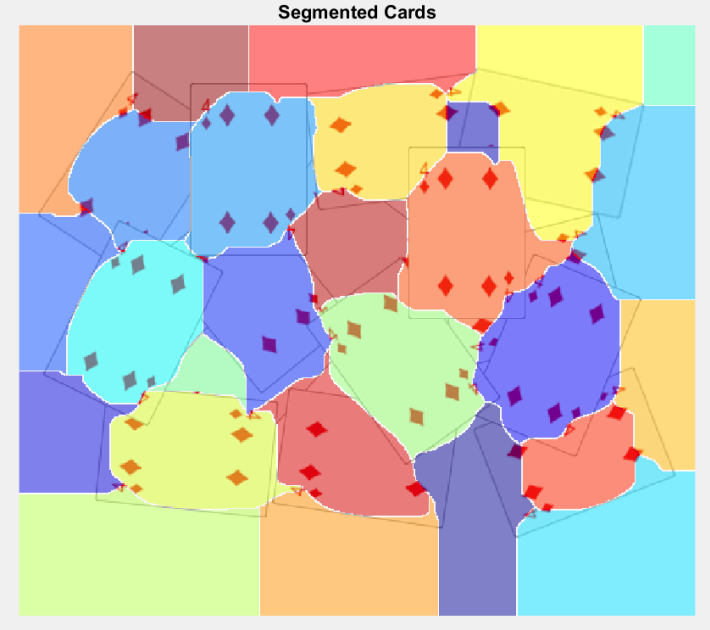
The watershed algorithm is particularly useful in scenarios where objects touch or overlap, making it difficult to segment them using simpler methods like thresholding and contour detection. These methods might treat a group of touching objects as a single object rather than multiple objects. The watershed algorithm, on the other hand, can detect and extract each individual object, even if they are touching or overlapping.

a. Using Cards.png, segment the cards.

 TO 

(Example is indicative. You can show the results in any way you like)

```
% Load the image
im = imread('Cards.png');

% Convert the image to grayscale
gray = rgb2gray(im);

% Apply Otsu's thresholding
bw = imbinarize(gray);

% Compute the Euclidean Distance Transform
D = bwdist(~bw);

% Blur the distance transform to reduce noise
sigma = 10;
kernel = fspecial('gaussian', 4*sigma+1, sigma);
D_blurred = imfilter(D, kernel, 'symmetric');

% Apply the watershed algorithm
L = watershed(max(D_blurred(:)) - D_blurred);

% Display the segmented image
imshow(labeloverlay(im, L));
title('Segmented Cards');
```


Segmented Cards