Concordia University

Department of Computer Science & Software Engineering

COMP 478/6771 Image Processing

Assignment 4

Name: Mohammad Al-Shariar

ID: 40263705

Date:02-12-2024

Q.1

$P_1m_1 + P_2m_2 = m_G$ —------ equation 1

$P_1 + P_2 = 1$ —------ equation 2

$\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2$ —------ equation 3

Use the value of m_G in equation 3,

$\sigma_B^2 = P_1(m_1 - (P_1m_1 + P_2m_2))^2 + P_2(m_2 - (P_1m_1 + P_2m_2))^2$ —------ equation 4

Simplify $(m_1 - m_G)$ and $(m_2 - m_G)$:

For $(m_1 - m_G)$,

$$m_1 - m_G = m_1 - (P_1m_1 + P_2m_2) = P_2(m_1 - m_2)$$

For $(m_2 - m_G)$,

$$m_2 - m_G = m_2 - (P_1m_1 + P_2m_2) = P_1(m_2 - m_1)$$

From equation 4,

$$\sigma_B^2 = P_1(P_2(m_1 - m_2))^2 + P_2(P_1(m_2 - m_1))^2$$

Since $(m_1 - m_2)^2 = (m_2 - m_1)^2$:

$$\sigma_B^2 = P_1P_2^2(m_1 - m_2)^2 + P_2P_1^2(m_1 - m_2)^2$$

$$\sigma_B^2 = P_1P_2(P_1 + P_2)\ (m_1 - m_2)^2$$

From equation 2 ($P_1 + P_2 = 1$):

$$\sigma_B^2 = P_1P_2\ (m_1 - m_2)^2$$

Q.2

A structural element sliding across the binary image I is called erosion. The structuring element must entirely fit inside the foreground pixels of I in order for a pixel to stay 1 (foreground) in the eroding output.

Erosion Rules:

The output's center pixel stays at 1 if the structuring element fits perfectly (all of its 1s overlap with the foreground in I).

The output's center pixel becomes 0 if the structuring element does not fit.

Regions outside of I are padded with 1s (foreground) to manage border effects.

The Binary Image I(Given in the question):
I = $[(0\ 0\ 0\ 0\ 0)(0\ 1\ 1\ 1\ 0)(0\ 1\ 1\ 1\ 0)(0\ 1\ 1\ 1\ 0)(0\ 0\ 0\ 0\ 0)]$

This displays a binary image in which the background pixels (0) around the foreground pixels (1), which form a central block.

Now,
b = $[(1\ 1\ 1)(1\ 1\ 1)(1\ 1\ 1)]$

- All of the elements in the 3x3 square structural element b are 1.
- For the center pixel to be white after erosion, this structural element requires that all nine pixels in a 3x3 neighborhood be white (foreground).

c = $[(0\ 1\ 0)(1\ 1\ 1)(0\ 1\ 0)]$

- c is a cross-shaped structuring element.
- It only checks 5 pixels (the center pixel and its 4 immediate neighbors) for white pixels.
- This structuring element is less restrictive compared to b, as fewer pixels need to match.

The provided binary picture To deal with the border effect, I is padded with 1s outside the boundary. The structuring element can fully analyze pixels close to the edges thanks to padding.

Padding the Image:
$I_{padded} = [(1\ 1\ 1\ 1\ 1\ 1\ 1)(1\ 0\ 0\ 0\ 0\ 0\ 1)(1\ 0\ 1\ 1\ 1\ 0\ 1)(1\ 0\ 1\ 1\ 1\ 0\ 1)\ (1\ 0\ 1\ 1\ 1\ 0\ 1)(1\ 0\ 0\ 0\ 0\ 0\ 1)(1\ 1\ 1\ 1\ 1\ 1\ 1)]$

Erosion with b:

- All 9 nearby pixels must overlap with the foreground pixels in the 3x3 square b (1).
- Regarding I's central pixels:
  - For instance: In I (the block's center), pixel (3,3): In the padding image, the neighborhood: $[(1\ 1\ 1)\ (1\ 1\ 1)\ (1\ 1\ 1)]$

Here, b does not fit because there are 0s in the top-left corner of the neighborhood (from the padding).

- As b requires all 9 pixels to fit, no pixel in I satisfies the condition. Thus, the eroded output is:

  Eroded I using b:

  $I_{eroded\_b} = [(0\ 0\ 0\ 0\ 0)(0\ 0\ 0\ 0\ 0)(0\ 0\ 0\ 0\ 0)(0\ 0\ 0\ 0\ 0)(0\ 0\ 0\ 0\ 0)]$

Erosion with c:

- Just 5 pixels are needed for the cross c to overlap with the foreground (the centre and its four neighbors).
- For the pixels in the center of I:

- Example: Pixel (3,3) in I: Neighbourhood in the padded image:
  [(1 1 1) (1 1 1) (1 1 1)]

  In this case, c fits since the center and cross neighbors—the necessary 5 pixels—are 1.

- Likewise, in I, c fits all pixels except those that are close to the core block's boundaries, which will erode to 0.

Eroded I using c:

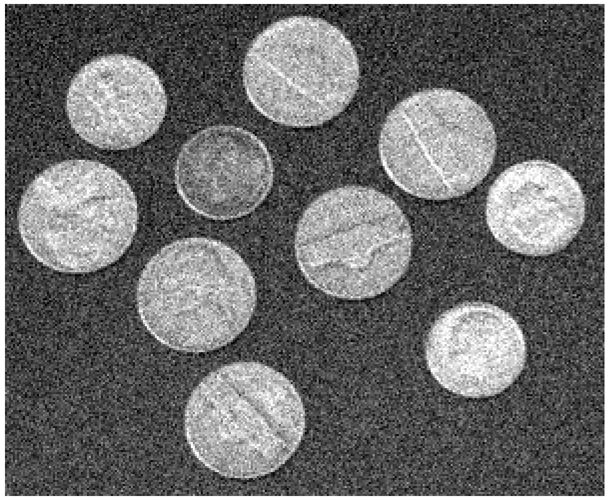
$$I_{eroded\_c} = [(0\ 0\ 0\ 0\ 0)(0\ 0\ 1\ 0\ 0)\ (0\ 1\ 1\ 1\ 0)(0\ 0\ 1\ 0\ 0)(0\ 0\ 0\ 0\ 0)]$$

Code:

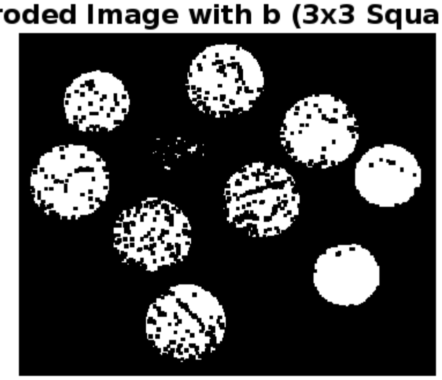
```
% Load the binary image
img = imread('noisycoin.png');
img = im2bw(img);
b = [1 1 1; 1 1 1; 1 1 1];
c = [0 1 0; 1 1 1; 0 1 0];
% Apply erosion with structuring element b
eroded_b = imerode(img, b);
% Apply erosion with structuring element c
eroded_c = imerode(img, c);
figure;
imshow(img);
title('Original Image');
figure;
subplot(1, 2, 1);
imshow(eroded_b);
title('Eroded Image with b (3x3 Square)');
subplot(1, 2, 2);
imshow(eroded_c);
title('Eroded Image with c (Cross)');
```

Output:

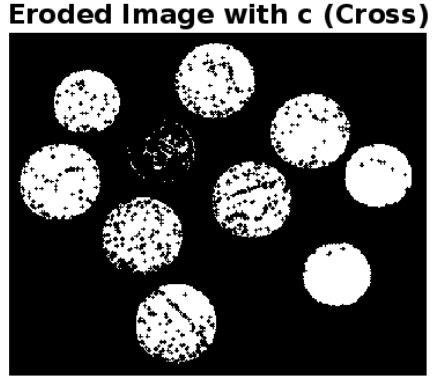
Original Image



Eroded Image with b (3x3 Square)



Eroded Image with c (Cross)

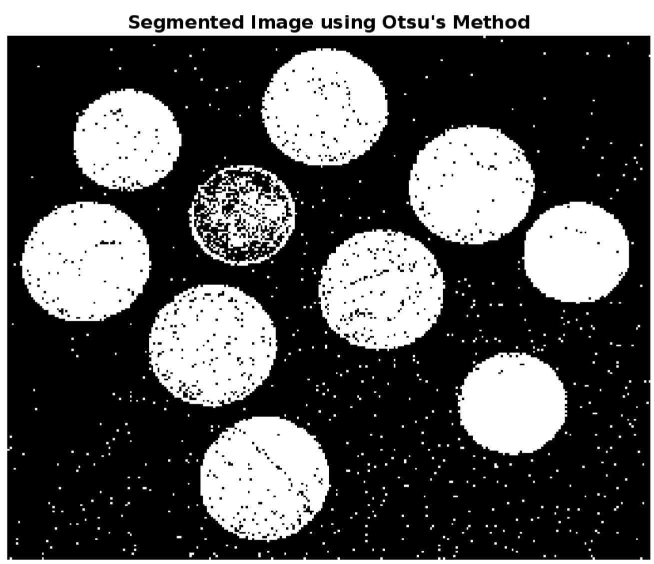


Part ii: Programming questions

Q.1(a)

```
% Load the noisy coin image
img = imread('noisycoin.png');
img_gray = im2double(img);
% Display the original image
figure;
imshow(img_gray);
title('Original Noisy Coin Image');
%% Part (a): Apply Otsu's Algorithm to Segment the Coins
otsu_thresh = graythresh(img_gray); % Compute the threshold
using Otsu's method
binary_otsu = imbinarize(img_gray, otsu_thresh); % Segment the
image
% Display the segmented image
figure;
imshow(binary_otsu);
title('Segmented Image using Otsu's Method');
```

Output:

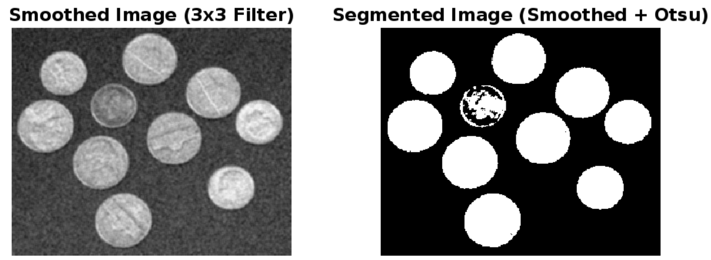


Q.1(b)

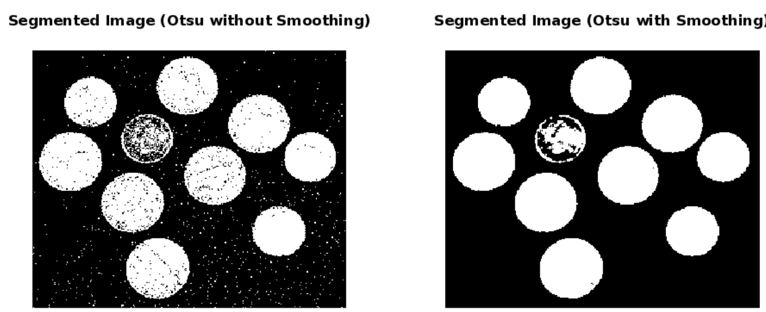
```
%% Part (b): Smooth the Image with a 3x3 Averaging Filter, then
Apply Otsu's Algorithm
% Apply a 3x3 averaging filter
```

```
filter = fspecial('average', [3 3]); % Create a 3x3 averaging
filter
smoothed_img = imfilter(img_gray, filter, 'replicate'); % Smooth
the image
% Apply Otsu's algorithm on the smoothed image
otsu_thresh_smooth = graythresh(smoothed_img); % Compute the
threshold
binary_otsu_smooth = imbinarize(smoothed_img,
otsu_thresh_smooth); % Segment
% Display the smoothed image and segmentation result
figure_handle2 = figure;
subplot(1, 2, 1);
imshow(smoothed_img);
title('Smoothed Image (3x3 Filter)');
subplot(1, 2, 2);
imshow(binary_otsu_smooth);
title('Segmented Image (Smoothed + Otsu)');
saveas(figure_handle2, 'Smoothed Image and Segmented Image
together.png');
% Comparison of Part (a) and Part (b) results
figure_handle3 = figure;
subplot(1, 2, 1);40.
imshow(binary_otsu);
title('Segmented Image (Otsu without Smoothing)', 'FontSize', 8,
'Position', [size(binary_otsu, 2) / 2, -20]);
subplot(1, 2, 2);
imshow(binary_otsu_smooth);
title('Segmented Image (Otsu with Smoothing)', 'FontSize', 8,
'Position', [size(binary_otsu_smooth, 2) / 2, -20]);
saveas(figure_handle3, 'Comparison of Part (a) and Part (b)
Results.png');
```

Output:



Comparison of Part (a) and Part (b) Results:

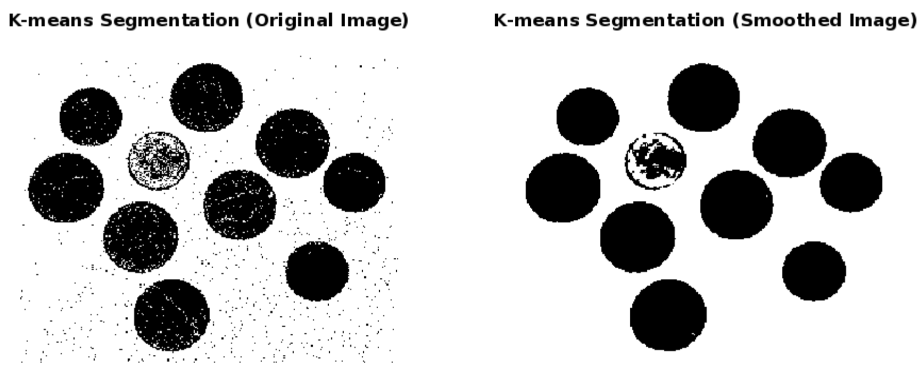


Q.1(c)

```
%% Part (c): K-means Segmentation
% Ensure the image is of type single
img_gray_single = im2single(img_gray); % Convert to single
precision
% Apply K-means segmentation on the original image
```

```
L_original = imsegkmeans(img_gray_single, 2); % Segment into 2
clusters
seg_kmeans_original = (L_original == 1); % Extract one cluster
% Apply K-means segmentation on the smoothed image
smoothed_img_single = im2single(smoothed_img); % Convert
smoothed image to single precision
L_smoothed = imsegkmeans(smoothed_img_single, 2); % Segment into
2 clusters
seg_kmeans_smoothed = (L_smoothed == 1); % Extract one cluster
% Display K-means segmentation results
figure_handle4 = figure;
subplot(1, 2, 1);
imshow(seg_kmeans_original);
title('K-means Segmentation (Original Image)' , 'FontSize', 8,
'Position', [size(binary_otsu, 2) / 2, -20]);
subplot(1, 2, 2);
imshow(seg_kmeans_smoothed);
title('K-means Segmentation (Smoothed Image)', 'FontSize', 8,
'Position', [size(binary_otsu, 2) / 2, -20]);
saveas(figure_handle4, 'K-means Segmentation.png');
```

Output:

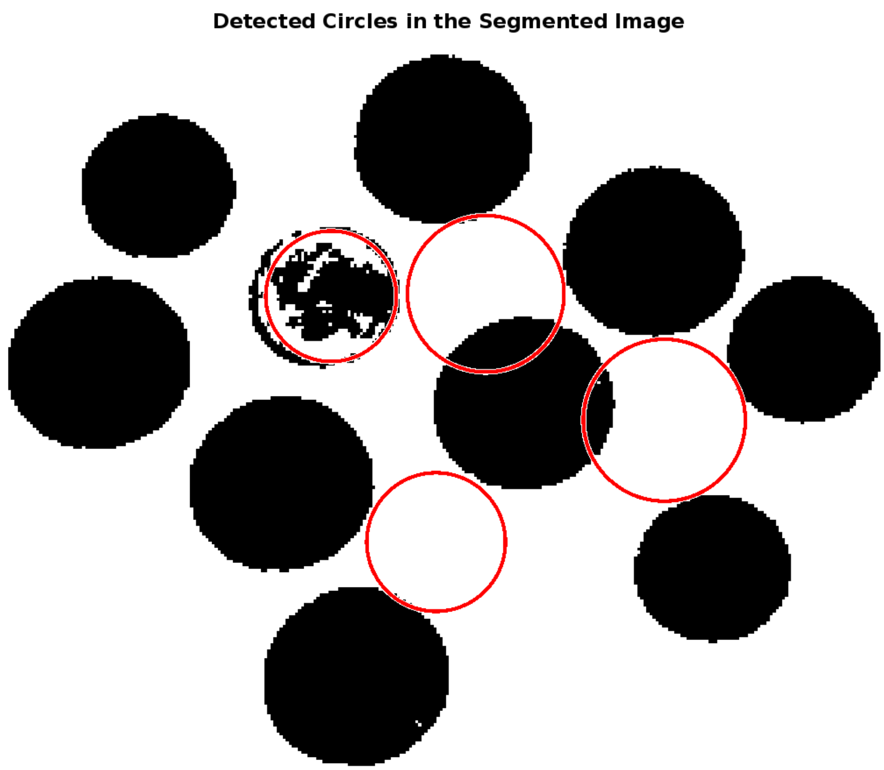


Q.1(d)

```
%% Part (d): Detect Circles Using Hough Transform
% Input: Segmentation results from the denoised image (Part c)
segmented_denoised_image = seg_kmeans_smoothed; % Use the result
from Part (c)
% Detect circles in the segmented image
[centers, radii] = imfindcircles(segmented_denoised_image, [20
50], 'Sensitivity', 0.95);
% Check if circles are detected
if isempty(centers)
   disp('No circles were detected. Try improving segmentation or
adjusting the radius range.');
else
   % Visualize the detected circles
   figure_handle5 = figure;
   imshow(segmented_denoised_image);
   title('Detected Circles in the Segmented Image');
   hold on;
   viscircles(centers, radii, 'EdgeColor', 'r'); % Overlay
detected circles
   hold off;
```

```
    saveas(figure_handle5, 'Detecting Circles in the Segmented
Image.png');
    % Display the centroid coordinates and radii
    disp('Centroid coordinates and radii of detected circles:');
    disp(table(centers(:, 1), centers(:, 2), radii, ...
        'VariableNames', {'X_Center', 'Y_Center', 'Radius'}));
end
```

Output:



The End