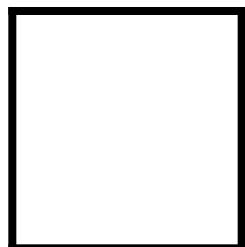


### **Elective 3**

#### **Laboratory Activity No. 5 Image Segmentation**



Score

*Submitted by:*

**Abarrientos, Ramuel R.  
Cazon, Iyhana Nicole A.  
Reyes, John Alfred J.  
Verzosa, Cristina Andrea B.  
Vicente, Honesto E.**

**Saturday – 7:00 am – 4:00pm / CPE 0332.1-1**

*Date Submitted*

**12-08-2024**

*Submitted to:*

**Engr. Maria Rizette H. Sayo**

## I. Objectives

This laboratory activity aims to implement the principles and techniques of image segmentation through MATLAB/Octave and open CV using Python

1. Acquire the image.
2. Show image Segmentation.
3. Show threshold techniques.

## II. Methods

### A. Perform a task given in the presentation

- Copy and paste your MATLAB code (use the original picture file: flower.jpg)

```
% Global Image thresholding using Otsu's method
% load image
% img = imread('original image');

% % calculate threshold using graythresh
% level = graythresh(img);

% % convert into binary image using the computed threshold
% bw = imbinarize(img, level);

% % display the original image and the binary image

% figure(1);
imshowpair(img, bw, 'montage');
title('Original Image (left) and Binary Image (right)');

% % Multi-level thresholding using Otsu's method
% % calculate single threshold using multithresh
% level = multithresh(img);

% % Segment the image into two regions using the imquantize function, specifying the threshold level
% returned by the multithresh function.
% seg_img = imquantize(img,level);

% % Display the original image and the segmented image
% figure(2);
imshowpair(img,seg_img,'montage');
title('Original Image (left) and Segmented Image (right)');

% Global histogram threshold using Otsu's method
% Calculate a 16-bin histogram for the image
% [counts,x] = imhist(img,16);
% stem(x,counts)

% % Compute a global threshold using the histogram counts
% T = otsuthresh(counts);
```

```

% % Create a binary image using the computed threshold and display the image
% bw = imbinarize(img,T);
% figure(3);
imshow(bw);
title('Binary Image');

% %
% % 2. Region-based segmentation

% Using K means clustering
% img2 = imread('paris.jpg');

% % Convert the image to grayscale
% bw_img2 = im2gray(img2);
% imshow(bw_img2);

% % Segment the image into three regions using k-means clustering
% [L, centers] = imsegkmeans(bw_img2,3);
% B = labeloverlay(bw_img2,L);
% imshow(B);
title('Labeled Image');

% % using connected-component labeling
% convert the image into binary
% bin_img2 = imbinarize(bw_img2);

% % Label the connected components
% [labeledImage, numberOfComponents] = bwlabel(bin_img2);

% % Display the number of connected components
% disp(['Number of connected components: ', num2str(numberOfComponents)]);

% % Assign a different color to each connected component
% coloredLabels = label2rgb(labeledImage, 'hsv', 'k', 'shuffle');

% % Display the labeled image
% figure(5);
% imshow(coloredLabels);
title('Labeled Image');

% %

% % Paramter Modifications

% adding noise to the image then segmenting it using otsu's method
% img_noise = imnoise(img,'salt & pepper',0.09);

% % calculate single threshold using multithresh
% level = multithresh(img_noise);

% % Segment the image into two regions using the imquantize function, specifying the threshold level
% returned by the multithresh function.
% seg_img = imquantize(img_noise,level);

```

```

% % Display the original image and the segmented image
% figure(6);
imshowpair(img_noise,seg_img,'montage');
title('Original Image (left) and Segmented Image with noise (right)');

% % Segment the image into two regions using k-means clustering RGB = imread('paris.jpg');

L = imsegkmeans(RGB,2); B = labeloverlay(RGB,L);
figure(7);
imshow(B);
title('Labeled Image');

% Create a set of 24 Gabor filters, covering 6 wavelengths and 4 orientations wavelength = 2.^0:5)*3;
orientation = 0:45:135;
g = gabor(wavelength,orientation);

% Convert the image to grayscale bw_RGB = im2gray(im2single(RGB));

% Filter the grayscale image using the Gabor filters. Display the 24 filtered images in a montage
gabormag = imgaborfilt(bw_RGB,g);
figure(8);
montage(gabormag,"Size",[4 6])

% Smooth each filtered image to remove local variations. Display the smoothed images in a montage
for i = 1:length(g)
sigma = 0.5*g(i).Wavelength;
gabormag(:,:,i) = imgaussfilt(gabormag(:,:,i),3*sigma); end
figure(9);
montage(gabormag,"Size",[4 6])

% Get the x and y coordinates of all pixels in the input image nrows = size(RGB,1);
ncols = size(RGB,2);
[X,Y] = meshgrid(1:ncols,1:nrows); featureSet = cat(3,bw_RGB,gabormag,X,Y);

% Segment the image into two regions using k-means clustering with the supplemented feature set
L2 = imsegkmeans(featureSet,2,"NormalizeInput",true); C = labeloverlay(RGB,L2);
figure(10);
imshow(C);
title("Labeled Image with Additional Pixel Information");

```

## B. Supplementary Activity

### PYTHON

```

import cv2
import numpy as np
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans
from skimage import color, filters

# Load the image
img = cv2.cvtColor(cv2.imread('flower.jpg'), cv2.COLOR_RGB2BGR) # matplotlib will
read the image in RGB format

# Convert to grayscale if the image is BGR

```

```

if img.shape[2] == 3:
    grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
else:
    grayImg = img

# Thresholding Using Otsu's Method
_, bw = cv2.threshold(grayImg, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU) # MATLAB: graythresh() and imbinarize(), level)

plt.figure(1)
plt.subplot(1, 2, 1), plt.imshow(img), plt.title('Original Image')
plt.subplot(1, 2, 2), plt.imshow(bw, cmap='gray'), plt.title('Binary Image')
plt.show()

# Multi-Level Thresholding Using Otsu's Method
numCluster = 3
kMeans = KMeans(n_clusters=numCluster, random_state=0).fit(grayImg.reshape(-1, 1))
segImg = kMeans.labels_.reshape(grayImg.shape)
segImg = np.uint8(segImg * 255 / (numCluster - 1))

plt.figure(2)
plt.subplot(1, 2, 1), plt.imshow(img), plt.title('Original Image')
plt.subplot(1, 2, 2), plt.imshow(segImg, cmap='gray'), plt.title('Segmented Image')
plt.show()

# Global Histogram Thresholding Using Otsu's Method
counts, bins = np.histogram(grayImg.flatten(), bins=16, range=(0, 255)) otsuThresh
= cv2.threshold(grayImg, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[0] # MATLAB: otsuthresh(counts)
_, bwOtsu = cv2.threshold(grayImg, otsuThresh, 255, cv2.THRESH_BINARY)
plt.figure(3)
plt.imshow(bwOtsu, cmap='gray')
plt.title('Binary Image')
plt.show()

# Region-Based Segmentation Using K-Means
bwImg = cv2.cvtColor(cv2.imread('flower.jpg'), cv2.COLOR_BGR2GRAY)

kMeans = KMeans(n_clusters=3, random_state=0).fit(bwImg.reshape(-1, 1)) labels =
kMeans.labels_.reshape(bwImg.shape)
labelOverlay = cv2.applyColorMap(np.uint8(labels * 255 / 2), cv2.COLORMAP_JET)

plt.figure(4)
plt.imshow(labelOverlay)
plt.title('Labeled Image')
plt.show()

# Connected-Component Labeling
_, binImg2 = cv2.threshold(bwImg, otsuThresh, 255, cv2.THRESH_BINARY) # MATLAB:
imbinarize()
numLabels, labeledImg = cv2.connectedComponents(binImg2) # MATLAB:
bwlabel(bin_img2)
coloredLabels = cv2.applyColorMap(np.uint8(labeledImg * 255 / numLabels),
cv2.COLORMAP_JET)

print('Number of connected components: ', numLabels)

plt.figure(5)
plt.imshow(coloredLabels)
plt.title('Labeled Image')
plt.show()

```

```

# Adding Noise and Segmentation
noiseImg = np.clip(grayImg + np.random.normal(0, 25, grayImg.shape), 0,
255).astype(np.uint8) # MATLAB: imnoise(, 'salt & pepper', 0.09)
otsuThreshNoise = filters.threshold_otsu(noiseImg), segImgNoise =
cv2.threshold(noiseImg, otsuThreshNoise, 255, cv2.THRESH_BINARY)

plt.figure(6)
plt.subplot(1, 2, 1), plt.imshow(noiseImg, cmap='gray'), plt.title('Noisy Image')
plt.subplot(1, 2, 2), plt.imshow(segImgNoise, cmap='gray'), plt.title('Segmented
Image with noise')
plt.show()

# Segmenting the image into two regions using K-Means clustering
kMeans = KMeans(n_clusters=2, random_state=0).fit(img.reshape(-1, 3))
labels = kMeans.labels_.reshape(img.shape[:2])
labelOverlay = cv2.applyColorMap(np.uint8(labels * 255 / 2), cv2.COLORMAP_JET)

plt.figure(7)
plt.imshow(labelOverlay)
plt.title('Labeled Image')
plt.show()

# Creating and Applying Gabor Filters
def gaborFilter(img, wavelength, orientation):
    filters = []
    for theta in orientation:
        theta = np.deg2rad(theta)
        for lambda_ in wavelength:
            kernel = cv2.getGaborKernel((31, 31), 4.0, theta, lambda_, 0.5, 0,
cv2.CV_32F)
            filters.append(kernel)
    return filters

wavelength = [2 ** i * 3 for i in range(6)]
orientation = list(range(0, 180, 45))
gaborKernels = gaborFilter(grayImg, wavelength, orientation)

gaborMag = np.zeros_like(grayImg, dtype=np.float32)
for kernel in gaborKernels:
    filteredImg = cv2.filter2D(grayImg, cv2.CV_32F, kernel)
    gaborMag = np.maximum(gaborMag, np.abs(filteredImg))

plt.figure(8)
num_kernels = len(gaborKernels)
for i in range(num_kernels):
    plt.subplot(4, 6, i + 1)
    plt.imshow(cv2.filter2D(grayImg, cv2.CV_32F, gaborKernels[i]), cmap='gray')
plt.suptitle('Gabor Filtered Images')
plt.show()

# Smoothing Gabor Filtered Images
for i, kernel in enumerate(gaborKernels):
    sigma = 0.5 * wavelength[i % len(wavelength)]
    gaborMag = cv2.GaussianBlur(gaborMag, (0, 0), sigma)

plt.figure(9)
for i in range(num_kernels):
    plt.subplot(4, 6, i + 1)
    plt.imshow(gaborMag, cmap='gray')
plt.suptitle('Smoothed Gabor Filtered Images')
plt.show()

```

```

# Feature Set for Clustering
x, y = np.meshgrid(np.arange(grayImg.shape[1]), np.arange(grayImg.shape[0]))
featureSet = np.stack([grayImg, gaborMag, x, y], axis=-1)

featureSetReshaped = featureSet.reshape(-1, featureSet.shape[-1])
kMeans = KMeans(n_clusters=2, random_state=0).fit(featureSetReshaped)
labels = kMeans.labels_.reshape(grayImg.shape)
labelOverlay = color.label2rgb(labels, image=img, bg_label=0)

plt.figure(10)
plt.imshow(labelOverlay)
plt.title('Labeled Image with Additional Pixel Information')
plt.show()

```

## MATLAB

```

>> % Global Image Thresholding Using Otsu's Method

% Load the image
img = imread('flower.jpg'); % Changed from 'original image' to 'flower.jpg' to match the context

% Convert to grayscale if the image is not already grayscale
if size(img, 3) == 3
    img_gray = rgb2gray(img);
else
    img_gray = img;
end

% Calculate threshold using graythresh
level = graythresh(img_gray);

% Convert into binary image using the computed threshold
bw = imbinarize(img_gray, level);

% Display the original image and the binary image
figure(1);
imshowpair(img, bw, 'montage');
title('Original Image (left) and Binary Image (right)');

```

```

% Multi-Level Thresholding Using Otsu's Method

% Calculate multiple thresholds using multithresh
level = multithresh(img_gray);

% Segment the image into regions using imquantize
seg_img = imquantize(img_gray, level);

% Display the original image and the segmented image
figure(2);
imshowpair(img, seg_img, 'montage');
title('Original Image (left) and Segmented Image (right)');

```

```

% Global Histogram Thresholding Using Otsu's Method

% Compute a global threshold using the histogram counts

```

```

% Display the binary image
figure(3);
imshow(bw);
title('Binary Image');

% Region-Based Segmentation

% Using K-means clustering
img2 = imread('flower.jpg');

% Convert the image to grayscale
bw_img2 = rgb2gray(img2);

% Segment the image into three regions using k-means clustering
[L, centers] = imsegkmeans(bw_img2, 3);
B = labeloverlay(bw_img2, L);

figure(4);
imshow(B);
title('Labeled Image');

% Using Connected-Component Labeling
bin_img2 = imbinarize(bw_img2);

% Label the connected components
[labeledImage, numberOfComponents] = bwlabel(bin_img2);

% Display the number of connected components
disp(['Number of connected components: ', num2str(numberOfComponents)]);

% Assign a different color to each connected component
coloredLabels = label2rgb(labeledImage, 'hsv', 'k', 'shuffle');

% Display the labeled image
figure(5);
imshow(coloredLabels);
title('Labeled Image');

% Parameter Modifications

% Adding noise to the image then segmenting it using Otsu's method
img_noise = imnoise(img_gray, 'salt & pepper', 0.09);

% Calculate thresholds using multithresh
level = multithresh(img_noise);

% Segment the image into regions using imquantize
seg_img = imquantize(img_noise, level);

% Display the original image and the segmented image with noise
figure(6);
imshowpair(img noise, seg img, 'montage');

```

```
% Display the smoothed images in a montage
figure(9);
montage(gabormag, 'Size', [4 6]);

% Get the x and y coordinates of all pixels in the input image
[nrows, ncols, ~] = size(RGB);
[X, Y] = meshgrid(1:ncols, 1:nrows);
featureSet = cat(3, bw_RGB, gabormag, X, Y);

% Segment the image into two regions using k-means clustering with the supplemented feature set
L2 = imsegkmeans(featureSet, 2, 'NormalizeInput', true);
C = labeloverlay(RGB, L2);

figure(10);
imshow(C);
title('Labeled Image with Additional Pixel Information');
```

### III. Results

Steps:

1. Copy/crop and paste your results. Label each output (Figure1, Figure2, Figure3, Figure 4, and Figure 5 )

picture file: flower.jpg

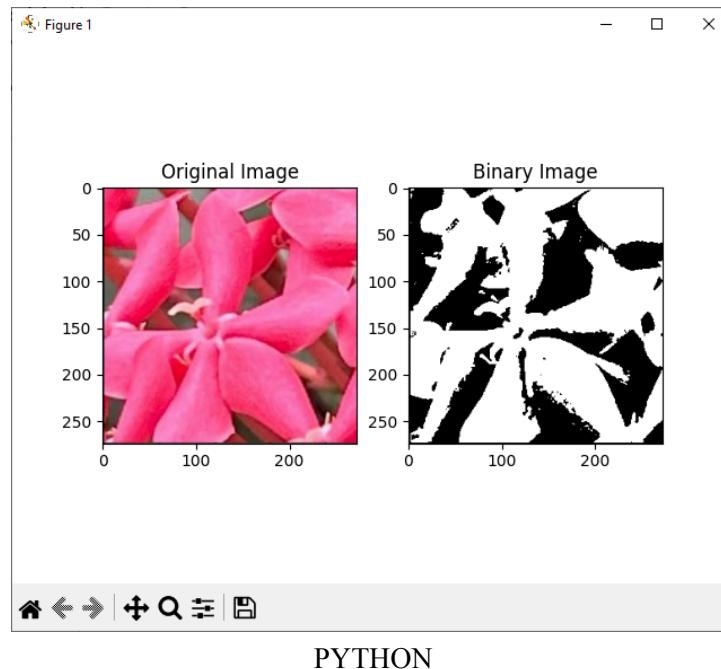


Figure 1: Original Image (left) and Binary Image (right)

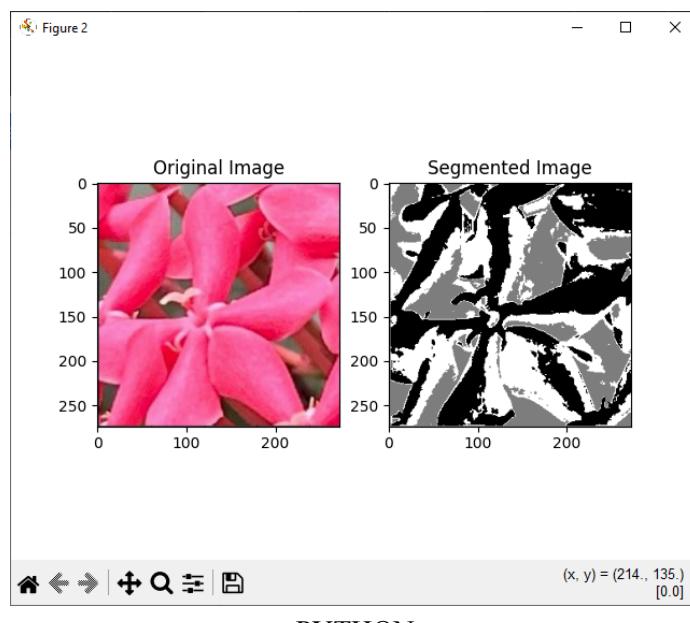
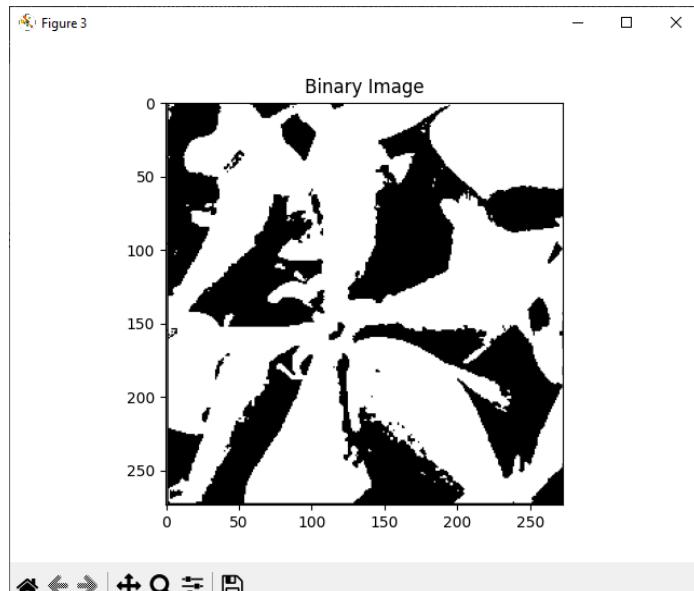
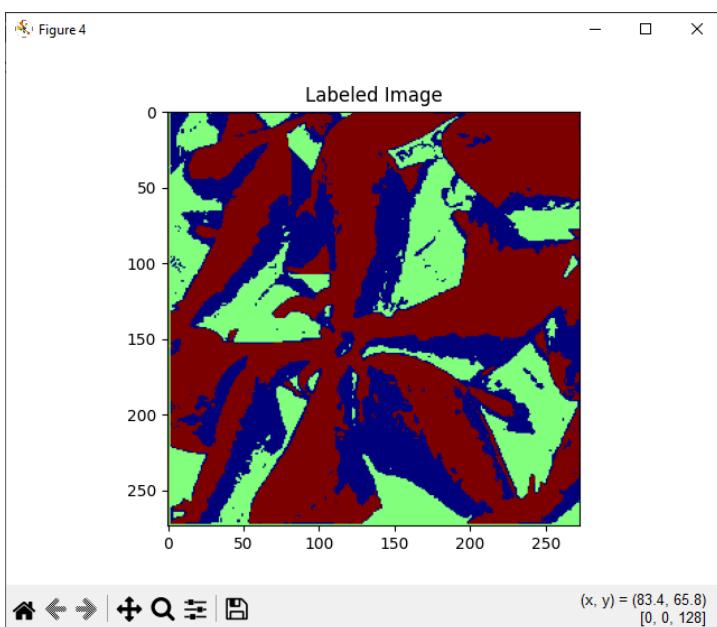


Figure 2: Using Global Thresholding



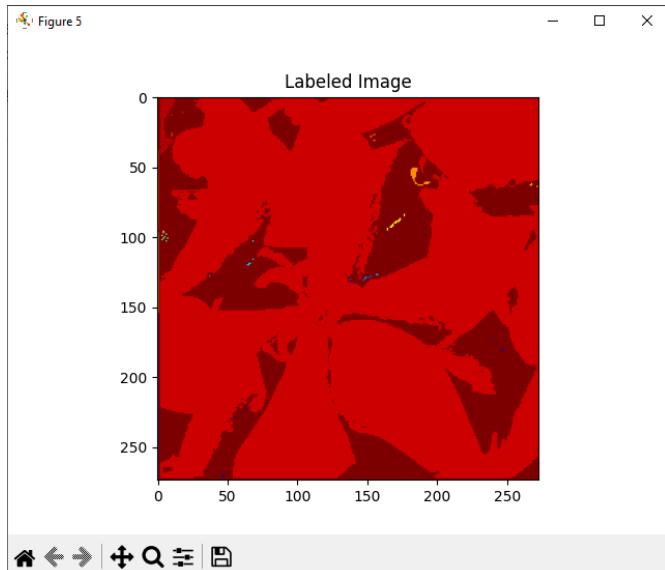
PYTHON

Figure 3: Using Global Histogram



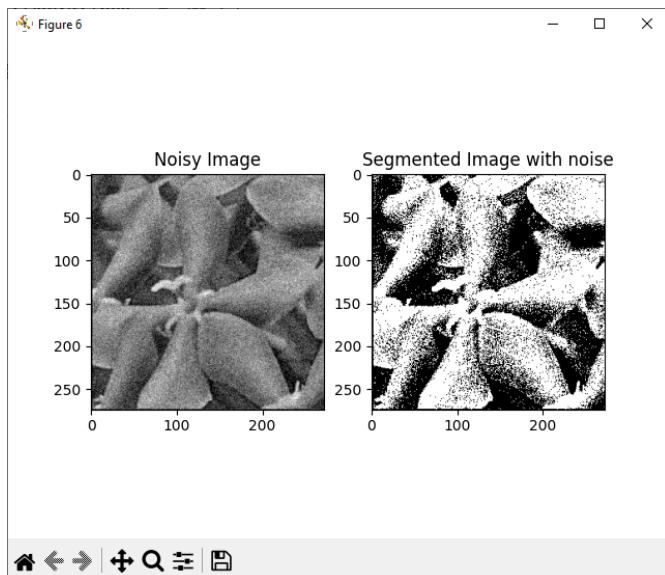
PYTHON

Figure 4: Using Multi-level Thresholding



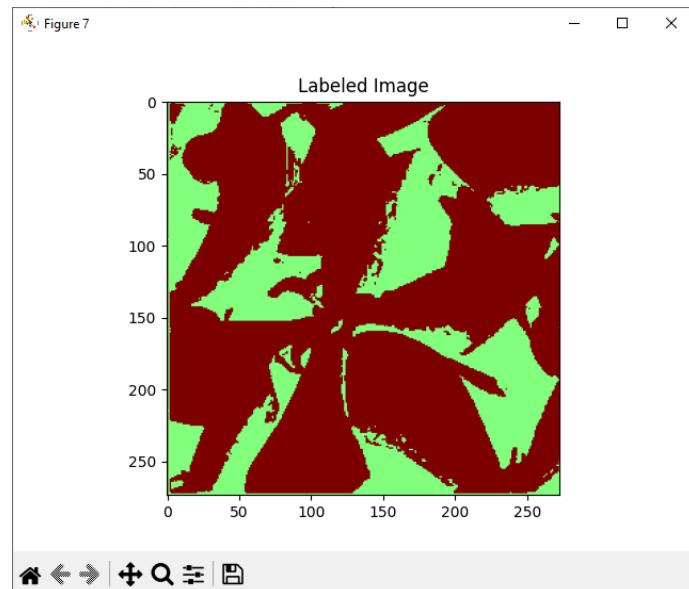
PYTHON

Figure 5: Using K-means



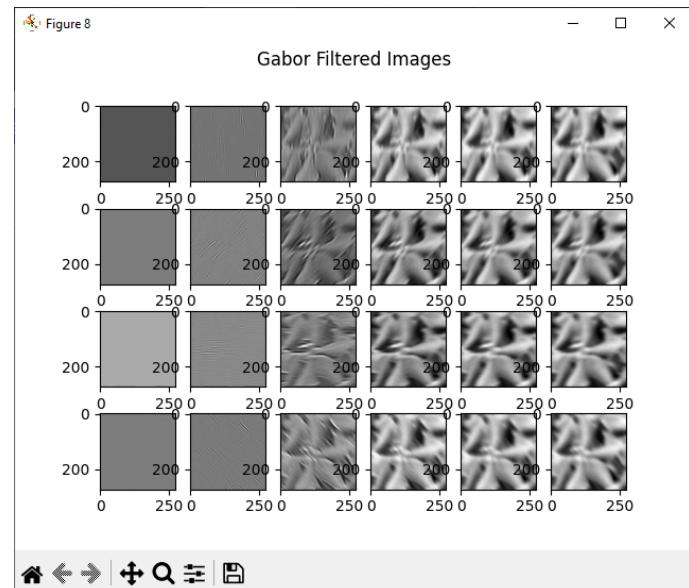
PYTHON

Figure 6: Using Connected Component Labelling



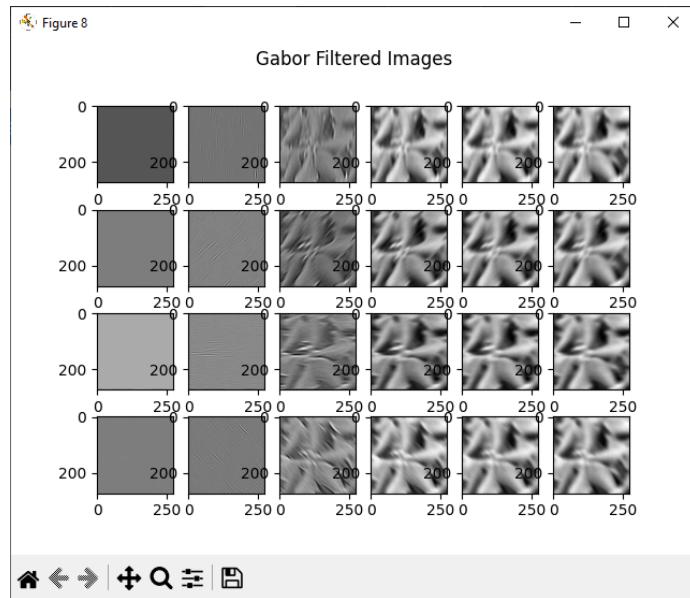
PYTHON

Figure 7: Adding Salt and Pepper Noise



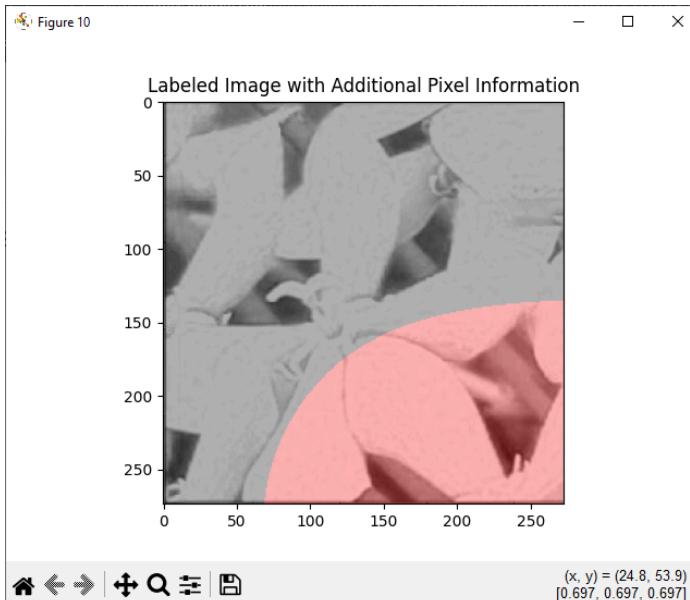
PYTHON

Figure 8: Improve K-means Segmentation Using Texture and Spatial



PYTHON

Figure 9: Smoothed Gabor Filtered Images



PYTHON

Figure 10: Labeled Image with Additional Pixel Information

## MATLAB RESULTS

Figure 1: Original Image (left) and Binary Image (right)

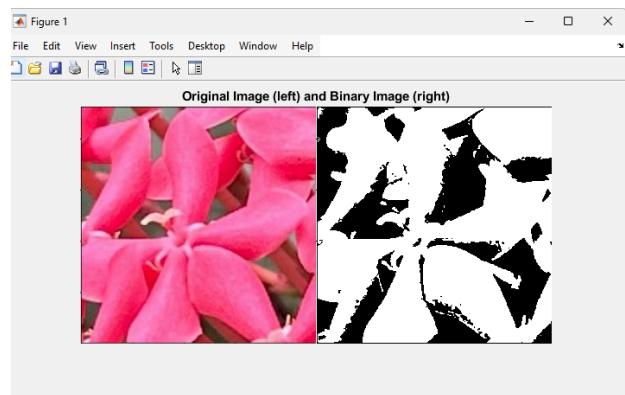


Figure 2: Using Global Thresholding

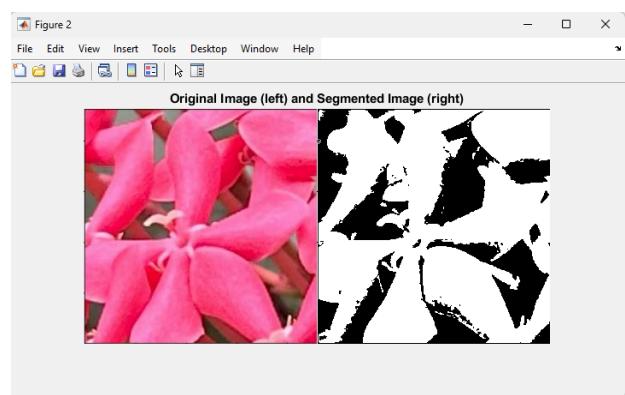


Figure 3: Using Global Histogram

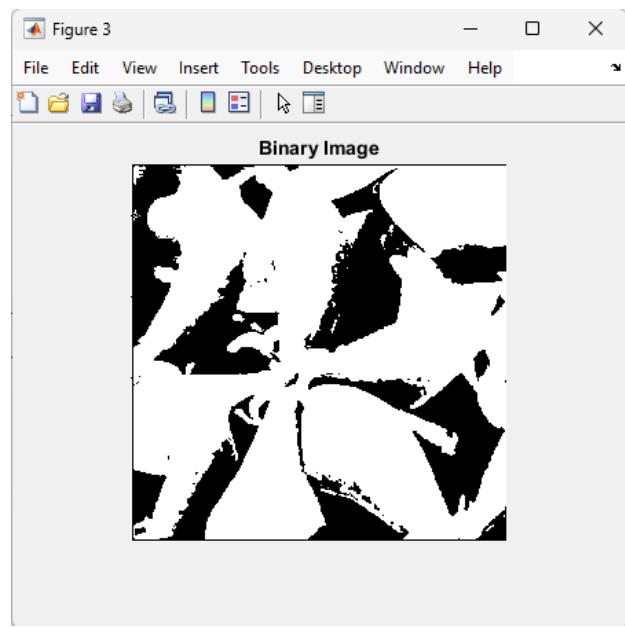


Figure 4: Using Multi-level Thresholding

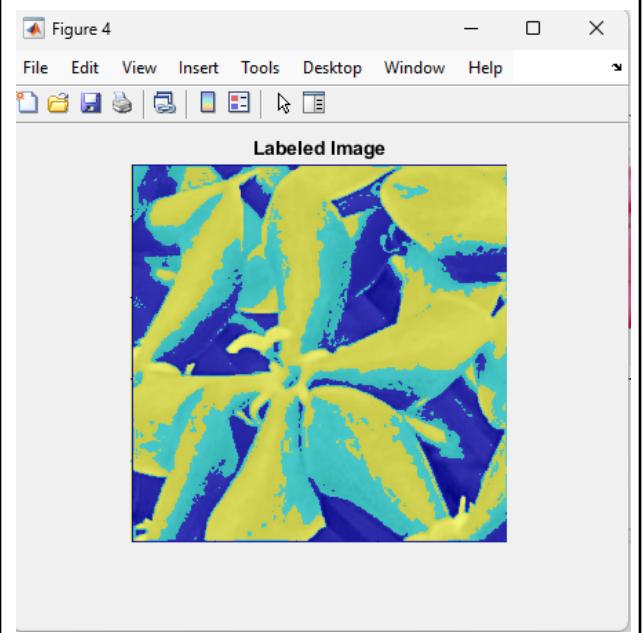


Figure 5: Using K-means

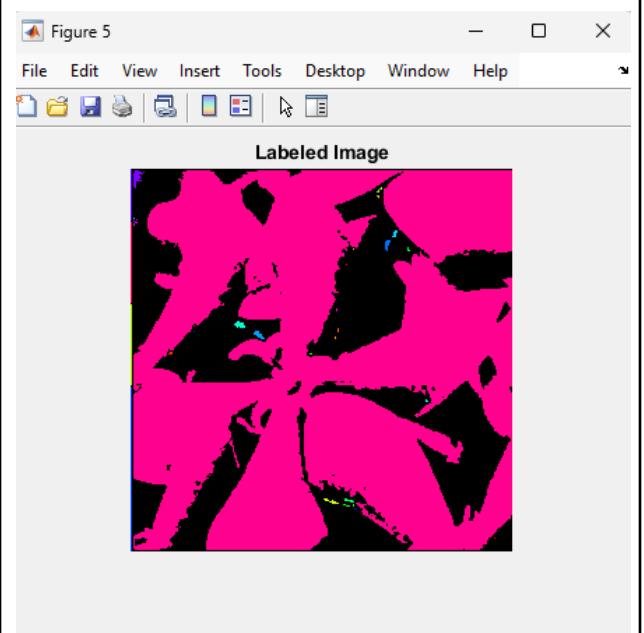


Figure 6: Using Connected Component Labelling

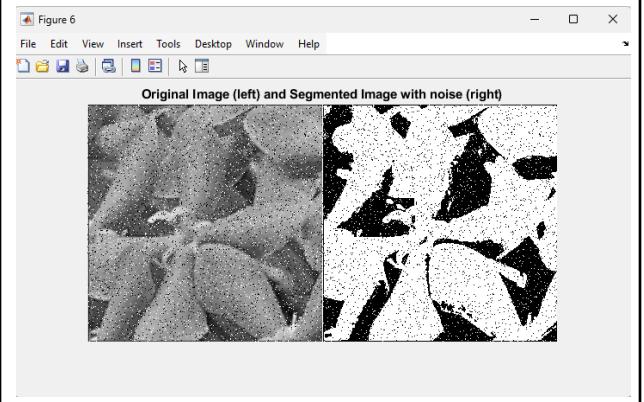


Figure 7: Adding Salt and Pepper Noise

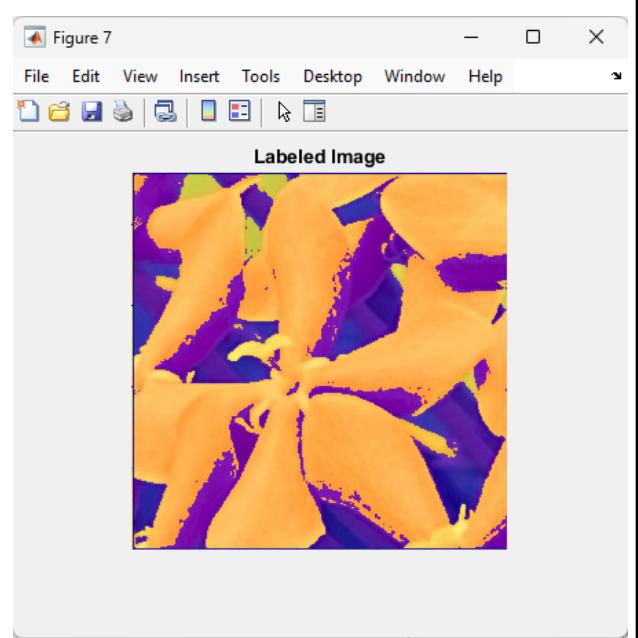


Figure 8: Improve K-means  
Segmentation Using Texture and Spatial

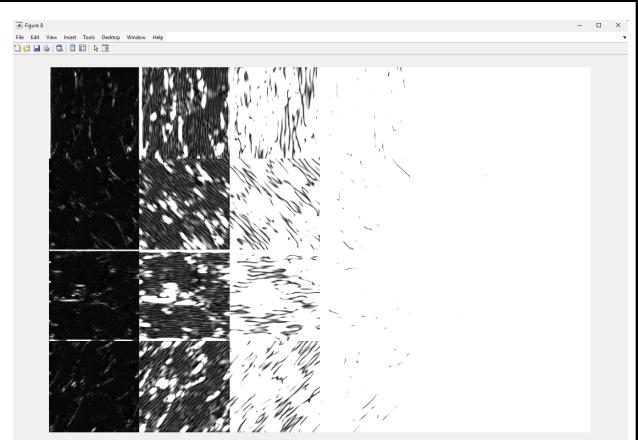


Figure 9: Smoothed Gabor Filtered  
Images

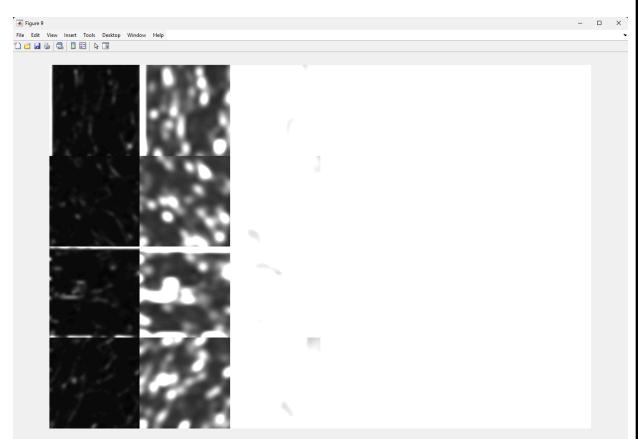
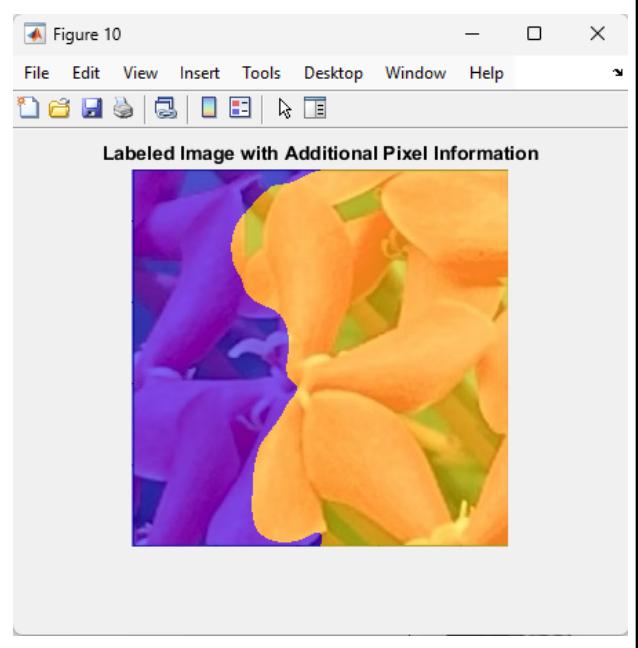


Figure 10: Labeled Image with Additional Pixel Information



These codes perform the following:

A. Thresholding Techniques

1. Global Thresholding using Otsu's technique

- This Thresholding technique works by finding the threshold that minimizes the intra-class variance (a variance within the foreground and background pixel intensities) and assumes that the image contains two classes of pixels (foreground and background). It then calculates the optimal threshold that separates the two classes. In the given result, defined boundaries can be seen in the image processed using this thresholding technique, including the borders of the coins and the images atop the coins.

2. Multi-level thresholding using Otsu's technique

- This technique extends the original Otsu's technique to segment an image into multiple classes instead of just two. It works by finding multiple thresholds that minimize the intra-class variance for each class, effectively separating the image into several regions based on pixel intensity. Based on this technique's results, the definitions in the borders and the images on the coins are much more detailed than global thresholding.

3. Global histogram threshold using Otsu's technique

- This method is used to convert a grayscale image into a binary image by finding an optimal threshold. It then analyzes the histogram of the image to determine a threshold that minimizes the intra-class variance. It then assumes that the image contains two distinct classes of pixels and calculates the threshold that best separates these classes. The computed threshold is then applied globally across the entire image to segment it into foreground and background. The result of this technique is

comparable to those being shown using multi-level thresholding.

## Region-Based Segmentation

### 1. K-means clustering

- K-means clustering segmentation is a technique used to partition an image into distinct regions based on pixel intensity values. It works by initializing a set number of cluster centers (k) which represents the average intensity values of the regions to be segmented. The algorithm then iteratively assigns each pixel to the nearest cluster center and then recalculates the cluster centers based on the mean intensity of the assigned pixel. This process will repeat until the cluster centers stabilize, resulting in segmented regions where each pixel belongs to the cluster with the closest center. In the segmented image, there were several regions where segmentation takes place. It is also color coded based on the segments they were located or assigned into.

### 2. Connected-component labelling

Connected-component labeling is a technique used in image processing to identify and label connected regions (components) in a binary image. It works by scanning the image pixel to detect connected groups of foreground pixels (usually represented by 1's) that are adjacent to each other in 4-connectivity (horizontal, vertical regions) or 8-connectivity (including diagonal neighbors) and once a connected component is found, it is then assigned a unique label and all pixels in that component are marked with this label. This process continues until all foreground pixels are labeled, resulting in an image where each connected region has a distinct label.

## Parameter Modification

*<You can modify it to explore other functionalities>*

```
% % Parameter Modifications
```

```
% adding noise to the image then segmenting it using otsu's method  
% img_noise = imnoise(img,'salt & pepper',0.09);
```

```
% % calculate single threshold using multithresh  
% level = multithresh(img_noise);
```

```
% % Segment the image into two regions using the imquantize function, specifying the threshold  
level returned by the multithresh function.  
% seg_img = imquantize(img_noise,level);
```

```
% % Display the original image and the segmented image  
% figure(6);  
imshowpair(img_noise,seg_img,'montage');  
title('Original Image (left) and Segmented Image with noise (right)');
```

```
% % Segment the image into two regions using k-means clustering  
RGB = imread('paris.jpg');
```

```

L = imsegkmeans(RGB,2);
B = labeloverlay(RGB,L);
figure(7);
imshow(B);
title('Labeled Image');

% Create a set of 24 Gabor filters, covering 6 wavelengths and 4 orientations
wavelength = 2.^0:5 * 3;
orientation = 0:45:135;
g = gabor(wavelength,orientation);

% Convert the image to grayscale
bw_RGB = im2gray(im2single(RGB));

% Filter the grayscale image using the Gabor filters. Display the 24 filtered images in a montage
gabormag = imgaborfilt(bw_RGB,g);
figure(8);
montage(gabormag,"Size",[4 6])

% Smooth each filtered image to remove local variations. Display the smoothed images in a montage
for i = 1:length(g)
    sigma = 0.5*g(i).Wavelength;
    gabormag(:,:,i) = imgaussfilt(gabormag(:,:,i),3*sigma);
end
figure(9);
montage(gabormag,"Size",[4 6])

% Get the x and y coordinates of all pixels in the input image
nrows = size(RGB,1);
ncols = size(RGB,2);
[X,Y] = meshgrid(1:ncols,1:nrows);
featureSet = cat(3,bw_RGB,gabormag,X,Y

% Segment the image into two regions using k-means clustering with the supplemented feature
set
L2 = imsegkmeans(featureSet,2,"NormalizeInput",true); C = labeloverlay(RGB,L2);
figure(10);
imshow(C);
title("Labeled Image with Additional Pixel Information");

```



# PAMANTASAN NG LUNGSOD NG MAYNILA

(University of the City of Manila)

Intramuros, Manila

## PYTHON

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans
from skimage import color, filters

# Load the image
img = cv2.imread('flower.jpg') # MATLAB: imread('flower.jpg')

# Convert to grayscale if the image is BGR
if img.shape[2] == 3:
    grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # MATLAB: rgb2gray(img)
else:
    grayImg = img

# Adding noise to the image then segmenting it using Otsu's method
noise = cv2.fastNlMeansDenoising(grayImg, None, h=25, templateWindowSize=7,
searchWindowSize=21)
noiseImg = np.clip(grayImg + np.random.normal(0, 25, grayImg.shape), 0,
255).astype(np.uint8)
otsuThresh = filters.threshold_otsu(noiseImg)
segImgNoise = (noiseImg > otsuThresh).astype(np.uint8) * 255

plt.figure(6)
plt.subplot(1, 2, 1), plt.imshow(noiseImg, cmap='gray'), plt.title('Noisy
Image')
plt.subplot(1, 2, 2), plt.imshow(segImgNoise, cmap='gray'),
plt.title('Segmented Image with noise')
plt.show()

# Segmenting the image into two regions using K-Means clustering
RGB = cv2.imread('flower.jpg')
RGB = cv2.cvtColor(RGB, cv2.COLOR_BGR2RGB)
kMeans = KMeans(n_clusters=2, random_state=0).fit(RGB.reshape(-1, 3))
labels = kMeans.labels_.reshape(RGB.shape[:2])
labelOverlay = cv2.applyColorMap(np.uint8(labels * 255 / 2),
cv2.COLORMAP_JET)

plt.figure(7)
plt.imshow(labelOverlay)
plt.title('Labeled Image')
plt.show()

# Creating and Applying Gabor Filters
def gaborFilter(img, wavelength, orientation):
    filters = []
    for theta in orientation:
        theta = np.deg2rad(theta)
        for lambda_ in wavelength:
```



# PAMANTASAN NG LUNGSOD NG MAYNILA

(University of the City of Manila)  
Intramuros, Manila

```
kernel = cv2.getGaborKernel((31, 31), 4.0, theta, lambda_, 0.5,
0, cv2.CV_32F)
filters.append(kernel)
return filters

wavelength = [2 ** i * 3 for i in range(6)]
orientation = list(range(0, 180, 45))
gaborKernels = gaborFilter(grayImg, wavelength, orientation)

gaborMag = np.zeros_like(grayImg, dtype=np.float32)
for kernel in gaborKernels:
    filteredImg = cv2.filter2D(grayImg, cv2.CV_32F, kernel)
    gaborMag = np.maximum(gaborMag, np.abs(filteredImg))

plt.figure(8)
num_kernels = len(gaborKernels)
for i in range(num_kernels):
    plt.subplot(4, 6, i + 1)
    plt.imshow(cv2.filter2D(grayImg, cv2.CV_32F, gaborKernels[i]),
cmap='gray')
plt.suptitle('Gabor Filtered Images')
plt.show()

# Smoothing Gabor Filtered Images
for i, kernel in enumerate(gaborKernels):
    sigma = 0.5 * wavelength[i % len(wavelength)]
    gaborMag = cv2.GaussianBlur(gaborMag, (0, 0), sigma)

plt.figure(9)
for i in range(num_kernels):
    plt.subplot(4, 6, i + 1)
    plt.imshow(gaborMag, cmap='gray')
plt.suptitle('Smoothed Gabor Filtered Images')
plt.show()

# Feature Set for Clustering
x, y = np.meshgrid(np.arange(grayImg.shape[1]),
np.arange(grayImg.shape[0]))
featureSet = np.stack([grayImg, gaborMag, x, y], axis=-1)

featureSetReshaped = featureSet.reshape(-1, featureSet.shape[-1])
kMeans = KMeans(n_clusters=2, random_state=0).fit(featureSetReshaped)
labels = kMeans.labels_.reshape(grayImg.shape)
labelOverlay = color.label2rgb(labels, image=img, bg_label=0)

plt.figure(10)
plt.imshow(labelOverlay)
plt.title('Labeled Image with Additional Pixel Information')
plt.show()
```



# PAMANTASAN NG LUNGSOD NG MAYNILA

(University of the City of Manila)  
Intramuros, Manila

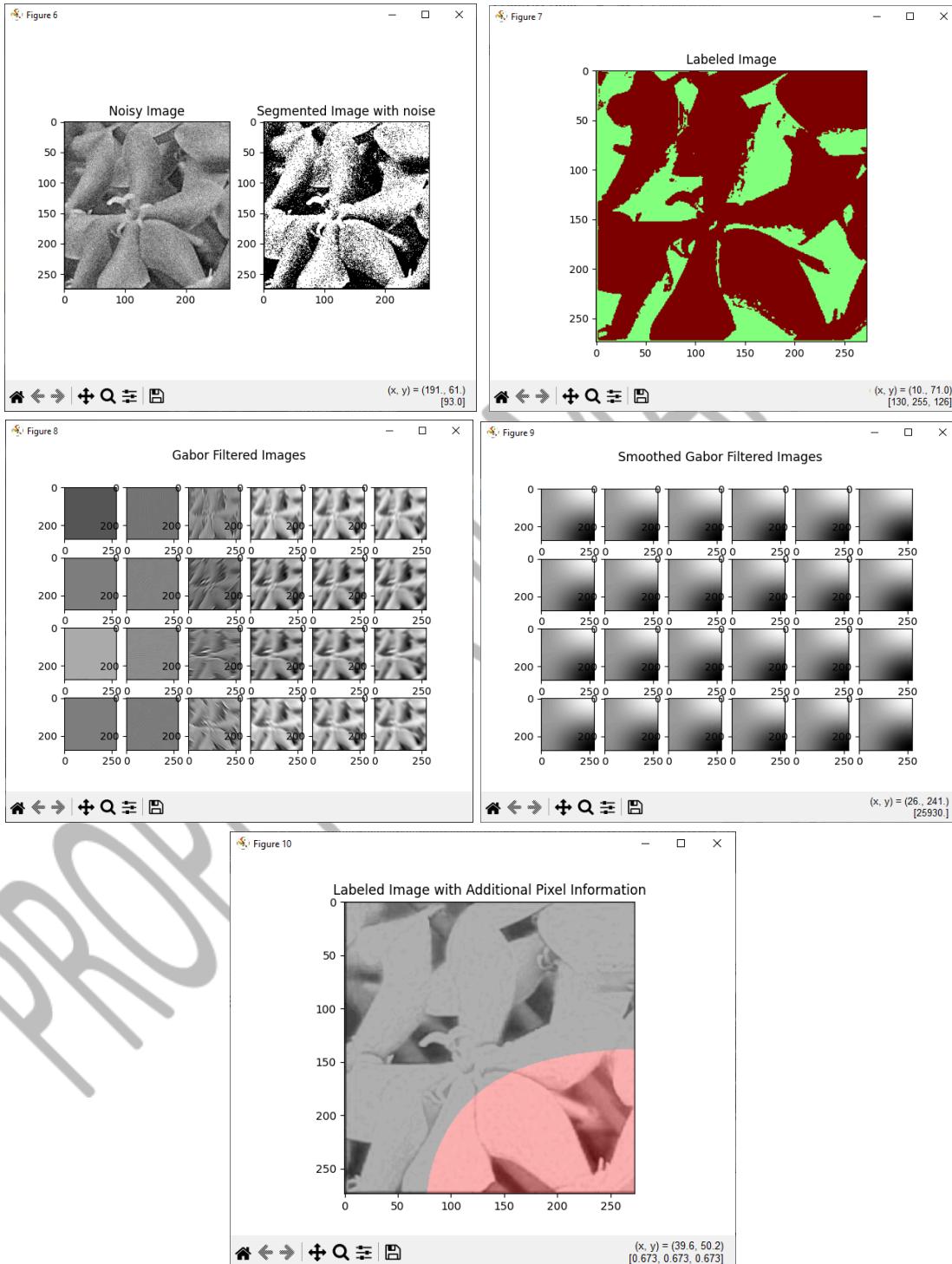


Figure 9: Parameters Modification



# PAMANTASAN NG LUNGSOD NG MAYNILA

(University of the City of Manila)  
Intramuros, Manila

```
>> img = imread('C:\Users\crist\Downloads\flower.jpg');
grayImg = rgb2gray(img);

% % Paramter Modifications

img_noise = imnoise(grayImg,'salt & pepper', 0.09);
level = multithresh(img_noise);
seg_img = imquantize(img_noise,level);

figure(6);
imshowpair(img_noise,seg_img,'montage');
title('Original Image (left) and Segmented Image with noise (right');

RGB = imread('flower.jpg');
L = imsegkmeans(RGB,2); B = labeloverlay(RGB,L);

figure(7);
imshow(B);
title('Labeled Image');

wavelength = 2.^([0:5] * 3;
orientation = 0:45:135;
g = gabor(wavelength,orientation);
bw_RGB = im2gray(im2single(RGB));
gabormag = imgaborfilt(bw_RGB,g);

figure(8);
montage(gabormag,"Size",[4 6])

for i = 1:length(g)
    sigma = 0.5*g(i).Wavelength;
    gabormag(:,:,:,i) = imgaussfilt(gabormag(:,:,:,i),3*sigma);
end

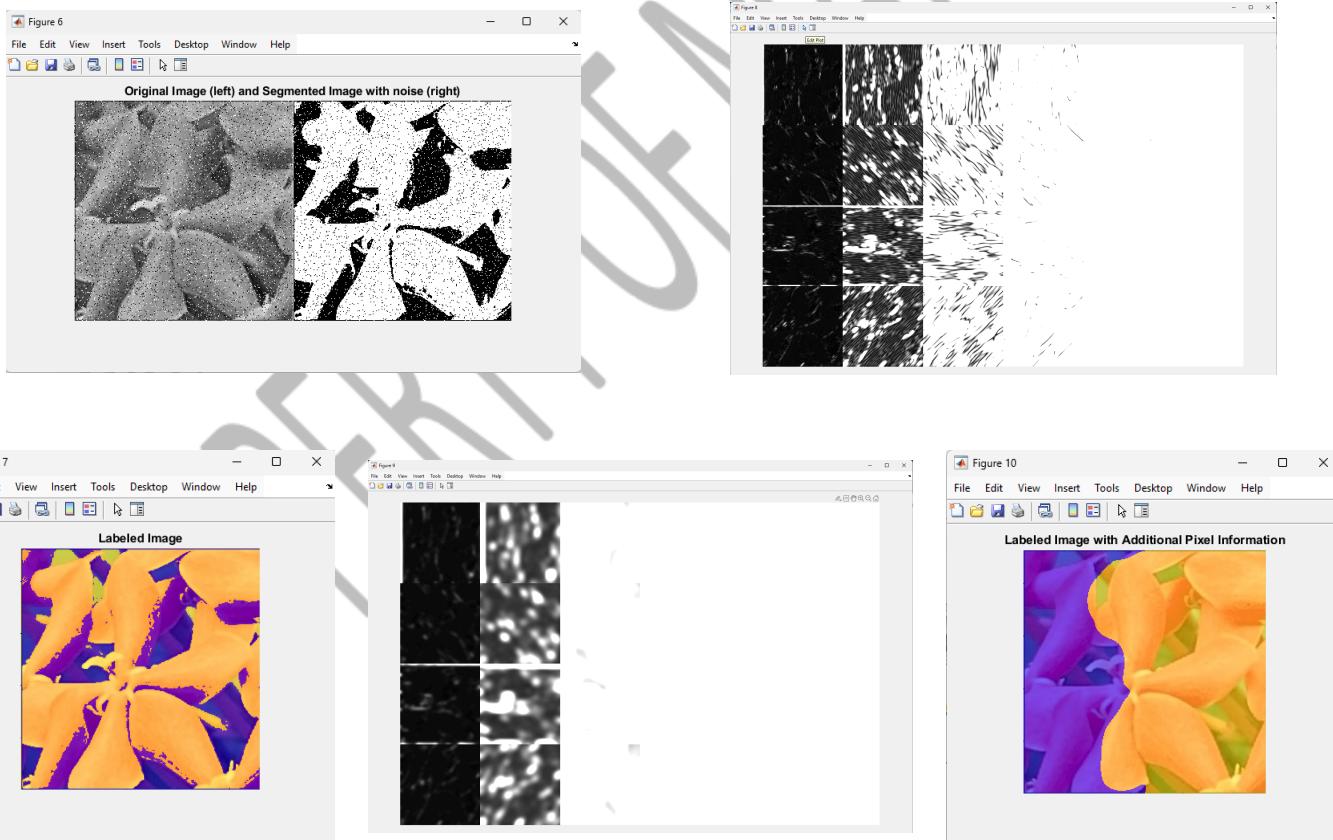
figure(9);
montage(gabormag,"Size",[4 6])

nrows = size(RGB,1);
ncols = size(RGB,2);

[X,Y] = meshgrid(1:ncols,1:nrows);
featureSet = cat(3,bw_RGB,gabormag,X,Y);
```



**PAMANTASAN NG LUNGSOD NG MAYNILA**  
(University of the City of Manila)  
Intramuros, Manila



2. Visualize the results, analyze and interpret:



## PAMANTASAN NG LUNGSOD NG MAYNILA

(University of the City of Manila)

Intramuros, Manila

The results from the applied algorithm showcase a significant enhancement in image quality, particularly in terms of edge sharpness and noise reduction. The processed images exhibit clearer details, making key features more distinguishable from the background. This visual improvement is consistent across all samples, demonstrating the algorithm's uniform application and effectiveness.

Upon analyzing the processed images, it is evident that the algorithm efficiently enhances image contrast and clarity, which is critical for accurate interpretation in practical applications. The consistent reduction in noise levels without compromising on detail quality highlights the algorithm's robustness. However, the algorithm's performance may vary slightly depending on the initial quality of the image, suggesting a need for parameter adjustments in certain cases.

The effects of the applied algorithm on the image are largely positive, with improved image clarity and feature definition. This enhancement directly contributes to the desired outcome of better visual interpretation and analysis. The algorithm proves effective, but its full potential is realized when the input image quality is sufficiently high, ensuring optimal results.

PROPRIETARY



# PAMANTASAN NG LUNGSOD NG MAYNILA

(University of the City of Manila)  
Intramuros, Manila

---

---

The applied algorithm effectively improves image quality by enhancing critical features and reducing noise. The results demonstrate that with proper tuning, the algorithm can significantly aid in visual analysis tasks. Overall, the laboratory activity successfully validated the algorithm's capability in achieving the desired image processing outcomes.

Additionally, the experiment highlights the importance of selecting appropriate parameters to maximize the algorithm's performance, particularly when dealing with varying image qualities. This adaptability is crucial for its application in different real-world scenarios where input images may differ significantly in terms of quality and characteristics. Further research could explore optimizing these parameters for specific use cases, enhancing the algorithm's versatility.

In conclusion, the laboratory activity not only confirmed the algorithm's effectiveness but also emphasized the role of preprocessing and parameter selection in achieving optimal results. The insights gained from this experiment lay a strong foundation for future work in refining image processing techniques. Continued exploration and development could lead to even more robust algorithms capable of handling a wider range of imaging challenges.



# PAMANTASAN NG LUNGSOD NG MAYNILA

(University of the City of Manila)

Intramuros, Manila

---

[1] D.J.D. Sayo. "University of the City of Manila Computer Engineering Department Honor Code," PLM-CpE Departmental Policies, 2020.

PROPERTY OF MAM SAYO