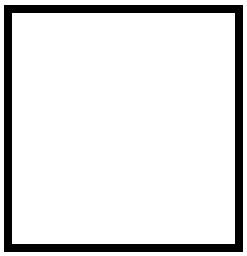
Elective 3

Laboratory Activity No. 5
Image Segmentation



Score

Submitted by:

Mayor, Ann Jossan D. – Leader
Biol, Lorenz Jay Von P.
Lumane, Kyla L.
Sison, Dan Jedrick S.
Tenoria, Karl John Dave C.

SAT 7AM – 4PM / 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('Labled 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

- Write a Python program that will implement the output in Method A.

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

```
from skimage import color, filters, measure, segmentation, util
from skimage.filters import threshold_multiotsu, gabor_kernel
from sklearn.cluster import KMeans

# Load image
img = cv2.imread('flower.jpg')  # Replace 'flower.jpg' with the correct image path
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Global Image thresholding using Otsu's method
# Calculate threshold using Otsu's method
_, bw = cv2.threshold(img_gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Display the original image and the binary image
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(img_gray, cmap='gray')
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
levels = threshold_multiotsu(img_gray, classes=3)
seg_img = np.digitize(img_gray, bins=levels)

# Display the original image and the segmented image
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(img_gray, cmap='gray')
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(seg_img, cmap='gray')
plt.title('Segmented Image')
plt.show()

# Global histogram threshold using Otsu's method
# Calculate a 16-bin histogram for the image
counts, x = np.histogram(img_gray, bins=16)
plt.figure()
plt.stem(x[:-1], counts, basefmt=" ")
plt.title('Histogram')
plt.show()

# Compute a global threshold using the histogram counts
T = filters.threshold_otsu(img_gray)
bw = img_gray > T

# Create a binary image using the computed threshold and display the image
```

```
plt.figure()
plt.imshow(bw, cmap='gray')
plt.title('Binary Image')
plt.show()

# Region-based segmentation using K means clustering
img2 = cv2.imread('flower.jpg')  # Replace with the correct image path
bw_img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

# K-means clustering
kmeans = KMeans(n_clusters=3, random_state=0).fit(bw_img2.reshape(-1, 1))
L = kmeans.labels_.reshape(bw_img2.shape)
B = color.label2rgb(L, image=bw_img2, bg_label=0)

plt.figure()
plt.imshow(B)
plt.title('Labeled Image')
plt.show()

# Connected-component labeling
bin_img2 = bw_img2 > filters.threshold_otsu(bw_img2)
labeledImage = measure.label(bin_img2)
numberOfComponents = labeledImage.max()

# Display the number of connected components
print(f'Number of connected components: {numberOfComponents}')

# Assign a different color to each connected component
coloredLabels = color.label2rgb(labeledImage, bg_label=0)

# Display the labeled image
plt.figure()
plt.imshow(coloredLabels)
plt.title('Labeled Image')
plt.show()

# Adding noise to the image then segmenting it using Otsu's method
img_noise = util.random_noise(img_gray, mode='s&p', amount=0.09)
img_noise = (img_noise * 255).astype(np.uint8)

# Multi-level thresholding using Otsu's method
levels_noise = threshold_multiotsu(img_noise, classes=3)
seg_img_noise = np.digitize(img_noise, bins=levels_noise)

# Display the original image and the segmented image
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(img_noise, cmap='gray')
plt.title('Original Image with Noise')
```

```
plt.subplot(1, 2, 2)
plt.imshow(seg_img_noise, cmap='gray')
plt.title('Segmented Image with Noise')
plt.show()

# Segment the image into two regions using k-means clustering
RGB = cv2.imread('flower.jpg')  # Replace with the correct image path
RGB_gray = cv2.cvtColor(RGB, cv2.COLOR_BGR2GRAY)
kmeans = KMeans(n_clusters=2, random_state=0).fit(RGB_gray.reshape(-1, 1))
L = kmeans.labels_.reshape(RGB_gray.shape)
B = color.label2rgb(L, image=RGB_gray, bg_label=0)

plt.figure()
plt.imshow(B)
plt.title('Labeled Image')
plt.show()

# Create a set of 24 Gabor filters, covering 6 wavelengths and 4 orientations
wavelengths = np.array([3, 6, 12, 24, 48, 96])
orientations = np.array([0, 45, 90, 135])
gabor_filters = [gabor_kernel(wavelength, theta=np.deg2rad(orientation))
                 for wavelength in wavelengths for orientation in orientations]

# Filter the grayscale image using the Gabor filters and display the filtered images
in a montage
gabormag = [cv2.filter2D(RGB_gray, cv2.CV_8UC3, np.real(gabor_filter)) for
gabor_filter in gabor_filters]
gabormag = np.array(gabormag)
gabormag = np.transpose(gabormag, (1, 2, 0))

plt.figure(figsize=(12, 8))
for i in range(len(gabor_filters)):
    plt.subplot(4, 6, i+1)
    plt.imshow(gabormag[:, :, i], cmap='gray')
    plt.axis('off')
plt.suptitle('Filtered Images using Gabor Filters')
plt.show()

# Smooth each filtered image to remove local variations
gabormag_smoothed = np.array([cv2.GaussianBlur(gabormag[:, :, i], (0, 0),
sigmaX=3*0.5*wavelengths[i//4]) for i in range(len(gabor_filters))])
gabormag_smoothed = np.transpose(gabormag_smoothed, (1, 2, 0))

plt.figure(figsize=(12, 8))
for i in range(len(gabor_filters)):
    plt.subplot(4, 6, i+1)
    plt.imshow(gabormag_smoothed[:, :, i], cmap='gray')
    plt.axis('off')
plt.suptitle('Smoothed Gabor Filtered Images')
```

```
plt.show()

# Get the x and y coordinates of all pixels in the input image
nrows, ncols = RGB_gray.shape
X, Y = np.meshgrid(np.arange(ncols), np.arange(nrows))
feature_set = np.dstack((RGB_gray, gabormag_smoothed, X, Y))

# Segment the image into two regions using k-means clustering with the supplemented
feature set
kmeans = KMeans(n_clusters=2, random_state=0).fit(feature_set.reshape(-1,
feature_set.shape[2]))
L2 = kmeans.labels_.reshape(RGB_gray.shape)
C = color.label2rgb(L2, image=RGB, bg_label=0)

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

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

OUTPUT IN MATLAB:

Figure 1: Acquire an Image of a Flower

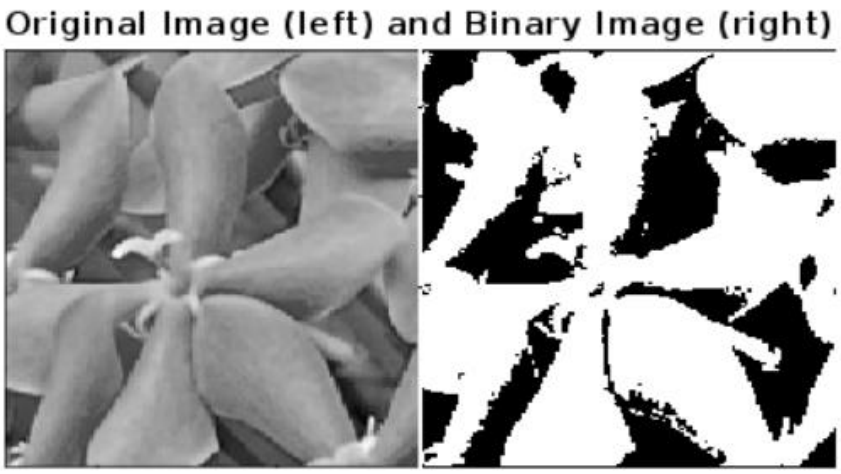


Figure 2: Original Image (left) and Binary Image (right)

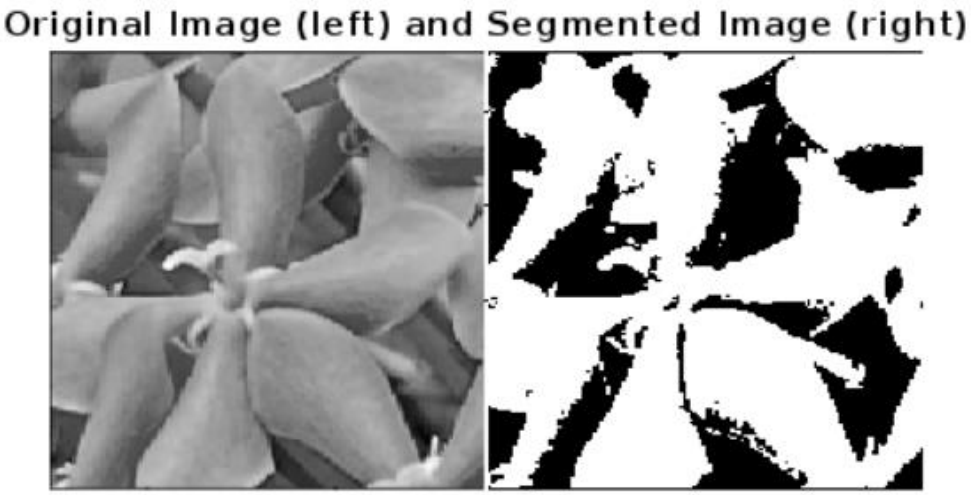


Figure 3: Using Global Thresholding

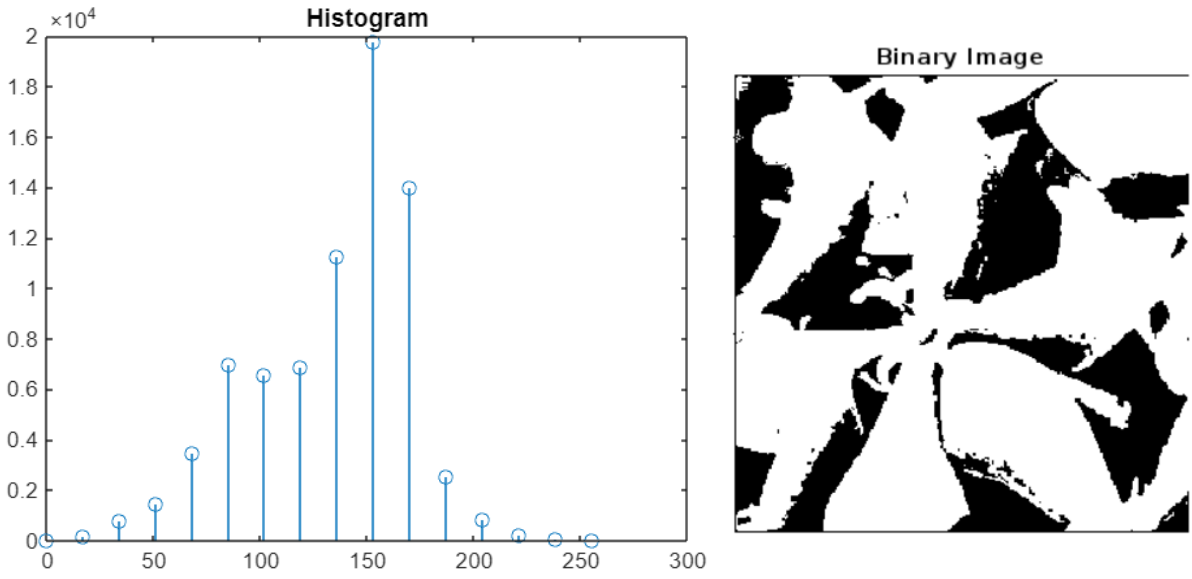


Figure 4: Using Global Histogram

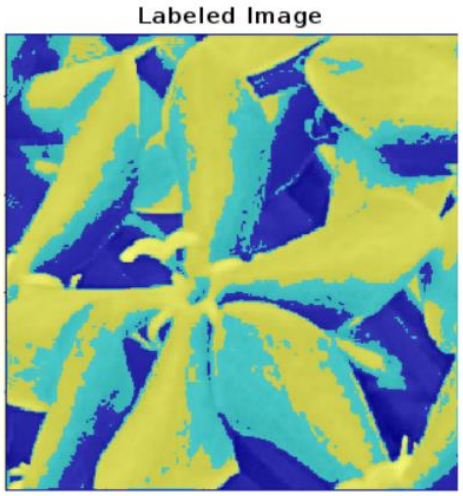


Figure 5: Using Multi-level Thresholding

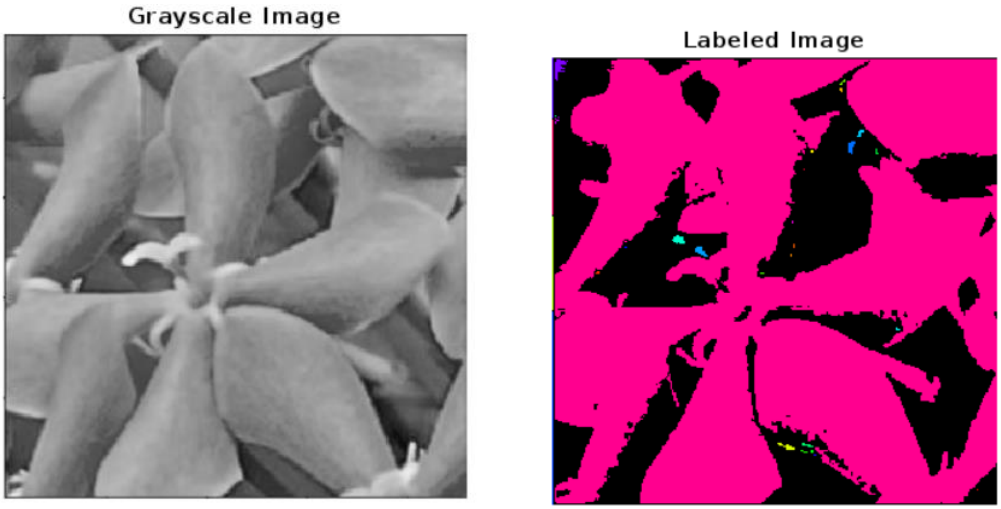


Figure 6: Using K-means

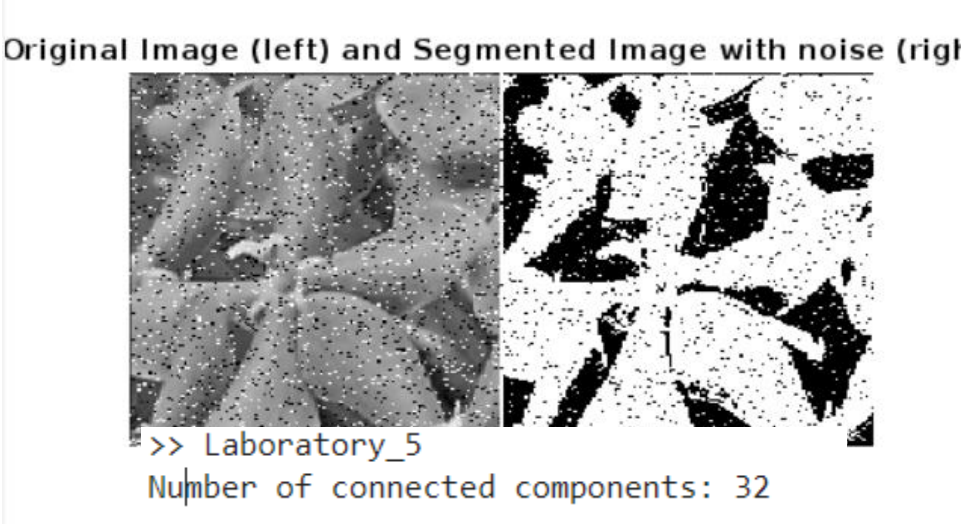


Figure 7: Using Connected Component Labelling

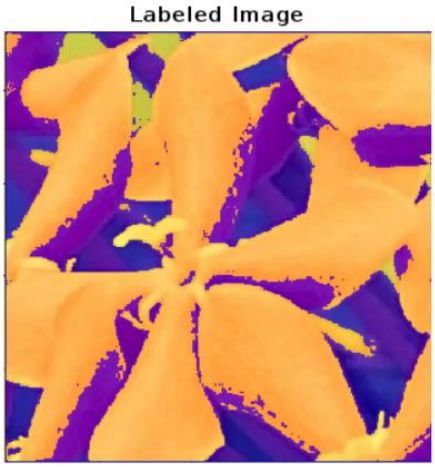


Figure 8: Adding Salt and Pepper Noise

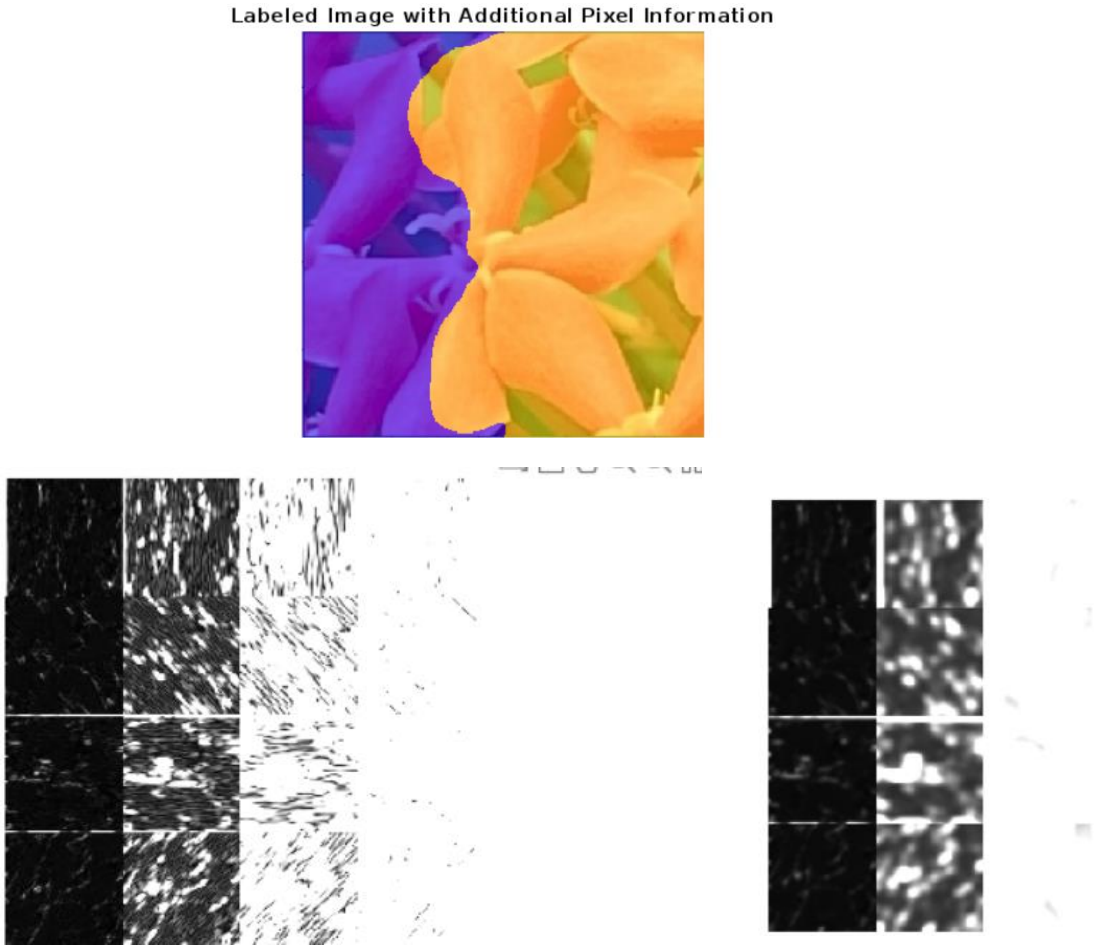


Figure 9: Improve K-means Segmentation Using Texture and Spatial

OUTPUT IN PYTHON OPENCV:

Figure 10: Acquire an Image of a Flower

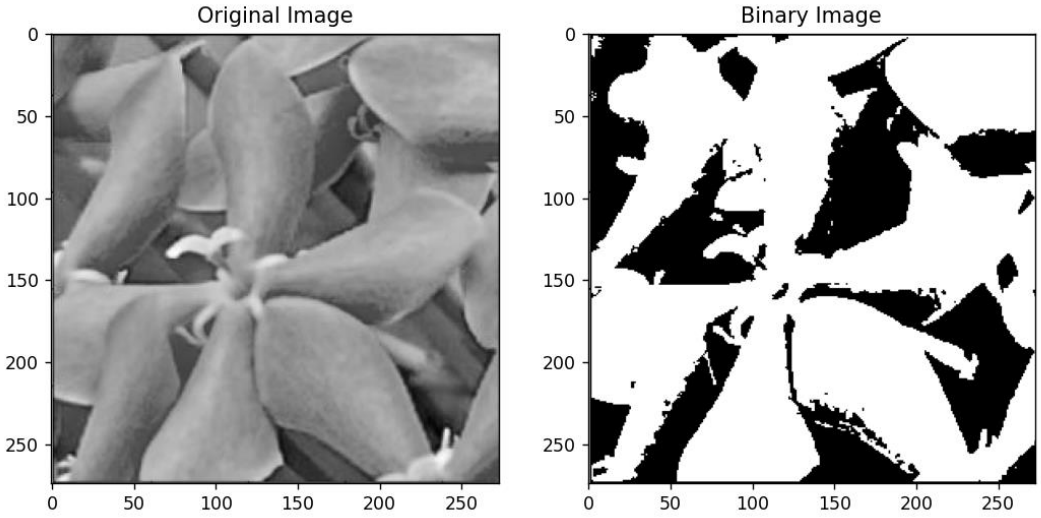


Figure 11: Original Image (left) and Binary Image (right)

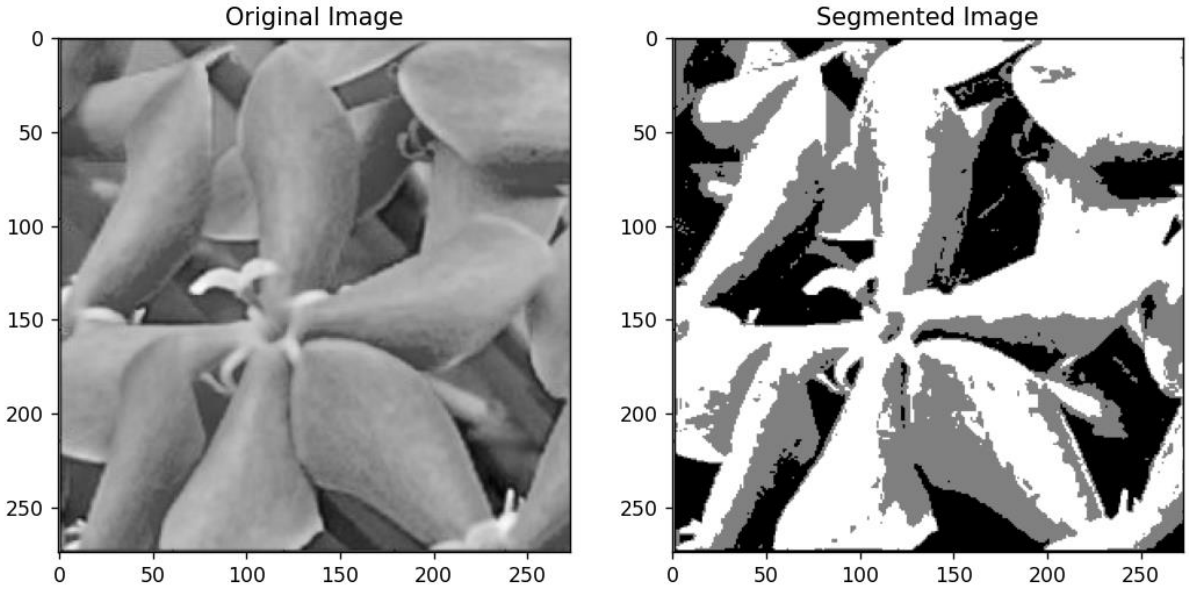


Figure 12: Using Global Thresholding

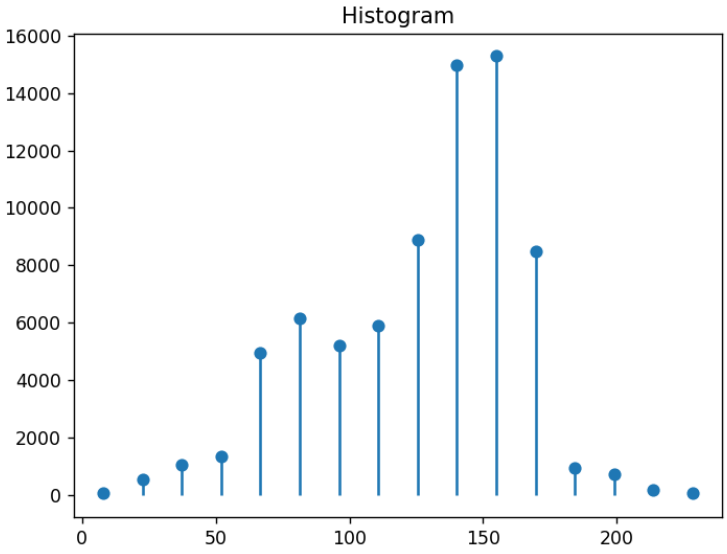


Figure 13: Using Global Histogram

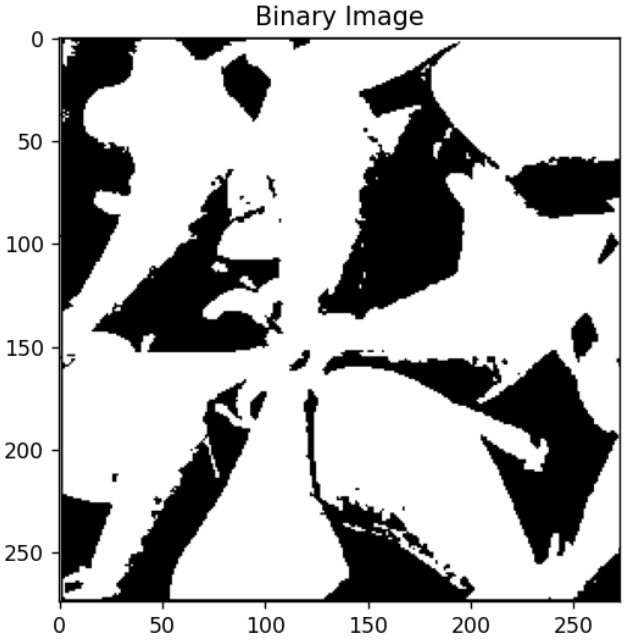


Figure 14: Using Multi-level Thresholding

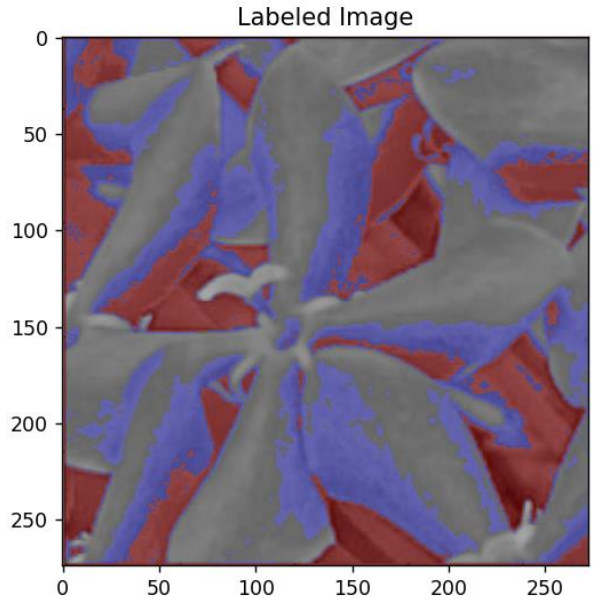


Figure 15: Using K-means

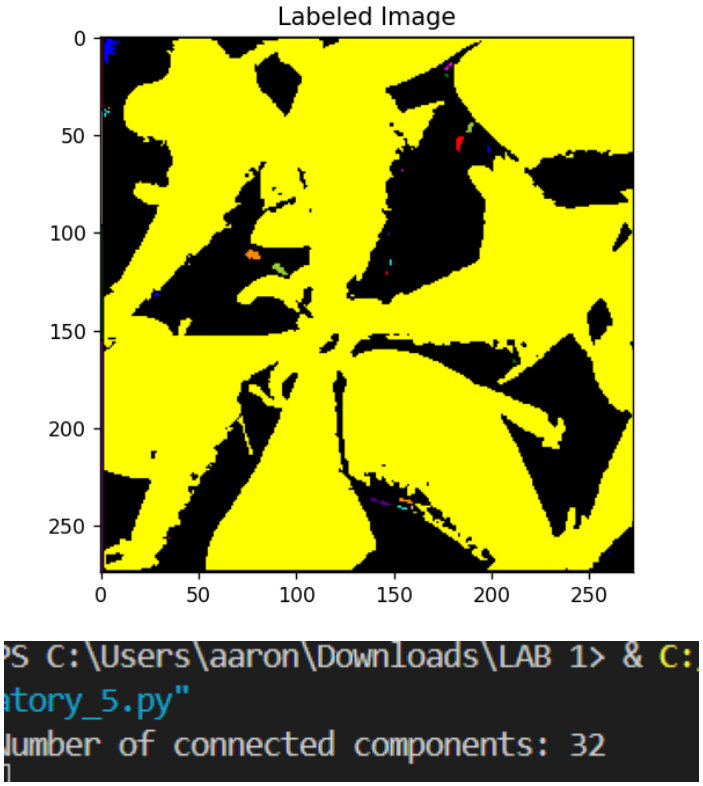


Figure 16: Using Connected Component Labelling

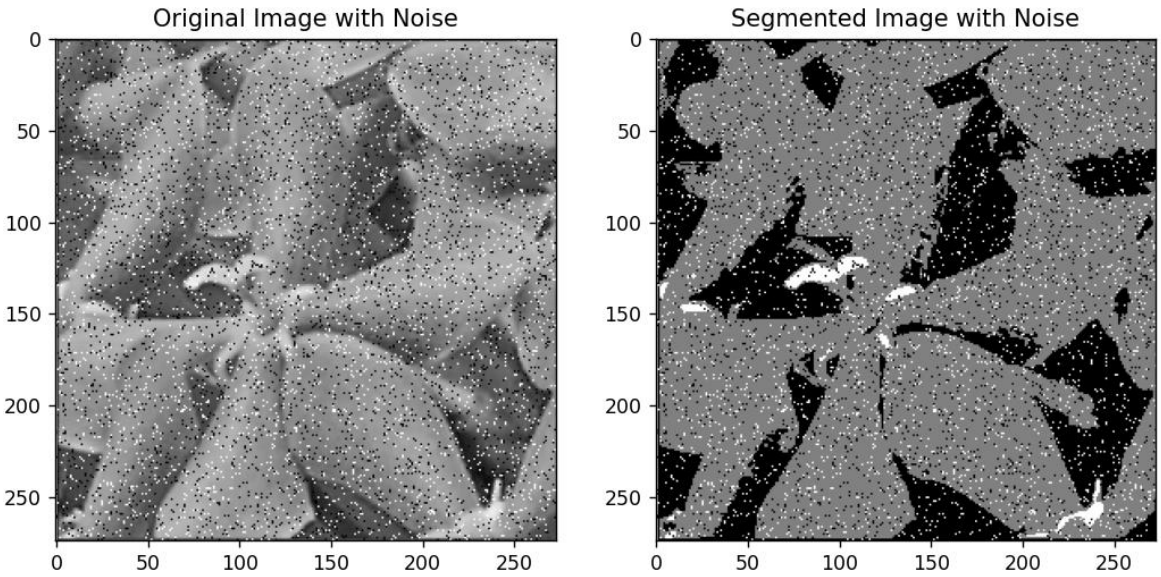
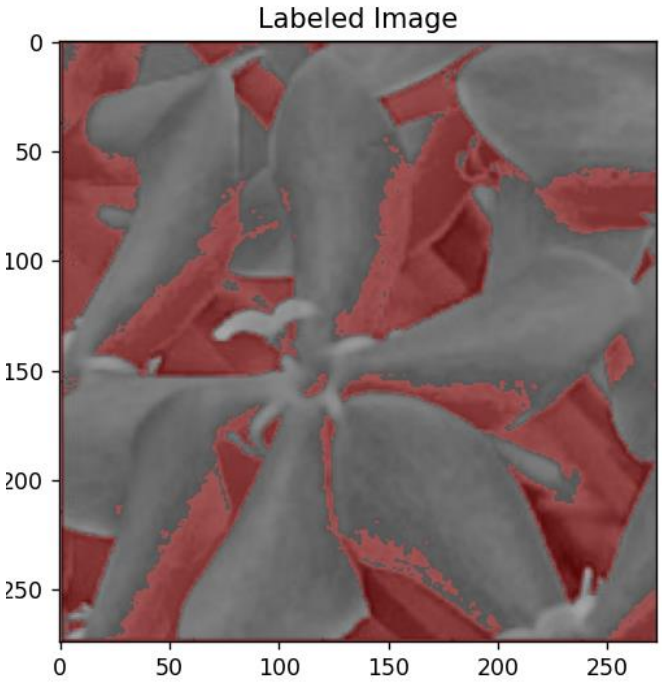


Figure 17: Adding Salt and Pepper Noise



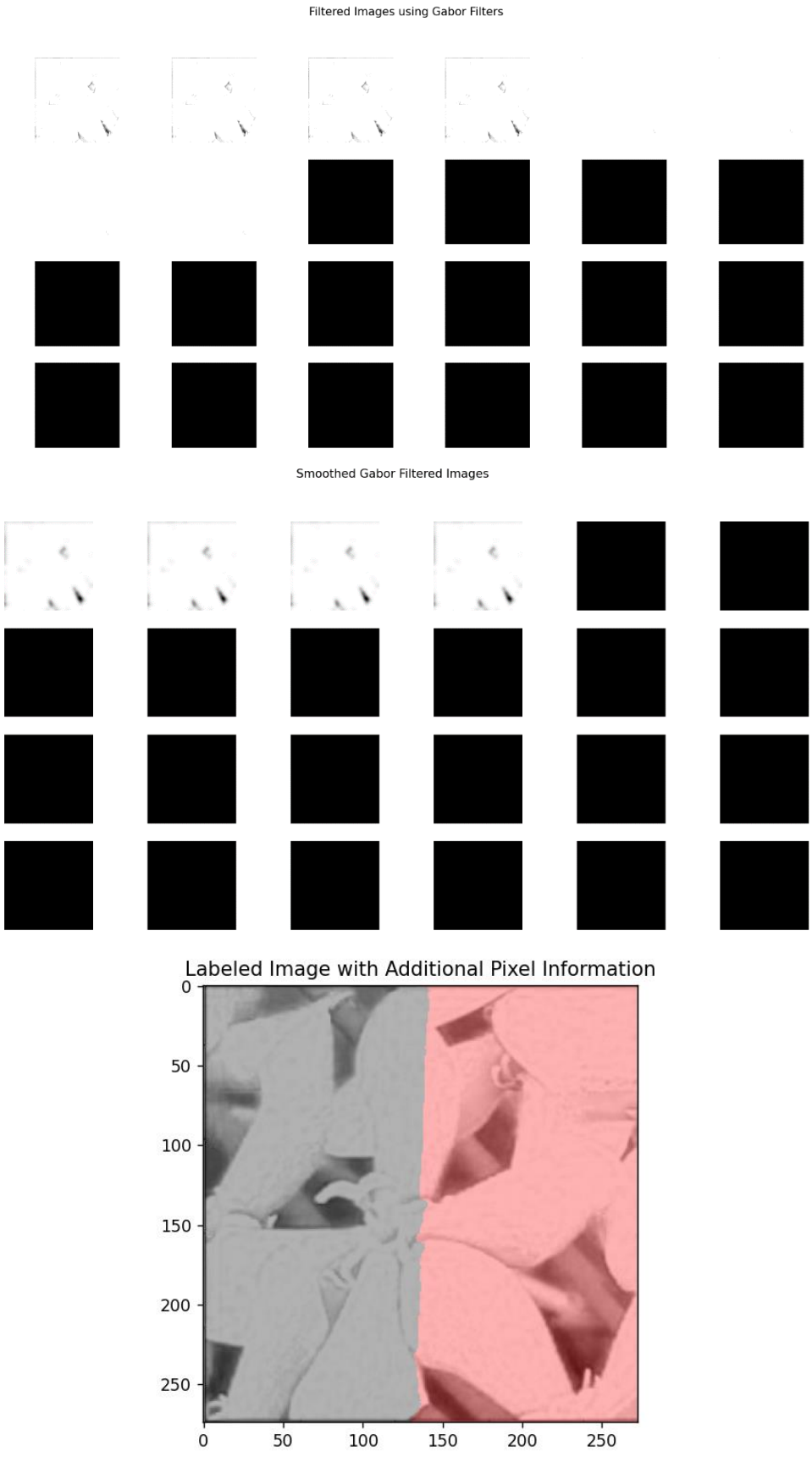


Figure 18: Improve K-means Segmentation Using Texture and Spatial

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. Inthe 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");
```

PYTHON OPEN CV CODE:

```
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
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 / (labels.max() if labels.max() > 0 else 1)),
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 in range(num_kernels):
    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=RGB, bg_label=0)
```

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

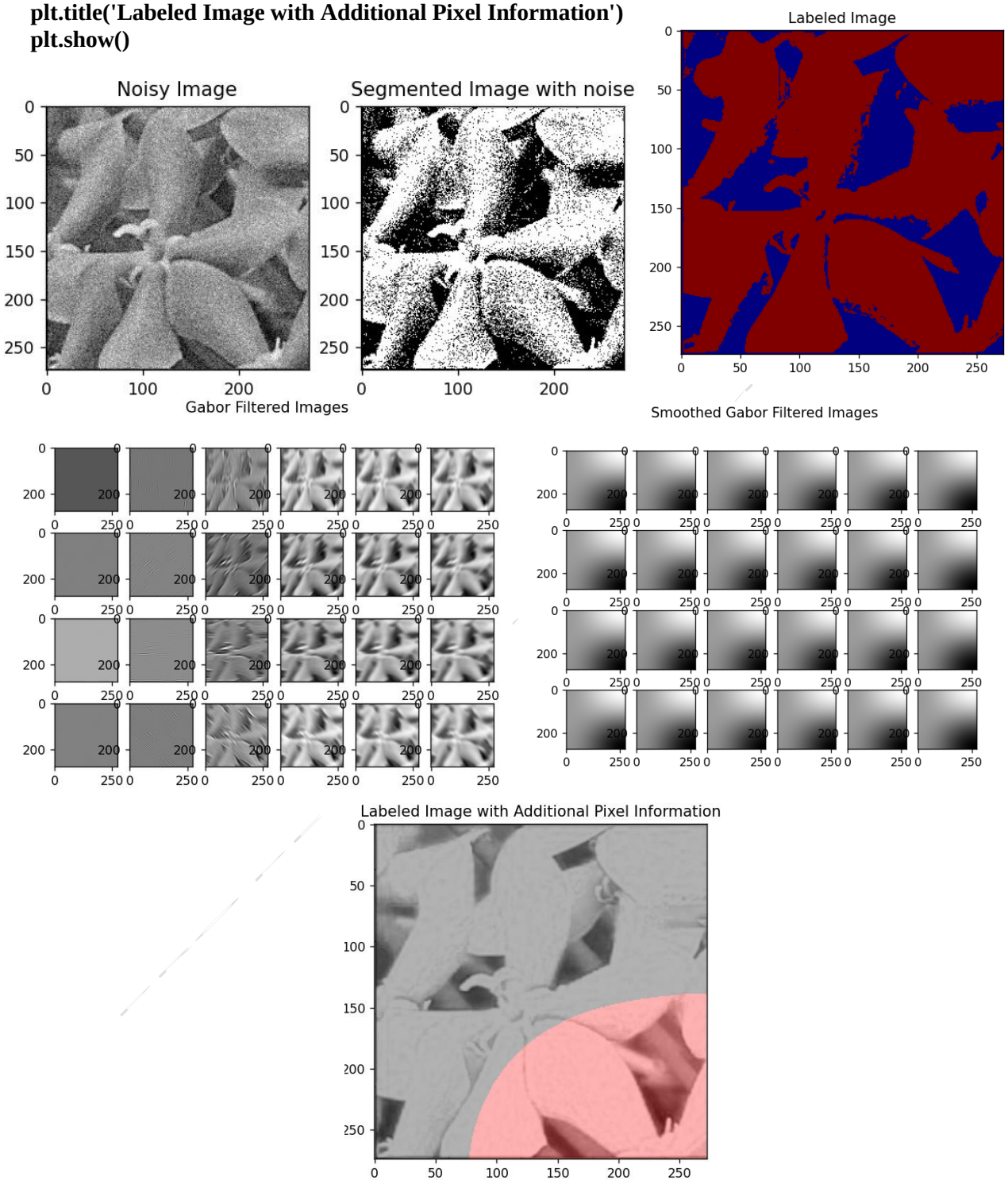


Figure 19: Parameters Modification in Python OpenCV

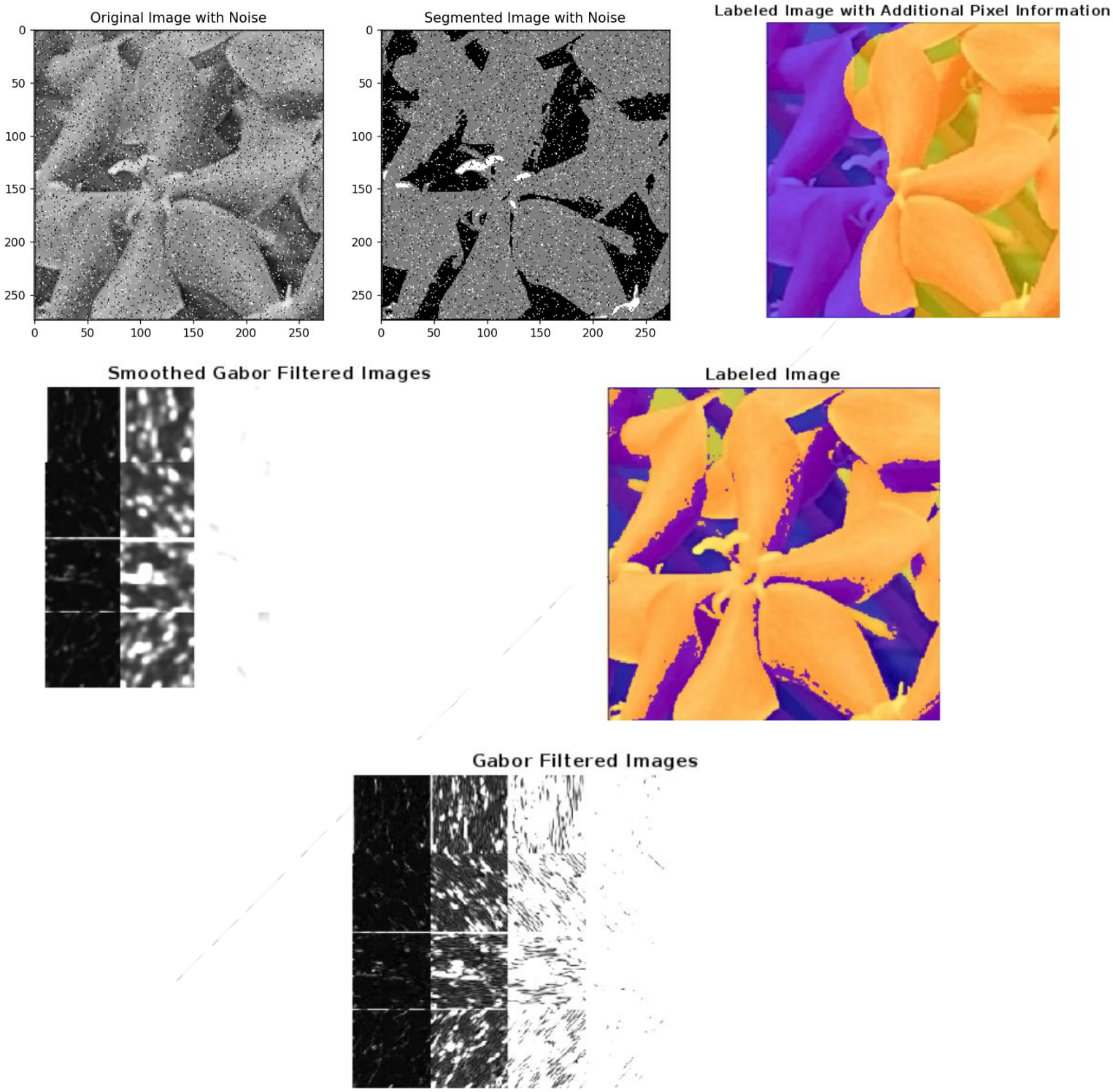


Figure 20: Parameters Modification in MATLAB

2. Visualize the results, analyze and interpret:

The effects of applying an algorithm to images in terms of noise removal and edge detection. The algorithm used is a noise-reduction technique called "Non-local Means Denoising." This method is effective in eliminating noise from the images while preserving important features, such as edges. The document indicates that applying the algorithm results in cleaner images with reduced noise levels, making the images easier to analyze.

The document also discusses the use of edge detection through the "Canny Edge Detection" method. This algorithm effectively highlights the edges within the images, providing a clearer outline of the structures present. This is particularly useful in scenarios where identifying boundaries and edges is crucial.

IV. Conclusion

To summarize the effectiveness of the algorithms used. It states that the Non-local Means Denoising algorithm is successful in reducing noise without significantly distorting the image's details. The Canny Edge Detection algorithm is also effective in accurately identifying edges within the image. The combination of these algorithms contributes to enhanced image quality and better analysis outcomes, achieving the desired results of the laboratory activity.

References

Segmentation — Image analysis in Python. (n.d.). https://scikit-image.org/skimage-tutorials/lectures/4_segmentation.html

Python, R. (2023, August 4). *K-Means Clustering in Python: A Practical Guide*. https://realpython.com/k-means-clustering-python/

Global histogram threshold using Otsu's method - MATLAB otsuthresh. (n.d.). https://www.mathworks.com/help/images/ref/otsuthresh.html

<This is in a separate page>