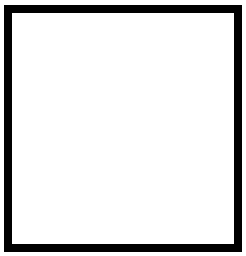
Elective 3

Laboratory Activity No. 5
Image Segmentation



Score

Submitted by:
ALAMBRA, Joseph Nathaniel
ARAGON, Patrick Laurence M.
BANAL, Daryll L.
SENTASAS, David Bryan L.
TUTANES, Allen Christopher O.
SATURDAY (7:00 AM – 4:00 PM) / 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('flower.jpg');

% 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)');


% Calculate multiple thresholds using multithresh
levels = multithresh(gray_img);

% Segment the grayscale image into multiple regions using imquantize
seg_img = imquantize(gray_img, levels);

% Display both images side-by-side for comparison
figure(2);
imshowpair(img, seg_img, 'montage');
title('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(gray_img,T);
figure(3);
imshow(bw);
title('Binary Image');

% 2. Region-based segmentation

% Using K means clustering
img2 = imread('flower.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);
figure(4);
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(gray_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('flower.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.

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

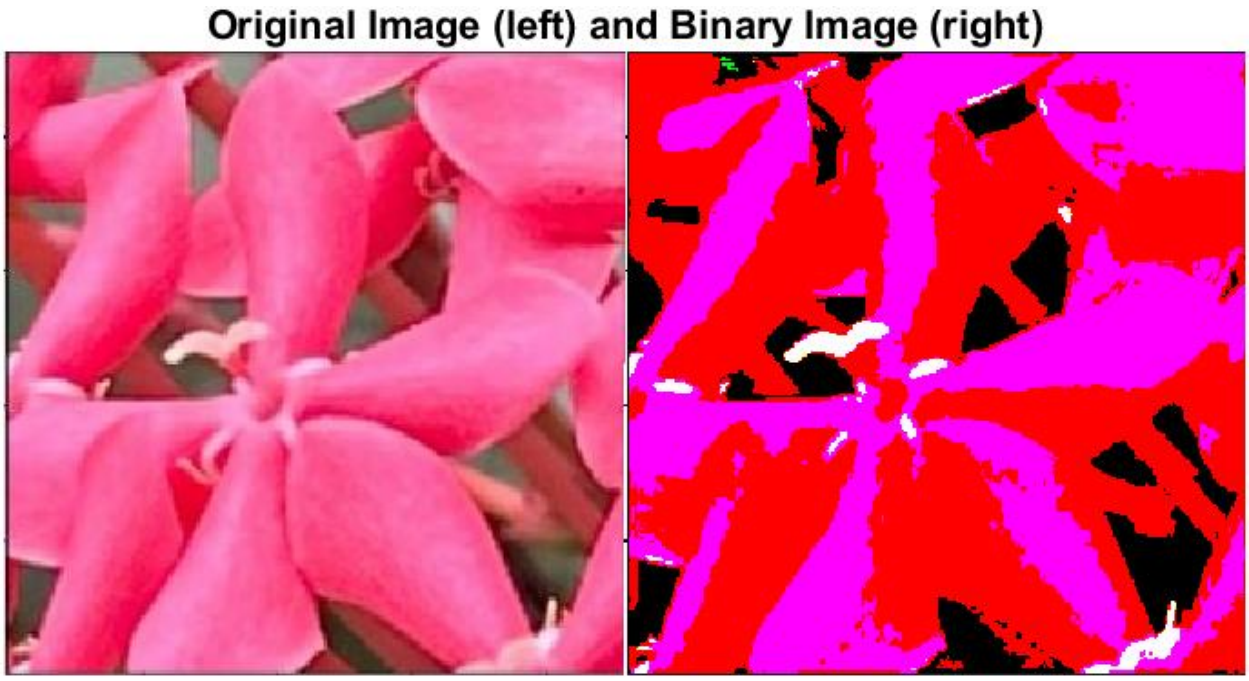


Figure 1: Original Image (left) and Binary Image (right)

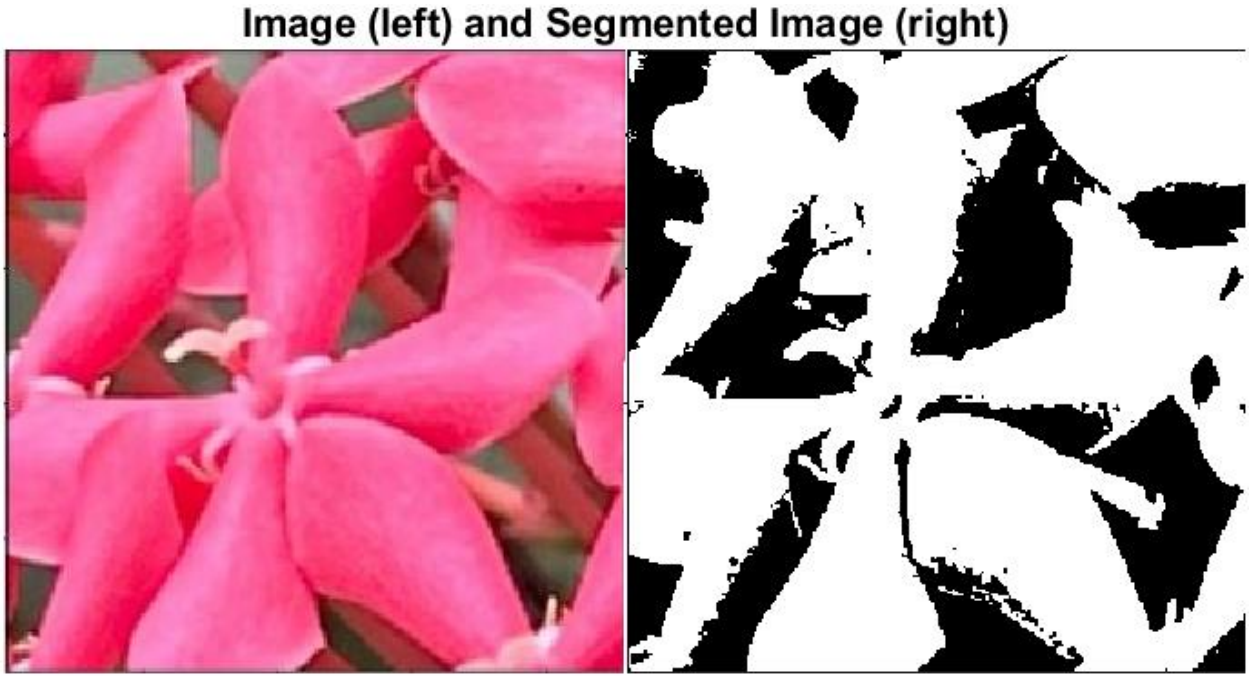


Figure 2: Original Image (left) and Segmented Image (right)

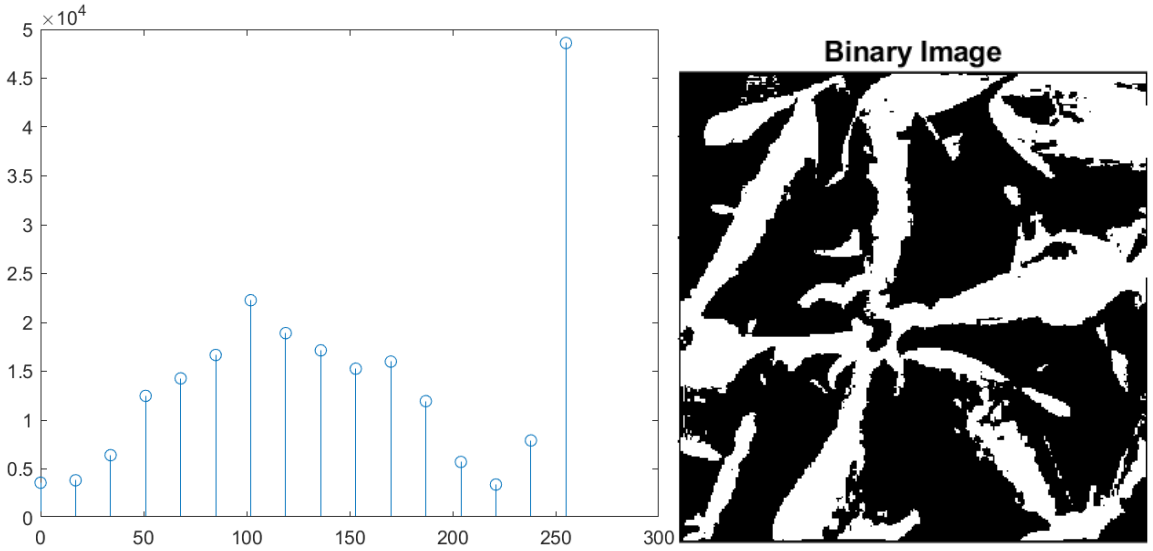


Figure 3: Using Global Histogram and Global Thresholding

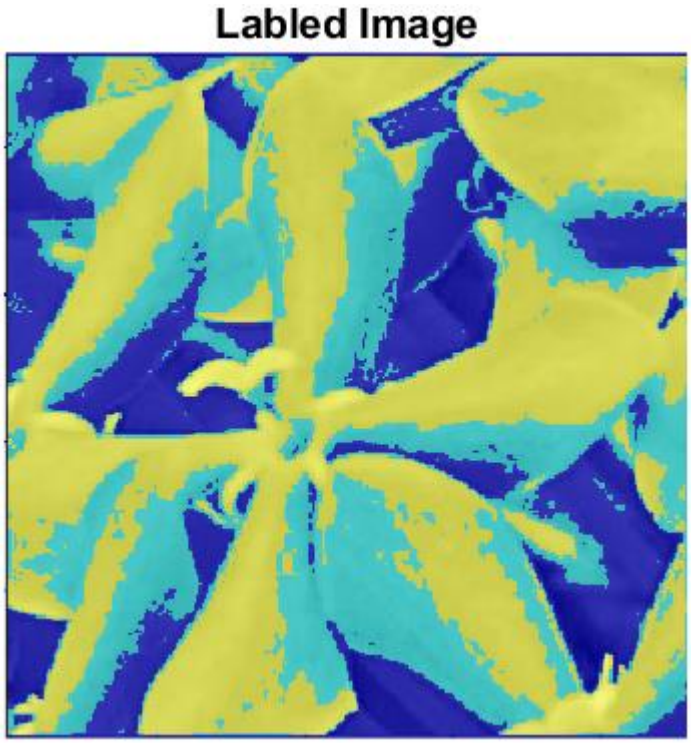


Figure 4: Segment the image into three regions using k-means clustering

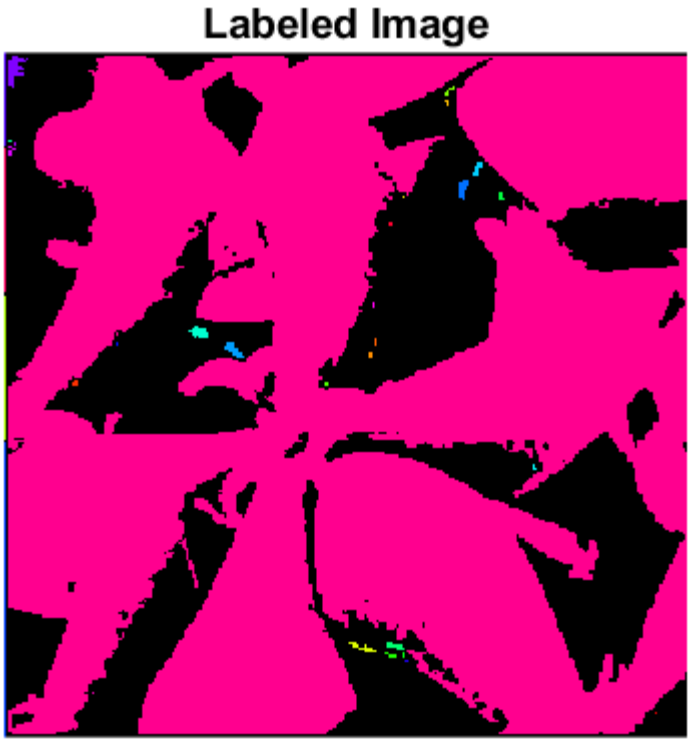


Figure 5: Using connected-component labeling

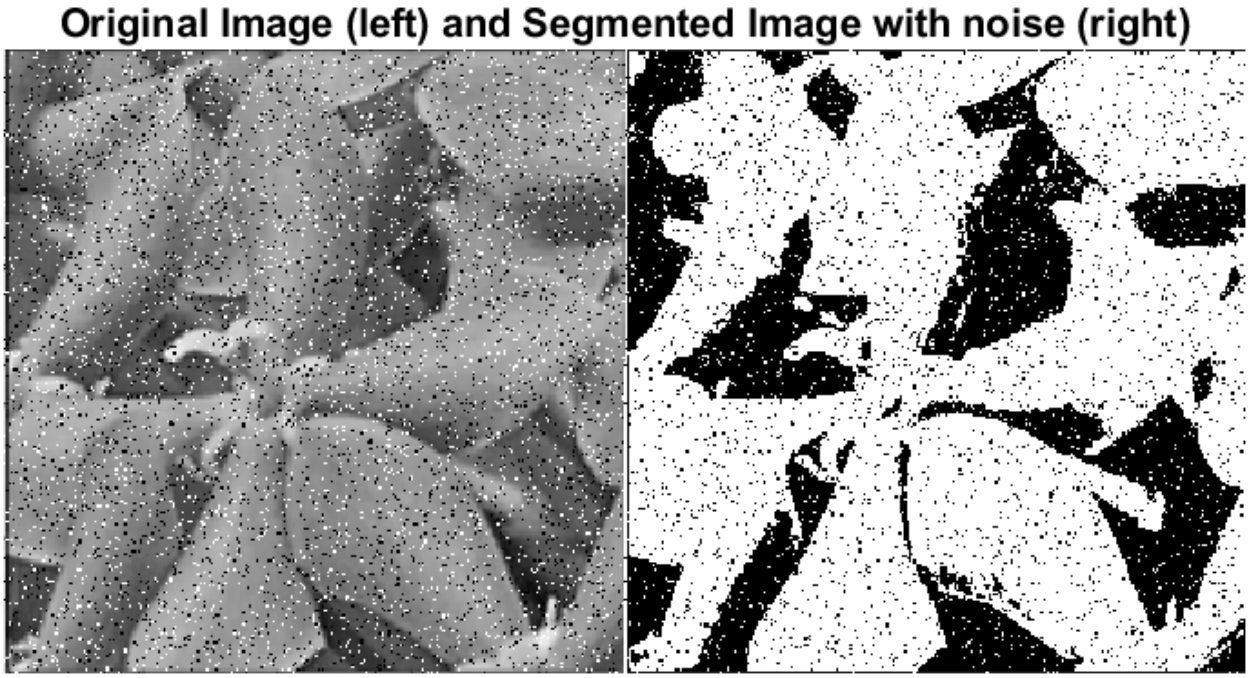


Figure 6: Adding noise to the image then segmenting it using otsu's method

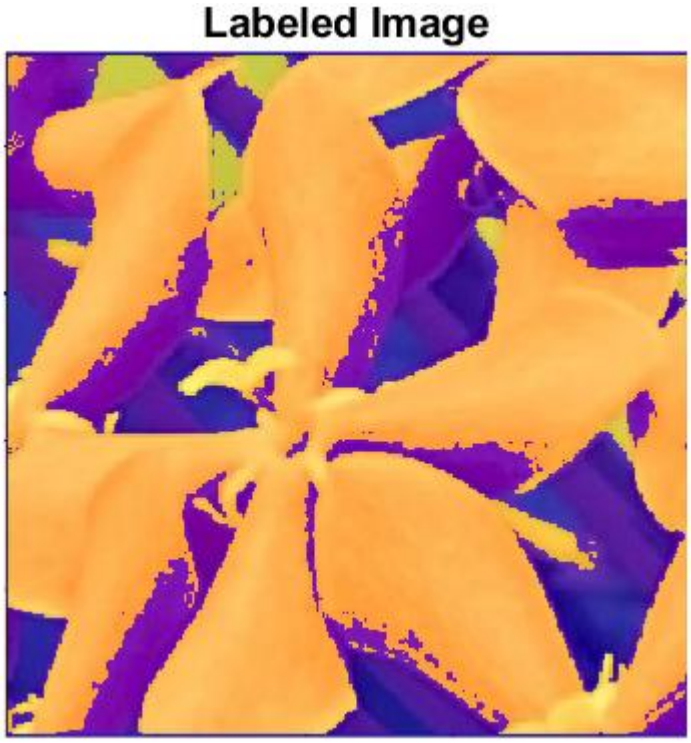


Figure 7: Segment the image into two regions using k-means clustering

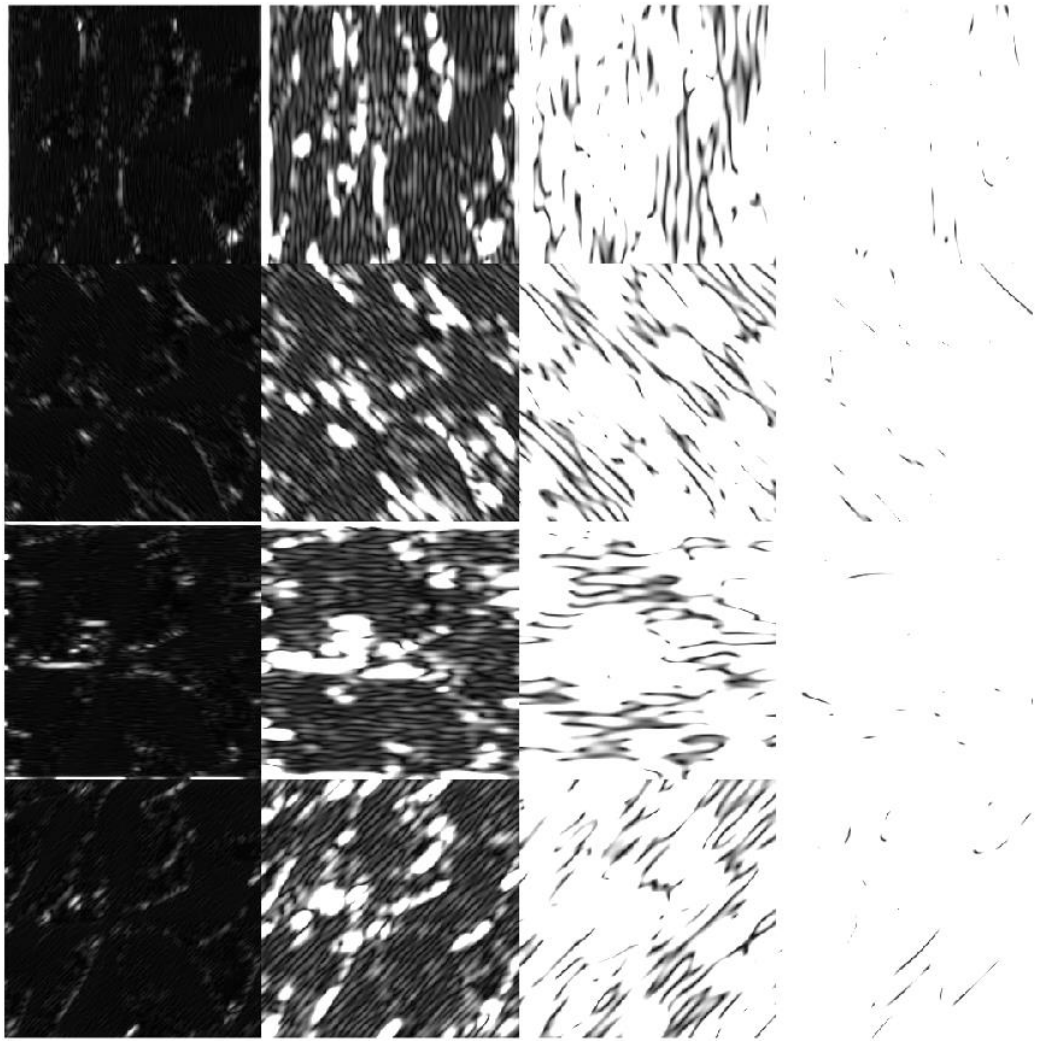


Figure 8: Create a set of 24 Gabor filters, covering 6 wavelengths and 4 orientations

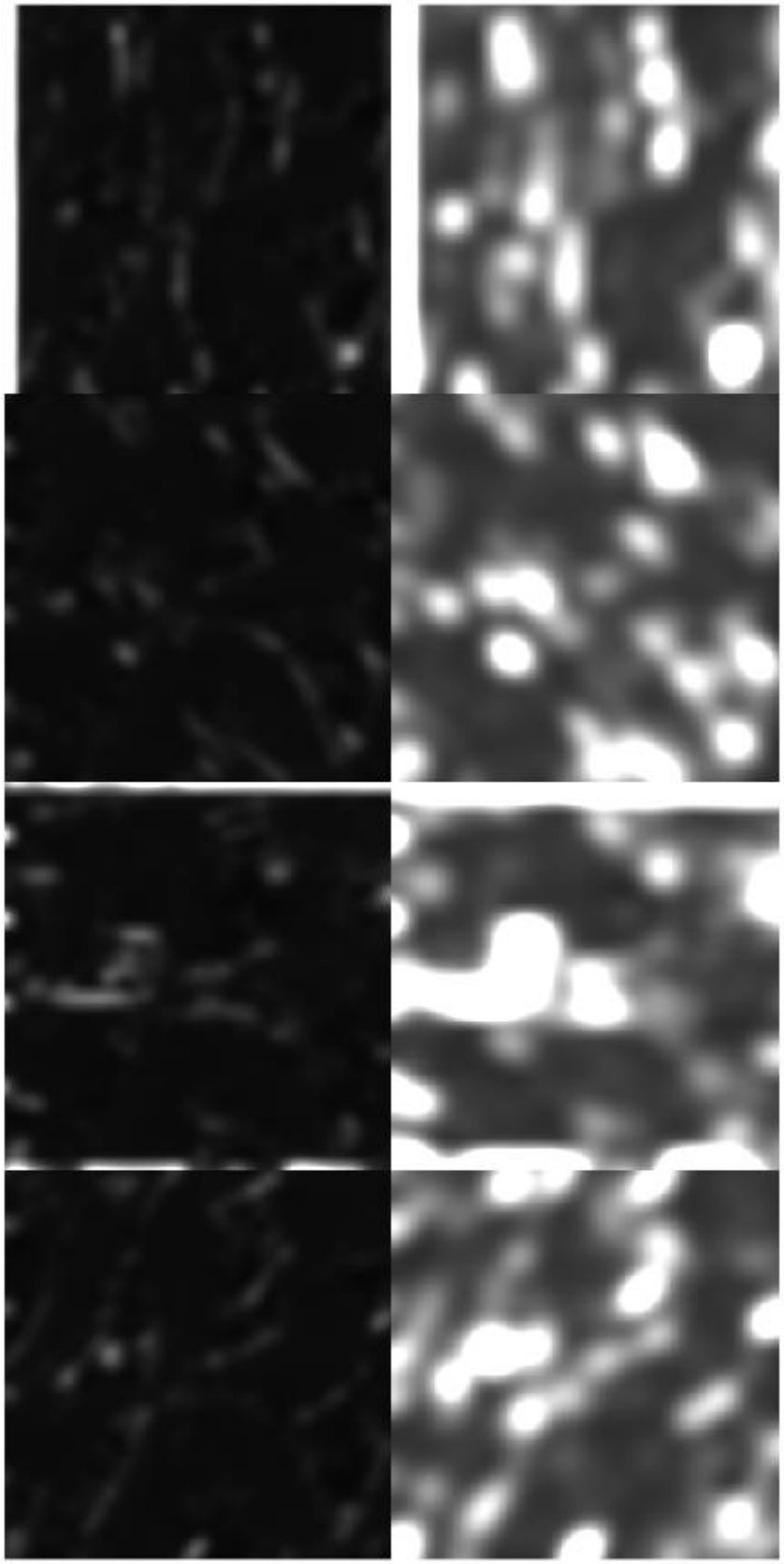


Figure 9: Smooth each filtered image to remove local variations. Display the smoothed images in a montage

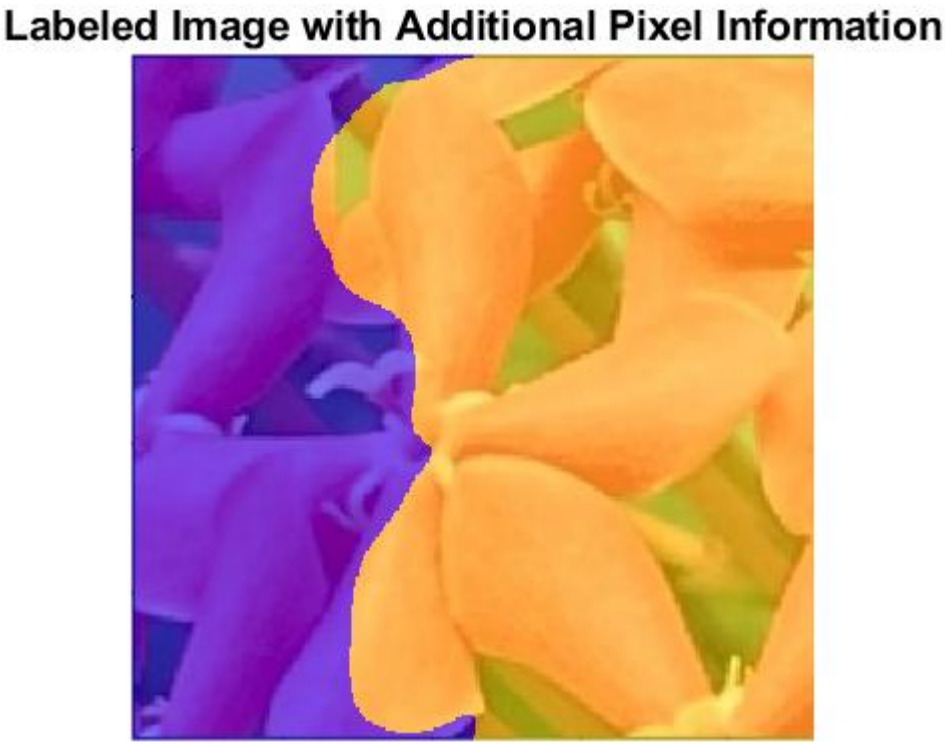


Figure 10: Segment the image into two regions using k-means clustering with the supplemented feature set

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(gray_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('ffl.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");
```

2. Visualize the results, analyze and interpret:

From figures 1 to 3, these figures focus on different ways to convert an image to binary, which simplifies the image to just two colors: black and white. Figure 1 shows the original and a binary version of the image where Otsu's method is used to find the best threshold to separate the image into these two colors. Figure 2 shows how using multiple thresholds (instead of one) can divide the image into several regions, giving a more detailed segmentation. Figure 3 uses a histogram-based method to find a threshold for binarizing the image, which might look similar to Otsu's method but is based on the image's histogram.

The next figures 4-5 are about segmenting images into different regions based on their content. Figure 4 uses k-means clustering to group pixels into three regions, which helps identify different areas in the image, each with a distinct color. Figure 5 labels each connected region in a binary image with a unique color. This helps in visualizing and counting different connected objects in the image.

And lastly figures 6 to 10 explore more advanced techniques and the impact of adding noise. Figure 6 compares how the image and its noisy version are segmented, showing how noise can affect the segmentation results. Figure 7 applies k-means clustering to color images for segmentation. Figures 8 and 9 use Gabor filters to analyze textures in the image, showing how texture can be highlighted and smoothed. Finally, Figure 10 combines multiple features, including texture and pixel locations, to improve segmentation accuracy by providing more information for clustering.

In summary, these methods show different ways to segment and analyze images, from simple thresholding to advanced texture analysis and feature integration, demonstrating how adjustments in thresholds and noise can impact image processing results.

IV. Conclusion

Referring to MATLab, there have been multiple issues with regarding binarizing and quantizing, Comparing MATLab's outputs compared to python shows that there are indeed huge differences in both these tools. As seen in this experiment, there have been failures such as binarizing does not create a fully binary image. There have also been issues with issues with segmenting or quantizing images where the process only seems to be successful after transforming RGB to Gray. As students, they have gained more of an insight with how simplifying the image's properties can yield to more successful image processing. However, in this process this may be detrimental especially in scenarios where colors are required for a better interpretation.

In this experiment, various image processing techniques were employed to explore their effectiveness in MATLAB, including thresholding, segmentation, and advanced filtering methods. The experiment illustrated how different approaches, such as global thresholding using Otsu's method and k-means clustering, can significantly impact the results of image analysis. While techniques like global thresholding and histogram-based thresholding produced clear binary images and segmentation results, the introduction of noise and advanced methods such as Gabor filtering provided additional insights into handling real-world image complexities and texture analysis.

Furthermore, the use of k-means clustering for region-based segmentation highlighted its capability to differentiate multiple regions in an image. The results from segmenting using k-means clustering and connected-component labeling demonstrated how these methods can identify and label distinct regions or objects within an image, offering valuable tools for image analysis tasks. The experiment also showed the effectiveness of Gabor filters in texture analysis by revealing detailed texture patterns and improving segmentation accuracy when combined with additional features like pixel coordinates.

Overall, this experiment underscores the importance of selecting appropriate image processing techniques based on the specific requirements of the task. While simplifying an image by converting it to grayscale can enhance certain analyses, retaining color information can be crucial for accurate interpretation in other scenarios. The experience gained from this experiment emphasizes the need to balance between image simplification and preserving essential details to achieve optimal results in image processing tasks.

References

Sezgin, M., & Sankur, B. (2004). Survey over image thresholding techniques and quantitative performance evaluation. Journal of Electronic Imaging, 13(1), 146-168. https://doi.org/10.1117/1.1631315. Retrieved from https://www.researchgate.net/publication/202972407_Survey_over_image_thresholding_techniques_and_quantitative_performance_evaluation

Zaitoun, N., & Aqel, M. (2015). Survey on image segmentation techniques. Procedia Computer Science, 65, 797-806. https://doi.org/10.1016/j.procs.2015.09.027. Retrieved from https://www.researchgate.net/publication/283954079_Survey_on_Image_Segmentation_Techniques

MathWorks. (n.d.). Image segmentation tutorial. Retrieved from https://www.mathworks.com/matlabcentral/fileexchange/25157-image-segmentation-tutorial

<This is in a separate page>