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

Elective 3

Laboratory Activity No. 5
Image Segmentation

Score

Submitted by:
Cacal, Chad R.
Dela Cruz, Alfonso Rafael C.
Palabay, Joven Carl B.
Suyu, Chester T.
SAT 7:00AM – 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('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');
```

B. Supplementary Activity

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

```
import cv2
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from skimage import color, filters
from skimage import measure
from scipy.ndimage import convolve
from skimage.filters import threshold_otsu, gabor_kernel, gabor
from skimage.util import random_noise
img = cv2.imread('flower.jpg')

# Global Image thresholding using Otsu's method
level = cv2.threshold(img,127,255,cv2.THRESH_BINARY)[1]
img_pair = cv2.hconcat([img,level])
cv2.imshow('Original Image (left) and Binary Image (right)',img_pair)

# Multi-level thresholding using Otsu's method
img2_PIL = Image.open('flower.jpg').convert('RGB')
img2_PIL = img2_PIL.quantize(2).convert('RGB')
seg_img = np.array(img2_PIL)[:,:,::-1].copy()
img_pair2 = cv2.hconcat([img,seg_img])
cv2.imshow('Original Image (left) and Segmented Image (right)', img_pair2)

# Create a binary image using the computed threshold and display the image
cv2.imshow('Binary Image', level)

# Region-Based Segmentation
# Using K-Means Clustering
img_Kmeans = img.reshape(-1,3)
kmeans = KMeans(n_clusters=3, n_init=10)
kmeans.fit(img_Kmeans)
segmented_img = kmeans.cluster_centers_[kmeans.labels_]
segmented_img = segmented_img.reshape(img.shape)
plt.imshow(segmented_img/255)
plt.axis('off')
plt.title('Labled Image')

# Using connected-component labeling, convert the image into binary
img_gray = cv2.imread('flower.jpg', cv2.IMREAD_GRAYSCALE)
thresh = threshold_otsu(img_gray)
bin_img2 = img_gray > thresh
labeledImage, numberOfComponents = measure.label(bin_img2,return_num=True)
print(f'Number of connected components: {numberOfComponents}')

# Paramter Modifications
# Adding noise to the image then segmenting it using otsu's method
img_noise = random_noise(img,mode='s&p', amount=0.09)
img_noise = (img_noise * 255).astype(np.uint8)
img_noise_gray = cv2.cvtColor(img_noise, cv2.COLOR_BGR2GRAY)
level = threshold_otsu(img_noise_gray)
seg_img = (img_noise_gray > level).astype(np.uint8) * 255
seg_img = cv2.cvtColor(seg_img, cv2.COLOR_GRAY2BGR)
img_pair = np.hstack([img_noise, seg_img])
```

```
cv2.imshow('Original Image (left) and Segmented Image with noise
(right)',img_pair)

#Segment the image into two regions using k-means clustering
img_Kmeans = seg_img.reshape(-1,3)
kmeans = KMeans(n_clusters=2, n_init=10)
kmeans.fit(img_Kmeans)
segmented_img = kmeans.cluster_centers_[kmeans.labels_]
segmented_img = segmented_img.reshape(seg_img.shape)
plt.imshow(segmented_img/255)
plt.axis('off')
plt.title('Labeled Image')
plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

III. Results

Steps:

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

MATLAB Results

Figure 1a. Acquire an Image of a Flower

Figure 2a. Original Image (left) and Binary Image (right)

Figure 3a. Original Image (left) and Binary Image (right)

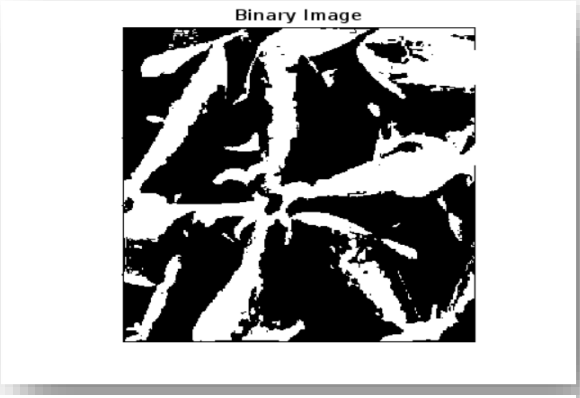

Figure 4a. Using Global Histogram

Figure 5a. Using Multi-level Thresholding

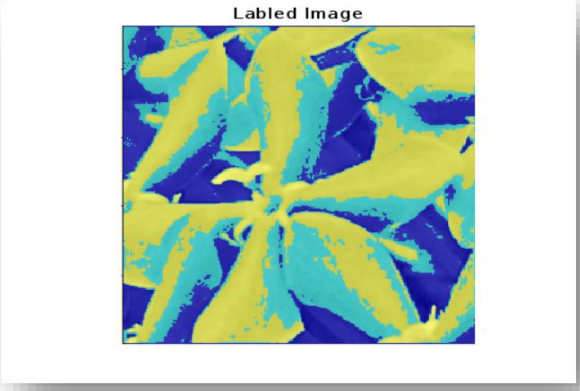

Figure 6a. Using K-means

Figure 7a. Using Connected Component Labelling

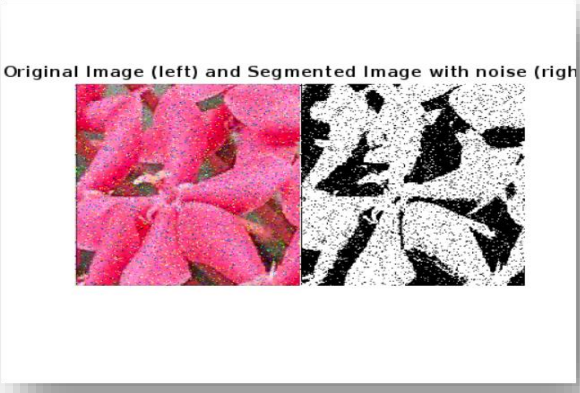

Figure 8a. Adding Salt and Pepper Noise

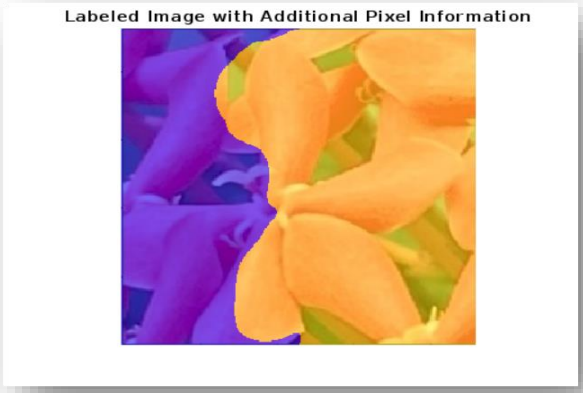

Figure 9a. Improve K-means Segmentation Using Texture and Spatial

Octave Results

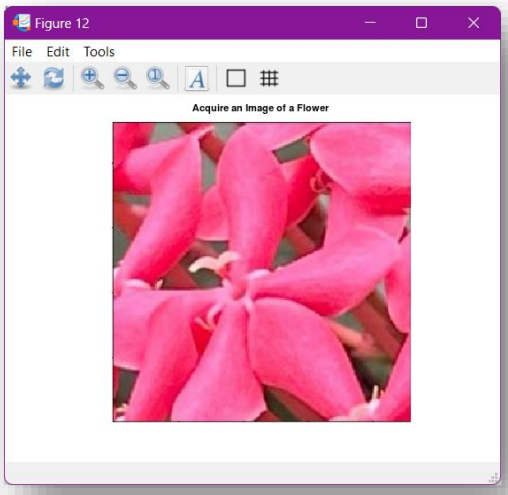

Figure 1b. Acquire an Image of a Flower

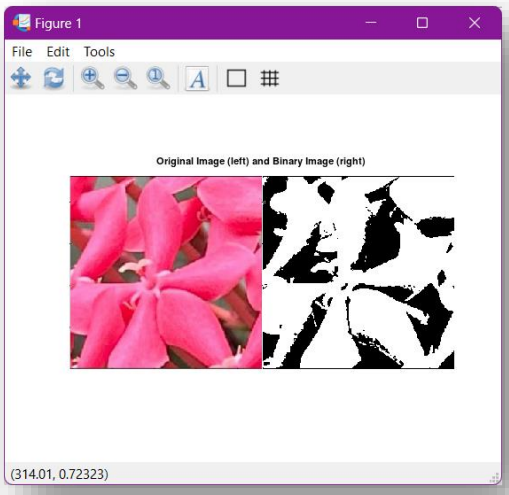

Figure 2b. Original Image (left) and Binary Image (right)

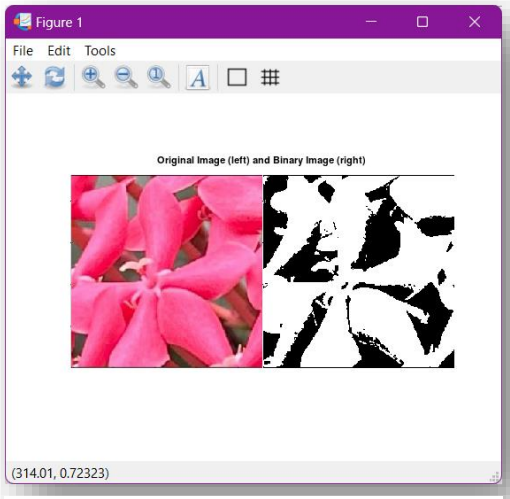

Figure 3b. Original Image (left) and Binary Image (right)

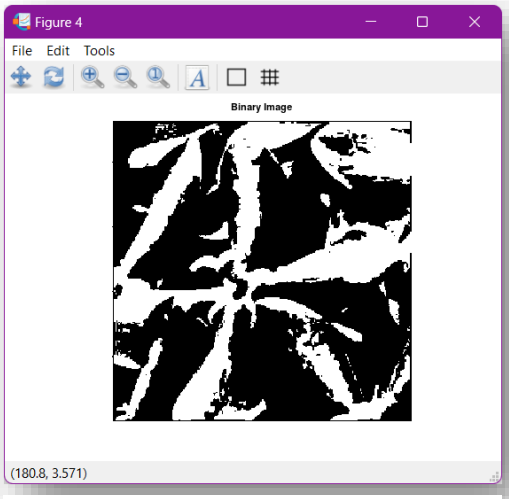

Figure 4b. Using Global Histogram

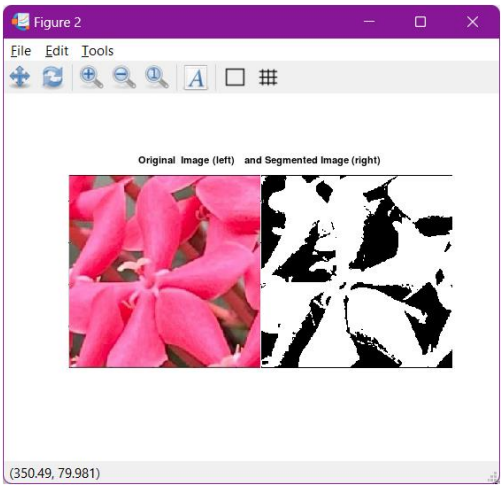

Figure 5b. Using Multi-level Thresholding

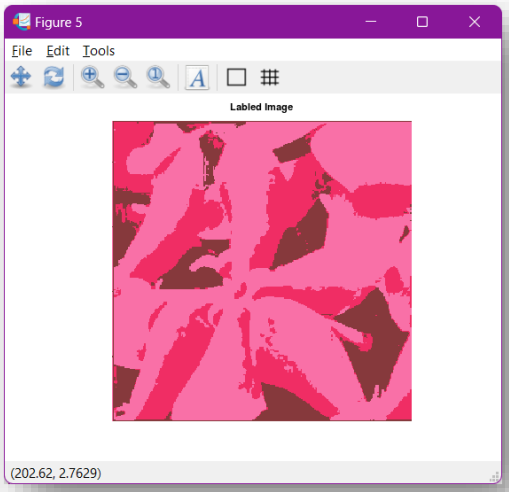

Figure 6b. Using K-means

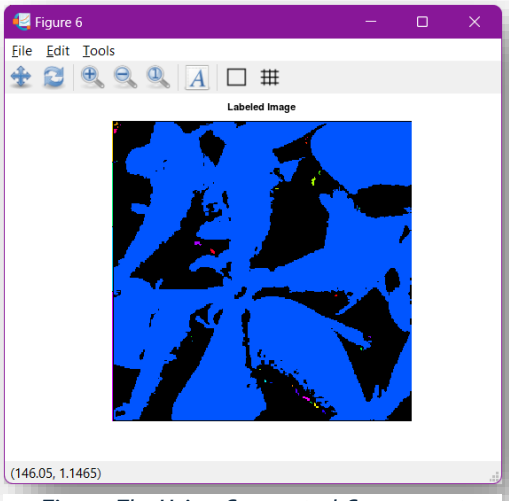

Figure 7b. Using Connected Component Labelling

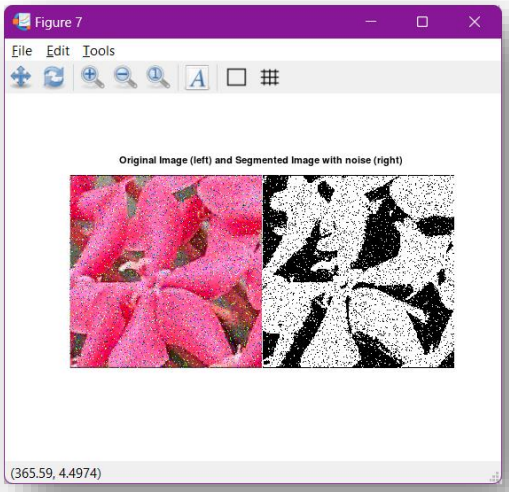

Figure 8b. Adding Salt and Pepper Noise

Python Image

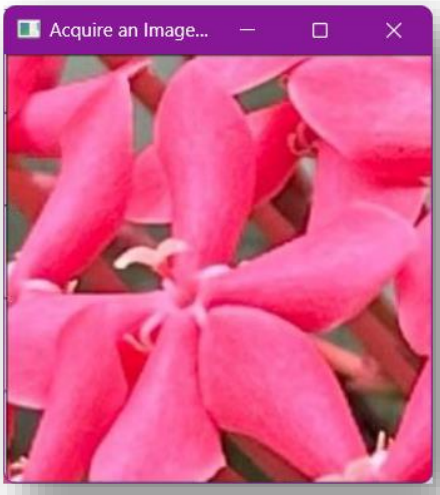

Figure 1c. Acquire an Image of a Flower

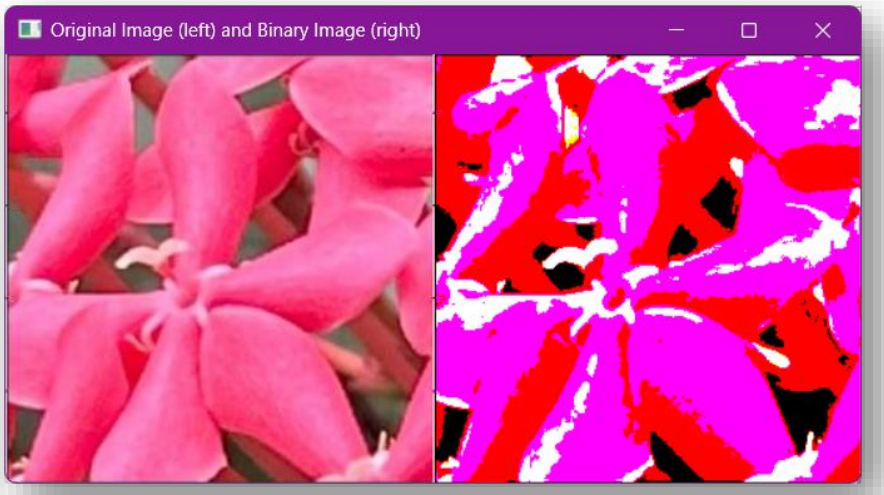

Figure 2c. Original Image (left) and Binary Image (right)

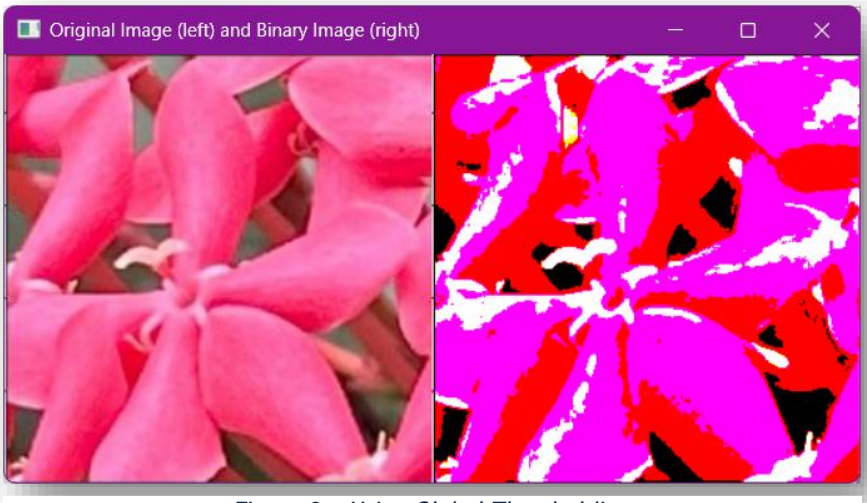

Figure 3c. Using Global Thresholding

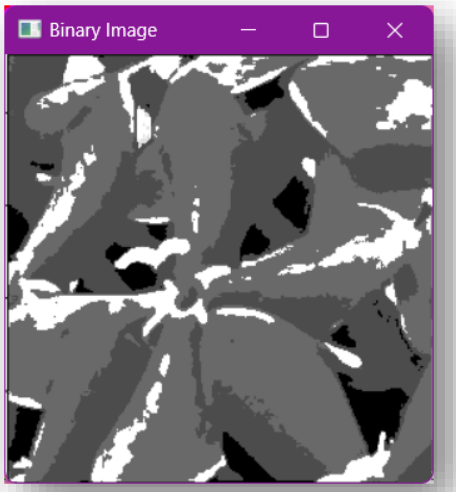

Figure 4c. Using Global Histogram

Figure 5c. Using Multi-level Thresholding

Figure 6c. Using K-means

Figure 8c. Adding Salt and Pepper Noise

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

```
% % 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");
```

2. Visualize the results, analyze and interpret:

When comparing image acquisition, segmentation, and thresholding techniques across MATLAB, Octave, and Python, distinct differences in performance, accuracy, and ease of use emerge due to the varying capabilities of their respective libraries and implementations.

MATLAB provides a highly optimized and user-friendly environment for image processing, with robust built-in functions like imread for image acquisition, and powerful segmentation tools such as imsegkmeans, activecontour, and watershed. These functions are integrated seamlessly within MATLAB's ecosystem, offering advanced options and delivering accurate, reliable results. MATLAB also excels in thresholding techniques, with functions like graythresh (for Otsu's method) and multithresh that are highly optimized for performance. This makes MATLAB particularly strong in providing a comprehensive, intuitive environment for both basic and advanced image processing tasks.

Octave, being an open-source alternative to MATLAB, offers similar functionality but with some limitations. While it supports image acquisition, segmentation, and thresholding through its image package, the range of supported formats, the sophistication of algorithms, and the overall performance may not match MATLAB's level of optimization. As a result, the output quality in Octave might be less refined, particularly for complex image processing tasks. Additionally, Octave's segmentation and thresholding capabilities may require more manual scripting, leading to potential inconsistencies in the results.

Python, on the other hand, provides a flexible and powerful environment for image processing, especially with libraries like OpenCV, Scikit-Image, and PIL (Pillow). These libraries offer comprehensive tools for image acquisition, segmentation, and thresholding, rivaling MATLAB's capabilities. Python's flexibility allows for custom implementations and integration with a wide range of third-party algorithms, which can result in high-quality output. However, the performance and accuracy of Python's image processing tasks may vary depending on the specific libraries and implementations used. While Python can match MATLAB in many aspects, it may require more setup and a deeper understanding of the underlying algorithms to achieve comparable results.

IV. Conclusion

In summary, MATLAB stands out for its ease of use, advanced functionality, and consistent performance, making it a preferred choice for professional image processing tasks. Octave, while useful as a free alternative, may fall short in terms of optimization and refinement. Python offers significant flexibility and power, making it a versatile tool for image processing, though it might require more manual effort to achieve results on par with MATLAB.

References

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

GeeksforGeeks. (2023, Jan 03). *Convert image to binary using Python*. https://www.geeksforgeeks.org/convert-image-to-binary-using-python/

W3Schools. (n.d.). *Machine Learning – K-means*. https://www.w3schools.com/python/python_ml_k-means.asp

Matplotlib. (n.d.). *matplotlib.pyplot.stem*. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.stem.html

Roboflow. (n.d.). *Convert PIL Image to cv2 image*. https://roboflow.com/use-opencv/convert-pil-image-to-cv2-image

NeuralNine. (n.d.). *Image Segmentation with K-Means Clustering in Python [Video]*. https://www.youtube.com/watch?v=X-Y91ddBqaQ

OpenCV. (n.d.). *Image Thresholding*. https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html