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

Elective 3

Laboratory Activity No. 5
Image Segmentation

Score

Submitted by:
Bolocon, Joniel R.

De Guzman, Jastine V.

De Juan, Lord Welchie P.

Mercado, Ed-Vir G.

Puno, Harold Dennis R.

Saturday 7:00 AM – 4:00 PM / CPE 0332.1-1

Date Submitted
11-08-2024

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

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

# Load image
img = cv2.imread('flower.jpg')
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Display the original image as Figure 1
plt.figure(1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.show()

# Global Thresholding using Otsu's Method
_, bw = cv2.threshold(gray_img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

plt.figure(2)
plt.subplot(1, 2, 1), plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)), 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
thresholds = filters.threshold_multiotsu(gray_img, classes=3)
regions = np.digitize(gray_img, bins=thresholds)

# Convert segmented image to RGB
seg_img_rgb = color.label2rgb(regions, image=img, bg_label=0)

plt.figure(3)
plt.subplot(1, 2, 1), plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)), plt.title('Original
Image')
plt.subplot(1, 2, 2), plt.imshow(seg_img_rgb), plt.title('Segmented Image')
plt.show()
```

```
# Histogram Thresholding using Otsu's Method
counts, bins = np.histogram(gray_img.flatten(), bins=16, range=(0, 256))
otsu_thresh = threshold_otsu(gray_img)
_, bw_otsu = cv2.threshold(gray_img, otsu_thresh, 255, cv2.THRESH_BINARY)

plt.figure(4)
plt.bar(bins[:-1], counts, width=bins[1] - bins[0])
plt.title('16-bin Histogram')
plt.xlabel('Intensity Value')
plt.ylabel('Pixel Count')
plt.show()

# Compute histogram counts
counts, bins = np.histogram(gray_img.flatten(), bins=256, range=(0, 256))

# Compute global threshold using Otsu's method
otsu_thresh = filters.threshold_otsu(gray_img)

# Create a binary image using the computed threshold
_, bw = cv2.threshold(gray_img, otsu_thresh, 255, cv2.THRESH_BINARY)

# Convert binary image to RGB for display
bw_rgb = cv2.cvtColor(bw, cv2.COLOR_GRAY2RGB)

# Display the original image and the binary image side by side
plt.figure(5)
plt.subplot(1, 2, 1), plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)), plt.title('Original
Image')
plt.subplot(1, 2, 2), plt.imshow(bw_rgb), plt.title('Binary Image')
plt.show()

# Region-Based Segmentation using K-means
kmeans = KMeans(n_clusters=3, random_state=0).fit(gray_img.reshape(-1, 1))
seg_img = kmeans.labels_.reshape(gray_img.shape)
seg_img_rgb = color.label2rgb(seg_img, image=img, bg_label=0)

plt.figure(6)
plt.imshow(seg_img_rgb)
plt.title('Labeled Image')
plt.show()

# Connected-Component Labeling
_, bin_img = cv2.threshold(gray_img, otsu_thresh, 255, cv2.THRESH_BINARY)
num_labels, labeled_img = cv2.connectedComponents(bin_img)
colored_labels = cv2.applyColorMap(np.uint8(labeled_img * 255 / num_labels),
cv2.COLORMAP_JET)

print('Number of connected components:', num_labels)
```

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

# Parameter Modifications with Noise
# Add noise to the RGB image
noise_img = np.clip(img + np.random.normal(0, 25, img.shape).astype(np.uint8), 0, 255)

# Segment the noisy RGB image
kmeans_noise = KMeans(n_clusters=3, random_state=0).fit(noise_img.reshape(-1, 3))
labels_noise = kmeans_noise.labels_.reshape(noise_img.shape[:2])
label_overlay_noise = color.label2rgb(labels_noise, image=noise_img, bg_label=0)

plt.figure(8)
plt.subplot(1, 2, 1), plt.imshow(cv2.cvtColor(noise_img, cv2.COLOR_BGR2RGB)), plt.title('Noisy
Image')
plt.subplot(1, 2, 2), plt.imshow(label_overlay_noise), plt.title('Segmented Image with Noise')
plt.show()

# Segment the original RGB image into regions using K-means
kmeans_rgb = KMeans(n_clusters=2, random_state=0).fit(img.reshape(-1, 3))
labels_rgb = kmeans_rgb.labels_.reshape(img.shape[:2])
label_overlay_rgb = color.label2rgb(labels_rgb, image=img, bg_label=0)

plt.figure(9)
plt.imshow(label_overlay_rgb)
plt.title('Segmented Image')
plt.show()

# Create and apply Gabor filters
def gabor_filter(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))
gabor_kernels = gabor_filter(gray_img, wavelength, orientation)

gabor_mag = np.zeros_like(gray_img, dtype=np.float32)
for kernel in gabor_kernels:
    filtered_img = cv2.filter2D(gray_img, cv2.CV_32F, kernel)
    gabor_mag = np.maximum(gabor_mag, np.abs(filtered_img))
```

```
plt.figure(10)
num_kernels = len(gabor_kernels)
for i in range(num_kernels):
    plt.subplot(4, 6, i + 1)
    plt.imshow(cv2.filter2D(gray_img, cv2.CV_32F, gabor_kernels[i]), cmap='gray')
plt.suptitle('Gabor Filtered Images')
plt.show()

# Smooth each filtered image
for i, kernel in enumerate(gabor_kernels):
    sigma = 0.5 * wavelength[i % len(wavelength)]
    gabor_mag = cv2.GaussianBlur(gabor_mag, (0, 0), sigma)

plt.figure(11)
for i in range(num_kernels):
    plt.subplot(4, 6, i + 1)
    plt.imshow(gabor_mag, cmap='gray')
plt.suptitle('Smoothed Gabor Filtered Images')
plt.show()

# Feature set and K-means clustering
x, y = np.meshgrid(np.arange(gray_img.shape[1]), np.arange(gray_img.shape[0]))
feature_set = np.stack([gray_img, gabor_mag, x, y], axis=-1)

feature_set_reshaped = feature_set.reshape(-1, feature_set.shape[-1])
kmeans_feature = KMeans(n_clusters=2, random_state=0).fit(feature_set_reshaped)
labels_feature = kmeans_feature.labels_.reshape(gray_img.shape)
label_overlay_feature = color.label2rgb(labels_feature, image=img, bg_label=0)

plt.figure(12)
plt.imshow(label_overlay_feature)
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

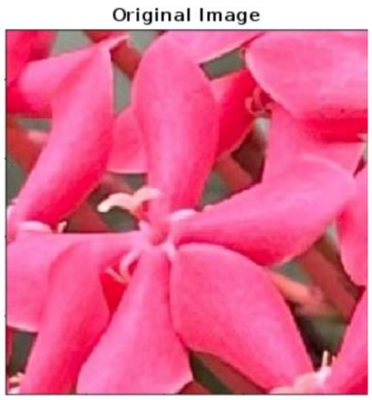


Figure 1: Acquire an Image of a Flower (MATLAB)

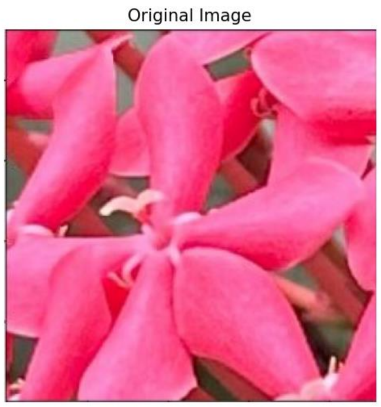


Figure 1.1: Acquire an Image of a Flower (PYTHON)

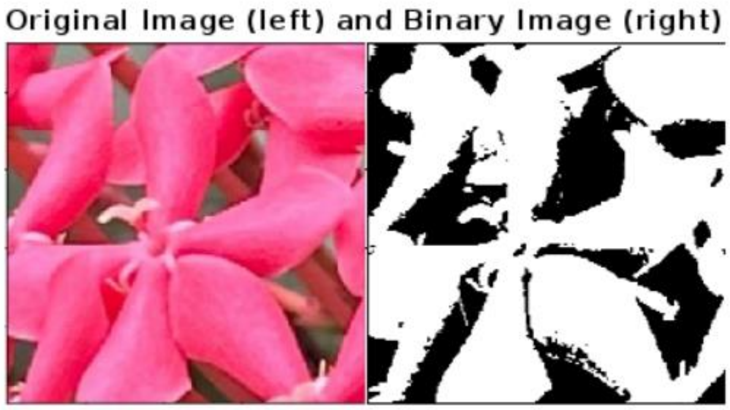


Figure 2: Original Image (left) and Binary Image (right) (MATLAB)

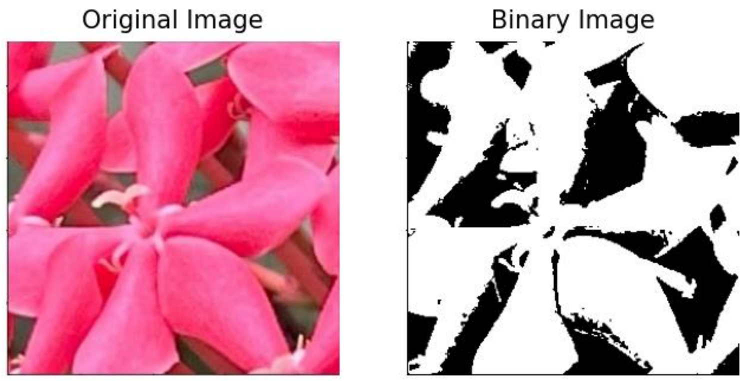


Figure 2.1: Original Image (left) and Binary Image (right) (PYTHON)

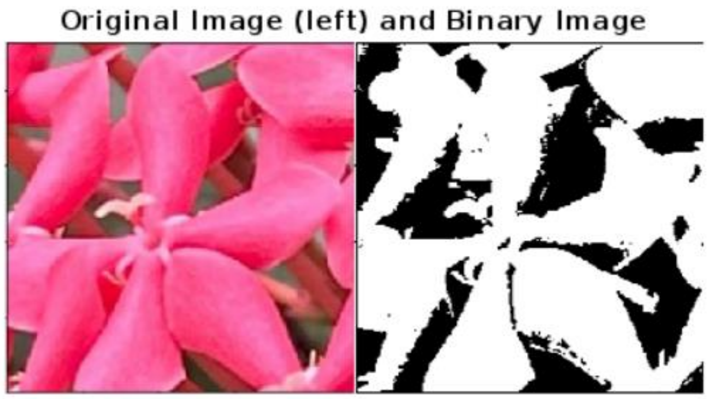


Figure 3: Using Global Thresholding (MATLAB)

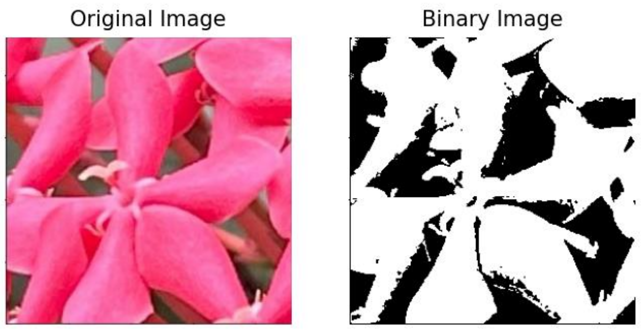


Figure 3.1: Using Global Thresholding (PYTHON)

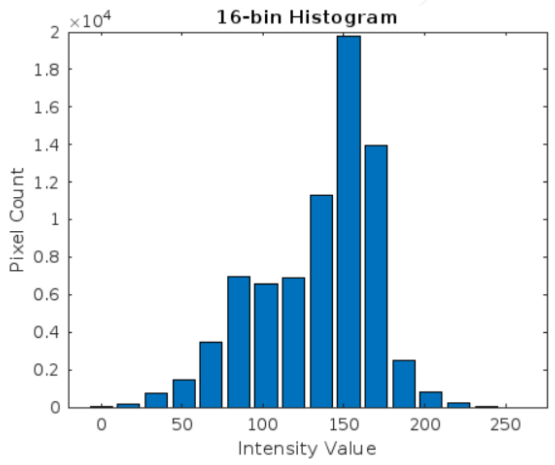


Figure 4: Using Global Histogram (MATLAB)

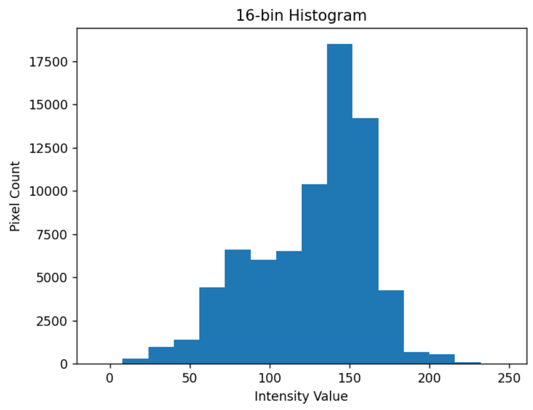


Figure 4.1: Using Global Histogram (PYTHON)

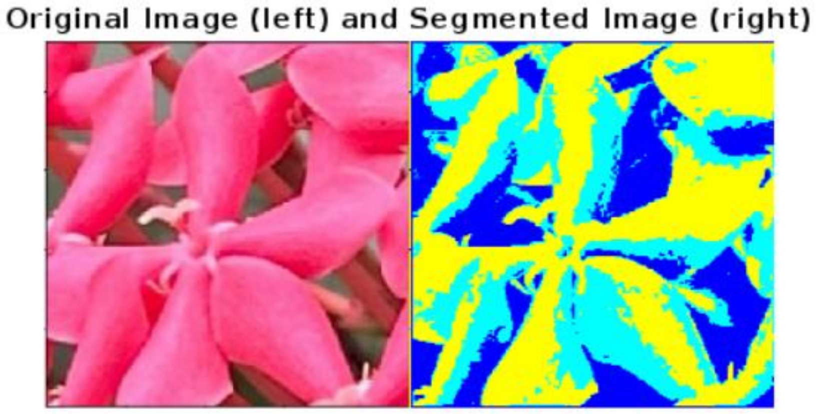


Figure 5: Using Multi-level Thresholding (MATLAB)

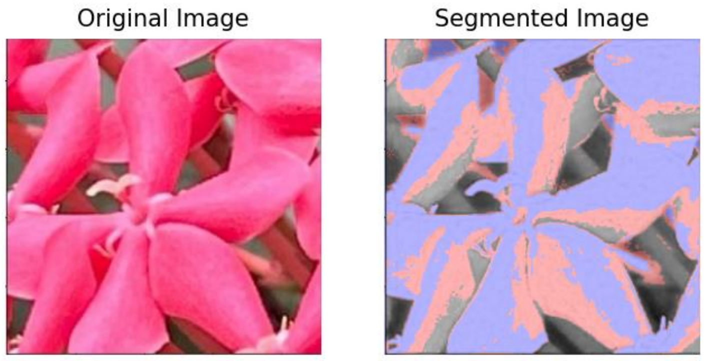


Figure 5.1: Using Multi-level Thresholding (PYTHON)

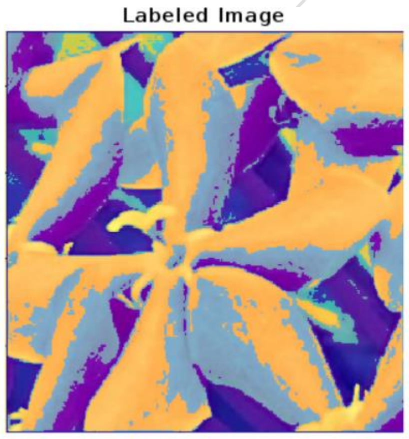


Figure 6: Using K-means (MATLAB)

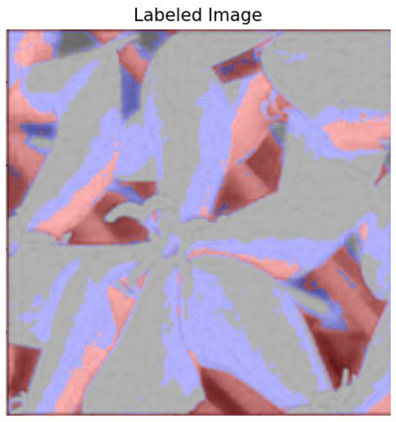


Figure 6.1: Using K-means (PYTHON)

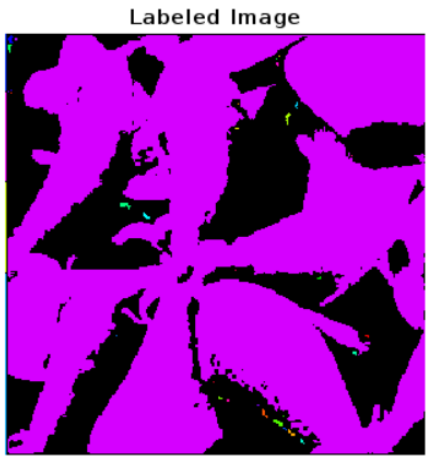


Figure 7: Using Connected Component Labelling (MATLAB)

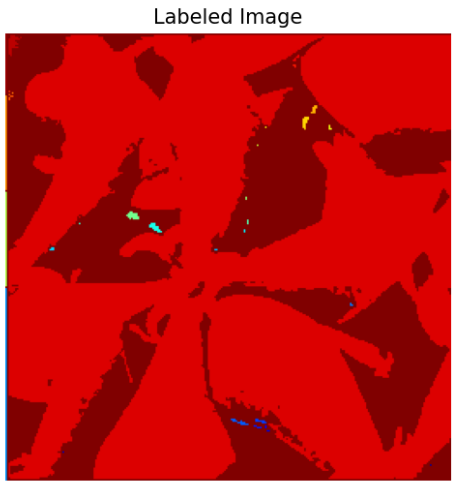


Figure 7.1: Using Connected Component Labelling (PYTHON)

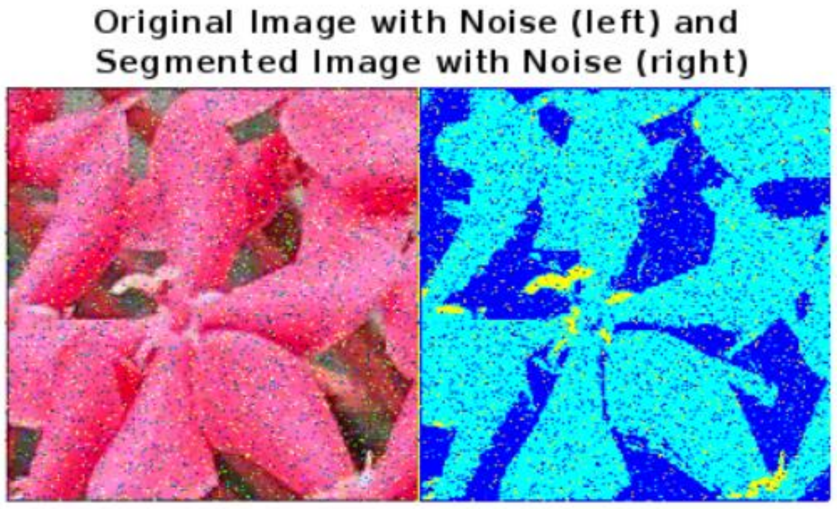

Original Image with Noise (left) and
Segmented Image with Noise (right)

Figure 8: Adding Salt and Pepper Noise (MATLAB)

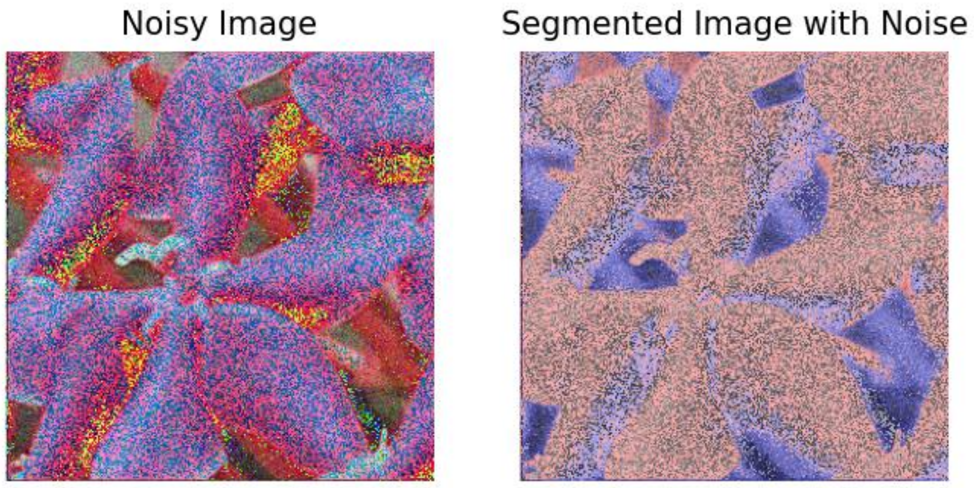


Figure 8.1: Adding Salt and Pepper Noise (PYTHON)

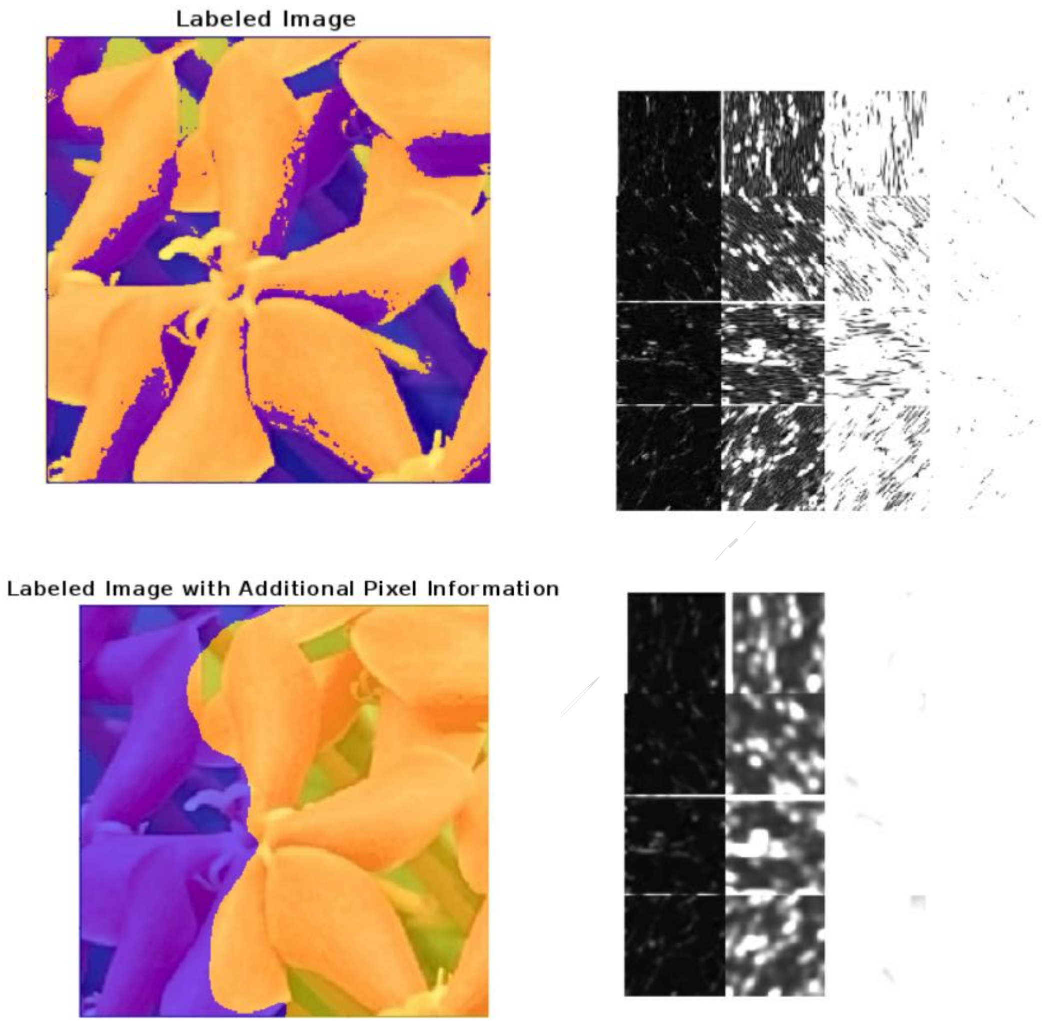


Figure 9: Improve K-means Segmentation Using Texture and Spatial (MATLAB)

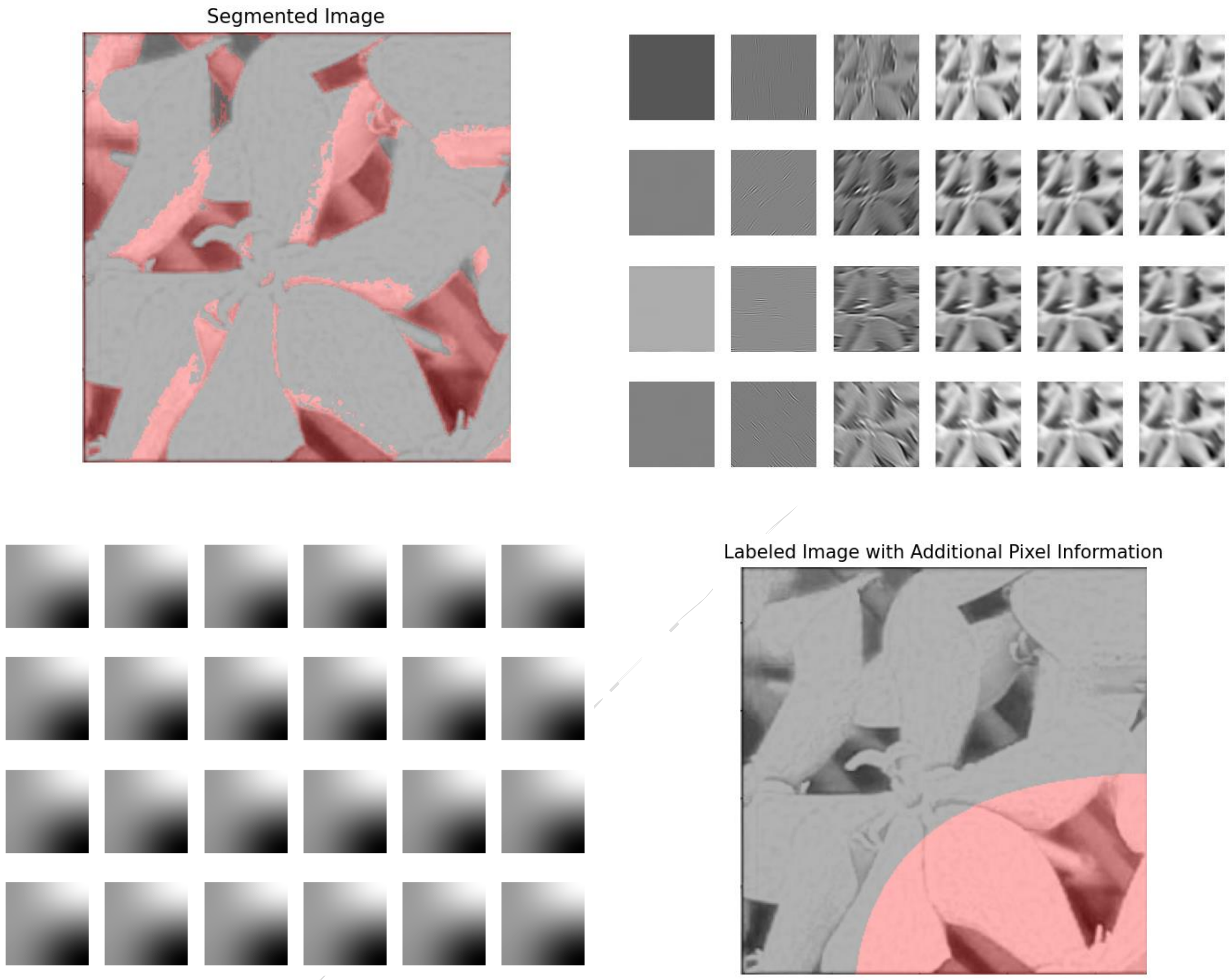


Figure 9.1: Improve K-means Segmentation Using Texture and Spatial (PYTHON)

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

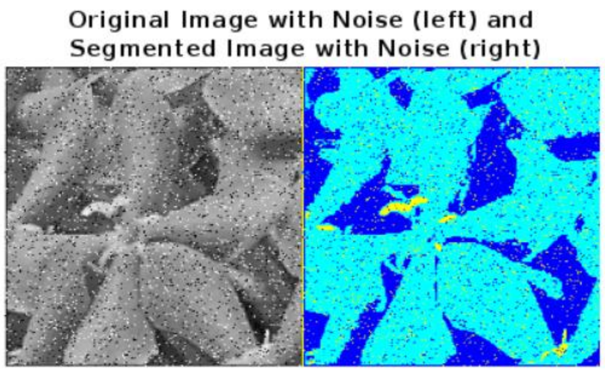
Connected-component labeling is a technique used in image processing to identify and label connected regions (components) in a binary image. It works by scanning the image pixel to detect connected groups of foreground pixels (usually represented by 1's) that are adjacent to each other in 4-connectivity (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.

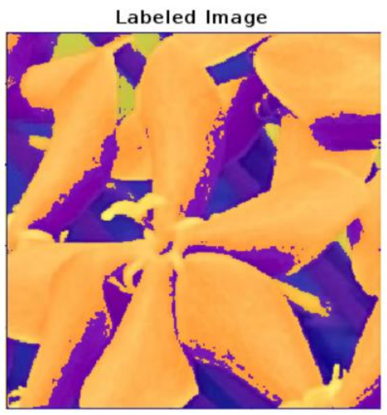
Parameter Modification

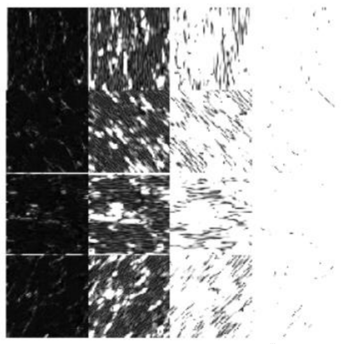
```
% % Parameter Modifications

% 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;
```

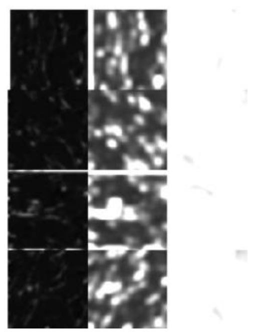
```
orientation = 0:45:135;
g = gabor(wavelength,orientation);

% 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");
```


Original Image with Noise (left) and
Segmented Image with Noise (right)







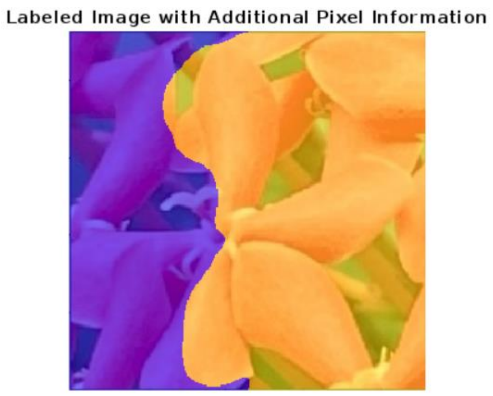


Figure 9: Parameters Modification (MATLAB)

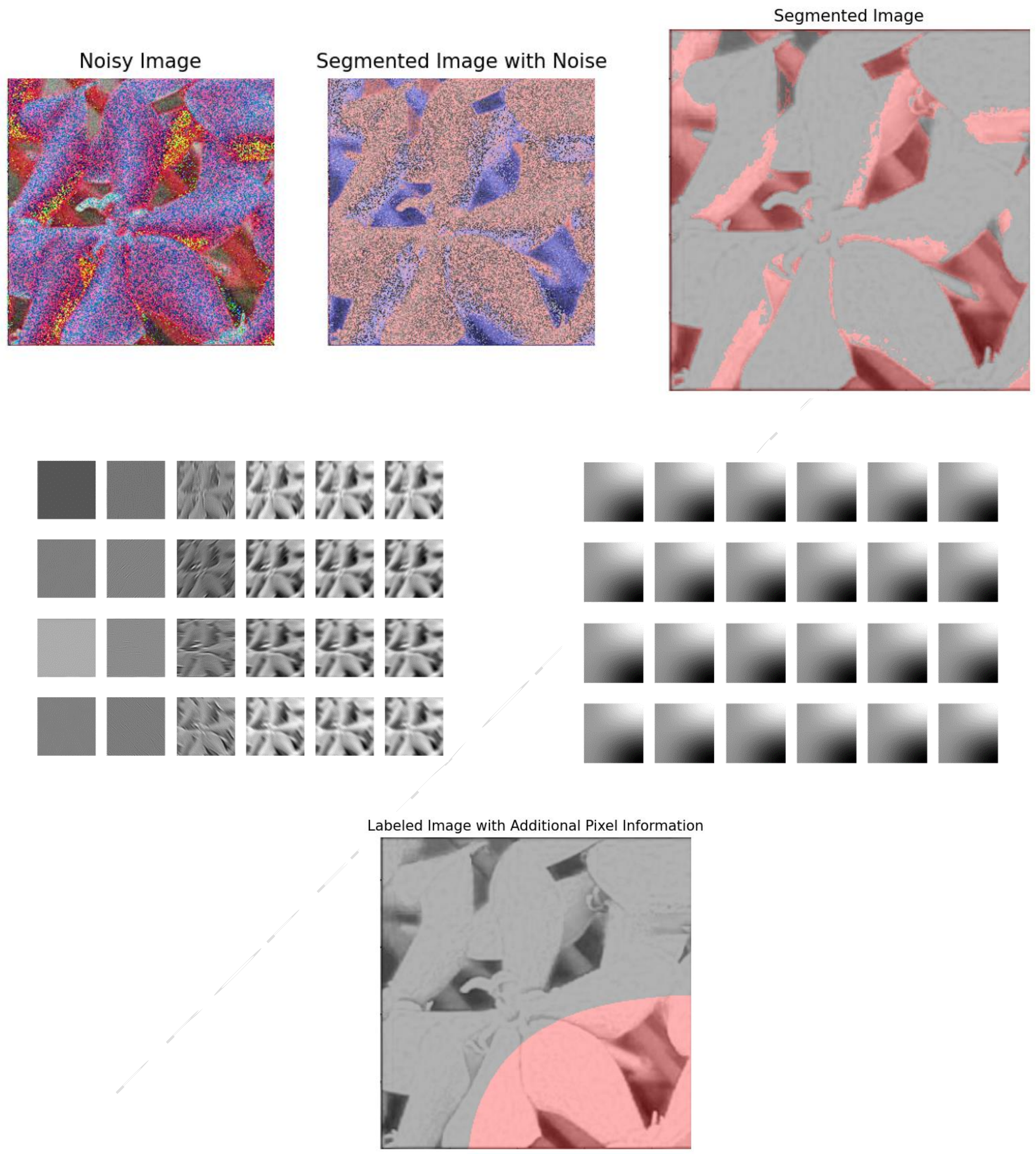

Noisy Image
Segmented Image with Noise
Segmented Image
Labeled Image with Additional Pixel Information

Figure 9.1: Parameters Modification (PYTHON)

2. Visualize the results, analyze and interpret:

The analysis covers image processing techniques, specifically focusing on thresholding and region-based segmentation. Firstly, Global Thresholding with Otsu's Method separates an image into two classes by minimizing intra-class variance, resulting in clear boundaries, such as those around coin edges. Building on this, Multi-level Thresholding refines the approach by segmenting the image into multiple classes, which captures more detailed features. In a similar vein, Global Histogram Thresholding also converts a grayscale image to binary by optimizing thresholding through histogram analysis, achieving results comparable to multi-level thresholding. Additionally, K-means Clustering further segments the image into distinct regions based on pixel intensity, while Connected-Component Labeling isolates and labels connected pixel groups in binary images.

IV. Conclusion

In conclusion, after successfully completing this laboratory exercise, we applied various image processing techniques, including image acquisition and conversion from RGB color to grayscale, as in previous laboratory exercises. However, this exercise specifically focused on image segmentation and thresholding techniques. It included additional processing techniques such as displaying Global Thresholding and Multi-level Thresholding using Otsu's Method in Binary, displaying histograms, performing Region-Based Segmentation using k-means, applying Connected-Component Labeling, adding noise, and using Gabor filters. Furthermore, this activity utilized the same tools as previous exercises, namely MATLAB and Python, with both displaying the images appropriately, with slight differences in output, such as image clarity and color accuracy.

References

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

<This is in a separate page>