PAMANTASAN NG LUNGSOD NG MAYNILA

(University of the City of Manila)

Intramuros, Manila

Elective 3

Laboratory Activity No. 1

Image Acquisition and Manipulation

Score

Submitted by:

Alambra, Joseph Nathaniel
Aragon, Patrick Laurence
Banal, Daryll
Sentasas, David Bryan
Tutanes, Allen Christopher

Saturday (7:00 – 4:00 pm) / CPE 0332.1-1

Date Submitted

24-07-2024

Submitted to:

Engr. Maria Rizette H. Sayo

I. Objectives

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

- Acquire the image.
- Rotate the image by 30 degrees.
- Flip the image horizontally.

II. Methods

A. Perform a task given in the presentation

- Copy and paste your MATLAB code

```
% Read the image
img = imread('C:/Users/Allen/Downloads/orange.png');
% Rotate by 30 degrees
rotated_img = imrotate(img, 30);

% Flip horizontally
flipped_img = fliplr(rotated_img);

% Display results
figure(1);
plot(1,1);
imshow(img);
title('Original Image');
figure(2);
plot(1,1);
imshow(rotated_img);
title('Rotated 30°'); figure(3); plot(1,1);
imshow(flipped_img); title('Rotated & Flipped');
```

B. Supplementary Activity

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

Source Code:

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

# Read the image
img = cv2.imread("C:/Users/Elitebook 840 G7/Documents/3rd Year - 3rd
Sem/Elective (Laboratory)/orange.png")

# Rotate by 30 degrees
(h, w) = img.shape[:2]
center = (w // 2, h // 2)
M = cv2.getRotationMatrix2D(center, 30, 1.0)
rotated_img = cv2.warpAffine(img, M, (w, h))

# Flip horizontally
flipped_img = cv2.flip(rotated_img, 1)

# Convert BGR to RGB for displaying using matplotlib
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
rotated_img_rgb = cv2.cvtColor(rotated_img, cv2.COLOR_BGR2RGB)
flipped_img_rgb = cv2.cvtColor(flipped_img, cv2.COLOR_BGR2RGB)

# Display results
plt.figure(1)
plt.imshow(img_rgb)
plt.title('Original Image')

plt.figure(2)
plt.imshow(rotated_img_rgb)
plt.title('Rotated 30°')

plt.figure(3)
plt.imshow(flipped_img_rgb)
plt.title('Rotated & Flipped')

plt.show()
```

C. Results

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

TASK A

picture file: orange.png

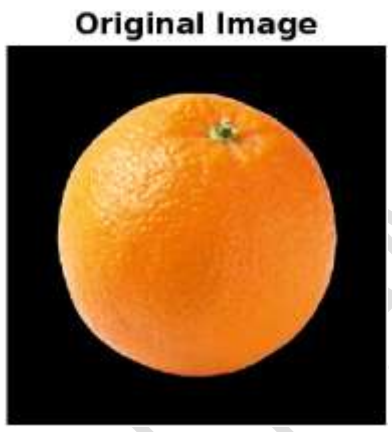

Figure 1: Orange

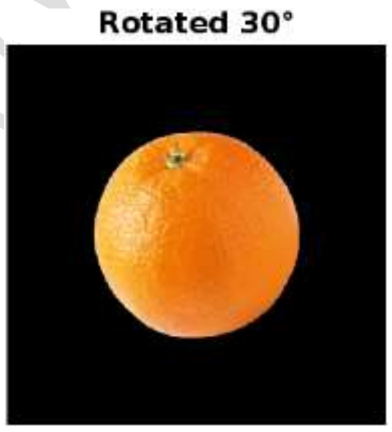

Figure 2: Orange Rotated by 30 Degrees

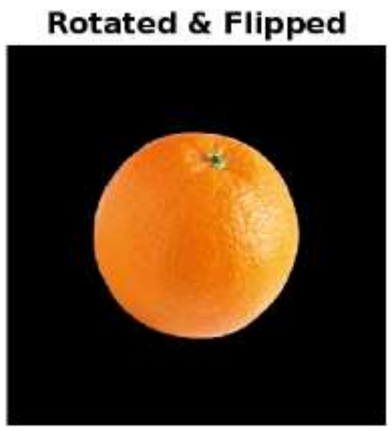

Figure 3: Orange Flip Horizontally

2. Copy/crop and paste your results. Label each output (Figure1, Figure2, Figure3)

TASK B

picture file: orange.png

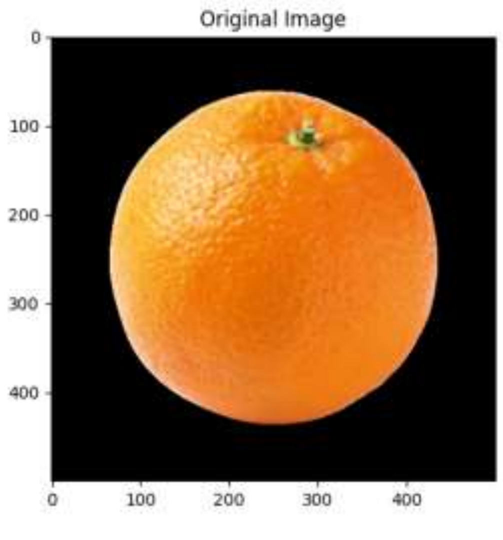

Figure 4: Orange

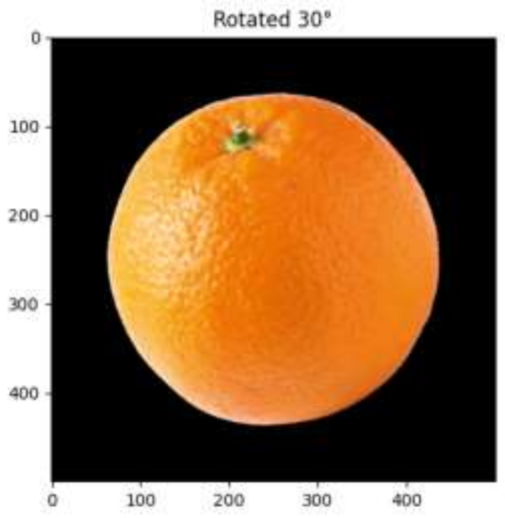

Figure 5: Orange Rotated by 30 Degrees

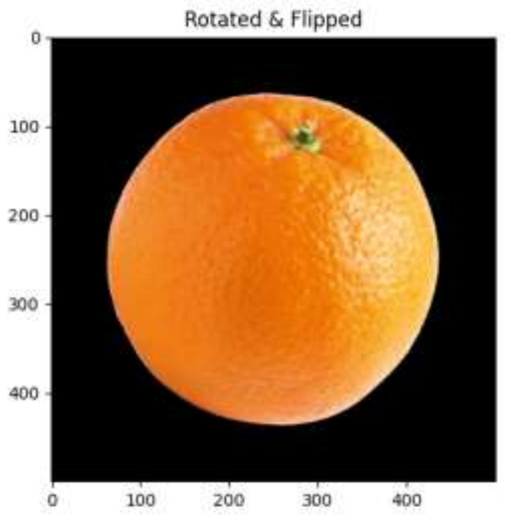

Figure 6: Orange Flip Horizontally

2. Visualize the results, analyze and interpret:

< Discuss the effects of the applied algorithm on the image and its effectiveness in achieving the desired outcome. Handwritten>

TASK A

In TASK A, the image is manipulated using Online MATLAB. The first step involves rotating the image by 30 degrees using the ***imrotate*** function. This function is particularly user-friendly, as it only requires specifying the image and the rotation angle. The 30-degree rotation tilts the image to the right, altering its orientation significantly. Following this, the image is flipped horizontally using the ***fliplr*** function, which mirrors the image along the vertical axis. This sequence of transformations is both intuitive and straightforward, leveraging MATLAB's built-in functions for basic image manipulation. The original image, the rotated image, and the rotated & flipped image are displayed sequentially using MATLAB's ***imshow*** function, which provides immediate visual feedback. The simplicity of this approach lies in its ease of implementation, making it an efficient method for quick and effective image transformations.

TASK B

In TASK B, the image manipulation is performed using PyCharm with the OpenCV library. The process begins with reading the image using ***cv2.imread***, which loads the image into the program. To rotate the image by 30 degrees, the team used ***cv2.getRotationMatrix2D*** to create a rotation matrix, specifying the center of the image, the rotation angle, and the scale. This matrix is then applied to the image using ***cv2.warpAffine***, which performs the affine transformation based on the rotation matrix. This method offers precise control over the rotation center, allowing for more nuanced adjustments. After rotating the image, the team achieved the horizontal flip using ***cv2.flip***, which mirrors the image along the vertical axis, similar to MATLAB's ***fliplr*** function. Before displaying the images, they were converted from BGR to RGB format using ***cv2.cvtColor*** to ensure correct color representation in Matplotlib, as OpenCV uses BGR by default. The original, rotated, and rotated & flipped images are displayed using Matplotlib's ***imshow*** function, which provides advanced visualization capabilities. This approach, while slightly more complex, offers extensive control and flexibility in image processing, making it suitable for more detailed and customizable manipulations.

Effectiveness Comparison

In TASK A, The approach using Online MATLAB is straightforward and easy to implement. The functions ***imrotate*** and ***fliplr*** are intuitive, requiring minimal parameters to execute the transformations. This makes MATLAB a great choice for quick and efficient image manipulations, particularly when the task does not demand extensive customization or control over the transformation parameters. The immediate visual feedback provided by MATLAB's ***imshow*** function further enhances the ease of use, making it accessible for users who require rapid results without delving into complex configurations. While in TASK B, the approach using PyCharm and OpenCV is more detailed and offers greater precision. The use of ***cv2.getRotationMatrix2D*** and ***cv2.warpAffine*** allows for exact control over the rotation center and angle, providing a higher degree of customization. Additionally, converting images from BGR to RGB ensures accurate color representation in Matplotlib, enhancing the visualization quality. This method is particularly beneficial for advanced image processing tasks that require fine-tuned adjustments and comprehensive manipulation capabilities. The versatility of OpenCV, combined with the advanced plotting features of Matplotlib, makes this approach suitable for users who need robust and flexible image processing solutions.

To sum it up, the applied algorithms in both tasks are effective in achieving the desired image transformations, with TASK B providing additional advantages in terms of control and visualization.

IV. Conclusion

In this laboratory activity, the group successfully acquired and manipulated images using both Online MATLAB and OpenCV in Python with PyCharm. The initial step involved reading the image file, which was seamlessly executed using the ***imread*** function in MATLAB and cv2.imread in PyCharm. This foundational step enabled subsequent image processing tasks, demonstrating the efficiency and ease of image acquisition in both environments.

The group's next objective was to rotate the image by 30 degrees. In MATLAB, this was achieved using the ***imrotate*** function, which allowed for a simple and efficient way to specify the rotation angle and apply the transformation. In PyCharm, the group utilized ***cv2.getRotationMatrix2D*** to create a rotation matrix, and ***cv2.warpAffine*** to apply this matrix to the image. Both methods effectively altered the image's orientation, providing clear visual confirmation of the rotation. The precise control over the rotation center in PyCharm added an extra layer of flexibility, highlighting its advanced capabilities in image processing.

Finally, the group flipped the images horizontally to complete the transformation. MATLAB's fliplr function and PyCharm's ***cv2.flip function*** were used to mirror the images along the vertical axis. Both approaches accomplished the task efficiently, with the results displayed using MATLAB's ***imshow*** and Matplotlib's ***imshow*** functions respectively. The group met this objective with equal effectiveness in both environments, showcasing the robustness of the applied algorithms in achieving the desired image transformations.

Throughout the activity, the group observed that both MATLAB and OpenCV in Python provided robust and versatile tools for image processing. MATLAB offered a more straightforward and user-friendly approach for basic image manipulation tasks, while OpenCV in PyCharm provided more granular control and flexibility, especially for more advanced transformations.

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>