Image Processing Homework #1

Implementations of Power-Law (Gamma) Transformation, Histogram Equalization, and Laplacian Sharpening

Advisor : Jin-Jang Leou (柳金章)
Student : Chih-Chia Lo (羅治嘉)
Registration Number : 610410002
Date Due : 2021/11/07 23:59
Date Handed In : 2021/10/29 09:00

I. Technical Description

A. Power-Law (Gamma) Transformation

A variety of devices used for image capture, printing, and display respond according to a power law, where the exponent in the power-law equation is referred to as gamma. As the illustrated example shown in Fig. 1., the process used to correct this power-law response phenomena is called gamma correction. [1]

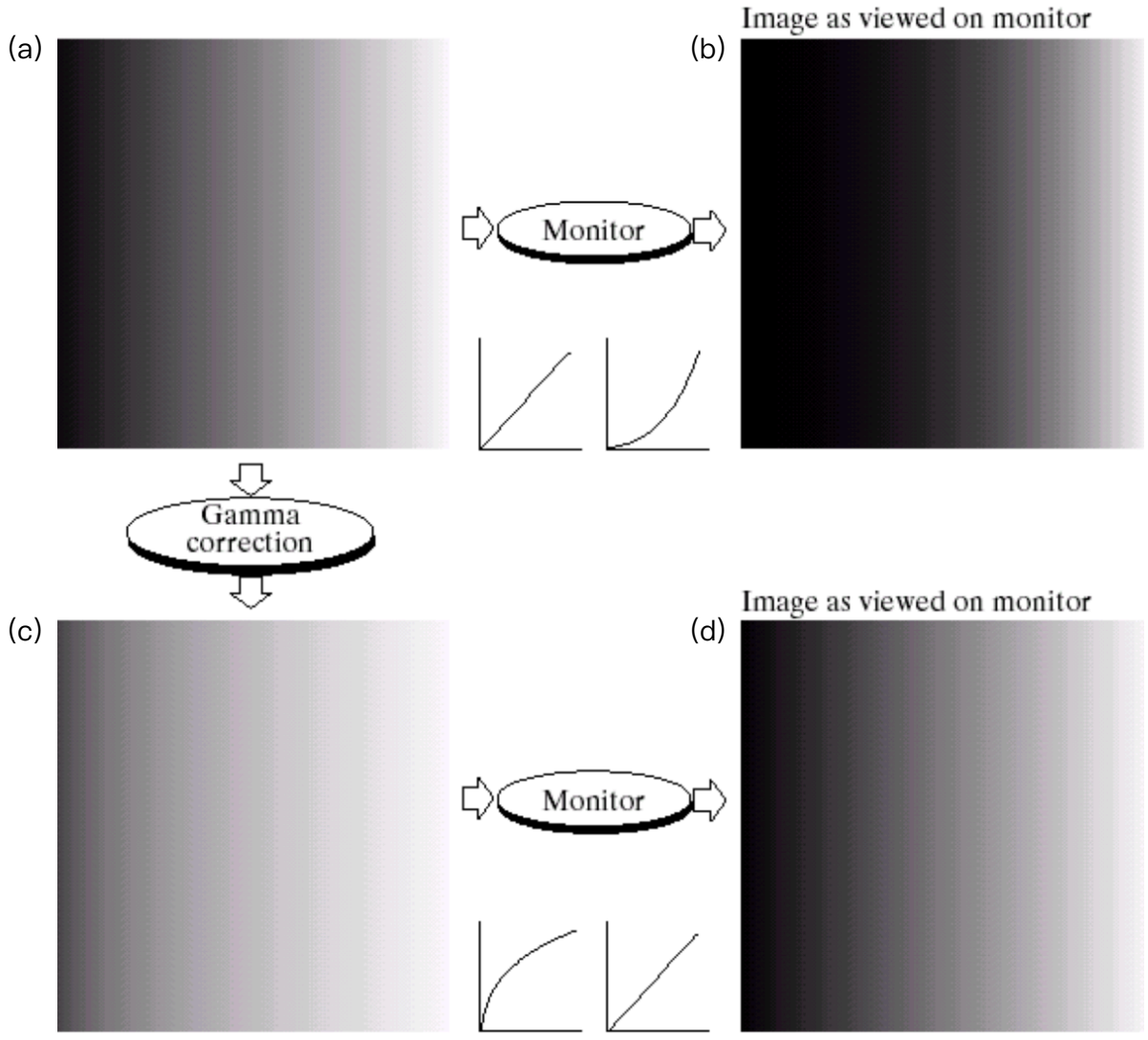

Fig. 1. (a) Linear–wedge gray–scale image. (b) Response of monitor to linear wedge. (c) Gamma–corrected wedge (d) Output of monitor.

The first part of the implementations is Power-Law (Gamma) Transformation, which is commonly used for alternating illumination of an image. The following formula is how Power-Law (Gamma) Transformation functions:

$$g(x, y) = c \cdot f(x, y)^{\gamma} \tag{1}$$

where c (is usually set to 1) and γ denote positive constants, and $g(x, y)$ and $f(x, y)$ denote the pixels of output image and input image respectively. The following Fig. 2. is the source code of Power-Law (Gamma) Transformation:

```
#Power-Law (Gamma) Transformation
def gamma_correction(input_img, const, gamma):
    if (const > 0) & (gamma > 0.0):
        output_img = const * ((input_img / 255) ** gamma) * 255 #Divided 255 and multiplies gamma
        return np.array(output_img).astype(int) #Convert all float value in the array into integer
    else:
        return input_img
```

Fig. 2. Implementations of Gamma Correction in Python.

B. Histogram Equalization

This approach usually increases the global contrast of many images, especially when the image is represented by a narrow range of intensity values. Through this adjustment, the intensities can be better distributed on the histogram utilizing the full range of intensities evenly. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the highly populated intensity values which is used to degrade image contrast. [2]

The first step of Histogram Equalization is to count the numbers of each pixel value and all pixels, so that we can obtain the probability of occurrence of gray level in an image can be approximated by:

$$p_x(i) = p(x = i) = \frac{n_i}{n}, \quad 0 \leq i < L \tag{2}$$

where n_i denotes the count of each pixel value, n denotes the count of all pixels, and L denotes the total pixels in an image. After that, calculate the cumulative density function (cdf) of each pixel:

$$cdf_x(i) = \sum_{j=0}^{i} p_x(j), \tag{3}$$

where $p_x(j)$ is from equation (2).
The following Fig. 3. is the source code of how Histogram Equalization is implemented:

```python
def myHistEq(input_img):
    hist, bins = np.histogram(input_img.ravel(), 256, [0,255])
    pdf = hist / input_img.size
    cdf = pdf.cumsum()
    return np.around(cdf * 255).astype(np.uint8)[input_img]
```

Fig. 3. Implementations of Histogram Equalization in Python.

C. *Laplacian Sharpening*

The response of a 2-D isotropic filter (or a rotation-invariant filter) is independent of the direction of the discontinuities in the image to which the filter is applied. To achieve Laplacian Sharpening, we implement 3 x 3 convolutions on the input images. The first, second, and third derivative masks of Laplacian Sharpening are as follow:

(a)

0	−1	0
−1	5	−1
0	−1	0

(b)

−1	−1	−1
−1	9	−1
−1	−1	−1

(c)

1	−2	1
−2	5	−2
1	−2	1

Fig. 4. (a) First derivative mask of Laplacian Sharpening. (b) Second derivative mask of Laplacian Sharpening. (c) Third derivative mask of Laplacian Sharpening.

And Fig. 5. is the source code of the Laplacian Sharpening, using 3 x 3 Laplacian masks to multiply every pair of 3 x 3 input image's pixels, and then add up these 9 values to obtain the center image pixel. Noting that the edge pixels are left unchanged due to the lack of neighboring pixels.

```python
def myLaplacian(input_img, kernel):
    Laplacian_operator = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
    rows = len(input_img)
    cols = len(input_img[0])
    result = np.arange(rows*cols*3).reshape(rows,cols,3)
    input_img = np.transpose(input_img)
    for i in range(0, rows):
        for j in range(0, cols):
            for k in range(0, 3):
                if (i < rows - 1) & (i > 0) & (j < rows - 1) & (j > 0):
                    result[i][j][k] = sum(sum(kernel*(np.transpose([index[i-1:i+2] for index in input_img[k][j-1:j+2]]))))
                if result[i][j][k] > 255:
                    result[i][j][k] = 255
                elif result[i][j][k] < 0:
                    result[i][j][k] = 0
    return result
```

Fig. 5. Implementations of Laplacian Sharpening in Python.

D. Main Processes of Programming Implementations

First of all, take input images that are in HW1_test_images folder and assign these three input images to each variables (Fig. 6.). Secondly, establish three plots for different implementations and input variables into our custom functions (Fig.7., Fig.8., and Fig.9 are Power-Law Transformation, Histogram Equalization, and Laplacian Sharpening respectively).

```
48    Jetplane_file = './HW1_test_image/Jetplane.bmp'
49    Lake_file = './HW1_test_image/Lake.bmp'
50    Peppers_file = './HW1_test_image/Peppers.bmp'
51    Jetplane_img = cv2.imread(Jetplane_file) #Input image value [0, 255]
52    Lake_img = cv2.imread(Lake_file)
53    Peppers_img = cv2.imread(Peppers_file)
```

Fig. 6. Input images and assign to variables.

```
60    fig1.add_subplot(rows, cols, 1)
61    plt.imshow(Jetplane_img)
62    plt.title('Original Jetplane')
63    fig1.add_subplot(rows, cols, 2)
64    Jetplane_output = gamma_correction(Jetplane_img, 1, 0.4)
65    plt.imshow(Jetplane_output)
66    plt.title('Power-Law Jetplane' + ' (γ = 0.4)')
67    fig1.add_subplot(rows, cols, 3)
68    Jetplane_output = gamma_correction(Jetplane_img, 1, 1.2)
69    plt.imshow(Jetplane_output)
70    plt.title('Power-Law Cameraman' + ' (γ = 1.2)')
71    fig1.add_subplot(rows, cols, 4)
72    Jetplane_output = gamma_correction(Jetplane_img, 1, 2.5)
73    plt.imshow(Jetplane_output)
74    plt.title('Power-Law Jetplane' + ' (γ = 2.5)')
75    fig1.add_subplot(rows, cols, 5)
76    plt.imshow(Lake_img)
77    plt.title('Original Lake')
78    fig1.add_subplot(rows, cols, 6)
79    Lake_output = gamma_correction(Lake_img, 1, 0.4)
80    plt.imshow(Lake_output)
81    plt.title('Power-Law Lake' + ' (γ = 0.4)')
82    fig1.add_subplot(rows, cols, 7)
83    Lake_output = gamma_correction(Lake_img, 1, 1.2)
84    plt.imshow(Lake_output)
85    plt.title('Power-Law Lake' + ' (γ = 1.2)')
86    fig1.add_subplot(rows, cols, 8)
87    Lake_output = gamma_correction(Lake_img, 1, 2.5)
88    plt.imshow(Lake_output)
89    plt.title('Power-Law Lake' + ' (γ = 2.5)')
90    fig1.add_subplot(rows, cols, 9)
91    plt.imshow(Peppers_img)
92    plt.title('Original Peppers')
93    fig1.add_subplot(rows, cols, 10)
94    Peppers_output = gamma_correction(Peppers_img, 1, 0.4)
95    plt.imshow(Peppers_output)
96    plt.title('Power-Law Peppers' + ' (γ = 0.4)')
97    fig1.add_subplot(rows, cols, 11)
98    Peppers_output = gamma_correction(Peppers_img, 1, 1.2)
99    plt.imshow(Peppers_output)
100   plt.title('Power-Law Peppers' + ' (γ = 1.2)')
101   fig1.add_subplot(rows, cols, 12)
102   Peppers_output = gamma_correction(Peppers_img, 1, 2.5)
103   plt.imshow(Peppers_output)
104   plt.title('Power-Law Peppers' + ' (γ = 2.5)')
105   plt.tight_layout()
```

Fig. 7. Input variables and display Power-Law Transformation images.

```
107   #Histogram Equalization
108   fig2 = plt.figure(figsize = (18, 9))
109   fig2.canvas.manager.set_window_title('Histogram Equalization')
110   rows = 3
111   cols = 4
112   fig2.add_subplot(rows, cols, 1)
113   plt.imshow(Jetplane_img)
114   plt.title('Original Jetplane')
115   fig2.add_subplot(rows, cols, 2)
116   plt.hist(Jetplane_img.ravel(), 256, [0, 255])
117   plt.title('Histogram of Original Jetplane')
118   fig2.add_subplot(rows, cols, 3)
119   Jetplane_output = myHistEq(Jetplane_img)
120   plt.imshow(Jetplane_output)
121   plt.title('Histogram Equalization Jetplane')
122   fig2.add_subplot(rows, cols, 4)
123   plt.hist(Jetplane_output.ravel(), 256, [0, 255])
124   plt.title('Histogram of Histogram Equalization Jetplane')
125   fig2.add_subplot(rows, cols, 5)
126   plt.imshow(Lake_img)
127   plt.title('Original Lake')
128   fig2.add_subplot(rows, cols, 6)
129   plt.hist(Lake_img.ravel(), 256, [0, 255])
130   plt.ylim(0, 8000)   #Make subplot easier to see
131   plt.title('Histogram of Original Lake')
132   fig2.add_subplot(rows, cols, 7)
133   Lena_output = myHistEq(Lake_img)
134   plt.imshow(Lake_output)
135   plt.title('Histogram Equalization Lake')
136   fig2.add_subplot(rows, cols, 8)
137   plt.hist(Lake_output.ravel(), 256, [0, 255])
138   plt.ylim(0, 8000)   #Make subplot easier to see
139   plt.title('Histogram of Histogram Equalization Lake')
140   fig2.add_subplot(rows, cols, 9)
141   plt.imshow(Peppers_img)
142   plt.title('Original Peppers')
143   fig2.add_subplot(rows, cols, 10)
144   plt.hist(Peppers_img.ravel(), 256, [0, 255])
145   plt.ylim(0, 12000)   #Make subplot easier to see
146   plt.title('Histogram of Original Peppers')
147   fig2.add_subplot(rows, cols, 11)
148   Peppers_output = myHistEq(Peppers_img)
149   plt.imshow(Peppers_output)
150   plt.title('Histogram Equalization Peppers')
151   fig2.add_subplot(rows, cols, 12)
152   plt.hist(Peppers_output.ravel(), 256, [0, 255])
153   plt.ylim(0, 12000)  #Make subplot easier to see
154   plt.title('Histogram of Histogram Equalization Peppers')
155   plt.tight_layout()
```

Fig. 8. Input variables and display Histogram Equalization images.

```
157     #Laplacian Sharpening
158     fig4 = plt.figure(figsize = (16, 9))
159     fig4.canvas.manager.set_window_title('Laplacian Image Sharpening')
160     rows = 3
161     cols = 4
162     fig4.add_subplot(rows, cols, 1)
163     plt.imshow(Jetplane_img)
164     plt.title('Original Jetplane')
165     fig4.add_subplot(rows, cols, 2)
166     Jetplane_output = myLaplacian(Jetplane_img, KernelBank["Laplacian_first"])
167     plt.imshow(Jetplane_output)
168     plt.title('Laplacian first-order derivative Jetplane')
169     fig4.add_subplot(rows, cols, 3)
170     Jetplane_output = myLaplacian(Jetplane_img, KernelBank["Laplacian_second"])
171     plt.imshow(Jetplane_output)
172     plt.title('Laplacian second-order derivative Jetplane')
173     fig4.add_subplot(rows, cols, 4)
174     Jetplane_output = myLaplacian(Jetplane_img, KernelBank["Laplacian_third"])
175     plt.imshow(Jetplane_output)
176     plt.title('Laplacian third-order derivative Jetplane')
177     fig4.add_subplot(rows, cols, 5)
178     plt.imshow(Lake_img)
179     plt.title('Original Lake')
180     fig4.add_subplot(rows, cols, 6)
181     Lake_output = myLaplacian(Lake_img, KernelBank["Laplacian_first"])
182     plt.imshow(Lake_output)
183     plt.title('Laplacian first-order derivative Lake')
184     fig4.add_subplot(rows, cols, 7)
185     Lake_output = myLaplacian(Lake_img, KernelBank["Laplacian_second"])
186     plt.imshow(Lake_output)
187     plt.title('Laplacian second-order derivative Lake')
188     fig4.add_subplot(rows, cols, 8)
189     Lake_output = myLaplacian(Lake_img, KernelBank["Laplacian_third"])
190     plt.imshow(Lake_output)
191     plt.title('Laplacian third-order derivative Lake')
192     fig4.add_subplot(rows, cols, 9)
193     plt.imshow(Peppers_img)
194     plt.title('Original Peppers')
195     fig4.add_subplot(rows, cols, 10)
196     Peppers_output = myLaplacian(Peppers_img, KernelBank["Laplacian_first"])
197     plt.imshow(Peppers_output)
198     plt.title('Laplacian first-order derivative  Peppers')
199     fig4.add_subplot(rows, cols, 11)
200     Peppers_output = myLaplacian(Peppers_img, KernelBank["Laplacian_second"])
201     plt.imshow(Peppers_output)
202     plt.title('Laplacian second-order derivative Peppers')
203     fig4.add_subplot(rows, cols, 12)
204     Peppers_output = myLaplacian(Peppers_img, KernelBank["Laplacian_third"])
205     plt.imshow(Peppers_output)
206     plt.title('Laplacian third-order derivative Peppers')
207     plt.tight_layout()
208     plt.show()
```

Fig. 9. Input variables and display Laplacian Sharpening images.

Last but not least, after obtaining our desired images, we display them on three different plots to compare one another easier. Custom functions are afore-mentioned in ***A***, ***B***, and **C** sections in Technical Description.

II. Experimental Results

A. Datasets and Environments

This project contains a dataset of HW1_test_image from ecourse2, which there are three grayscale images given in this dataset. The implementation programming language of this project is Python 3, and the interpreter version of Python is Python 3.9.6 64-bit.

B. Comparisons of Power-Law (Gamma) Transformation

The following figure is the Power-Law (Gamma) Transformation results with different gamma values (*c* is set to 1):

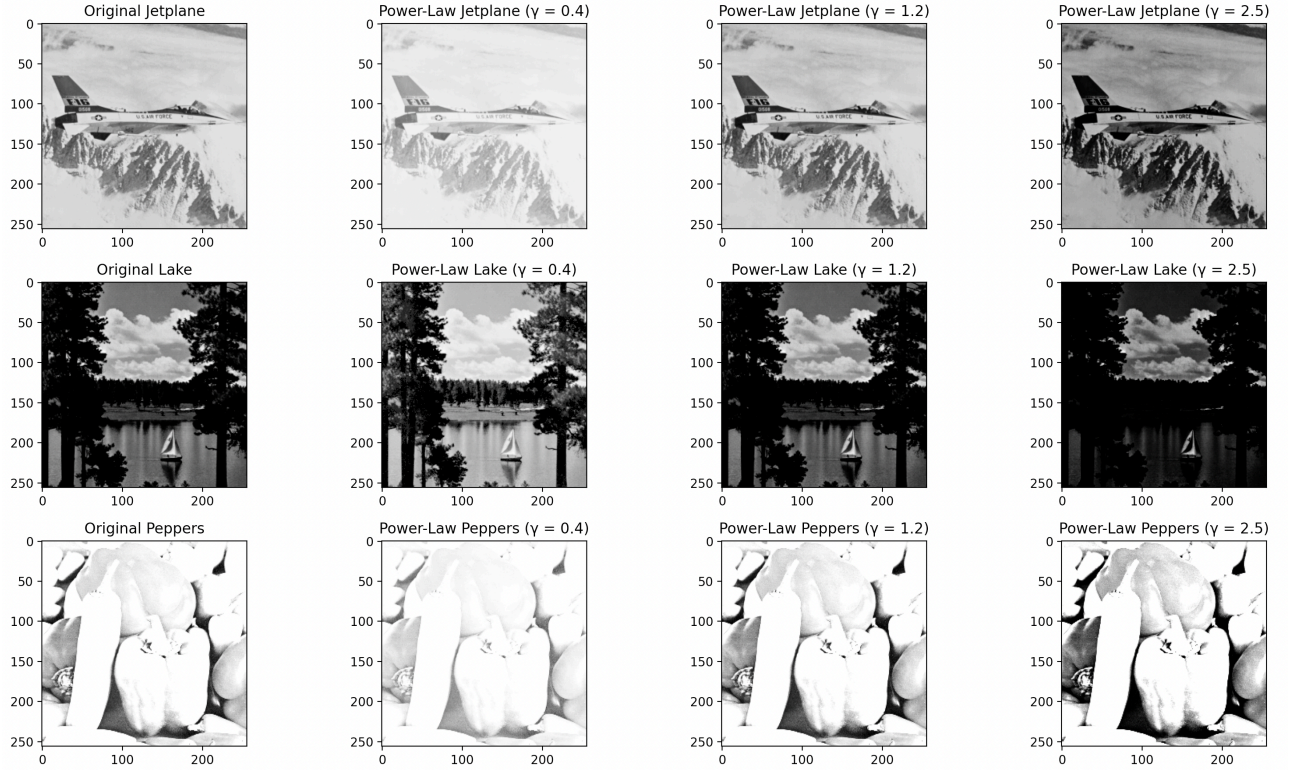

Fig. 10. Comparisons with different values of gamma correction

where the first column displays the three original images, the second column displays the images processed with gamma value set to 0.4, the third column displays the images processed with gamma value set to 1.2, and the last column displays the images processed with gamma value set to 2.5.

C. Comparisons of Histogram Equalization

The following figure is the Histogram Equalization results of histogram images and statistical histograms:

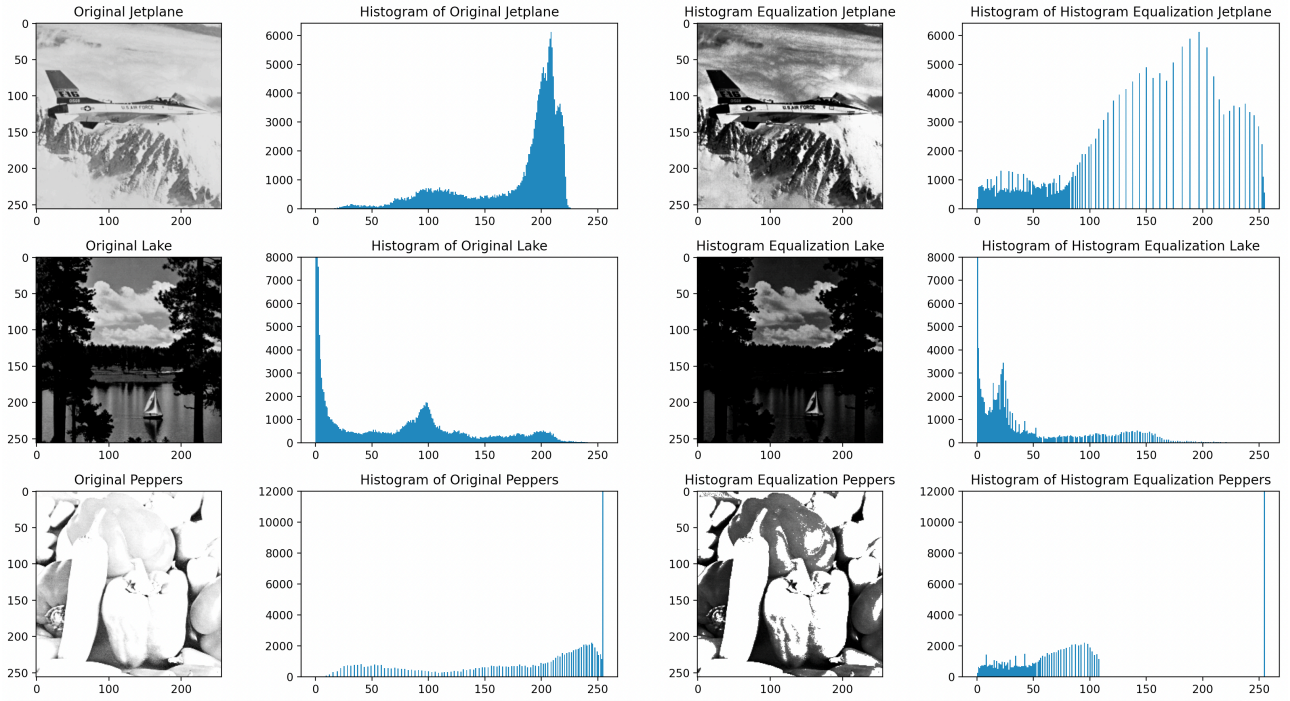

Fig. 11. Comparisons of Histogram Equalization and histograms of corresponding images

where the first column displays the three original images, the second column displays the histograms of the corresponding images, the third column displays the images after Histogram Equalization, and the last column displays the histograms of the post-processed images.

D. Comparisons of Laplacian Sharpening

The following figure is the Laplacian Sharpening results of different derivative, that is, three orders of derivative are used:

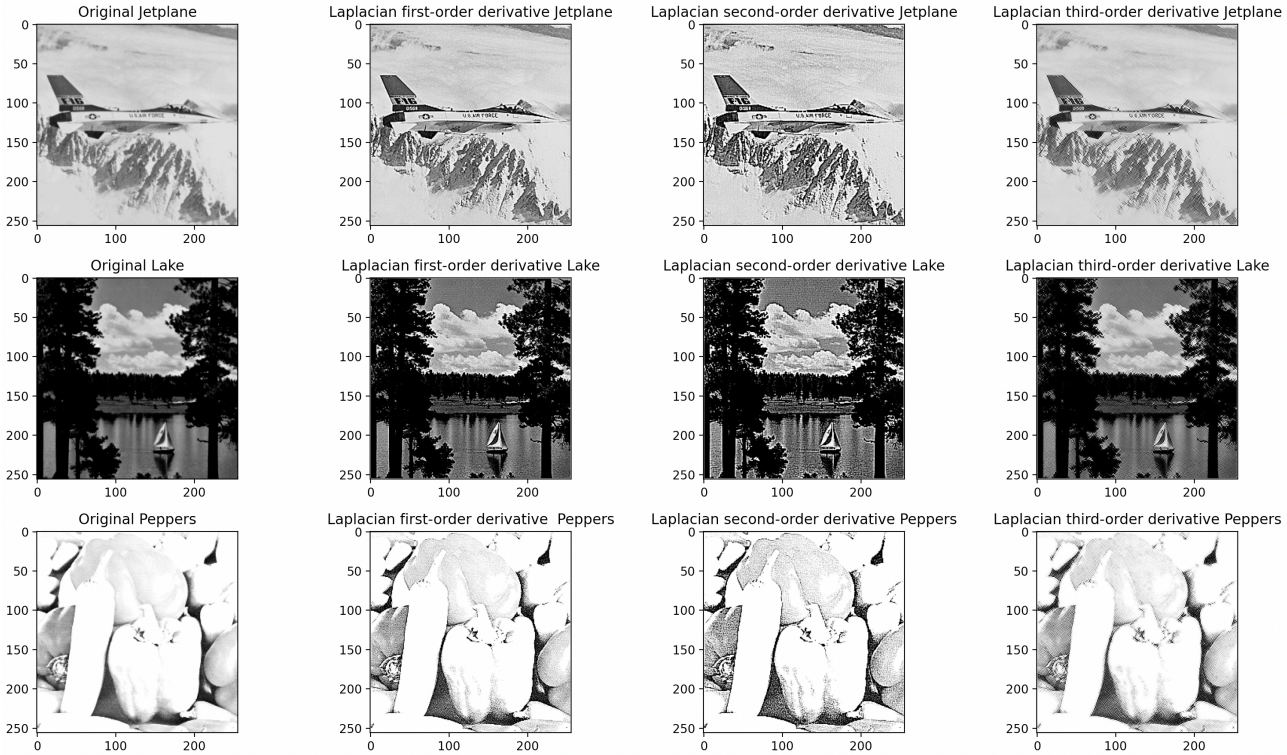

Fig. 12. Comparisons of Laplacian Sharpening

where the first column displays the three original images, the second column displays the usage of the first-order derivative Laplacian mask, the third column displays the usage of the second-order derivative of Laplacian mask, and the last column displays the usage of the third-order derivative of Laplacian mask.

III. Discussions

A. Power-Law (Gamma) Transformation

In Fig. 6., if γ is set to 1, then the output image is as same as the original image, if γ is larger than 1, it is darker than the original image, and vice versa. As far as we're concerned, Gamma Correction is used to adjust brightness of an image by controlling γ factor.

B. Histogram Equalization

In Fig. 7., it is self-evident that Histogram Equalization can increase image's global contrast. It functions successfully on the Jetplane image because pixels are scattered across different pixel values, not gathered in one specific pixel value, while the other two images do not perform well since its pixels are flocked into one specific pixel value, making the contrast worse than the original image. From our

perspectives, Histogram Equalization is only suitable for the image that its pixels do not flock immensely into certain specific pixel values.

C. Laplacian Sharpening

In Fig. 8., after processing Laplacian Sharpening, the first and the second derivative Laplacian Sharpening perform well, making the original images much more clear. However, the third derivative Laplacian Sharpening processes overly that makes the original images a bit vague.

IV. References and Appendix

[1] R. C. Gonzalez and R. E. Woods, “Digital Image Processing, 4th Ed., Pearson, New York, NY, 2018.” in Chapter 3 Image Enhancement Gamma Correction.
[2] Ruye Wang, “Histogram Equalization - Wikipedia” in https://en.wikipedia.org/wiki/Histogram_equalization.