ROBT 310 Image Processing

Homework Project 2

By Zarema Balgabekova

1. Introduction

The aim of this Homework Project is to apply the knowledge obtained during lectures on quantization, intensity transformations, and histogram processing in practice. This is done by implementing warmup and main tasks using MATLAB. The warmup part consists of two tasks. The first of them is to implement Nearest-Neighbor Interpolation, while the second is to compare histogram equalization and local histogram equalization. In the main part, a visual illusion is created, in which 2 colors of a grayscale image (8 colors of a color image) are perceived as more colors. We implement it using the Floyd-Steinberg dithering technique. Furthermore, Dispersed Dots (Bayer) dithering is chosen as an additional technique. The details on how this work has been performed are provided below.

2. Warmup Part

There are two tasks in this part.

2.1. Nearest-Neighbor Interpolation technique

Nearest-Neighbor Interpolation is the simplest interpolation technique, in which an enlarged image is created by assigning to new locations the intensity of their nearest neighbors in the image.

In this task, we implement this technique without using built-in functions from MATLAB image processing toolbox except image read, write, and show functions.

MATLAB commands:

```
function robt310_project2_interpolation(input_file_name,
output_file_name, scale_factor)

    input_img = imread(input_file_name);
    [row, col, colorChannel] = size(input_img);
    output_img = uint8(zeros(round(row*scale_factor),
round(col*scale_factor), colorChannel));
    output_size = size(output_img);

    for i_output = 1:output_size(1)
        for j_output = 1:output_size(2)
            i_input = ceil(i_output/scale_factor);
            j_input = ceil(j_output/scale_factor);
            output_img(i_output, j_output, :) = input_img(i_input, j_input,
:);
        end
    end
    imwrite(output_img, output_file_name);
    imshow(output_img)
    title('Output image (Nearest-Neighbor Interpolation)')
end
```

The function prototype has been provided in the manual. First, we read an input image using the function imread( ) and save the length of each dimension of this image separately with size( ). After that we pre-allocate the size of the output image with zeros( ), and uint8( ) is used because the command zeros( ) provides a double output. The length of rows and columns of the output image is equal to the length of input image multiplied by a scale

factor, and round( ) is used because the scale factor can be not only the whole number. Then in nested for-loop, each new location in the output image is assigned to the intensity of its nearest neighbor in the input image. This is done using the function ceil( ), which rounds floating-point number to the nearest integer greater than or equal to it. Img(I, j, :) notation is used to do this procedure for each color channel if the image is color. If the image is grayscale, the procedure is done for one color channel only. Finally, the output image is written into a file using the function imwrite( ).

Obtained output images and corresponding input images are presented below.

Figure 1: Grayscale input image with size 183×275

Figure 2: Grayscale output image with size 915×1375 (scale factor = 5)

Figure 3: Color input image with size 225×225×3

Figure 4: Color input image with size 675×675×3 (scale factor = 3)

From Figures 1-4, it is seen that Nearest-Neighbor Interpolation results in severe distortion of edges. In other words, the output image is pretty rough. Furthermore, smaller input image and larger scale factor provide rougher and more distorted output image.

2.2. Histogram Equalization and Local Histogram Equalization

Histogram equalization is used to enhance the contrast of an image by modifying the intensity distribution of its histogram.

In this task, we use the provided image and apply histogram equalization and local histogram equalization to it.

MATLAB commands:

```
function robt310_project2_histogram_equalize(input_file_name)
    input_img = imread(input_file_name);
    %histogram equilization
    hist_eq_img = histeq(input_img);
    figure
    imshow(hist_eq_img)
    title('Histogram Equalization')
    %local histogram equalization
    fun = @(block_struct) histeq(block_struct.data);
    loc_hist_eq_img = blockproc(input_img, [40 40], fun);
    figure
    imshow(loc_hist_eq_img)
    title('Local Histogram Equalization')
end
```

The function prototype has been given to us. First, we read the input image with the help of imread( ). Then we perform histogram equalization using the function histeq( ), whereas local histogram equalization is implemented with the function blockproc( ), for which block processing function is histeq( ) and the neighborhood is 40×40.

The original image and two obtained results are as follows.

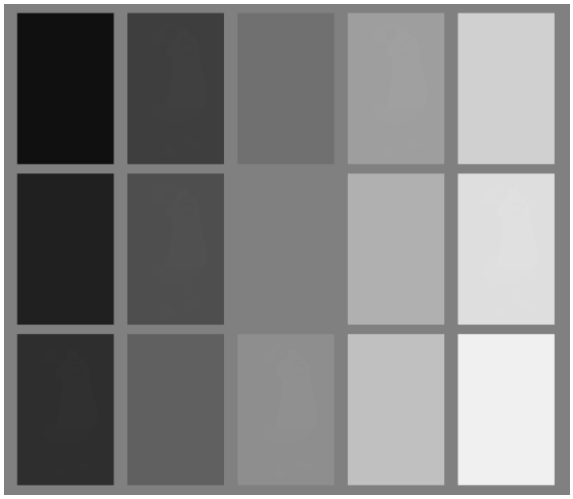

Figure 5: Original image

Figure 6: Histogram equalization

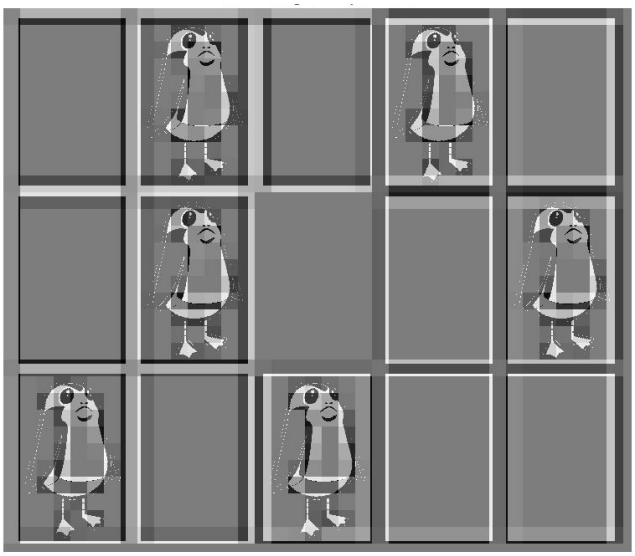

Figure 6: Local histogram equalization

It is seen from Figure 5 that birds are depicted in the original image. However, we can barely see them because of the low contrast. Applying histogram equalization, we enhance the contrast of the image as a whole. As a result, we see the birds more clearly than before, but the contrast is still not enough to see the details (birds) as clear as possible. Conversely, applying local histogram equalization, we increase the contrast of the image block by block. This allows us to see the birds distinctly. However, local histogram equalization results in slightly rough image, especially in the areas with birds. Therefore, both histogram equalization and local histogram equalization have their benefits and drawbacks. The best solution could be to use the combination of them.

3. Main Part

In the main task, Floyd-Steinberg dithering is implemented to obtain a visual illusion, in which human eyes see more colors than it is in reality. Dispersed dots/Bayer dithering is chosen as an additional technique. The function prototype has been provided in the manual. Floyd-Steinberg dithering is implemented if the parameter part of the function is equal to 0, and Bayer dithering is implemented if it equals 1. This is regulated with if-elseif statements.

MATLAB code is not provided in the report because it is too large. However, it can be seen in the corresponding MATLAB files, where comments are provided for clarification.

3.1. Floyd-Steinberg Dithering

Floyd-Steinberg dithering is a type of error-diffusion dithering, in which the fractions of the quantization error of a pixel are added to its neighboring pixels. The algorithm is performed from top to bottom, left to right so that already quantized pixels are not affected with the quantization error.

The pseudo code of this technique has been provided in the manual. So, we just implement it with some adjustments:

1. Image is converted into a double array not to have problems with the unsigned integers operations. At the end, it is again converted to uint8

2. `new_pixel = find_closest_palette_color(old_pixel)` is implemented as `new_pixel = round(old_pixel/255)*255`
   In other words, binarization is performed. For grayscale images, color palette consists of 2 colors (black, white). For color images, it consists of $2^3 = 8$ colors (black, white, red, green, blue, yellow, magenta, and cyan).
3. Image(i, j, :) notation is used as in warmup task 1.1
4. Special attention is given to left, right, and bottom edges of the image because x+1, x-1, y+1 indexing can cause problems when dealing with the last row and the first and the last columns. This is controlled by if-elseif-else statements

Input and output images are provided below.

Figure 7: Original grayscale image

Figure 8: Floyd-Steinberg dithering

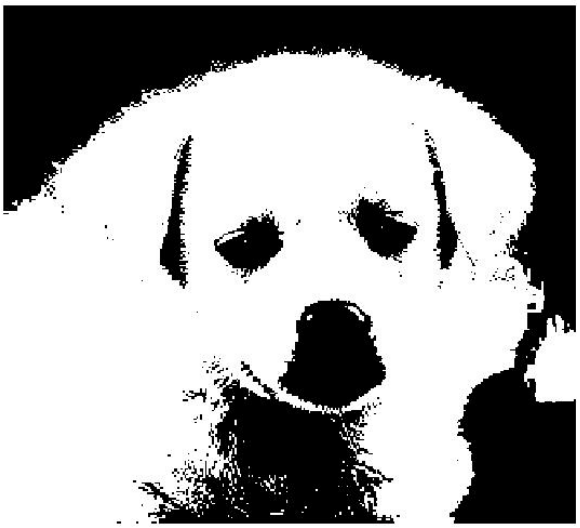

Figure 9: Binarization of original image obtained with `round(img./255)*255`
(provided to illustrate the improvement obtained by the dithering)

Figure 10: Original color image

Figure 11: Floyd-Steinberg dithering

Figure 12: Binarization of original image obtained with `round(img./255)*255`
(provided to illustrate the improvement obtained by the dithering)

From Figures 7-12, it is seen how much Floyd-Steinberg dithering improves the result obtained by binarization though both techniques use the same set of colors (2 for a grayscale image, 8 for a color image). In the case of Floyd-Steinberg dithering, however, it seems that we see almost all colors from the original images.

3.2. Dispersed dots/Bayer dithering

Bayer dithering is a type of ordered dithering, in which threshold maps are used to perform the quantization. In the case of Bayer dithering, Bayer matrices are used as threshold maps.

The algorithm to perform Bayer dithering is as follows:

1. Choose a Bayer matrix.
   We have chosen 8×8 Bayer matrix because it gives a suitable result
2. Compare each pixel of an image with the corresponding element of the Bayer matrix part by part (the size of part is equal to the size of the matrix). For color images, it is done for every color channel.
   To do this, we use a nested for-loop, within which we use a modulo operator to map the pixel to the matrix element. Also, image is converted to double intensity precision so that we could compare pixels with the elements of a threshold matrix
3. If pixel is less than the corresponding threshold, set its value to black (0), and set it to white (255) otherwise

This is done with if-else statements

This algorithm could be further improved if we do not compare pixels with thresholds but offset their values with the corresponding threshold values and if a bigger threshold matrix is used.

The following output images are obtained for the same original images used in Floyd-Steinberg dithering.

Figure 13: Bayer dithering for grayscale image

Figure 14: Bayer dithering for color image

From Figures 13-14, Bayer dithering also improves the result obtained with binarization. Furthermore, it is seen that Bayer and Floyd-Steinberg dithering techniques have their characteristic patterns.

4. Conclusion

To conclude, in the course of this Homework Project, we implemented warmup exercises and the visual illusion for the main part using MATLAB. In warmup exercises, Nearest-Neighbor Interpolation was implemented, and histogram equalization and local histogram equalization were compared. It was found out that Nearest-Neighbor Interpolation results in a quite rough image, and both equalizations have their own drawbacks and benefits. In main part, a visual illusion was obtained with Floyd-Steinberg and Bayer dithering algorithms. In my opinion, Floyd-Steinberg dithering provides a slightly better result than the latter. Nonetheless, both algorithms give much better results than the binarization.

References

[1] C. A. Bouman, Digital Image Processing, Jan. 20, 2021. Available: https://engineering.purdue.edu/~bouman/ece637/notes/pdf/Halftoning.pdf

[2] Ordered Dithering. Available: https://en.wikipedia.org/wiki/Ordered_dithering