ROBT 310 Image Processing

Homework Project 1

By Zarema Balgabekova

1. Introduction

This Homework Project aims to apply the knowledge obtained during lectures in practice. This is done by implementing warmup and main tasks using MATLAB. The warmup part consists of five small tasks connected with the creation of different images. In the main part, a photo mosaic must be obtained. Furthermore, a personal image dataset should be created, and a patch shape other than a rectangular must be generated. The details on how this work has been implemented are provided below.

2. Warmup Part

In this part, five 1000×1000 8-bit images are created avoiding for loops and if statements.

2.1. A grayscale image of constant intensity 100

MATLAB commands:

```
I = uint8(ones(1000, 1000));
I_new = 100*I;
figure
imshow(I_new)
title('A grayscale image of constant intensity 100')
```

First, 1000×1000 matrix of ones is created using the command ones(), uint8() is used because the command ones() produces a double output. Then, the obtained matrix is multiplied by 100 to obtain 1000×1000 matrix of hundreds. After using imshow(), the following image is produced.

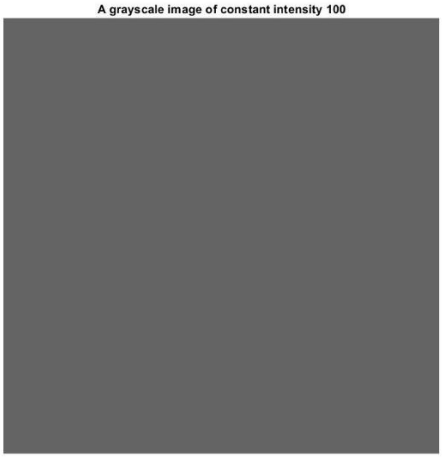

Figure 1: A grayscale image of constant intensity 100

2.2. A grayscale image with alternating black and white vertical stripes, each of which 4 pixels wide

MATLAB commands:

```
I2 = uint8([zeros(1000, 4), 255*ones(1000, 4)]);
I2_new = repmat(I2, 1, 125);
figure
imshow(I2_new)
title('Alternating black and white vertical stripes')
```

First, we create a pattern consisting of one black and one white vertical stripes using zeros() and ones(). After that the command repmat() is used to repeat the columns of this pattern 125 times to obtain 1000×1000 matrix. The final image is obtained using imshow().

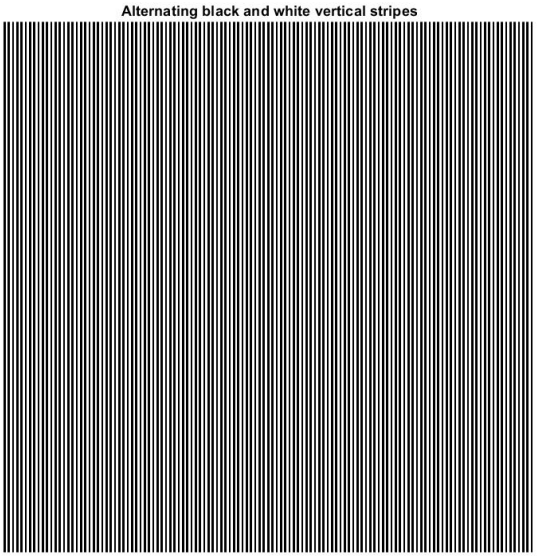

Figure 2: A grayscale image with alternating black and white vertical stripes, each of which 4 pixels wide

2.3. A grayscale image with a ramp intensity distribution, described by I(x, y) = x/2
MATLAB commands:

```
[y,x] = meshgrid(0:999, 0:999);
I3 = x/2;
I3_new = uint8(I3);
figure
imshow(I3, [0 255])
title('Ramp intensity distribution')
```

Here, meshgrid() is used to create square grid coordinates with grid size 1000×1000. Then matrix x is divided by 2 to obtain a ramp intensity distribution. uint8() is used because the command meshgrid() produces a double output. Using imshow(I3, [0 255]), we obtain the following image.

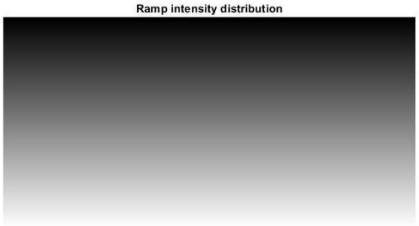

Figure 3: A grayscale image with a ramp intensity distribution, described by I(x, y) = x/2

2.4. A grayscale image with a Gaussian intensity distribution centered at (128, 128), described by $I(x,y) = 255exp - (\frac{(x-128)^2+(y-128)^2}{200^2})$

MATLAB commands:

```
[y,x] = meshgrid(0:999,0:999);
I4 = 255*exp(-((x - 128).^2 + (y - 128).^2)/200^2);
I4_new = uint8(I4);
imshow(I4_new, [0 255]);
title('Gaussian intensity distribution')
```

In this task, meshgrid() is used again, and then a new matrix is obtained using the formula of a gaussian distribution. The produced image is as follows.

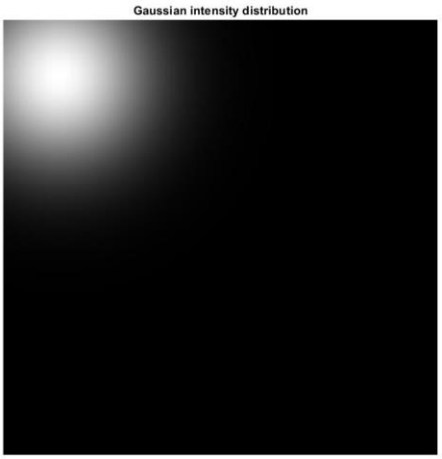

Figure 4: A grayscale image with a Gaussian intensity distribution centered at (128, 128)

2.5. A color image where the upper left quadrant is yellow, the lower left quadrant is red, the upper right quadrant is green, and the lower right quadrant is black

MATLAB commands:

```
I5 = zeros(1000, 1000, 3);
I5(1:500, 1:500, 1) = 255; I5(1:500, 1:500, 2) = 255;
I5(1:500, 1:500, 3) = 0;  %yellow
I5(1:500, 500:1000, 1) = 0; I5(1:500, 500:1000, 2) = 255;
I5(1:500, 500:1000, 3) = 0;   %green
I5(500:1000, 1:500, 1) = 255; I5(500:1000, 1:500, 2) = 0;
I5(500:1000, 1:500, 3) = 0;   %red
I5_new = uint8(I5);
figure
imshow(I5_new)
title('A color image')
```

First, 1000×1000×3 matrix of zeros is created using zeros() because the lower right quadrant is black. Then other colors are set using the proper indexing and RGB color codes, and

uint8() is used because zeros() produces a double output. As a result, the following image is generated.

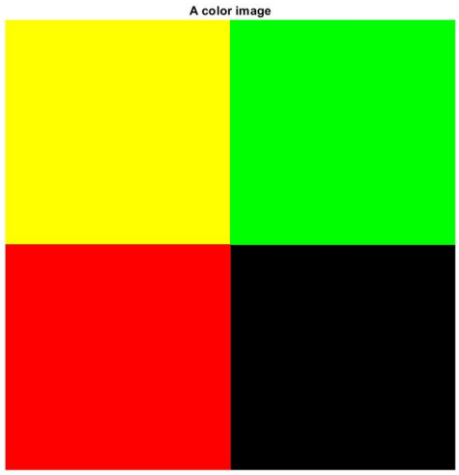

Figure 5: A color image where the upper left quadrant is yellow, the lower left quadrant is red, the upper right quadrant is green, and the lower right quadrant is black

3. Main Part

3.1. Mosaic using rectangular patches

To implement a mosaic, I decided to choose the second method described in the manual because it produces mosaics of a higher quality.

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.

The whole procedure of creating a mosaic can be divided into several steps:

1. Load the dataset and scale images within it to obtain patches using the input from the user. In other words, the size of patches is adjustable.
2. Divide an original target image into sections equal to the size of patches.
3. Choose the patch that is the closest to the section of the image using the technique described in the manual and replace this section with it. Repeat for each section.
4. After replacing all sections with appropriate patches, combine these new sections to obtain a new image – mosaic.

To do the first step, the datastore from provided dataset is created using the command imageDatastore(), and then all images are loaded into a cell to access them later. After getting the size of patches from a user using input(), all images inside the cell are resized using the command imresize().

To implement the second step, first, the original target image is resized so that the whole number of patches could fit into it. Then the image is split into three color channels to divide each of them into sections using the command mat2cell(). After that all color channels of each section are combined again with the help of command cat(). Finally, all sections are saved into another cell.

For the third step, the function compare_pieces() has been created to compare each section of the original image with each patch using the chosen method. Images are converted to double precision with im2double() because there are difficulties with the image subtraction when images are in unsigned integer format. Furthermore, the functions sum(), and sqrt() are used within compare_pieces(). The difference of each section with each patch is saved into a vector. Finding the index of the minimum value in this vector, we obtain the closest patch. Then we replace the section of the image with it and save the result into another cell.

In the fourth step, after replacing all sections of the original image these patches are combined using the command cell2mat() to obtain the mosaic.

Original image and obtained results for the provided dataset are presented below.

Figure 6: Original image

Figure 7: Mosaic with patches of size 50×120

Figure 8: Mosaic with patches of size 28×64

Figure 9: Mosaic with patches of size 10×10

From Figures 7-9, it is seen that with the decreasing size of patches the mosaic becomes smoother and more detailed. However, some colors of the original image change, for example, the color of the cat's sweater. The reason could be that the provided database lacks images with the suitable color. Therefore, it is reasonable to check another dataset.

Mosaics generated from a personal image dataset are as follows.

Figure 10: Mosaic with patches of size 50×120

Figure 11: Mosaic with patches of size 10×10

(generated using the personal database)

Comparing Figures 10 and 11 with Figures 7 and 9, we see that the provided dataset produces better colors of the wall and the cat's paw, whereas the personal dataset provides better colors for everything else. Of course, the mosaic can still be improved by a more suitable dataset. Therefore, it is needed to carefully choose datasets when generating mosaics.

It is worth to see the mosaics produces from another target image.

Another original image and mosaics generated from two datasets are presented below.

Figure 12: Another original image

Figure 13: Mosaic with patches of size 20×30

(using provided dataset)

Figure 14: Mosaic with patches of size 20×30 (using personal dataset)

From Figures 13-14, it is seen that for the chosen original image both datasets give quite similar results.

3.2. Mosaic using circular patches

Circle has been chosen as a different shape of patches. The steps to create a mosaic from circles are the same as for rectangular patches, except the following additional steps.

1. Make squares from rectangular patches
2. Make circular patches from squares
3. Divide original image into squares of the size of square patches
4. Divide original image into circles using squares in the previous step

To make squares from rectangular patches, the number of rows and columns are compared, and then the command imresize() makes squares according to a smaller number. All patches are saved into a new cell to access them later.

To make circles from squares, the function make_circle() has been created. It creates an auxiliary image, in which a white circle is drawn on a black square. Multiplying this auxiliary image by a patch element by element, we obtain circular patches (circular images on black squares).

Similarly, the original image is divided into square and then circular patches. The results are saved into cells.

The following results are obtained for the personal dataset.

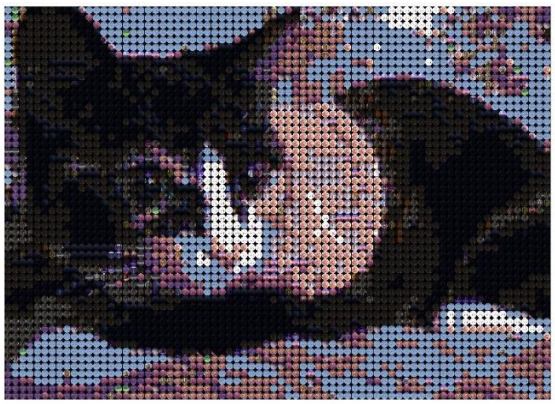

Figure 15: Circular mosaic with patches of size 50×120

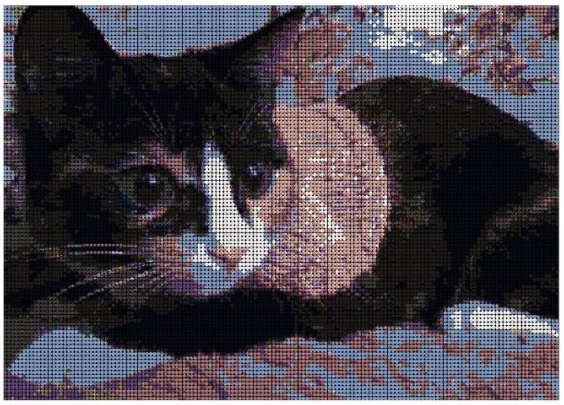

Figure 16: Circular mosaic with patches of size 25×25

Figure 17: Circular mosaic with patches of size 30×30

Comparing Figures 10 and 15, it is seen that Figure 15 demonstrates a smoother mosaic because though we choose patches of size 50×120, they become 50×50 for circular patches.

4.Conclusion

To conclude, in the course of this Homework Project, we implemented warmup exercises and the image mosaic for the main part using MATLAB. In warmup exercises, five different images were created. In main part, a mosaic from rectangular and circular patches was obtained. It was found out that the quality of a mosaic depends not only on the technique to obtain it but also on the dataset of images from which the mosaic is generated. The personal dataset seems to produce better result than the provided one for the first chosen original image. In contrast, both datasets give similar results for the second chosen image. Overall, all obtained results can be further improved by using another dataset or a technique to create a mosaic.