Report

1. **Task 1**

"hw2_1.m" - solution for task 1. Slices the input gray image.

- Input: Reads gray image file "gray_image.jpg" from the same directory.
- Output: Creates 8 image files "bit<k>.tif" in the same directory, where <k> - appropriate bit from slicing. Shows 8 images concatenated together in one figure using imshow.
- slice function from "slice.m" was used to separate bit planes.

This task was one of the easiest problems. The only difficulty here was showing all the 8 images. Initially, I have shown all images in different figures. However, then I have decided to show all images in one figure by concatenating all together.

I have implemented slicing function in a separate script, because I knew that I will repeat this process several times.

2. **Task 2**

"hw2_2.m" - solution for task 2. Generates new images from the most significant 2 and 3 bits of the input gray image.

- Input: Reads gray image file "gray_image.jpg" in the same directory.
- Output: Creates 2 files "most2bits.jpg" and "most3bits.jpg", images generated from most significant 2 and 3 bits of input image respectively. Shows 2 figures: images generated from most significant 2 and 3 bits of input image respectively.
- slice function from "slice.m" was used to separate bit planes.

This task also was not difficult, because I have already implemented slicing function and I just used this function and then combine some planes using standard matrix operations.

3. **Task 3**

"hw2_3.m" - solution for task 3. Applies filtering on the input image.

- Input: Reads gray image file "gray_image.jpg" in the same directory.
- Output: Creates file "filtered.jpg", the filtered image. Shows image: filtered image.
- filtering function from "filtering.m" was used to apply the given mask on the input image.

For this task I have implemented filtering function that takes two arguments: image and mask. The difficulty was that image format is uint8 and its range is limited. So, calculations were not precise. Then I have converted the image into float numbers with function im2double, but the problem was that when this function converts the integer to double it adds 1. Then, I have searched how to properly convert image matrix into float, and found standard function double(), and it worked correctly. Also, I have extended the input matrix by 0's so that the mask could be applied for the edge pixels, but this required some calculations.

4. **Task 4**

"hw2_4.m" - solution for task 4. Applies Sobel edge detection on the input image by filtering with vertical and horizontal edge detection masks.

- Input: Reads gray image file "gray_image.jpg" in the same directory.
- Output: Creates file "sobel.jpg", the merge of two images from filtering by horizontal and vertical edge detection masks. Shows image: the merge of two images from filtering by horizontal and vertical edge detection masks.
- filtering function from "filtering.m" was used to apply the given mask on the input image.
- The threshold for the edge pixels was set to 128.

This task was not difficult because I have already implemented filtering function, and I just used this function with appropriate mask.

5. **Task 5**

"hw2_5.m" - solution for task 5. Applies Laplacian edge detection on the input image.

- Input: Reads gray image file "gray_image.jpg" in the same directory.
- Output: Creates file "laplacian.jpg", the filtered image. Shows image: the filtered image.
- filtering function from "filtering.m" was used to apply the given mask on the input image.
- The threshold for the edge pixels was set to 128.

The same as in task 4.

6. **Task 6**

"hw2_6.m" - solution for task 6. Applies Sobel edge detection on the input image and shows the lines.

- Input: Reads gray image file "gray_image.jpg" in the same directory.
- Output: Creates file "lines.jpg" (rgb image) where the lines are shown with red color. Shows rgb image where the lines are shown with red color.
- filtering function from "filtering.m" was used to apply the given mask on the input image.

This task was the hardest one. The problems I have faced:

- The θ may be negative, but a matrix cannot use negative indexes. So, I have replaced θ by 2*pi-θ, it was positive now but the indicating the same angle.
- The values of θ and ρ may be equal to 0, but a matrix cannot use 0 indexes. TO solve this problem, I have used shifting the result values by 1, so the minimum value was 1.
- The value of ρ may be negative. I have converted such lines to another form: ρ = -ρ and θ = θ - pi. So, this was the same line but defined in different form.

I have played with several values of n, and noticed that when n is too big some lines are repeated several times, and when n is too small some lines are skipped. So, I have set n to 20. Also, I have played with scale values for θ and ρ. When the scale is too small, the different lines may be shown as single, and if the scale is too big, the same line parameters may not converge to one point. I have set the maximum value of ρ as the diagonal of the image, and the maximum value of θ was set to 2*pi.