CS5680/6680 – Fall Semester 2023
Assignment 3 – Filter Techniques for Image Enhancement, Edge Detection, and Noise Removal
Due: 11:59 p.m. Sunday, October 8, 2023
Total Points: 35 points

Warm-up Exercise:
Matlab provides one useful function "edge" to find edges in an intensity image. It supports six edge-finding methods including Sobel, Prewitt, Roberts, Laplacian of Gaussian, zero-cross, and Canny methods. Among these six methods, Sobel and Canny edge detectors are two commonly used ones. Please type "edit edge" on the command line to bring up the implementation of this function. Read the code to understand the following:

- How to find the default threshold for the Sobel edge detector.
- The implementation of each of the five steps of the Canny edge detector summarized in https://en.wikipedia.org/wiki/Canny_edge_detector.

Problem I: Exercises on Low-pass and High-pass Filters in the Spatial Domain [Total: 15 points]
Note: If boundary extension is needed, please pad the boundary with 0's. Your functions should accommodate any square filter with an odd number of rows and columns.

1. [5 points]
Implement an **AverageFiltering** function to perform a filtering operation (i.e., convolution operation), as explained in class, on the input image. This function has two input parameters im and mask, where im is the original grayscale image and mask is the square filter with an odd number of rows and columns. Make sure that your function shows appropriate error messages when the mask does not possess the **three properties** of the low-pass filter (i.e., all elements in the mask are positive, the sum of all elements is 1, and the elements are symmetric around the center). Note: Both input and output images of the **AverageFiltering** function should be an array with the same size and the same data type uint8. **You are NOT allowed to call any built-in filtering or convolution functions.**

Call **AverageFiltering** function to process the noisy image ***Circuit*** using a **standard** 5×5 averaging filter and a **weighted** 3×3 averaging filter, respectively. Display the original image and two processed images in Figure 1 with appropriate titles.

2. [5 points]
Implement a **MedianFiltering** function to perform a filtering operation, as explained in class, on the input image. This function has two input parameters im and mask, where im is the original grayscale image and mask is the square filter with an odd number of rows and columns. Make sure that your function shows appropriate error messages when any of the elements in the mask are not **positive integers**. Note: Both input and output images of the **MedianFiltering** function should be an array with the same size and the same data type uint8. **You are NOT allowed to call any built-in median filtering functions.**

Call this function to process the same noisy image ***Circuit*** using a **standard** 3×3 median filter and a **weighted** 3×3 median filter **M** = [1 2 1 ; 2 4 2; 1 2 1], respectively. Display the original image and two processed images in Figure 2 with appropriate titles.

Note: The standard median filter is a special kind of the weighted median filter. Each value in the median filter indicates the number of copies of the corresponding masked value in the original image involved in the standard median filtering.

3. [5 points]
Use the **strong** 3×3 Laplacian mask to filter the image ***Moon*** by calling an **appropriate built-in function**. Use the formula **Enhanced Image = Original Image + Filtered Image** to get the final enhanced image (Hint:

This formula indicates that one of the two strong Laplacian masks should be used). Use **imshow** to display four images including the original image, the filtered image (display it by setting the values larger than 255 as 255 and setting the values smaller than 0 as 0), the scaled filtered image whose intensities fall in the range of [0, 255], and the enhanced image, in Figure 3 with appropriate titles (Refer to Figure 3.40 in the textbook or the same figure on slide 57 of my notes).

Problem II: Exercises on Edge Detectors in the Spatial Domain [Total: 8 points]

Implement a **FindEdgeInfo** function to locate the important edges and compute the edge histogram of the input image. This function has two input parameters im and bin, where im is the original grayscale image and bin is the number of bins. It returns two outputs edges and edgeHist, where edges contains important edges in the input image (i.e., im) and edgeHist contains the counts of the orientation (angles) of the edge in each bin, which can be computed by: $\theta = \arctan\left(G_x / G_y\right)$, where *Gx* is the gradient component produced by the horizontal edge detector and *Gy* is the gradient component produced by the vertical edge detector (Refer to slide 63 of my notes. *Gx* is the filter at left and *Gy* is the filter at right). **You are NOT allowed to call any built-in edge functions and histogram functions.**

Call this function to compute a 30-bin edge histogram (e.g., dividing the entire range of edge orientations into 30 equal intervals and get the counts of angles falling in each interval) of the image ***Rice***. Display the original image, the image with the important edges, and the edge histogram in Figure 4 with appropriate titles. Note: The entire range of the edge orientations should cover 360 degrees.

Problem III: A Practical Problem [12 points]

Use the techniques explained in class to get rid of the streaks (stripes) in the image "**Text.gif**". Please write a function "RemoveStripes" with the original grayscale image as the input and the cleaned grayscale image as the output. Display the original, some important intermediate images or plots, and the "cleaned" image in a few figures with appropriate titles. **[10 points]**

You may test your function RemoveStripes on **Text1.gif** to see whether it can get rid of its stripes. **[2 points]**