James Murphy
Image Processing HW #3

1 : What effect would setting to zero the lower-order bit planes have on the histogram of an image in general? Why?

- The lower order bit planes are 0-3 and the higher bit planes are 4-7. Setting zero to the lower order bit planes would get rid of all odd members of the histogram. There are 2^8 bits, for a total of 256. When zeroing out the lower bitplanes, the image will slightly lose color and be dimmed.

1a : What would be the effect on the histogram if we set to zero the higher-order bit planes instead? Why? Support your argument with an example

- If you zero out the higher order bit planes, the image will become much darker and lose a lot of its low-frequency parts and mess with its composition.

2 : Implement your own MatLAB function with performs RGB histogram matching on color images.

- In this question, the first thing that you want to do is split the 2 images into their separate 3 RGB values.

```
%Turn the 2 images into  a double that can be operated on
mercury = double(mercury);
techno = double(techno);

%Grab the vector red
mercury_red = mercury(:,:,1);
techno_red = techno(:,:,1);

%Grab the vector for green
mercury_green = mercury(:,:,2);
techno_green = techno(:,:,2);

%Grab the vector for blue
mercury_blue = mercury(:,:,3);
techno_blue = techno(:,:,3);

%Take the mapped RGB values of mercury and techno and store it as
%individual RGB values
result_red = uint8(map_RGB(mercury_red,techno_red));
result_green = uint8(map_RGB(mercury_green, techno_green));
result_blue = uint8(map_RGB(mercury_blue, techno_blue));
```

- Next, you have to calculate the histogram of the input images, techno trousers and mercury

```
for i = 1:row
    for j = 1:col

        %get the value of the pixel at the current index i, j
        hist_value = img_input(i,j) + 1 ;

        %Increment each histogram value by 1
        hist_result(2, hist_value) = hist_result(2, hist_value) + 1;
    end
end
```

- Add sum with the histogram_result(2, i) and then take that sum and divide it by the total amount of pixels and store it in the j, i pair.

```
    % Take the row * column to find the total num of pixels
    pixel_count = row*col;
    sum = 0;

    % In a loop from 2^1 -> 2^8
    for i = 1 : 256
        % Add sum with sum + the (2, i) value in the matrix for each pixel
        sum = sum + hist_result(2,i);

        %Take the ration of the sum divided by the total pixel count and
        %store it in it's 2, i pair

        hist_result(2,i) = sum / pixel_count;

    end
end
```

- Next, the histogram table needs to be calculated. The histogram table contains the pixel values. For each pixel k, take the sum of all numbers >= k, and take the ratio of it over the total number of pixels.

```
for i = 1 : 256
    %Take the absolute value of the (2, i) pair and subtract it from
    %the input incoming to the table
    difference = abs(histogram(2,i) - input);
    %Check if the difference value is smaller than the current minimum
    if difference < minimum
        %If the minimum is greater than the difference, swap the values
        minimum = difference;
        %Take the pixel value output and set it equal to the current
        %iteration in the histogram matrix

        pixval_output = histogram(1,i);
    end
end
```

- Lastly, go through the output and update the pixel values of the swapped RGB values from the previous part

```
for i = 1 : row
    for j = 1 : col
        %To take the output, take the inputted table in the function
        %and to get the value for i, j, take the 2, ((i, j) + 1)
        %paitinh and store it into i, j
        img_output(i, j) = table_input(2, img_input(i,j)+1);
    end
end
```

Output :

- I used the built int MATLAB histogram match function comparison. It looks very close, so it is a good result. It may be my eyes but the hard coded image looks a little less bright than matlabs function.

MATLAB Histogram Function

Hard Coded Histogram

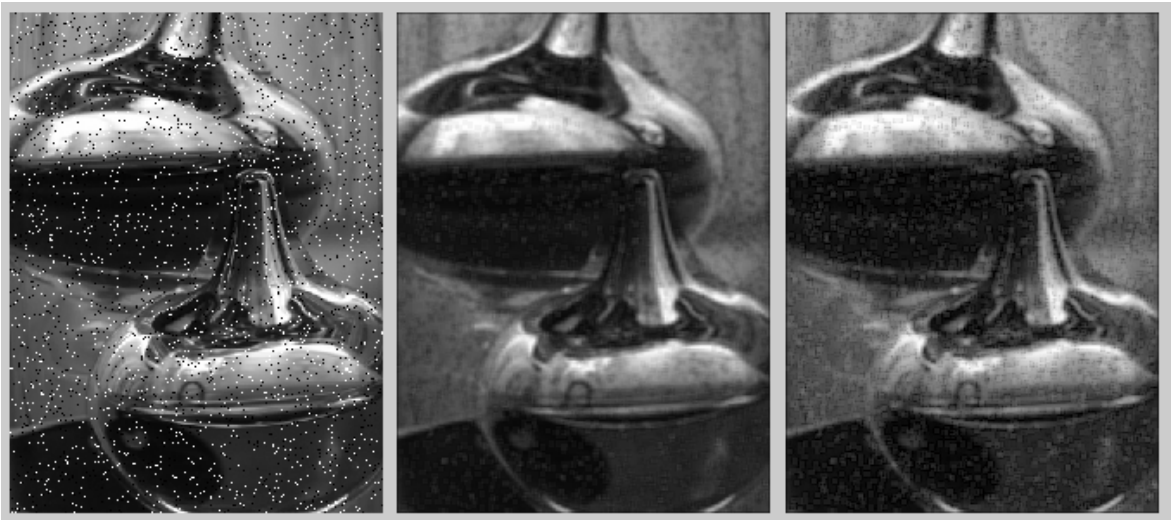
Question 3 :

- An averaging mask is the number of pixels in the window. Each pixel is then replaced by its average with the average of its nearest 4 or 8 pixels. This mask is used for smooth filtering and eliminating noise.

- A Laplacian mask is similar to an averaging mask, but instead of averaging the pixels around it, it assigns a different weighted value to each surrounding weight. The Laplacian filter highlights regions of high intensity change. It helps highlight the fine details of an image.

- I think that the order does matter. From what I read online, these functions are using the second derivative information about the intensity changes in neighboring pixels. The derivative is sensitive to noise. Since it's sensitive to noise and the average mask is used to reduce noise, the averaging mask should be applied first before using the Laplacian mask.

Example :



(Not my picture or results, taken from stack exchange.)