Image Processing Programming Assignment #1

In this assignment, our goal is to enhance the given images (6 in total) using the techniques of contrast adjustment, noise reduction, color correction, and so on. Multiple techniques have been implemented, and only the best result of each method will show in this report (Note that, all of them are achieved by trying to change parameters multiple times). The code is done with Matlab programming language and is released on https://github.com/KennethYapWL/NCTU-DIP-2020/tree/main/Project%201, and the references are listed on the last page of this report.

Those methods implemented are:

1. Contrast adjustment method
   → Piecewise linear stretching
   → Power law transformation
2. Smoothing and noise reduction method
   → Order statistic filtering, including mean filter, max filter, min filter, median filter
   → Midpoint filtering
   → Gaussian smoothing filtering
   → Alpha-trimmed filtering
   → Adaptive local noise reduction
   → Adaptive median filtering
3. Sharpening method
   → Laplacian filtering
4. Color Adjustment
   → Conversion and adjustment to HSV components

This report is organized as follow:

→ Section 1 (Experimental Results): This part will describe the best method and combination to enhance each of the given images. A brief explanation will also be provided in this section.
→ Section 2 (Observations and Discussions): This part will give brief descriptions of the other experiment I have tried, also the further observations of section 1 will be discussed in this section
→ Section 3 (Code Analysis): This part will show all the code used in this assignment
→ Section 4 (References): This part will list all the articles I have read.

Section 1: Experimental Results

This section is focused on the combination that achieved the greatest image enhancement during all the experiments that have been done. The result of the enhancement of each image will be demonstrated in this section, and list the parameters used of each step of the best combination.

A) The first given image (p1im1.png)

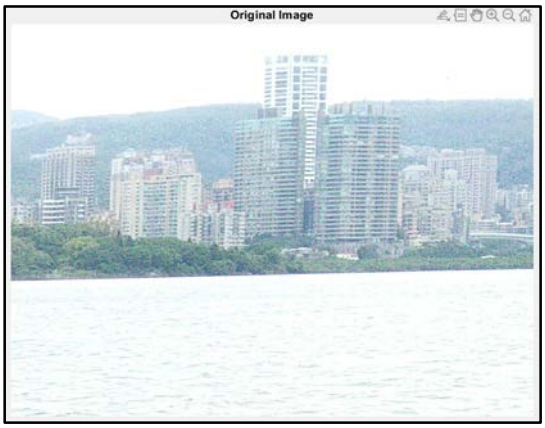

figure 1.A.1

figure 1.A.2

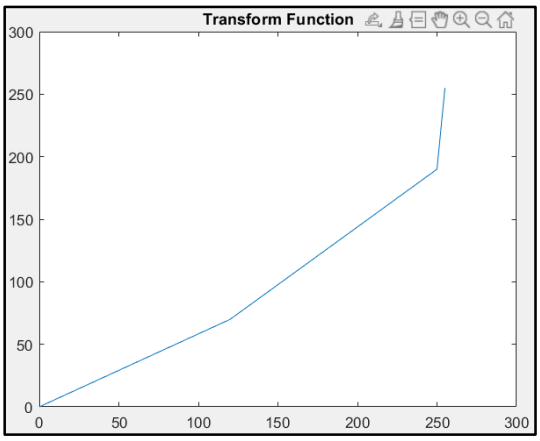

figure 1.A.3

As shown in the figures above, figure 1.A.1 is the original image. Both the foreground and background of the original image are bright. Thus, my very first thing to do is contrast enhancement. According to experiments have been done so far, the best combination is

1) Piecewise Linear Stretching, with settings, r1=120, s1=70, r2=250, s2=190. Figure 1.A.3 shows the transform function of these settings.
2) Adaptive Local Noise Reduction filtering, with a filter size of 3.
3) Color Correction by adjusting the HSV components, with H and S remain unchanged, and V subtract 40 (V = V - 40).

The parameters were chosen in step 1 is based on the examination of the image histogram. Note that steps 1 and 2 are applied only to the Intensity component of the HSI color space. Figure 1.A.2 shows the result of the combination, and more details can be found on the file “Solution_Img1.m” in the provided GitHub.

B) The second given image (p1im2.png)

figure 1.B.1

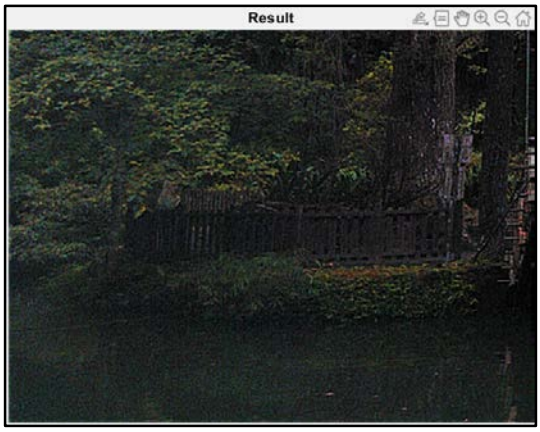

figure 1.B.2

As shown in the figures above, figure 1.B.1 is the original image. Both the foreground and background of the original image are dark. Based on the experiments, the power-law transformation perform better than piecewise linear stretching on the contrast adjustment of this image, the best combination is

1) Power Law Transformation, with parameters, c = 1.2 and γ = 0.9
2) Color correction by adjusting the HSV components, with H = H - 30°, S = S + 0.2 and V = V - 20.
3) Bilateral Filtering with a filter size of 3, σ_d = 15, and σ_g = 15.
4) Sharpening by using Laplacian Filter

Note that steps 1,3 and 4 are applied only to the Intensity component of the HSI color space. Figure 1.B.2 shows the result of the combination, and more details can be found on the file "Solution_Img2.m" in the provided GitHub.

C) The third given image (p1im3.png)

figure 1.C.1

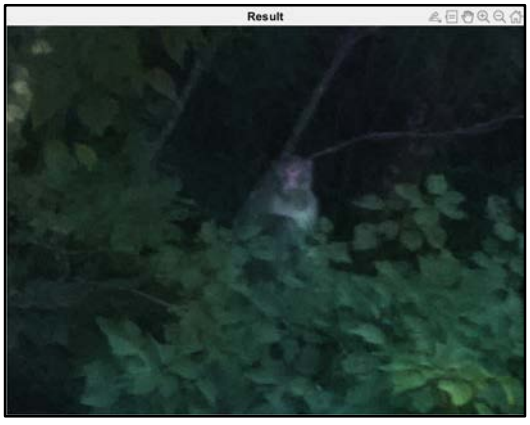

figure 1.C.2

As shown in the figures above, figure 1.C.1 is the original image. Both the foreground and background of the original image are dark. Based on the experiments, the contrast adjustment to RGB components separately results in better enhancement of the image. The best combination is

1) Power Law Transformation, with parameters, c = 1.4 and γ = 3.1 on Red component, c = 1.4 and γ = 2.7 on Green component, c = 1.4 and γ = 2.5 on Blue component.

2) Sharpening by using Laplacian Filter
3) Color correction by adjusting the HSV components, with H = H - 40°, S = S + 0.1 and V = V - 30.

Note that step 1 is applied on RGB components separately while step 2 is applied only to the Intensity component of the HSI color space, Figure 1.C.2 shows the result of the combination, and more details can be found on the file “Solution_Img3.m” in the provided GitHub.

D) The forth given image (p1im4.png)

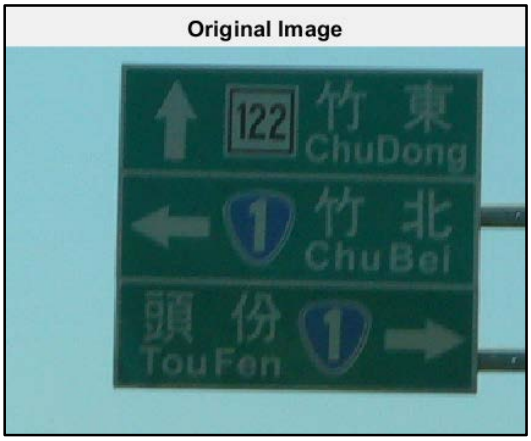

figure 1.D.1

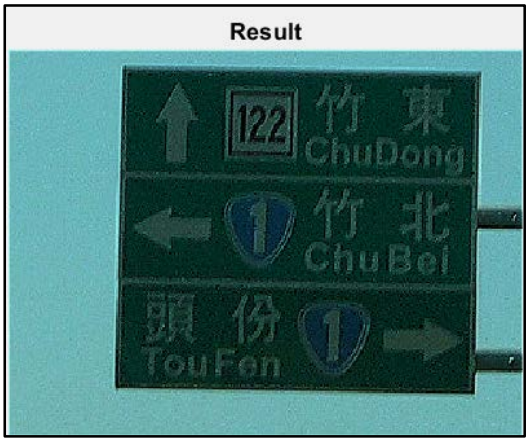

figure 1.D.2

As shown in the figures above, figure 1.D.1 is the original image. According to the experiments that have been done so far, contrast adjustment and smoothing techniques are not so effective in this image. The only two steps are applied to this image, the best combination is

1) Sharpening by using Laplacian Filter
2) Color correction by adjusting the HSV components, with H = H - 5°, S = S + 0.1 and V = V - 20.

Note that step 1 is applied only to the Intensity component of the HSI color space. Figure 1.D.2 shows the result of the combination, and more details can be found on the file “Solution_Img4.m” in the provided GitHub.

E) The fifth given image (p1im5.png)

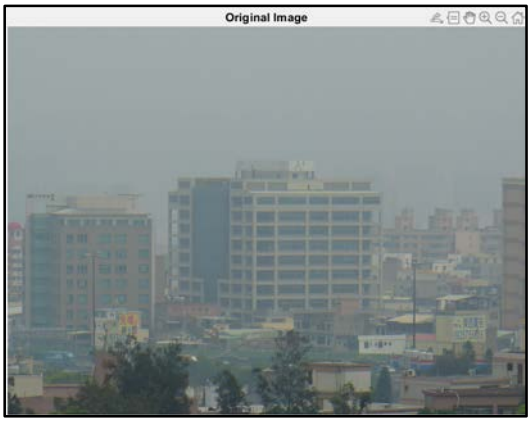

figure 1.E.1

figure 1.E.2

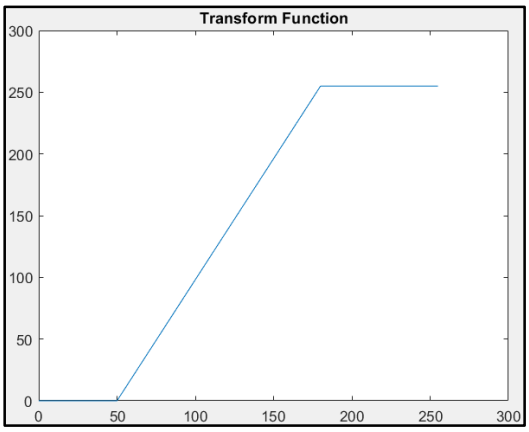

figure 1.D.3

As shown in the figures above, figure 1.E.1 is the original image. On the part of contrast adjustment, piecewise linear stretching output a better result than power-law transformation based on the experiments that have been tried so far. The best combination is

1) Piecewise Linear Stretching, with settings, r1=50, s1=0, r2=180, s2=255. Figure 1.D.3 shows the transform function of these settings.
2) Color Correction by adjusting the HSV components, with H = H - 5°, S = S + 0.09 and V = V - 25.
3) Sharpening by using Laplacian Filter.

The parameters were chosen in step 1 is based on the examination of the image histogram. Note that steps 1 and 3 are applied only to the Intensity component of the HSI color space. Figure 1.E.2 shows the result of the combination, and more details can be found on the file “Solution_Img5.m” in the provided GitHub.

F) The sixth given image (p1im6.png)

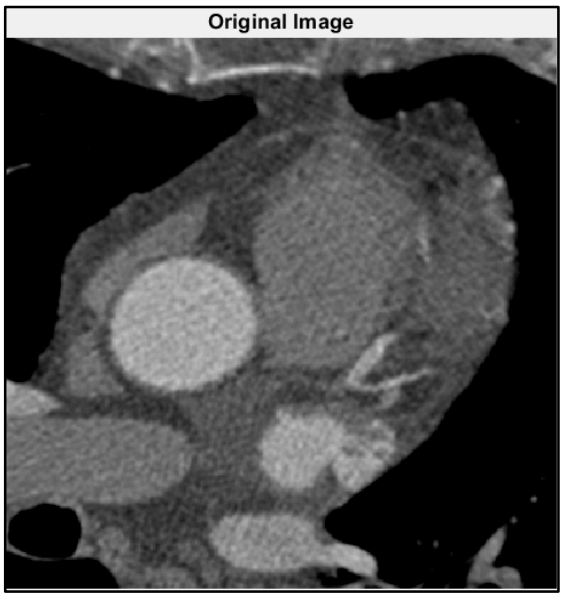

figure 1.F.1

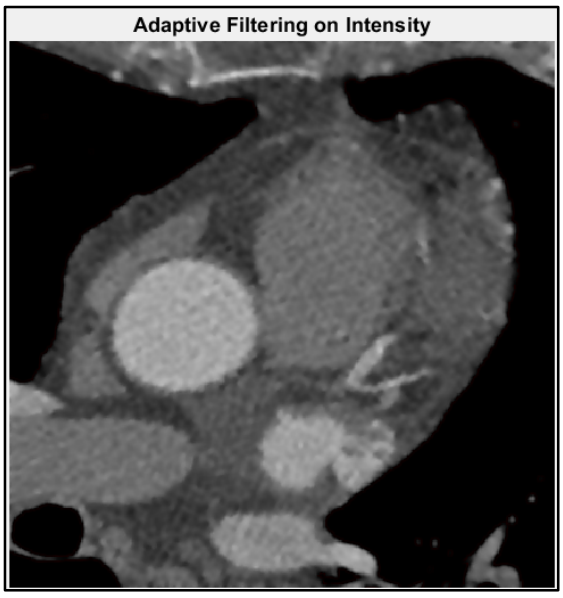

figure 1.F.2

As shown in the figures above, figure 1.F.1 is the original image. In this image, noise reduction is the only way allowed to do image enhancement. As stated at the beginning of the report, there are multiple denoising techniques have been implemented in this assignment. Some of them result very similarly, and some of them perform worse, further observation will be discussed in **Section 2**. The final smoothing technique chosen is the adaptive local noise reduction filtering (also named adaptive mean filtering), with a filter size of 5.

Figure 1.F.2 shows the result of the combination, and more details can be found on the file “Solution_Img6.m” in the provided GitHub.

Section 2: Observations and Discussions

Each of the best combination image enhancement described in section 1 is achieved by trying multiple times of changing the combination and the value of parameters of each step. During all these experiments, some of the observations have been found and will be shown in this section.

1. *Contrast Adjustment: Apply on RGB components separately vs Apply on Intensity component of the HSI color model.*

   It is a little bit different situation on applying contrast adjustment on the grayscale images (with only one channel dimension) and that on color images (usually with 3 channel dimensions, say RGB). Some of the articles or websites, like ref[13] said the adjustment can simply just loop over each color channel (RGB components), and some of them said it is better to apply only on the intensity component. During the experiments, I have done some observations on this issue.

figure 2.1.1

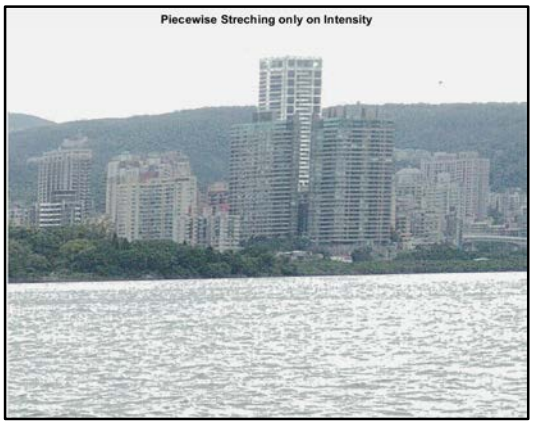

figure 2.1.2

figure 2.1.3

figure 2.1.4

As shown in the figures above, figure 2.1.1 and figure 2.1.2 are the results after applying the piecewise linear stretching on the first given image (refer to figure 1.A.1). Note that figure 2.1.1 is adjusted by applying the method on each of the RGB components and figure 2.1.2 is adjusted by applying the method on only the intensity component of HSI. We could see that figure 2.1.1 is more colorful, while figure 2.1.2 is more realistic. According to the ref[15], these happened due to it will not considering the relevance of R, G, and B channels if we immediately applying the method on RGB components separately.

However, applying the method on RGB components separately is way faster than applying the method on the intensity component of HSI (we needed to convert RGB to HSI, and convert it back to RGB after the adjustment done). Not all the examples will always suitable for doing adjustments on intensity component, figure 2.1.3, and figure 2.1.4 are the results after applying the power law transformation on the third given image (refer to figure 1.C.1). Surprisingly, adjustment to RGB separately performs better compared to the adjustment to intensity component.

More details can be found on the files “Experiment_img1.m” and “Experiment_img3.m” in the provided GitHub.

2. *Smoothing: Is a larger filter size results in more blur???*

The effectiveness of the smoothing filter depends on which noise does it applied. For example, the median filter is effective in removing the impulse noise, such as pepper-and-salt noise, the mean filter is effective in removing the Gaussian noise, the alpha-trimmed filtering is effective in removing the mixture of the Gaussian noise and the impulse noise.

As one of the parameters of the smoothing filtering, the filter size is also a factor that influences the effectiveness of the smoothing filter. During the smoothing experiments, I was wondering that how could filter size influence the result of the smoothing.

figure 2.2.1

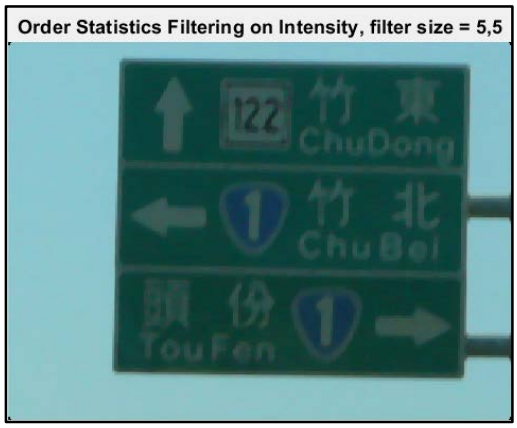

figure 2.2.2

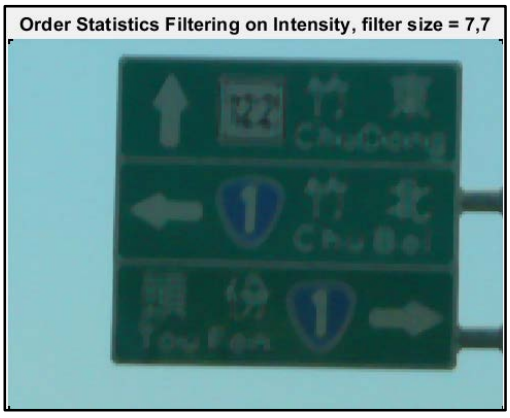

figure 2.2.3

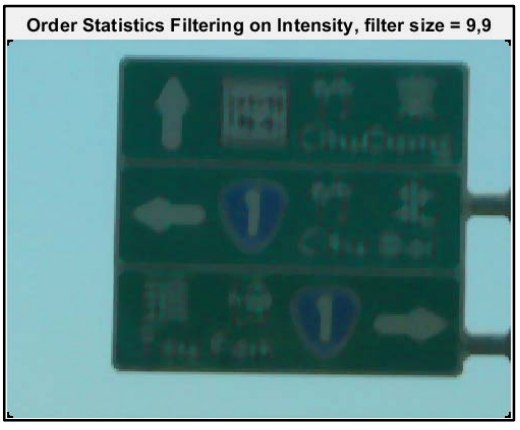

figure 2.2.4

As shown in the figures above, figure 2.2.1, figure 2.2.2, figure 2.2.3 and figure 2.2.4 are the results after applying the median filtering with filter size 3x3, 5x5, 7x7, 9x9 respectively.

The image becomes more blurry when applying a filter with a bigger size. Refer to ref[18], the larger kernels would remove more noise from the image, but they will also mean more undesirable artifacts (more blur).

Moreover, I have found that a larger filter size also results in a black edge in images. Notice that there are black edges in all of the corners of figure 2.2.2, figure 2.2.3, and figure 2.2.4. Also, the filter size is not necessary to be square size or odd numbers. (refer to ref[16] and ref[17]).

More details can be found on the file “Experiment_different_filter_size.m” in the provided GitHub.

Section 3: Code Analysis

Since the content in files “Experiment_img1.m”, “Experiment_img2.m”, “Experiment_img3.m”, “Experiment_img4.m”, “Experiment_img5.m”, “Experiment_img6.m”, and “Experiment_different_filter_size.m” are only used as trying multiple parameter values and multiple combinations to achieve the best image enhancement result, these experiment used files will not be shown in this section. (There are almost more than 200 lines of code in each of the files)

Except for all experiments used files, this section will show all the code used in this assignment. We will first overview all of the methods implemented in this assignment, then show the code that performs the best enhancement of each image.

Note that the explanations of each part of the code are shown in the comment (font color in light green).

1. Contrast Adjustment Code.

→ Piecewise Linear Stretching

```
function [imgMatr, transFunc] = piecewise_linear_streching(imgMatr,r1,s1,r2,s2)
    % --------------------------------
    % This function perform the piecewise linear transformation
    % with using the equation y = mx + c on each segment.
    % return transformed image matrix
    % --------------------------------

    imgMatr = uint8(imgMatr);

    % declare each starting point, ending point of each segment.
    p1 = [0,0];
    p2 = [r1,s1];
    p3 = [r2,s2];
    p4 = [255,255];

    % calculate gradients of each segment.
    m1 = (p2(2) - p1(2))/(p2(1) - p1(1));
    m2 = (p3(2) - p2(2))/(p3(1) - p2(1));
    m3 = (p4(2) - p3(2))/(p4(1) - p3(1));

    % calculate intercepts of each segment.
    c1 = p1(2) - m1*p1(1);
    c2 = p2(2) - m2*p2(1);
    c3 = p3(2) - m3*p3(1);

    % build transformation function
    transFunc = zeros(2,256);
    transFunc(1,:) = [0:255];
```

```
    for x = 0 : 255
        y = 0;
        if (x <= p2(1))
            y = m1 * x + c1;
        elseif (x > p2(1) && x <= p3(1))
            y = m2 * x + c2;
        else
            y = m3 * x + c3;
        end
        transFunc(2,x + 1) = y;
    end

    % transform the input image
    hg = size(imgMatr,1);
    wd = size(imgMatr,2);

    size(transFunc);

    for row = 1 : hg
        for col = 1 : wd
            origPxl = imgMatr(row,col);
            imgMatr(row,col) = transFunc(2, origPxl + 1);
        end
    end

end
```

→ Power Law Transformation

```
function imgMatr = power_law_transform(imgMatr,c,gamma)
    % --------------------------------
    % This function perform the power law transformation
    % return transformed image matrix
    % https://www.ques10.com/p/3690/determine-the-output-image-using-power-law-transfo/?
    % --------------------------------
    imgMatr = double(imgMatr)/255;
    imgMatr = c * (imgMatr).^gamma;
    imgMatr = uint8(imgMatr * 255);
end
```

2. Smoothing and Noise Reduction Code.

→ Order Statistics Filter

The code includes the implementation of the median filter, averaging filter, max filter, and min filter, which depends on the value of the parameter “Type”.

```
function transImgMatr = statistic_filtering(imgMatr,filtWd,filtHg,Type)
    % --------------------------------
    % This function perform the noise removal with statistic filtering
    % "mean" --> averaging filter
    % "max" --> max filter
    % "min" --> min filter
    % "median" --> median filter
    % return transformed image matrix
    % --------------------------------

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % zero padding
    shift_hg = int32(floor(filtHg / 2)); % calulate the extra space needed for hg
    shift_wd = int32(floor(filtWd / 2)); % calulate the extra space needed for wd
    paddMatr = zeros(hg + (shift_hg * 2),wd + (shift_wd * 2));
    for row = 1 : hg
        for col = 1 : wd
            paddMatr(row + shift_hg,col + shift_wd) = imgMatr(row,col);
        end
    end

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(hg,wd);
```

```
    % Do the spatial filtering computation
    for row = 1 : hg
        for col = 1 : wd
            % assign filter pixel values
            filter = paddMatr(row : row + (filtHg - 1),col : col + (filtWd - 1));

            if (Type == "mean")% assign the mean of the filter pixel values
                transImgMatr(row,col) = mean(filter(:));

            elseif (Type == "max")% assign the max of the filter pixel values
                transImgMatr(row,col) = max(filter(:));

            elseif (Type == "min")% assign the min of the filter pixel values
                transImgMatr(row,col) = min(filter(:));

            else % assign the median of the filter pixel values
                transImgMatr(row,col) = median(filter(:));

            end
        end
    end

    transImgMatr = uint8(transImgMatr);

end
```

→ Gaussian Smoothing Filter

```
function transImgMatr = gaussian_filtering(imgMatr,sigma,filtSize)
    % --------------------------------
    % This function perform the noise removal with gaussian filtering
    % return transformed image matrix
    % --------------------------------

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % replicate padding on image matrix
    shift = double(floor(filtSize / 2));
    paddMatr = padarray(imgMatr,[shift shift],'replicate');

    % Make 2D Gaussian kernel
    x=-ceil(shift):ceil(shift);
    Gauss1D = exp(-(x.^2/(2*sigma^2)));
    Gauss1D = Gauss1D/sum(Gauss1D(:));

    GaussX=reshape(Gauss1D,[length(Gauss1D) 1]);
    GaussY=reshape(Gauss1D,[1 length(Gauss1D)]);

    GaussMatr = zeros(length(GaussX),length(GaussY));
    for x = 1 : length(GaussX)
        for y = 1 : length(GaussY)
            GaussMatr(x,y) = GaussX(x) * GaussY(y);
        end
    end

    clearvars GaussX GaussY Gauss1D

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(hg,wd);
```

```
    % Do spatial filtering
    for row = 1 : hg
        for col = 1 : wd

            tmp = 0;
            for x = 1 : filtSize
                for y = 1 : filtSize
                    tmp = tmp + GaussMatr(x,y) * paddMatr(x + (row - 1), y + (col - 1));
                end
            end

            transImgMatr(row,col) = tmp;

        end
    end

    transImgMatr = uint8(transImgMatr);

end
```

→ Midpoint Filter

```
function transImgMatr = midpoint_filtering(imgMatr,filtWd,filtHg)
    % --------------------------------
    % This function perform the noise removal with midpoint filtering
    % return transformed image matrix
    % --------------------------------

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % zero padding
    shift_hg = int32(floor(filtHg / 2)); % calulate the extra space needed for hg
    shift_wd = int32(floor(filtWd / 2)); % calulate the extra space needed for wd
    paddMatr = zeros(hg + (shift_hg * 2),wd + (shift_wd * 2));
    for row = 1 : hg
        for col = 1 : wd
            paddMatr(row + shift_hg,col + shift_wd) = imgMatr(row,col);
        end
    end

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(hg,wd);
```

```
    % Do the spatial filtering computation
    for row = 1 : hg
        for col = 1 : wd
            % assign filter pixel values
            filter = zeros(filtHg, filtWd);
            for x = 1 : filtHg
                for y = 1 : filtWd
                    filter(x,y) = paddMatr(x + (row - 1), y + (col - 1));
                end
            end

            % assign the midpoint of the filter pixel values
            Max = max(filter(:));
            Min = min(filter(:));
            transImgMatr(row,col) = (Max + Min)/2;
        end
    end

    transImgMatr = uint8(transImgMatr);
end
```

→ Alpha-Trimmed Filter

```
function transImgMatr = alpha_trimmed_filtering(imgMatr,filtWd,filtHg,alpha)
    % --------------------------------
    % This function perform the noise removal with alpha trimmed filtering
    % return transformed image matrix
    % --------------------------------

    wd = size(imgMatr,1); % get the width of image
    hg = size(imgMatr,2); % get the height of image


    % zero padding on image matrix
    shiftWd = double(floor(filtWd / 2));
    shiftHg = double(floor(filtHg / 2));
    paddMatr = padarray(imgMatr,[shiftWd shiftHg],0);

    % number of element to be trimmed
    d = floor(floor(filtWd * filtHg * alpha)/2);

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(wd,hg);
```

```
    % Do the spatial filtering computation
    for row = 1 : wd
        for col = 1 : hg
            % assign filter pixel values
            filter = zeros(filtWd, filtHg);
            for x = 1 : filtWd
                for y = 1 : filtHg
                    filter(x,y) = paddMatr(x + (row - 1), y + (col - 1));
                end
            end

            list = sort(filter(1 : filtWd * filtHg)); % sort
            list = list(d + 1 : length(list)-d); % trimmed
            transImgMatr(row,col) = mean(list); % take mean value

        end
    end

    transImgMatr = uint8(transImgMatr);



end
```

→ Adaptive Filter

```
function transImgMatr = adaptive_filtering(imgMatr,filtWd,filtHg)
    %-----------------------------------
    % This function perform the noise removal with adaptive filtering
    % return transformed image matrix
    % refer to https://www.imageeprocessing.com
    % /2011/12/adaptive-filtering-local-noise-filter.html
    %-----------------------------------

    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % zero padding on image matrix
    shiftHg = double(floor(filtHg / 2));
    shiftWd = double(floor(filtWd / 2));
    paddMatr = padarray(imgMatr,[shiftHg shiftWd],0);

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(hg,wd);

    % create matrices of local variance,
    % local means,
    local_var = zeros(hg,wd);
    local_mean = zeros(hg,wd);
```

```
    % Do the spatial filtering computation
    for row = 1 : hg
        for col = 1 : wd
            % assign filter pixel values
            filter = paddMatr(row : row + (filtHg - 1),col : col + (filtWd - 1));

            % get local mean for the local region
            local_mean(row,col) = mean(filter(:));
            % get local variance for the local region
            local_var(row,col) = mean(filter(:).^2) - mean(filter(:)).^2;

        end
    end

    % calculate noise variance
    noise_var = mean(local_var(:));

    % replace local variances which smaller than noice variance
    local_var = max(local_var,noise_var);

    % apply adaptive expression formula to image matrix
    transImgMatr = imgMatr - (noise_var./local_var).*(imgMatr - local_mean);

    transImgMatr = uint8(transImgMatr);

end
```

→ Adaptive Median Filter

```
function transImgMatr = adaptive_median_filtering(imgMatr,filtSize)
    %-----------------------------------
    % This function perform the noise removal with adaptive median filtering
    % return transformed image matrix
    %
    % for code pls refer to :
    % 1.https://www.mathworks.com/matlabcentral/answers/247326-help-with-adaptive-median-filter
    % 2.https://github.com/mortezamg63/Adaptive-Median-Filter/blob/master/Adaptive_Median_filter.m
    %
    % and refer to it's algorithm explanation:
    % https://www.massey.ac.nz/~mjjohnso/notes/59731/presentations/Adaptive%20Median%20Filtering.doc
    %-----------------------------------

    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % filter maximum size
    Smax = 9;

    % replicate padding on image matrix
    shift = double(floor(Smax / 2));
    paddMatr = imgMatr;
    counterPadding=shift;
     while(counterPadding)
         paddMatr=[paddMatr(:,1) paddMatr paddMatr(:,end)];
         paddMatr=[paddMatr(1,:);paddMatr;paddMatr(end,:)];
         counterPadding=counterPadding-1;
     end

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(hg,wd);
```

```
    startIdx = Smax - shift;
    for row = startIdx : startIdx + (hg - 1)
        for col = startIdx : startIdx + (wd - 1)
            result = adap_med_computation(paddMatr,filtSize,Smax,row,col);
            transImgMatr(row - startIdx + 1, col - startIdx + 1) = result;
        end
    end

    transImgMatr = uint8(transImgMatr);


end
```

And sub-functions inside:

```
function output = adap_med_computation(paddMatr,filtSz,Smax,x,y)
    %-------------------------------------------------
    % Perform the adaptive median computation
    % paddMatr --> the input image with padding
    % filtSz --> current filter size
    % Smax --> maximum filter size
    % x,y --> current row index and current col index resp.
    % return transformed pixel value
    %-------------------------------------------------

    % get the area of neighborhood
    space = ceil((filtSz - 1)/2);
    ngbHood = paddMatr(x - space : x + space,y - space : y + space);

    % transformed pixel value
    output = 0;

    Zxy = paddMatr(x,y); % image curr index pixel value
    % assign local median, max, and min
    Zmed = median(ngbHood(:));
    Zmax = max(ngbHood(:));
    Zmin = min(ngbHood(:));

    A1 = Zmed - Zmin;
    A2 = Zmed - Zmax;
```

```
    % level A algorithm
    if (A1 > 0 && A2 < 0)
        % level B algorithm
        B1 = Zxy - Zmin;
        B2 = Zxy - Zmax;
        if (B1 > 0 && B2 < 0)
            output = Zxy;
            return;
        else
            output = Zmed;
            return;
        end
    else % continue level A algorithm
        if (filtSz < Smax)
            filtSz = filtSz + 2;
            output = adap_med_computation(paddMatr,filtSz,Smax,x,y);
            return;
        else
            output = Zxy;
        end
    end

end
```

→ Bilateral Filter

```
function transImgMatr = bilateral_filtering(imgMatr,filtSize,sigma_d,sigma_g)
    % ---------------------------------
    % This function perform the noise removal with bilateral filtering
    % return transformed image matrix
    % Code refer to https://www.mathworks.com/matlabcentral/fileexchange/12191-bilateral-filtering
    % ---------------------------------
    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % replicate padding on image matrix
    shift = double(floor(filtSize / 2));
    paddMatr = padarray(imgMatr,[shift shift],0);

    % Make 2D Gaussian kernel
    x=-ceil(shift) : ceil(shift);
    Gauss1D = exp(-(x.^2 / (2 * sigma_d^2)));
    Gauss1D = Gauss1D / sum(Gauss1D(:));

    GaussX=reshape(Gauss1D,[length(Gauss1D) 1]);
    GaussY=reshape(Gauss1D,[1 length(Gauss1D)]);

    GaussMatr = zeros(length(GaussX),length(GaussY));
    for x = 1 : length(GaussX)
        for y = 1 : length(GaussY)
            GaussMatr(x,y) = GaussX(x) * GaussY(y);
        end
    end

    clearvars GaussX GaussY Gauss1D

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(hg,wd);
```

```
    % Do spatial filtering
    for row = 1 : hg
        for col = 1 : wd

            % Extract local region
            L = paddMatr(row : row + (filtSize - 1),col : col + (filtSize - 1));
            % Compute Gaussian Intensity Weights
            H = exp(-(L - imgMatr(row,col).^2) / (2 * sigma_g^2));
            % Compute Bilateral filter response
            Bilt = H.* GaussMatr;

            transImgMatr(row,col) = sum(Bilt(:).* L(:)) / sum(Bilt(:));

        end
    end

    transImgMatr = uint8(transImgMatr);
end
```

3. Sharpening Code.

   → Laplacian Filter

```
function transImgMatr = laplacian_filtering(imgMatr,Lapl_filt)
    % --------------------------------
    % This function perform sharpening with laplacian filtering
    % return transformed image matrix
    % refer to https://bohr.wlu.ca/hfan/cp467/12/notes/cp467_12_lecture6_sharpening.pdf
    % pg13
    % --------------------------------

    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % zero padding on image matrix
    filtSize = 3;
    shift = double(floor(filtSize / 2));
    paddMatr = padarray(imgMatr,[shift shift],0);

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(hg,wd);

    %Lapl_filt = [1,1,1 ; 1,-8,1 ; 1,1,1];
    %if (filtType == 1)
        %Lapl_filt = [0,1,0 ; 1,-4,1 ; 0,1,0];
    %end


    filt = zeros(hg,wd);
    for row = 1 : hg
        for col = 1 : wd
            mul = paddMatr(row : row + (filtSize - 1),col : col + (filtSize - 1)).* Lapl_filt;
            filt(row,col) = sum(mul(:));
        end
    end

    transImgMatr = imgMatr - filt;
    transImgMatr = uint8(transImgMatr);
end
```

4. Color Space Conversion Code.

→ Conversion of RGB to HSI

```
function HSI = RGB_to_HSI(imgMatr)
    %-----------------------------------
    % This function perform the tranformation from RGB to HSI
    % return RGB
    % refer to https://www.imageeprocessing.com/2013/05/converting-rgb-image-to-hsi.html
    %-----------------------------------

    imgMatr = double(imgMatr);

    R = imgMatr(:,:,1) / 255;
    G = imgMatr(:,:,2) / 255;
    B = imgMatr(:,:,3) / 255;

    % Hue
    numerator = 1/2 * ((R - G) + (R - B));
    denominator = ((R - G).^2+((R - B).*(G - B))).^0.5;

    % To avoid divide by zero exception add a small number in the denominator
    H = acosd(numerator./(denominator + 0.000001));

    % If B>G then H= 360-Theta
    H(B > G) = 360 - H(B > G);

    % Normalize to the range [0 1]
    H = H / 360;

    % Saturation
    S = 1 - (3./ (sum(imgMatr,3) + 0.000001)).* min(imgMatr,[],3);

    % Intensity
    I = sum(imgMatr,3)./ 3;

    % HSI
    HSI = cat(3,H,S,I);

end
```

→ Conversion of HSI to RGB

```
function RGB = HSI_to_RGB(imgMatr)
    %----------------------------------
    % This function perform the tranformation from HSI to RGB
    % return RGB
    % refer to https://www.imageeprocessing.com/2013/06/convert-hsi-image-to-rgb-image.html
    %----------------------------------

    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    H = imgMatr(:,:,1) * 360;
    S = imgMatr(:,:,2);
    I = imgMatr(:,:,3);

    % Preallocate the R,G and B components
    R = zeros(hg,wd);
    G = zeros(hg,wd);
    B = zeros(hg,wd);

    % RG Sector(0<=H<120)
    % When H is in the above sector, the RGB components equations are
    B(H < 120) = I(H < 120).* (1 - S(H < 120));
    R(H < 120) = I(H < 120).* (1 + ((S(H < 120).* cosd(H(H < 120)))./ cosd(60 - H(H < 120))));
    G(H < 120) = 3.* I(H < 120) - (R(H < 120) + B(H < 120));
```

```
    % GB Sector(120<=H<240)
    % When H is in the above sector, the RGB components equations are
    % Subtract 120 from Hue
    H_adj = H - 120;
    R(H >= 120 & H < 240) = I(H >= 120 & H < 240).*(1 - S(H >= 120 & H < 240));
    G(H >= 120 & H < 240) = I(H >= 120 & H < 240).*(1 + ((S(H >= 120 & H < 240).*cosd(H_adj(H >= 120 & H <240)))./cosd(60 - H_adj(H >=120 & H < 240))));
    B(H >= 120 & H < 240) = 3.*I(H >= 120 & H < 240) - (R(H >= 120 & H < 240) + G(H >= 120 & H < 240));

    % BR Sector(240<=H<=360)
    % When H is in the above sector, the RGB components equations are
    % Subtract 240 from Hue
    H_adj = H - 240;
    G(H >= 240 & H <= 360) = I(H >= 240 & H <= 360).*(1 - S(H >= 240 & H <= 360));
    B(H >= 240 & H <= 360) = I(H >= 240 & H <= 360).*(1 + ((S(H >= 240 & H <= 360).*cosd(H_adj(H >= 240 & H <= 360)))./cosd(60 - H_adj(H >= 240 & H <= 360))));
    R(H >= 240 & H <= 360) = 3.*I(H >= 240 & H <= 360) - (G(H >= 240 & H <= 360) + B(H >= 240 & H <= 360));

    %Form RGB Image
    RGB = uint8(cat(3,R,G,B));

end
```

→ Conversion of RGB to HSV

```
function HSV = RGB_to_HSV(imgMatr)
    %----------------------------------
    % This function perform the tranformation from RGB to HSV
    % return HSV
    % refer to https://www.mathworks.com/matlabcentral/fileexchange/48864-rgb_to_hsv-m
    %----------------------------------

    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    imgMatr_R = imgMatr(:,:,1);
    imgMatr_G = imgMatr(:,:,2);
    imgMatr_B = imgMatr(:,:,3);

    H_matr = zeros(hg,wd);
    S_matr = zeros(hg,wd);
    V_matr = zeros(hg,wd);
```

```
    for row = 1 : hg
        for col = 1 : wd
            R = imgMatr_R(row,col);
            G = imgMatr_G(row,col);
            B = imgMatr_B(row,col);

            Max = max(R,max(G,B));
            Min = min(R,min(G,B));

            % Compute H
            H = 0;
            delta = Max - Min;
            if (delta == 0)
                H = 0; % undefined
            elseif (Max == R)
                H = 60 * (G - B) / delta;
                if (G < B) H = H + 360; end
            elseif (Max == G)
                H = 60 * (B - R) / delta + 120;
            elseif (Max == B)
                H = 60 * (R - G) / delta + 240;
            end

            % check if the value of H is negative
            if (H < 0)
                H = H + 360;
            end
            H = H / 360.0;


           % Compute S
           if (Max == 0)
               S = 0;
           else
            S = 1 - Min / Max;
           end
```

```
            % Compute V
            V = Max;

            H_matr(row,col) = H;
            S_matr(row,col) = S;
            V_matr(row,col) = V;

        end
    end

    HSV = cat(3,H_matr,S_matr,V_matr);

end
```

→ Conversion of HSV to RGB

```
function RGB = HSV_to_RGB(imgMatr)
    %-------------------------------------
    % This function perform the tranformation from HSV to RGB
    % return RGB
    % refer to https://www.rapidtables.com/convert/color/hsv-to-rgb.html
    %-------------------------------------

    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    imgMatr_H = imgMatr(:,:,1) * 360;
    imgMatr_S = imgMatr(:,:,2);
    imgMatr_V = imgMatr(:,:,3) / 255;

    R_matr = zeros(hg,wd);
    G_matr = zeros(hg,wd);
    B_matr = zeros(hg,wd);
```

```
    for row = 1 : hg
        for col = 1 : wd
            H = imgMatr_H(row,col);
            S = imgMatr_S(row,col);
            V = imgMatr_V(row,col);

            % transform to RGB
            C = V * S;
            X = C * (1 - abs( mod(H/60,2) - 1 ));
            m = V - C;

            R = 0; G = 0; B = 0;
            if (H >= 0 && H < 60)
                R = C; G = X; B = 0;
            elseif (H >= 60 && H < 120)
                R = X; G = C; B = 0;
            elseif (H >= 120 && H < 180)
                R = 0; G = C; B = X;
            elseif (H >= 180 && H < 240)
                R = 0; G = X; B = C;
            elseif (H >= 240 && H < 300)
                R = X; G = 0; B = C;
            elseif (H >= 300 && H < 306)
                R = C; G = 0; B = X;
            end


            R_matr(row,col) = (R + m) * 255;
            G_matr(row,col) = (G + m) * 255;
            B_matr(row,col) = (B + m) * 255;

        end
    end

    RGB = uint8(cat(3,R_matr,G_matr,B_matr));

end
```

5. Solutions of each image.

→ The solution to First Image (p1im1.png)

```
% ================================
% This file show the best method to enhance the image1
% ================================

clear all;
close all;
clc;
clf;

figIdx = 0;
% Image 1
imgMatr = imread("p1im1.png");
figIdx = figIdx + 1;
figure(figIdx);
imshow(imgMatr);
title("Original Image");


imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);
```

```
% check each distribution of channel (RGB)
histR = compute_histogram(imgMatr_R);
histG = compute_histogram(imgMatr_G);
histB = compute_histogram(imgMatr_B);


figIdx = figIdx + 1;
figure(figIdx);
subplot(3,1,1),bar(0:255,histR);
title("Distribution of RGB");
subplot(3,1,2),bar(0:255,histG);
subplot(3,1,3),bar(0:255,histB);


% check distribution of Intensity
histI = compute_histogram(imgMatr_I);

figIdx = figIdx + 1;
figure(figIdx);
bar(0:255,histI);
title("Distribution of Intensity");
```

```
% ========================================================================
% CONTRAST ADJUSTMENT
% ========================================================================
% use Piecewise Linear Stretching
[trans_I, transFunc] = piecewise_linear_streching(imgMatr_I,120,70,250,190);
transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = trans_I;
transImg = HSI_to_RGB(transImgHSI);

% Transform function
figIdx = figIdx + 1;
figure(figIdx);
plot(transFunc(1,:),transFunc(2,:));
title("Transform Function");

% check adjusted distribution of Intensity
histI = compute_histogram(trans_I);

figIdx = figIdx + 1;
figure(figIdx);
bar(0:255,histI);
title("Distribution of Intensity");
```

```
% Check the adjusted distribution of RGB
trans_R = transImg(:,:,1);
trans_G = transImg(:,:,2);
trans_B = transImg(:,:,3);

histR = compute_histogram(trans_R);
histG = compute_histogram(trans_G);
histB = compute_histogram(trans_B);

figIdx = figIdx + 1;
figure(figIdx);
subplot(3,1,1),bar(0:255,histR);
title("Distribution of RGB");
subplot(3,1,2),bar(0:255,histG);
subplot(3,1,3),bar(0:255,histB);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Piecewise Streching only on Intensity");


imgMatr = transImg;

imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);
```

```
% ========================================================================
% SMOOTHING DENOISING
% ========================================================================
% Adaptive local noise reduction filter
filtSz = 3;
trans_I = adaptive_filtering(imgMatr_I,filtSz,filtSz);
transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = trans_I;
transImg = HSI_to_RGB(transImgHSI);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Adaptive Filtering on Intensity");

imgMatr = transImg;

imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);
```

```
% ========================================================================
% COLOR CORRECTION
% ========================================================================
% Get the RGB and HSI components of the best result in last section
imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

% Convert RGB component into HSV
HSV = RGB_to_HSV(imgMatr);
changeIn_h = 0;
changeIn_s = 0;
changeIn_v = -40;

imgMatr_H = HSV(:,:,1) + changeIn_h;
imgMatr_S = HSV(:,:,2) + changeIn_s;
imgMatr_V = HSV(:,:,3) + changeIn_v;

transImgHSV = zeros(size(imgMatr));

transImgHSV(:,:,1) = imgMatr_H;
transImgHSV(:,:,2) = imgMatr_S;
transImgHSV(:,:,3) = imgMatr_V;
transImg = HSV_to_RGB(transImgHSV);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Result");
```

→ The solution to Second Image (p1im2.png)

```
% ================================
% This file show the best method to enhance the image2
% ================================

clear;
close all;
clc;
clf;

figIdx = 0;
% Image 2
imgMatr = imread("p1im2.png");
figIdx = figIdx + 1;
figure(figIdx);
imshow(imgMatr);
title("Original Image");


imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);
```

```
% check each distribution of channel (RGB)
histR = compute_histogram(imgMatr_R);
histG = compute_histogram(imgMatr_G);
histB = compute_histogram(imgMatr_B);


figIdx = figIdx + 1;
figure(figIdx);
subplot(3,1,1),bar(0:255,histR);
title("Distribution of RGB");
subplot(3,1,2),bar(0:255,histG);
subplot(3,1,3),bar(0:255,histB);


% check distribution of Intensity
histI = compute_histogram(imgMatr_I);

figIdx = figIdx + 1;
figure(figIdx);
bar(0:255,histI);
title("Distribution of Intensity");
```

```
% ========================================================================
% CONTRAST ADJUSTMENT
% ========================================================================
% power law transformation
% only on I (HSI)
trans_I = power_law_transform(imgMatr_I,1.2,0.9);
transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = trans_I;
transImg = HSI_to_RGB(transImgHSI);

% check adjusted distribution of Intensity
histI = compute_histogram(trans_I);

figIdx = figIdx + 1;
figure(figIdx);
bar(0:255,histI);
title("Distribution of Intensity");
```

```
% Check the adjusted distribution of RGB
trans_R = transImg(:,:,1);
trans_G = transImg(:,:,2);
trans_B = transImg(:,:,3);

histR = compute_histogram(trans_R);
histG = compute_histogram(trans_G);
histB = compute_histogram(trans_B);

figIdx = figIdx + 1;
figure(figIdx);
subplot(3,1,1),bar(0:255,histR);
title("Distribution of RGB");
subplot(3,1,2),bar(0:255,histG);
subplot(3,1,3),bar(0:255,histB);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Power Law Transformation only on Intensity");

imgMatr = transImg;

imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);
```

```
% ========================================================================
% COLOR CORRECTION
% ========================================================================
% Convert RGB component into HSV
HSV = RGB_to_HSV(imgMatr);
changeIn_h = -30/360;
changeIn_s = 0.2;
changeIn_v = -20;

imgMatr_H = HSV(:,:,1) + changeIn_h;
imgMatr_S = HSV(:,:,2) + changeIn_s;
imgMatr_V = HSV(:,:,3) + changeIn_v;

transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = imgMatr_V;
transImg = HSV_to_RGB(transImgHSI);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Color Correction on Intensity");

imgMatr = transImg;

imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);
```

```
% ========================================================================
% SMOOTHING AND NOISE REDUCTION
% ========================================================================
% bilateral filter
% only on I (HSI)
filtSz = 3;
sigma_d = 15;
sigma_g = 15;
trans_I = bilateral_filtering(imgMatr_I,filtSz,sigma_d,sigma_g);
transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = trans_I;
transImg = HSI_to_RGB(transImgHSI);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Bilateral Filtering on Intensity");

imgMatr = transImg;

imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);
```

```
% ========================================================================
% SHARPENING
% ========================================================================
% laplacian filter
% only on I (HSI)
%Lapl_filt = [0,1,0 ; 1,-4,1 ; 0,1,0];
Lapl_filt = [1,1,1 ; 1,-8,1 ; 1,1,1];
trans_I = laplacian_filtering(imgMatr_I,Lapl_filt);
transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = trans_I;
transImg = HSI_to_RGB(transImgHSI);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Result");

imgMatr = transImg;
```

→ The solution to Third Image (p1im3.png)

```
% ================================
% This file show the best method to enhance the image3
% ================================

clear;
close all;
clc;
clf;

figIdx = 0;
% Image 3
imgMatr = imread("p1im3.png");
figIdx = figIdx + 1;
figure(figIdx);
imshow(imgMatr);
title("Original Image");


imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);
```

```
% check each distribution of channel (RGB)
histR = compute_histogram(imgMatr_R);
histG = compute_histogram(imgMatr_G);
histB = compute_histogram(imgMatr_B);


figIdx = figIdx + 1;
figure(figIdx);
subplot(3,1,1),bar(0:255,histR);
title("Distribution of RGB");
subplot(3,1,2),bar(0:255,histG);
subplot(3,1,3),bar(0:255,histB);


% check distribution of Intensity
histI = compute_histogram(imgMatr_I);

figIdx = figIdx + 1;
figure(figIdx);
bar(0:255,histI);
title("Distribution of Intensity");
```

```
% ========================================================================
% CONTRAST ADJUSTMENT
% ========================================================================
% power law transformation
% on R,G,B separately
trans_R = power_law_transform(imgMatr_R,1.4,3.1);
trans_G = power_law_transform(imgMatr_G,1.4,2.7);
trans_B = power_law_transform(imgMatr_B,1.4,2.5);

% Check the adjusted distribution of RGB
histR = compute_histogram(trans_R);
histG = compute_histogram(trans_G);
histB = compute_histogram(trans_B);

figIdx = figIdx + 1;
figure(figIdx);
subplot(3,1,1),bar(0:255,histR);
title("Distribution of RGB");
subplot(3,1,2),bar(0:255,histG);
subplot(3,1,3),bar(0:255,histB);


transImg = cat(3,trans_R,trans_G,trans_B);
figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Power Law Transformation on RGB separately");

imgMatr = transImg;
```

```
% ========================================================================
% Sharpening
% ========================================================================
% Get the RGB and HSI components of the best result in last section
imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);

% 1. laplacian filter
% only on I (HSI)
Lapl_filt = [1,1,1 ; 1,-8,1 ; 1,1,1];
trans_I = laplacian_filtering(imgMatr_I,Lapl_filt);
transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = trans_I;
transImg = HSI_to_RGB(transImgHSI);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Laplacian Filtering on Intensity");
```

```
% ========================================================================
% Color Correction
% ========================================================================
% Get the RGB and HSI components of the best result in last section
imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

% Convert RGB component into HSV
HSV = RGB_to_HSV(imgMatr);
changeIn_h = -40/360;
changeIn_s = 0.10;
changeIn_v = -30;

imgMatr_H = HSV(:,:,1) + changeIn_h;
imgMatr_S = HSV(:,:,2) + changeIn_s;
imgMatr_V = HSV(:,:,3) + changeIn_v;

transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = imgMatr_V;
transImg = HSV_to_RGB(transImgHSI);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Result");
```

→ The solution to Forth Image (p1im4.png)

```
% ================================
% This file show the best method to enhance the image4
% ================================

clear;
close all;
clc;
clf;

figIdx = 0;
% Image 4
imgMatr = imread("p1im4.png");
figIdx = figIdx + 1;
figure(figIdx);
imshow(imgMatr);
title("Original Image");


imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);
```

```
% check each distribution of channel (RGB)
histR = compute_histogram(imgMatr_R);
histG = compute_histogram(imgMatr_G);
histB = compute_histogram(imgMatr_B);


figIdx = figIdx + 1;
figure(figIdx);
subplot(3,1,1),bar(0:255,histR);
title("Distribution of RGB");
subplot(3,1,2),bar(0:255,histG);
subplot(3,1,3),bar(0:255,histB);


% check distribution of Intensity
histI = compute_histogram(imgMatr_I);

figIdx = figIdx + 1;
figure(figIdx);
bar(0:255,histI);
title("Distribution of Intensity");
```

```
% ========================================================================
% SHARPENING
% ========================================================================
% Get the RGB and HSI components of the best result in last section
imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);

% 1. laplacian filter
% b. only on I (HSI)
Lapl_filt = [0,1,0 ; 1,-4,1 ; 0,1,0];
trans_I = laplacian_filtering(imgMatr_I,Lapl_filt);
transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = trans_I;
transImg = HSI_to_RGB(transImgHSI);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Laplacian Filtering on Intensity");

imgMatr = transImg;
```

```
% ========================================================================
% COLOR CORRECTION
% ========================================================================
% Get the RGB and HSI components of the best result in last section
imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

% Convert RGB component into HSV
HSV = RGB_to_HSV(imgMatr);
changeIn_h = -5/360;
changeIn_s = 0.1;
changeIn_v = -20;

imgMatr_H = HSV(:,:,1) + changeIn_h;
imgMatr_S = HSV(:,:,2) + changeIn_s;
imgMatr_V = HSV(:,:,3) + changeIn_v;

transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = imgMatr_V;
transImg = HSV_to_RGB(transImgHSI);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Result");

imgMatr = transImg;
```

→ The solution to Fifth Image (p1im5.png)

```
% ================================
% This file show the best method to enhance the image5
% ================================

clear;
close all;
clc;
clf;

figIdx = 0;
% Image 5
imgMatr = imread("p1im5.png");
figIdx = figIdx + 1;
figure(figIdx);
imshow(imgMatr);
title("Original Image");


imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);
```

```
% check each distribution of channel (RGB)
histR = compute_histogram(imgMatr_R);
histG = compute_histogram(imgMatr_G);
histB = compute_histogram(imgMatr_B);


figIdx = figIdx + 1;
figure(figIdx);
subplot(3,1,1),bar(0:255,histR);
title("Distribution of RGB");
subplot(3,1,2),bar(0:255,histG);
subplot(3,1,3),bar(0:255,histB);


% check distribution of Intensity
histI = compute_histogram(imgMatr_I);

figIdx = figIdx + 1;
figure(figIdx);
bar(0:255,histI);
title("Distribution of Intensity");
```

```
% ========================================================================
% CONTRAST ADJUSTMENT
% ========================================================================
% piecewise linear streching
% only on I (HSI)
[trans_I, transFunc] = piecewise_linear_streching(imgMatr_I,50,0,180,255);
transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = trans_I;
transImg = HSI_to_RGB(transImgHSI);

% Transform function
figIdx = figIdx + 1;
figure(figIdx);
plot(transFunc(1,:),transFunc(2,:));
title("Transform Function");

% check adjusted distribution of Intensity
histI = compute_histogram(trans_I);

figIdx = figIdx + 1;
figure(figIdx);
bar(0:255,histI);
title("Distribution of Intensity");
```

```
% Check the adjusted distribution of RGB
trans_R = transImg(:,:,1);
trans_G = transImg(:,:,2);
trans_B = transImg(:,:,3);

histR = compute_histogram(trans_R);
histG = compute_histogram(trans_G);
histB = compute_histogram(trans_B);

figIdx = figIdx + 1;
figure(figIdx);
subplot(3,1,1),bar(0:255,histR);
title("Distribution of RGB");
subplot(3,1,2),bar(0:255,histG);
subplot(3,1,3),bar(0:255,histB);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Piecewise Streching only on Intensity");

imgMatr = transImg;
```

```
% ========================================================================
% Color Correction
% ========================================================================
% Get the RGB and HSI components of the best result in last section
imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

% Convert RGB component into HSV
HSV = RGB_to_HSV(imgMatr);
changeIn_h = -5/360;
changeIn_s = 0.09;
changeIn_v = -25;

imgMatr_H = HSV(:,:,1) + changeIn_h;
imgMatr_S = HSV(:,:,2) + changeIn_s;
imgMatr_V = HSV(:,:,3) + changeIn_v;

transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = imgMatr_V;
transImg = HSV_to_RGB(transImgHSI);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Color Correction on Intensity");

imgMatr = transImg;
```

```
% ========================================================================
% Sharpening
% ========================================================================
% Get the RGB and HSI components of the best result in last section
imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);

% 1. laplacian filter
% b. only on I (HSI)
Lapl_filt = [0,1,0 ; 1,-4,1 ; 0,1,0];
trans_I = laplacian_filtering(imgMatr_I,Lapl_filt);
transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = trans_I;
transImg = HSI_to_RGB(transImgHSI);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Result");

imgMatr = transImg;
```

→ The solution to Sixth Image (p1im6.png)

```
% ================================
% This file show the best method to enhance the image6
% ================================

clear;
close all;
clc;
clf;

figIdx = 0;
% Image 6
imgMatr = imread("p1im6.png");
figIdx = figIdx + 1;
figure(figIdx);
imshow(imgMatr);
title("Original Image");


imgMatr_R = imgMatr(:,:,1);
imgMatr_G = imgMatr(:,:,2);
imgMatr_B = imgMatr(:,:,3);

HSI = RGB_to_HSI(imgMatr);
imgMatr_H = HSI(:,:,1);
imgMatr_S = HSI(:,:,2);
imgMatr_I = HSI(:,:,3);
```

```
% ========================================================================
% SMOOTHING AND NOISE REDUCTION
% ========================================================================
filtSz = 5;
% Adaptive local noise reduction filter
% only on I (HSI)
trans_I = adaptive_filtering(imgMatr_I,filtSz,filtSz);
transImgHSI = zeros(size(imgMatr));

transImgHSI(:,:,1) = imgMatr_H;
transImgHSI(:,:,2) = imgMatr_S;
transImgHSI(:,:,3) = trans_I;
transImg = HSI_to_RGB(transImgHSI);

figIdx = figIdx + 1;
figure(figIdx);
imshow(transImg);
title("Adaptive Filtering on Intensity");

imgMatr = transImg;
```

Section 4: References

1. https://www.ques10.com/p/3690/determine-the-output-image-using-power-law-transfo/?
2. https://www.imageeprocessing.com/2011/12/adaptive-filtering-local-noise-filter.html
3. https://www.mathworks.com/matlabcentral/answers/247326-help-with-adaptive-median-filter
4. https://github.com/mortezamg63/Adaptive-Median-Filter/blob/master/Adaptive_Median_filter.m
5. https://www.massey.ac.nz/~mjjohnso/notes/59731/presentations/Adaptive%20Median%20Filtering.doc
6. https://bohr.wlu.ca/hfan/cp467/12/notes/cp467_12_lecture6_sharpening.pdf
7. https://www.mathworks.com/matlabcentral/fileexchange/12191-bilateral-filtering
8. https://www.imageeprocessing.com/2013/05/converting-rgb-image-to-hsi.html
9. https://www.imageeprocessing.com/2013/06/convert-hsi-image-to-rgb-image.html
10. https://www.mathworks.com/matlabcentral/fileexchange/48864-rgb_to_hsv-m
11. https://www.rapidtables.com/convert/color/hsv-to-rgb.html
12. https://stackoverflow.com/questions/25648878/median-filter-for-color-images?fbclid=IwAR1p2TO2N4m2saARPc48G7jss_IJQH18CQvf-Jq1KdyJJtOMJprlEZlqOcE
13. https://www.mathworks.com/matlabcentral/answers/358032-how-to-apply-median-filter-to-a-color-image-with-out-using-rgb2gray-program-please?fbclid=IwAR029EoUPcFPuXssEuu8R4baBqtloYXMHfv43z3M_ZIJHw41fs54Fe3Y5d0
14. https://medium.com/@gary1346aa/%E5%B0%8E%E5%90%91%E6%BF%BE%E6%B3%A2%E7%9A%84%E5%8E%9F%E7%90%86%E4%BB%A5%E5%8F%8A%E5%85%B6%E6%87%89%E7%94%A8-78fdf562e749
15. https://hypjudy.github.io/2017/03/19/dip-histogram-equalization/?fbclid=IwAR2X2g569Dub9RWoTRqMs9X7pdIRmqLOE18uTA-wynKmnCv_eBxDmnTXq_k
16. https://dsp.stackexchange.com/questions/13419/why-do-we-always-use-square-kernels-for-filters
17. https://stackoverflow.com/questions/49003346/why-convolutional-nn-kernel-size-is-often-selected-as-a-square-matrix
18. https://www.researchgate.net/post/What_does_filter_size_refer_to_in_image_processing
19. https://photography.tutsplus.com/tutorials/unlikely-friends-grain-noise-and-sharpening--cms-26634