Computational Photography

Programming Assignment 1

Basics of Imaging

Sagnik Ghosh

February 2022

1 Demosaicing

1.1 Part 1

For Demosaicing, the function **implemented_demosaic** can be used, as given below. This function extracts each channel (with missing values) and subsequently performs interpolation to fill in the missing values using the **griddata** function. The **method** argument can be 'linear' or 'cubic'.

```
function RGB = implemented_demosaic(raw_img, bayer, method)
    red_channel = interpolation(raw_img, bayer, 1, method);
    green_channel = interpolation(raw_img, bayer, 2, method);
    blue_channel = interpolation(raw_img, bayer, 3, method);

    RGB = cat(3, red_channel, green_channel, blue_channel);
end

function channel = interpolation(raw_img, bayer, channel_num, method)
    channel = raw_img(bayer==channel_num);
    [x, y] = meshgrid(1:size(raw_img, 2), 1:size(raw_img, 1));

    xv = x(bayer == channel_num);
    yv = y(bayer == channel_num);

    channel = uint8(griddata(xv, yv, double(channel), x, y, method));
    channel = reshape(channel, size(raw_img));
end
```

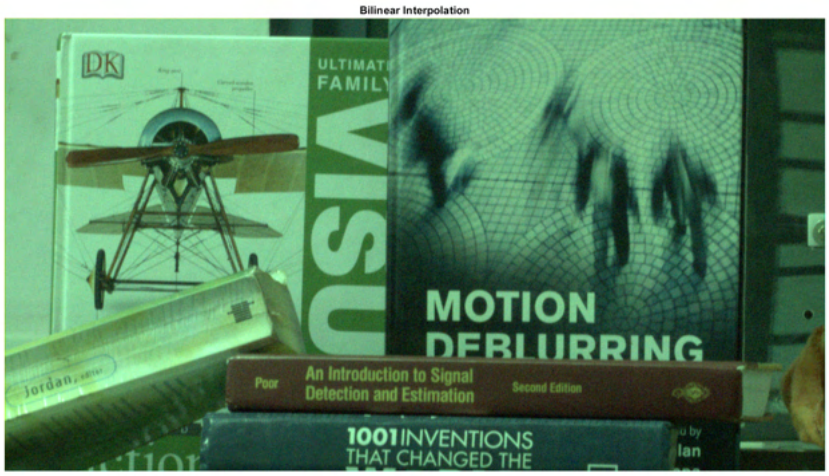


Figure 1: Bilinear Interpolation

1.2 Part 2

The **implemented_demosaic** has been used to perform demosaicing with bicubic interpolation by setting the **method** argument to 'cubic'. While bilinear interpolation fits a piece-wise linear function for interpolation and uses information from 4 adjacent pixels whereas bicubic interpolation fits a piece-wise cubic function and uses formation from 16 adjacent pixels. Hence, images obtained from bicubic interpolation are smoother. Comparing the obtained images, both the images look nearly identical. This could be because of the relatively small interpolation factor.

1.3 Part 3

The image can be demosaiced using the MATLAB built-in function **demosaic** as follows:

```
RGB = demosaic(RawImage1, 'rggb');
```

According to the documentation of the demosaic function, it uses an interpolation method called 'gradient coorected linear interpolation', which uses pixel values from other color channels for interpolation. Comparing the images, there is no apparent difference.

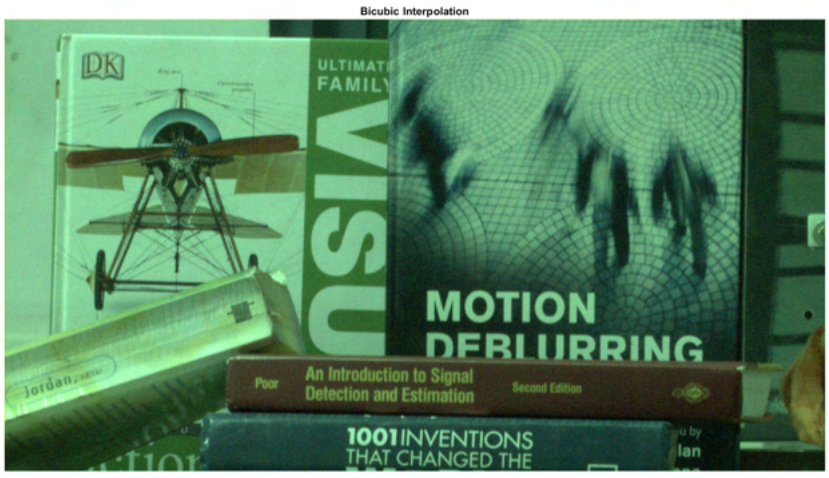


Figure 2: Bicubic Interpolation

Figure 3: MATLAB Demosaic Function

1.4 Part 4

The underlying assumption made while interpolating the missing values is that the scene is piece-wise smooth. This is because we are fitting a piece-wise linear curve for bilinear interpolation and a piece-wise cubic curve for bicubic interpolation.

This assumption can be violated when the scene consists of very high spatial frequency information. In that case, the high frequency information is lost or it can lead to unpleasant artifacts due to aliasing. As an example, sharp edges can get smoothed out after interpolation. This effect is not apparent in our case because of the small interpolation factor, but it becomes more prominent for higher interpolation factors.

1.5 Part 5

'kodim19.mat' can be demosaiced using the function **implemented_demosaic** and setting the **method** to 'cubic'.

1.6 Part 6

The chrominance channels of the given RGB image can be median filtered by using the following function which uses the MATLAB built-in **metfilt2** function.

```
function RGB_out = median_filter(RGB, filter_size)
    YCBCR = rgb2ycbcr(RGB);
    Y = YCBCR(:,:,1);
    CB = YCBCR(:,:,2);
    CR = YCBCR(:,:,3);
    CB = medfilt2(CB, [filter_size filter_size], 'symmetric');
    CR = medfilt2(CR, [filter_size filter_size], 'symmetric');
    YCBCR = cat(3, Y, CB, CR);
    RGB_out = ycbcr2rgb(YCBCR);
end
```

1.7 Part 7

In the comparison of the demosaiced, median filtered and the reference image, we can notice rainbow patterns in the demosaiced image on the fence which has a high spatial frequency. This is a well known problem called color moire. This happens when the spatial frequency exceeds the sensor resolution and during interpolation, it is interpreted as color information.

This effect manifests as outliers in the chrominance channels of the image. Consequently, this effect can be mitigated by removing the outliers in the chrominance channels by using a median filter of appropriate size. As we are not filtering the illuminance channel, the fine textures are preserved. In this experiment, a filter size of 15 was used. This was arrived at by noticing that a

smaller filter size was not very effective and a larger filter size reduced the color contrast.

Figure 4: Comparison between the demosaiced, median filtered and the reference images.

2 White Balancing and Tone Mapping

2.1 Part 1

Assuming that the average color of the scene is gray, we need to transform the image so that the mean of each channel is the same. Hence, the corresponding scale factors for each channel are:

$$S_R = \frac{\sum RGB}{3\sum R_{channel}}$$
$$S_G = \frac{\sum RGB}{3\sum G_{channel}}$$
$$S_B = \frac{\sum RGB}{3\sum B_{channel}}$$

This can be implemented as follows:

```
function RGB_white_balanced_gray = white_balance_gray(RGB)
    total_sum = sum(RGB, 'all');
    sum_R = sum(RGB(:,:,1), 'all');
    sum_G = sum(RGB(:,:,2), 'all');
    sum_B = sum(RGB(:,:,3), 'all');

    R_channel = RGB(:,:,1)*(total_sum/(3*sum_R));
    G_channel = RGB(:,:,2)*(total_sum/(3*sum_G));
    B_channel = RGB(:,:,3)*(total_sum/(3*sum_B));
```

```
    RGB_white_balanced_gray = cat(3, R_channel, G_channel, B_channel);
end
```

2.2 Part 2

The scale factors for each channel can be computed by mapping the specular highlight to $(1, 1, 1)$.

$$S_R = \frac{255}{R_{XY}}$$
$$S_G = \frac{255}{G_{XY}}$$
$$S_B = \frac{255}{B_{XY}}$$

2.3 Part 3

The scale factors for each channel can be computed by mapping the neutral pixel to a ratio of $1 : 1 : 1$.

$$S_R = \frac{R_{XY}+G_{XY}+B_{XY}}{3R_{XY}}$$
$$S_G = \frac{R_{XY}+G_{XY}+B_{XY}}{3G_{XY}}$$
$$S_B = \frac{R_{XY}+G_{XY}+B_{XY}}{3B_{XY}}$$

2.4 Part 4

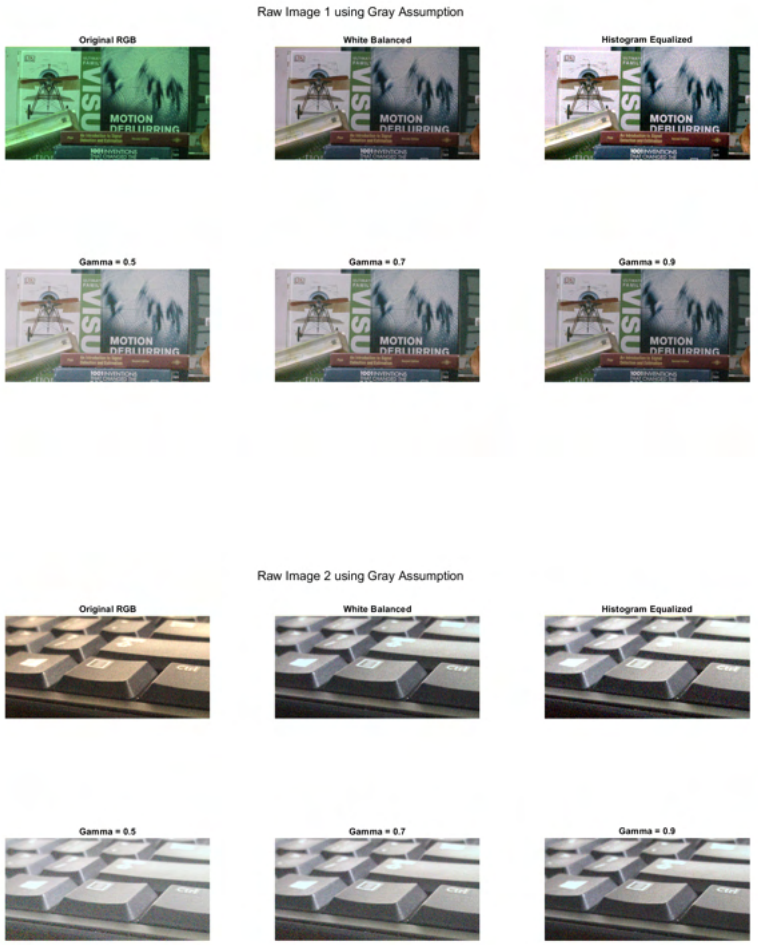
Histogram Equalization can be performed as follows:

```
RGB_histeq = histeq(RGB);
```

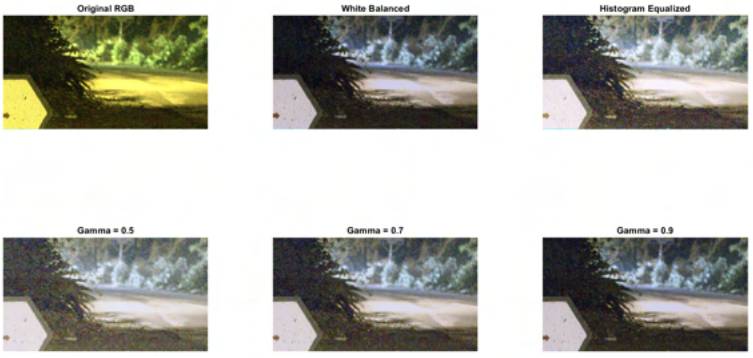
Gamma Correction can be performed as follows:

```
RGB_gamma = uint8(255*imadjust(double(RGB)/255,[0 1],[0 1],0.5));
```

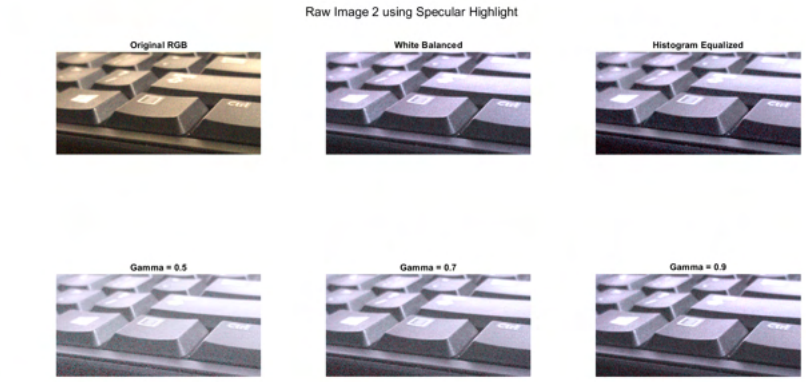
2.5 Results

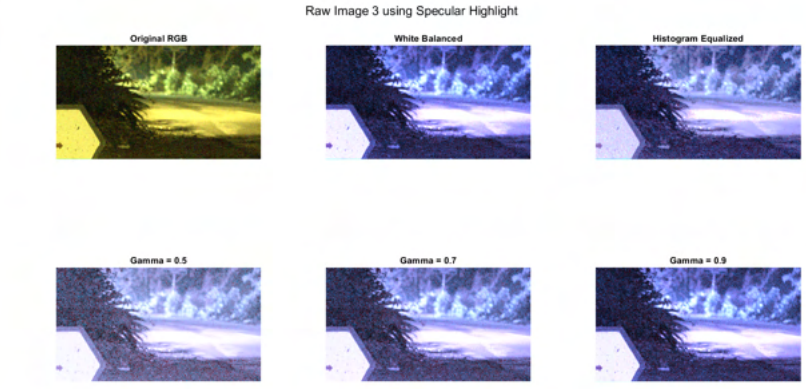

Raw Image 1 using Gray Assumption
Original RGB
White Balanced
Histogram Equalized
Gamma = 0.5
Gamma = 0.7
Gamma = 0.9
Raw Image 2 using Gray Assumption
Original RGB
White Balanced
Histogram Equalized
Gamma = 0.5
Gamma = 0.7
Gamma = 0.9

Raw Image 3 using Gray Assumption

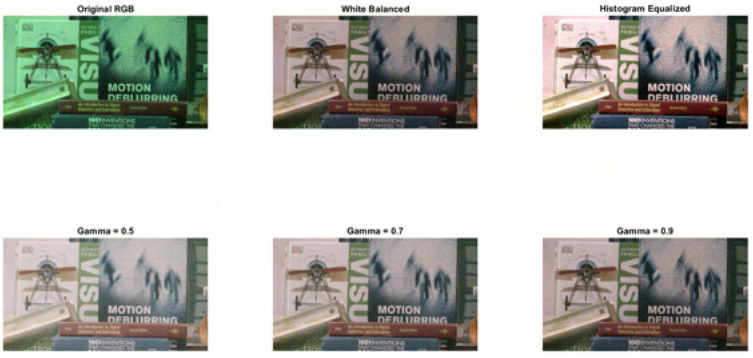


Raw Image 1 using Specular Highlight

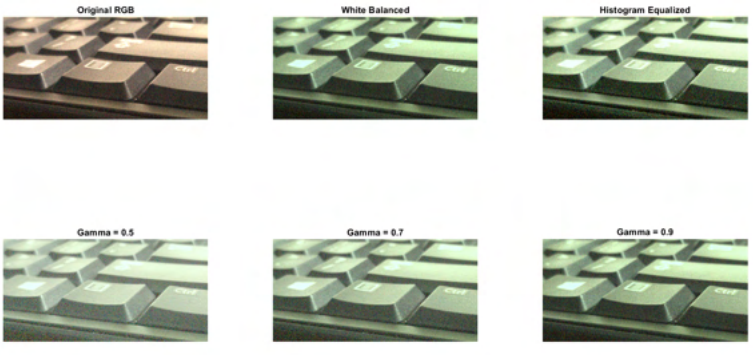

Raw Image 2 using Specular Highlight
Original RGB
White Balanced
Histogram Equalized
Gamma = 0.5
Gamma = 0.7
Gamma = 0.9

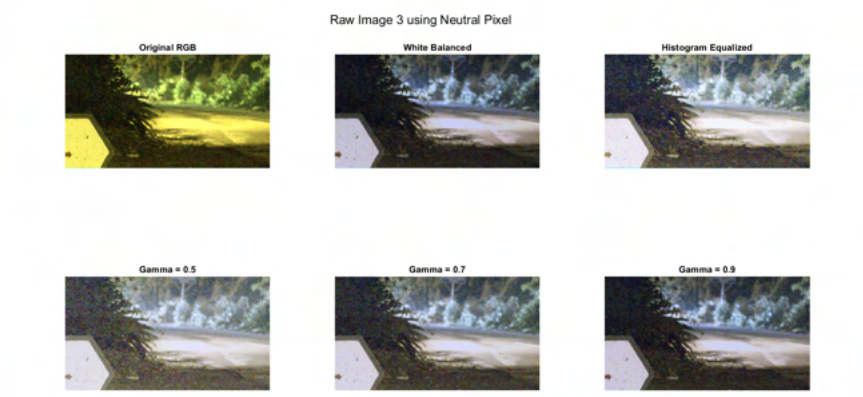

Raw Image 3 using Specular Highlight
Original RGB
White Balanced
Histogram Equalized
Gamma = 0.5
Gamma = 0.7
Gamma = 0.9

Raw Image 1 using Neutral Pixel



Raw Image 2 using Neutral Pixel




Raw Image 3 using Neutral Pixel
Original RGB
White Balanced
Histogram Equalized
Gamma = 0.5
Gamma = 0.7
Gamma = 0.9

3 Image Denoising

3.1 Results

Note that the regions for noise estimation for RawImage1 and RawImage3 were chosen to be the darkest 60×60 patch in each image.

Raw Image 1

Original

Denoised

Raw Image 2

Original

Denoised

Raw Image 3

3.2 Questions

3.2.1 Part 1

As the Gaussian function has infinite support, it needs to be truncated appropriately in order to implement it. While the Gaussian function has infinite support, the function drops quickly to values very close to zero as it moves away from the centre. Given a Gaussian function with standard deviation σ, the function becomes very close to zero after around 3σ on each side. Hence, we can truncate it to 3σ on each side. Theoretically, we can choose to truncate it beyond 3σ, but it would be computationally expensive and it would be very similar to 3σ.

3.2.2 Part 2

For a given value of σ_n, σ_r should be chosen of the same order of magnitude as σ_n. This is because if σ_r is chosen to be much smaller that σ_n, then the weights will be too sensitive to the difference in intensities of neighboring noisy pixels and it will assign those neighboring weights as zero and most of the weight will be concentrated at the centre pixel, and the convolution will not remove the noise.
On the other hand, choosing a σ_r much greater than σ_n will lead to the weights not being sensitive to difference in intensities for sharp edges, and the weights assigned on the other side of the sharp edge will be significant. This will lead to blurring of sharp edges.

4 Depth of Field

4.1 Part 1

We know,

$$\frac{1}{u} + \frac{1}{v} = \frac{1}{f} \quad (1)$$

We also know,

$$\frac{1}{u-du_1} + \frac{1}{v+DOF_+} = \frac{1}{f} \quad (2)$$
$$\frac{1}{u+du_2} + \frac{1}{v-DOF_-} = \frac{1}{f} \quad (3)$$

Using similar triangles,

$$u - du_1 = \frac{D}{W+D} \cdot u \quad (4)$$
$$u + du_2 = \frac{D}{D-W} \cdot u \quad (5)$$

We know $f = 16mm$.
We can obtain u from (1). We can substitute (4) in (2) to get DOF_+. We can substitute (5) in (3) to get DOF_-.
The notation used is the same as the notation used in the class slides. As it is given that the circle of confusion is 3 pixels, the effective pixel width $W = 3W = 10.35\mu m$.
The above method can be used to solve all 3 parts by plugging in the values.

4.1.1 Part (i)

We are given that $v = 3000mm$.
We can calculate the aperture diameter $D = \frac{f}{2.2} = \frac{16}{2.2}mm$.
Using the method as described above, we get $DOF_+ = 1083.93mm$.
Similarly, we get $DOF_- = 629.23mm$.
Hence, Depth of Field, $DOF = DOF_+ + DOF_- = 1083.93 + 629.23 = 1713.16mm$.
Boundary Distances $= 3000mm - DOF_-, 3000mm + DOF_+ = 2370.77mm, 4083.93mm$.

4.1.2 Part (ii)

We are given that $v = 3000mm$.
We can calculate the aperture diameter $D = \frac{f}{4} = \frac{16}{4}mm$.
Using the method as described above, we get $DOF_+ = 2797.87mm$.
Similarly, we get $DOF_- = 976.48mm$.
Hence, Depth of Field, $DOF = DOF_+ + DOF_- = 2797.87 + 976.48 = 3774.35mm$.
Boundary Distances $= 3000mm - DOF_-, 3000mm + DOF_+ = 2023.52mm, 5797.87mm$.

4.1.3 Part (iii)

We are given that $v = 1000mm$.
We can calculate the aperture diameter $D = \frac{f}{4} = \frac{16}{4}mm$.
Using the method as described above, we get $DOF_+ = 189.25mm$.

Similarly, we get $DOF_- = 137.28mm$.
Hence, Depth of Field, $DOF = DOF_+ + DOF_- = 189.25 + 137.28 = 326.53mm$.
Boundary Distances $= 1000mm - DOF_-, 1000mm + DOF_+ = 862.72mm, 1189.25mm$.

4.2 Part 2

The main observations we can make by comparing the results are as follows,

- As the f-number increases from 2.2 in (i) to 4 in (ii), the depth of field increases. This observation is consistent with our prior knowledge that depth of field increases with a decrease in aperture as the f-number is inversely proportional to the aperture diameter.
- Comparing the depth of field in (ii) and (iii), we can observe that keeping f-number constant, depth of field decreases with a decrease in object distance.
- We can also observe that $DOF_+ > DOF_-$ for all cases. This can be attributed to the non-linearity of the lens equation.

5 Note

This PDF has been compressed due to size restrictions on Moodle. Please run the code supplied to get the original images which have been labelled appropriately.