Agam Sidhu
3027948957
agamsidh@usc.edu
EE 569 Homework #1
February 1, 2026

Problem 1: Image Demosaicing and Histogram Manipulation

I. Motivation

Problem 1(a): Bilinear Demosicing

The goal of this problem is to explore bilinear demosaicing by reconstructing the image using a Bayer color filter array. The main reason we aim to do such a thing is because single sensors often struggle to capture full color information. Thus to rectify this, we apply image demosaicing to convert these images into full color images. Now for this problem, we are applying a simple demosaicing technique known as bilinear demosaicing to explore the tradeoff between adding color versus overall image quality.

Problem 1(b): Histogram Manipulation

For this problem we are investigating various histogram manipulation techniques to understand the contrast in enhancing the contrast of an image. In our day to day, there are many images that suffer from poor contrast which overall degrades images due to excessive illumination and poor hardware capturing these images. One example we as USC students have seen are the incident report images the school sends to students to help identify suspects. More times than not, the image description the school sends from DPS to students is a very blurry and unclear image. By adjusting the contrast, students may be better at spotting suspects. For this problem specifically, we will be looking at a Transfer based function and a bucket filling approach. With these approaches we can explore how different construction strategies can influence the final resulting contrast.

Problem 1(c): Contrast Limited Adaptive Histogram Equalization

The mission of this problem is to explore how Contrast Limited Adaptive Histogram Equalization (CLAHE) can further enhance images for the approach it holds. CLAHE specifically is used in conditions where images captured are foggy or hazy. This is done by the CLAHE dividing the image into subsections known as tiles and then applying the histogram equalization on these tiles. Thus for this problem, we are given a foggy image. We wish to see how the CLAHE approach differs from the two techniques discussed in question 1b.

II. Approach

Problem 1(a): Bilinear Demosaicing

For this problem, we use the bilinear Interpolation formula to estimate the missing color values of nearby pixels. We will set it so R(i,j), G(i,j), and B(i,j) represent the values of the color channel at the location (i,j) of the pixel.

Due to the information provided at piazza, we will be using RGGB to get our final image. Thus, we will use Green Channel Interpolation to evaluate our green value. This is denoted by:

$$\hat{G}(i,j) = \frac{1}{4}\left[G(i-1,j) + G(i+1,j) + G(i,j-1) + G(i,j+1)\right]$$

Where:

- The green value is estimated by using the values of its 4 neighbors

Red and Blue Interpolation is used when our red or blue values are missing. If the blue or red pixel location is missing, then our missing component is calculated by:

$$\hat{B}(i,j) = \frac{1}{4}\left[B(i-1,j-1) + B(i-1,j+1) + B(i+1,j-1) + B(i+1,j+1)\right]$$

Where:

- The missing value is calculated by estimating the 4 diagonal neighbors

At green pixel location where they are missing red or blue value, we apply this formula:

$$\hat{R}(i,j) = \frac{1}{2}\left[R(i,j-1) + R(i,j+1)\right] \quad \text{or} \quad \hat{R}(i,j) = \frac{1}{2}\left[R(i-1,j) + R(i+1,j)\right]$$

Where:

- The values are computed by using the two neighbors

Now our procedure for this problem is to first read the image and store it as a 2d array. Then, based on the Bayer pattern position, we identify the pixel type. After that we preserve the known color values at each location. After this step, we estimate the missing color components by applying the above formulas to get the values from neighboring pixels of the same color. We also introduce a technique known as clamping to ensure all neighbours stay within the bounds of the image. Finally, we combine this information and get our final RGB image.

Problem 1(b): Histogram Manipulation

For this problem, we set the histogram of the image to be:

$$h(k) = \text{number of pixels with intensity } k$$

When we normalize the histogram to get our probability mass function, we get our formula to be:

$$p(k) = \frac{h(k)}{N}$$

Where N is the total number of pixels in our image.
This is then used to get out cumulative distribution function (CDF):

$$\text{CDF}(k) = \sum_{i=0}^{k} p(i)$$

Method A: Transfer-Function-Based Histogram Equalization

In our method A, we get our transfer function from the CDF above and then apply it to each pixel. This is shown by:

$$T(k) = (L-1) \cdot \text{CDF}(k)$$

We later use this information to get the anxiety of the output pixel given by:

$$Y(i,j) = T\big(X(i,j)\big)$$

Method B: Bucket-Filling

The strategy used for this problem was to redistribute the intensities of the pixels so we maintain that our output intensity level has the same number of pixels. Our bucket is N/L, where N is the total number of pixels and L is our intensity level

Implementation

The procedure here is to read the image and store the values of the pixel in a 1D array. Once we do that, we will compute the histogram for the intensity levels. After that, based on the histogram we will compute the CDF and apply both our method A and method B. Once that is complete, we clamp the output to make sure we remain in our range of 0-255 and then we plot the histograms.

Problem 1(c): Contrast Limited Adaptive Histogram Equalization (CLAHE)

For this problem, we are tasked with enhancing the image using a collection of converting from RGB to YUV, applying the contrast to the Y channel, and then converting back to RGB. In order to do this, we use the conversion in the appendix to get out components of YUV as:

$$Y = 0.257R + 0.504G + 0.098B + 16$$

$$U = -0.148R - 0.291G + 0.439B + 128$$

$$V = 0.439R - 0.368G - 0.071B + 128$$

Where:

- Y is the brightness
- U and V are the color

Then we apply a contrast enhancement. First we do method A and B so we refer to the earlier equations to get our T(k) and n/l pixel values.

With the CLAHE however, we follow the tile section approach we mentioned earlier where we divide the image into a tiles of size tilesX * tilesY. Now after this we compute our histogram, set a clip limit to be C so we don't overamplify our data. Then we compute our CDF and create a transfer function based on this location information.

After we apply these methods, we convert the YUV back to RGB:

$$R = 1.164(Y' - 16) + 1.596(V - 128)$$

$$G = 1.164(Y' - 16) - 0.813(V - 128) - 0.391(U - 128)$$

$$B = 1.164(Y' - 16) + 2.018(U - 128)$$

And finally we read the image.

III. Results

Problem 1(a): Bilinear Demosaicing Results:

Figure 1: sailboats_cfa.raw

Figure 2: sailboats_bilinear

Figure 3: sailboats advanced

Problem 1(b): Histogram Equalization Results

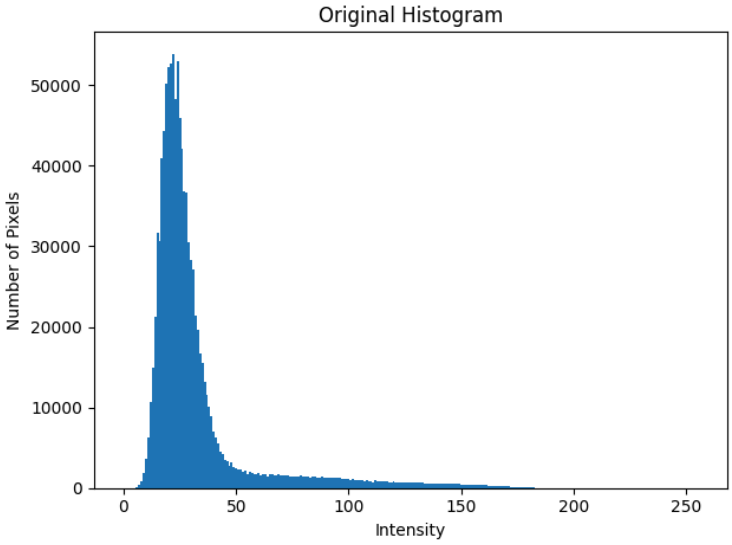


Figure 4.a: Original Histogram

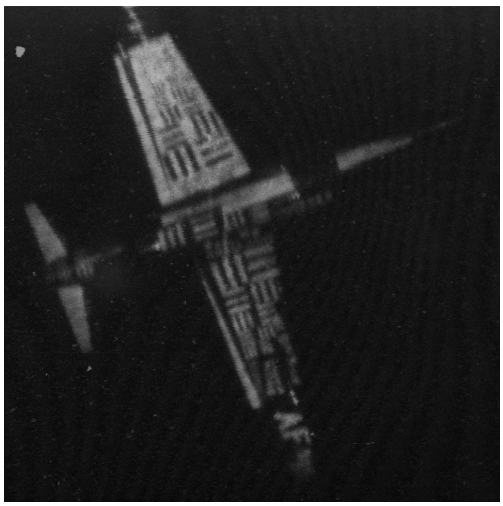
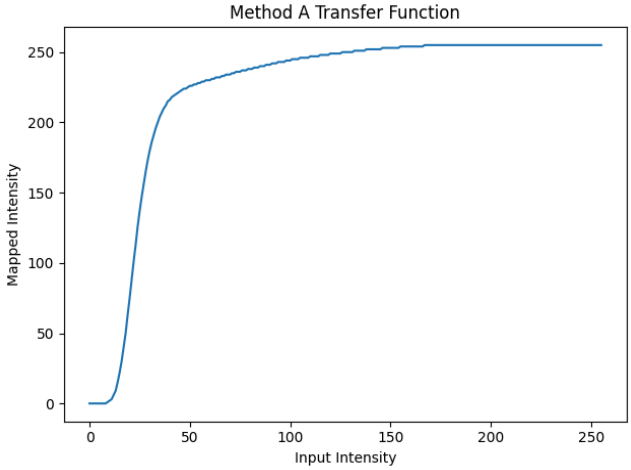


Figure 4.b Original Image



Method A:

Figure 5: Method A Transfer Function

Figure 6: Method A Image

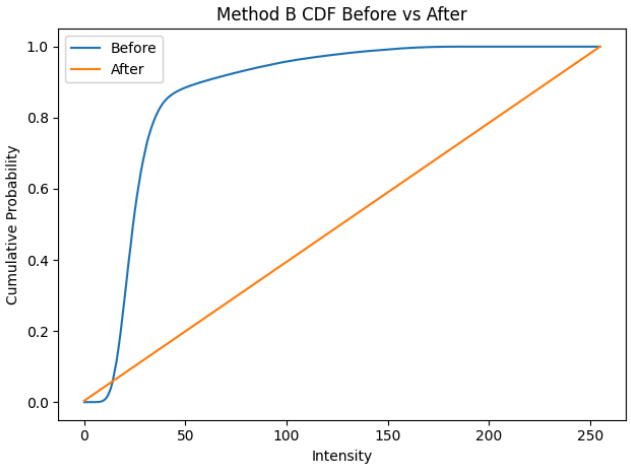


Figure 7: CDF Before and After

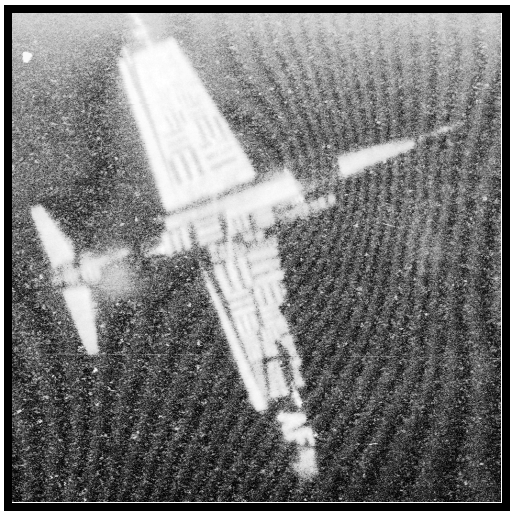


Figure 8: Method B Image

Method B:

Problem 1(c): CLAHE Results

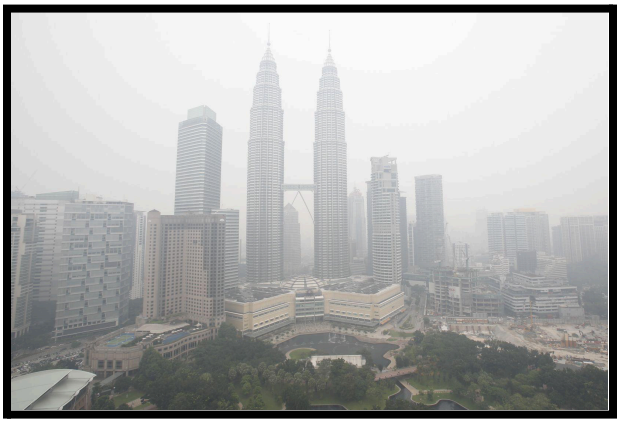


Figure 9: Original Foggy Image

Figure 10: Method A Tower

Figure 11: Method B Tower

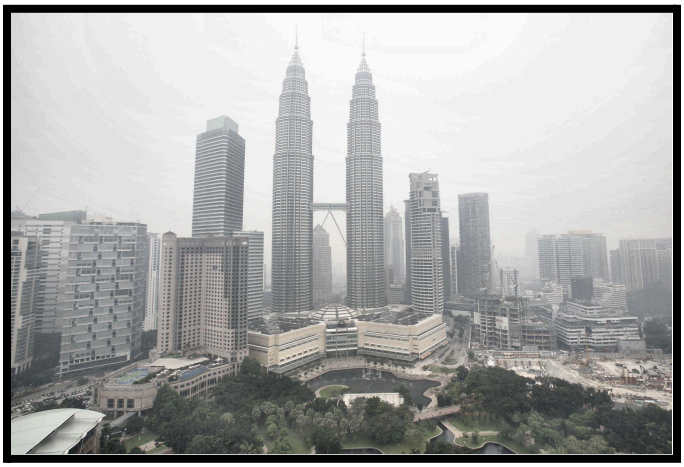


Figure 12: CLAHE 1.5 + 8x8

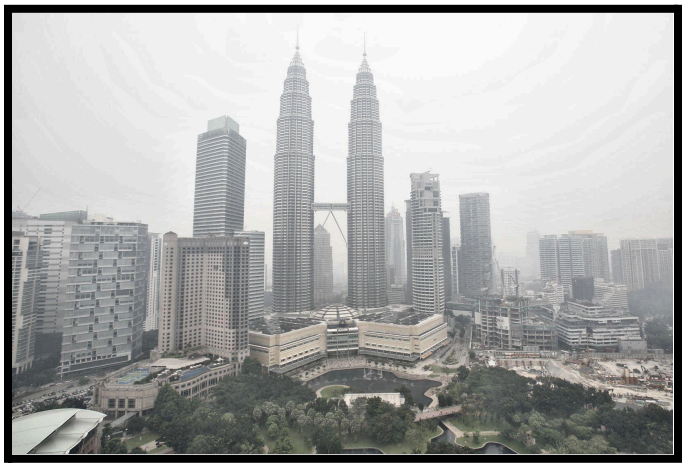


Figure 13: CLAHE 1.5 + 16x16

Figure 14: CLAHE 3 + 8x8

Figure 15: CLAHE 3 + 16x16

Discussion

Problem 1(a): Bilinear Demosaicing:

From our Figure 2, we can see that our bilinear demosaic result does a good job converting the color image from a grayscale image. Details, such as the sky, water, the various sail colors, and even the different color jackets worn by those riding the boat can be seen very clearly. Furthermore, it is easier to see details such as ropes in this new colored version of the sailboat.

Despite how detailed the bilinear approach is in comparison to the original grayscale image, when being compared to a more advanced algorithm, there are a couple of small details that are overlooked. For instance the quality of small details is a bit blurred. For instance the

details on the rope are slightly off when in comparison to the advanced image. Furthermore, some colors spill slightly over. Another clear example of this is the color of the rope in the bilinear color scheme and the color of the rope on the advanced algorithm. The advanced version is easy to spot and maintains the color vs the nonlinear approach which has other colors slightly bleeding into it.

Some suggestions to make a more improved performance for the bilinear approach is to add edge interpolation in the sense that we try to direct toward neighbors with similar intensity and go through neighbors in that direction because they would reflect the same directional shape that the image is trying to show. You could also add custom color correlation so instead of following the traditional RGB format, you can get a more fluid color schema that smoothly transitions the various color artifacts.

Problem 1(b): Histogram Equalization:

For the figure (6 & 8), we can see that both approaches drastically improve the image. Image 4.b (or the original image) is very dark and it is difficult to distinguish fine details on the image of the plane. Now specifically between Method A and Method B, Method B has a more aggressive color gradient in the sense that the image is brighter on white portion of the plane. This could be attributed to the fact that we redistribute the pixels so each intensity has the same amount of pixels. So, if we do this sharing distribution of pixels, then more intense color can be added since they would have the same number of pixels as a less intense region. Furthermore, in comparison to Method B, Method A maintains more stability because it maintains the intensity through the image transition, so for more visual neutral tasks Method A would be the preferred option. In my own personal preference, I like Method A because more intense colors can at times ruin a good image. For instance if someone is smiling in a photo and it is a little dark. If we apply method B, we will be able to see the person in the image but the person's teeth might be too bright, ruining/distorting the image in a manner that is poor.

Problem 1(c): CLAHE:

For this problem we applied both Method A and Method B to a foggy tower image, but this time we also applied CLAHE as well. CLAHE is a histogram equalization algorithm that uses tiles or local regions to get the image statistics instead of the entire image as a whole. For instance, CLAHE will divide the image into small overlapping tiles and then perform the histogram equalization based on the tile which can enhance local contrast.

Now based upon all of the images (Figures 10-15), the best images were all from the CLAHE side of things. Specifically image 15, as it almost completely reduced the noise coming from the fog making it the clearest image out of the bunch. Both Method A and B did a good job reducing the fog noise, however there was still some residue left over when you look at darker spots on the picture or if you look at the upper portion of the tower. Furthermore, the garden at the bottom is much more difficult to see small individual details in comparison to Figure 15 which

shows us the entire park, trees, and road that was more difficult to view in image 10 and 11. Out of all of the images, I think Figure 15 is the best image out of all of the CLAHE parameters because of less obscured details are on the entire image as well as the lack of any real residue haze on the image as the other CLAHE images still had some haze residue on the final image.

Problem 2: Image Denoising

I. Motivation

Problem 2(a): Basic Linear Denoising

The use of low pass linear filters (such as uniform and Gaussian) are used to reduce noise in images because of their simplicity and efficiency. Despite this benefit, parameters such as the type of filter and the size of the window can play a large role in the performance of reducing noise in images and affect the overall preservation of the image. Ultimately this portion of the homework explores how denoising quality can be influenced by varying linear filters and kernel sizes by comparing the quality with PSNR (peak signal to noise ratio) as the metric.

Problem 2(b): Edge-Preserving Denoising

Even though linear filters can reduce noise, fine structures and edges can be blurred. To mitigate this, introducing bilateral filters can be introduced since they add both intensity similarity and spatial proximity. In this section, we explore how introducing this type of nonlinear filtering may preserve edges while also reducing noise.

Problem 2(c): Non-Local Means (NLM) Filtering

A common issue that arises with local filters is that they rely only on the pixels near it. This can cause filters to not exploit repeated patterns found in the images. To alleviate this, we can use non-local means filtering as this approach averages pixels in similar neighborhoods across a much larger search radius. We will explore whether we can reach better performance in denoising with this approach, or with local linear and bilateral filters.

Problem 2(d): Denoising for Color Images

One key detail that has been explored in these earlier problems is that they have all been addressing noise with images in black and white. Images with color tend to be corrupted with mixed noise types. Thus, we will identify the types of noise present in the image and build a framework that successfully denoises the image while preserving its structure and original color.

II. Approach

Performance Metric

To evaluate the denoising performance across each model and approach in this section, we use Mean Squared Error (MSE) and Peak Signal to Noise Ratio (PSNR). MSE was described as:

$$\mathrm{MSE} = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}(Y(i,j)-X(i,j))^2$$

Where:

- N and M are the height and width of the image
- X(i,j) is the pixel value of the original noise-free image at the location (i,j)
- Y(i,j) is the pixel value of the filtered image at location (i,j)

We later use this MSE to help compute our PSNR:

The PSNR is then computed as:

$$\mathrm{PSNR\ (dB)} = 10\log_{10}\left(\frac{\mathrm{Max}^2}{\mathrm{MSE}}\right)$$

Where:

- Max is the maximum pixel intensity
- MSE is the mean square error between the original noise-free image X and the filtered image Y

We also use clamping to ensure that if a pixel's location is outside the image domain, then the index was clamped to the nearest valid coordinate in each dimension, so each filter computation is in the image boundary without introducing any additional, artificial padding.

Problem 2(a): Linear Filters

For the problem, we implemented two different linear filters: uniform and Gaussian. The formula used for linear filtering is:

$$Y(i,j) = \sum_{k=-r}^{r}\sum_{l=-r}^{r} w(k,l)\, I(i+k, j+l)$$

Where:

- w(k,l) is the filter weight at the offset (k,l)
- r is the filter radius
- Y(i,j) is the output image at position (i,j)

This formula was adjusted by each filter. For the uniform filter, where every pixel contributes equally within the window by assigning equal weights to all pixels in the window, had a formula (for a window of size N*N) to be:

$$w(k,l) = \frac{1}{N^2}$$

Which adjusted our linear filtering equation to be:

$$Y(i,j) = \frac{1}{N^2} \sum_{k=-r}^{r} \sum_{l=-r}^{r} I(i+k, j+l)$$

The Gaussian filter, which assigns higher weights closer to the center or the target pixel, to have a formula:

$$w(k,l) = \frac{1}{Z} \exp\left(-\frac{k^2 + l^2}{2\sigma^2}\right)$$

Where:

- Sigma controls the spread of the Gaussian
- Z is the normalizing constant we use to ensure that the sum of w(k,l) = 1

For this portion of the problem, we first read the image and store it as a 1D grayscale array. Then we generate the filter weights: uniform with a constant weight, and Gaussian with weights computed from Sigma, then normalize them to sum to 1. Then, for each pixel, we compute the weight sum over the window and clamp pixels near the edge of the image. Finally, we compare image quality and PSNR across different filter types and window sizes.

Problem 2(b): Bilateral Filtering

For this next problem, we apply a bilateral filter to denoise the image and preserve the edges of the image. The formula used for bilateral filtering is:

$$Y(i,j) = \frac{\sum_{k=-r}^{r} \sum_{l=-r}^{r} w_s(k,l)\, w_c(I(i,j) - I(i+k, j+l))\; I(i+k, j+l)}{\sum_{k=-r}^{r} \sum_{l=-r}^{r} w_s(k,l)\, w_c(I(i,j) - I(i+k, j+l))}$$

Where:

- Wc is the range (intensity) of the weight based on the values of the pixel difference
- R is the filter radius
- ws(k,l) is the spatial weight based on the pixel's location

The spatial weight uses a Gaussian to penalize pixels with:

$$w_s(k,l) = \exp\left(-\frac{k^2+l^2}{2\sigma_s^2}\right)$$

Where:

- Sigma is the control of the spatial extent of the filter

The range weight component that penalizes pixels with different intensities uses:

$$w_c(\Delta I) = \exp\left(-\frac{(I(i,j)-I(i+k,j+l))^2}{2\sigma_c^2}\right)$$

Where:

- Sigma controls how sensitive the intensity difference is

The implementation process of this problem is to first precompute spatial Gaussian weights with sigma for the window of the filter after reading the image. Then, for each pixel, we will iterate through a local neighborhood to get the spatial weight from pixel distance, get the range weight from the difference in intensity, and then multiply both weights to get the final bilateral weight. Then we will normalize the result by dividing it by the sum of all weights. Once we do this, we will use clamping to handle boundary pixels and finally evaluate the PSNR

Problem 2(c): Non-Local Means Filtering

In this problem, we use Non-Local Means Filtering which exploits the non-local self similarity in images by averaging the pixels in similar neighborhoods or patches regardless of their distance. The formula used to compute the denoise pixel Y(i,j) is:

$$Y(i,j) = \frac{\sum_{(k,l)\in\Omega} \exp\left(-\frac{\|P_{i,j}-P_{k,l}\|^2}{h^2}\right) I(k,l)}{\sum_{(k,l)\in\Omega} \exp\left(-\frac{\|P_{i,j}-P_{k,l}\|^2}{h^2}\right)}$$

Where:

- Pi,j is the local patch that is centered at the pixel (i,j)
- Omega is the search window centered at (i,j)
- (Pi,j - Pk,l) ^2 is the squared distance between the two patches

- I (k,l) is the input pixel (noisy input)
- H is the control of the decay of the exponential weight

Patch distance was calculated as:

$$\|P_{i,j} - P_{k,l}\|^2 = \sum_{u=-r}^{r} \sum_{v=-r}^{r} \left(I(i+u, j+v) - I(k+u, l+v)\right)^2$$

Where:

- (i,j) is the center of the reference patch
- (k,l) is the center of a neighbor patch
- U is the vertical offset
- V is the horizontal offset inside the patch
- R is the patch radius

The approach for this problem was to define the parameters of the patch window size, search window size, and filtering parameter (h). Then for each pixel, we will iterate over all pixels within that search window and compute the squared patch distance between the center patch and the neighbor patch. Also, we will convert the distance to a weight through the exponential function. Once that is done, we compute a weighted average of the neighboring pixel values. Then we normalize the output (we divide by the sum of the weights). Before we evaluate our denoising performance, we ensure that we handle our boundary pixels by using our clamping approach.

Problem 2(d): Color Image Denoising

After analyzing the image and identifying the noise, we applied similar denoising techniques used from earlier problems and extended them to color images. One thing that needed to be clear here is that since the input is represented in the RGB space (because it's a colored image) the denoising approach was applied independently to each color channel. This allows us to maintain the same filtering process we used for grayscale images here while ensuring color information is maintained.

We first applied median filtering to each color channel. This means that for each pixel, a median value of the local window would be selected and used to remove the salt and pepper noise while maintaining color transitions and edges.

After this filtering was applied, Gaussian filtering and Non-Local Means filtering were added to suppress additional Gaussian noise. The Gaussian filtering used spatially weighted averages to perform linear smoothing. NLM filtering on the other hand computed averages that were weighted on the similarity between the search window and the local patches. Thus allowing us to identify repeated patterns and structure similarity according to the color textures.

The implementation process boiled down into first separating the image into their corresponding color channels. Then median filtering was applied independently to each channel to remove the salt and pepper noise. After that Gaussian and NLM filters were applied to reduce the Gaussian noise/ After that, the denoise channels were recombined to form the transformed or denoised image. Once again, we handle the boundary pixels by clamping them so all pixels remain within the image bounds. Finally, we followed the same evaluation metric with the PSNR to evaluate the denoising performance.

III. Results

Problem 2(a): Linear Filtering

Figure 1: flower_grey.raw

Figure 2: flower_grey_noisy.raw

Figure 3: Uniform 3x3

Figure 4: Uniform 5x5

Figure 5: Gaussian 3x3

Figure 6: Gaussian 5x5

PSNR Values:

Uniform 3x3: 26.8963 dB	Uniform 5x5: 26.7450 dB
Gaussian 3x3: 26.853 dB	Gaussian 5x5: 27.5647 dB

Problem 2(b): Bilateral Filtering

Figure 7: Bilateral filtering result (σ_s = 2.0, σ_c = 50)

Figure 8: Bilateral filtering result (σ_s = 2.0, σ_c = 75)

Figure 9. Bilateral filtering result ($\sigma_s = 3.0$, $\sigma_c = 50$)

Figure 10. Bilateral filtering result ($\sigma_s = 3.0$, $\sigma_c = 75$)

PSNR Values:

Spatial = 2.0, Range = 50: 23.5605 dB	Spatial = 2.0, Range = 75: 26.3137 dB
Spatial = 3.0, Range = 50: 23.7713 dB	Spatial = 3.0, Range = 26.5570 dB

Problem 2(c): Non-Local Means Filtering

Figure 11. Non-Local Means result (h = 10)

Figure 12. Non-Local Means result (h = 18)

Figure 13. Non-Local Means result (h= 25)

Figure 14. Non-Local Means result (h=35)

PSNR Values:

H = 10: 20.9694 dB
H = 18: 22.8834 dB
H = 25: 24:8804 dB
H = 35: 26.986 dB

Problem 2(d): Color Image Denoising

Figure 15. flower.raw

Figure 16. Flower_noisy.raw

Figure 17. Median–Gaussian Denoising Result

Figure 18. Median–NLM Denoising Result

PSNR Values:

	PSNR Red	PSNR Green	PSNR Blue	PSNR Average
Median-Gaussian	28.2360 dB	28.1166 dB	28.2021 dB	28.1849 dB
Median-NLM	27.5384 dB	27.5094 dB	27.5826 dB	27.5435 dB

IV. Discussion

Problem 2(a): Linear Filtering:

For this problem, we apply a linear spatial filter to a grayscale image (flower_gray_noisy.raw). The two filters we used were uniform and Gaussian filters with both having varying filter sizes. The approach of the uniform filter was to assign equal weights to all pixels within the window. The Gaussian approach assigns higher weights to pixels that are closer to the middle of the window.

The PSNR values of the uniform filter highlight that a 3x3 size filter performed slightly better than the 5x5 uniform filter, but this was flipped for the Gaussian as the 5x5 filter performed better than the 3x3. Ultimately, the 5x5 Gaussian filter performed the best overall with a PSNR score of 27.5647 while all of the other filters and filter sizes in this problem had scores around 26 db. This can be attributed to the fact that the Gaussian approach preserves the edges of images more effectively because it reduces the influence of unrelated pixels as they are further from the middle while increasing the influence of closer pixels. Contrastly, the uniform filter

treats all pixels the same, yielding more edge blurring and an overall worse PSNR score. It is key to mention here that even though Gaussian filtering is the better method for this image here, choosing the correct filter size plays a critical role in the overall ability to suppress noise. Choosing too high of a filter can potentially corrupt the image structure thus making filter size a key parameter for linear denoising tasks.

Problem 2(b): Bilateral Filtering

For this problem, we are asked to introduce a nonlinear denoising technique known as bilateral filtering which adds spatial proximity and intensity similarity to reduce noise. This approach preserves edge through lowering the weight of neighbors who have large differences in intensity to the center pixel.

σs is the spatial parameter which means that it controls how far the filter will look in the plane of the image. If we have a small parameter, then we will only have local smoothing yielding limited noise reduction because we are looking over a smaller sized neighborhood. If we have larger parameters, then we increase our smoothing to go over a much wider neighborhood which can yield stronger denoising. Something that is key to note here is that if we keep increasing the size of our neighborhood, we can increase the potential loss of fine details in the image.

σc is the range parameter. This parameter controls how strong differences in intensity can be penalized. In other words, if we have a small range parameter then we focus on stronger edge preservation because we are saying that we want pixels with different intensities to not contribute as much. This may be good in terms of edge preservation but if we have a very textured image, we may struggle in noise reduction. Having a larger range parameter can yield stronger results because pixels that have different intensities can contribute. If we make the parameter too large, we can blur edges similar to what happens in Gaussian filtering.

From our results, we noticed that increasing our range parameter from 50 to 75, resulted in larger PSNR values with both models moving from 23.56 dB to 26.31 dB and 23.77 dB to 26.56 dB. Furthermore, what we also noticed is that slightly changing the spatial parameter only slightly increased the dB to an increase around 0.18 dB. In comparison to our linear filters, the bilateral filtering did better preserve edges even though it scored lower than the Gaussian PSNR. Since the bilateral filtering maintained structural details, the MSE may have scored higher leading to a worse PSNR. If you compare the images side by side, bilateral filtering yields a slightly less blurry image around its edges and regions with varying contrast. Thus, my argument would be that this approach did perform better than the linear filters because we prefer a higher quality image over a less noisy one.

Problem 2(c): Non-Local Means Filtering

For this filtering approach, the filter denoises the image by averaging pixels based on the similarity of their surrounding neighborhood rather than the spatial proximity. This means that

pixels that carry similar structures can assist in the filter value regardless of the distance from the pixel.

The four parameters that affect the overall performance of NLM filtering include the patch window size (M), search window size (n), filtering parameter (h) and the gaussian smoothing parameter (a). The M parameter influences the size of the local neighborhood we use for comparing the similarity. This means that if we have a large patch, we may reduce how sensitive the filter is to fine details, but we can have better measures of similarity. The N parameter controls the spatial extent in which patches are similar and can be searched. In other words if we have a large search window, we can add more potential candidate patches to improve our denoising performance, but the cost of running the computation increases. The filtering parameter (h) affects the decay of the weighting function. If we have a large value of h, then we can increase our noise reduction and reinforce averaging. But it leaves us susceptible to structural decay and oversmoothing if it is too large. The Gaussian (a) parameter correlates to the weighting of pixels in each patch where closer pixels to the center of the patch are emphasized when trying to compute similarity.

For this problem, the set up included using a patch window size of 5x5, a search window size of 21x21, a Gaussian parameter of 1.0 and a varying filtering parameter h (10, 18, 25 and 35) to explore the effect it has on denoising. In our experiment, we noticed that as we increased h, our PSNR increased (to a level of 26.986 dB). However in comparison to the PSNR scores of Gaussian and bilateral filters, it was around the same or slightly lower than them. This may be because NLM filtering focuses on preserving texture and the structure of the image which can be seen in Figure 11 and 12 as they both demonstrate how clear the image looks without the limited noise. Also I would argue that Figure 8 and 10 are noisier and worse in overall quality in comparison to the Figure 11 and 12 as they have more of the salt and pepper noise affecting the entire screen.

Problem 2(d): Color Image Denoising

From this noisy picture, we are able to see salt and paper noise as there are randomly disturbed pixels that are either dark or bright. We can also see that there is Gaussian noise present as there is fluctuating intensity across the various smooth regions of the image. As a result, to address, two approaches will be used and then we will pick the solution that best reflects our result. First, both approaches will use a median filter to suppress the salt and pepper noise. This was used because it prevents any impulsive noise from corrupting the subsequent smoothing operation as it eliminates outliers. Once we apply this filter, then test both using Gaussian and NLM filtering to address the remaining Gaussian noise present. If we decided to apply the Gaussian or NLM filter before we did the median filter then we risk degrading the denoising performance since the outliers would be spread out causing the median filter to fail to denoise the noise.

After applying both the Gaussian and NLM filters, the table displayed the PSNR metrics of both. What we noticed is that the Gaussian approach had a higher average PSNR with a score of

28.18 dB while the NLM approach only yielded a score of 27.54 dB. This means metrically speaking, Gaussian smoothing had a better error reduction. We also could see that when comparing both images, the overall quality of the Gaussian model was clearer with the NLM still having a grainy residue left over after the filtering.

Despite how well the Median-Gaussian approach was, there is a slight blurring around the edges when we compare it against the actual denoised image. Furthermore the NLM approach may have preserved every little detail, the expensive computation cost and the still grainy residue on the image.

Something can be used to further improve the performance of the images. For instance implementing a more edge-aware smoothing technique could be in play here. Using an edge aware system where different filter strengths in smooth and textured regions could be applied so we still preserve edge details. In addition to this, if we decide to add bilateral filtering after the median filtering, we may be able to maintain quality as bilateral filtering can help with maintaining the quality of edges while also suppressing any extra Gaussian noise. Thus providing further balance in preserving the structures of the image and reducing noise without overly increasing the computation cost.

Problem 3: Color Correction- Auto White Balancing

I. Motivation

For this problem, we are tasked with applying an auto white balancing technique to an image with color. The original image has unequal RGB channel intensities thus causing a different color cast. Our ultimate motivation is to apply the auto white balancing (AWB) so we balance these color channels so the color image quality reflects neutral colors well.

I. Approach

In this problem, we used the Grey World algorithm which assumed that the average of the colors in a natural image will sum to a neutral grey. Under this assumption, we can use the mean intensity of each color channel to derive scaling factors that move the RGB channels toward a channel average, reducing our global bias.

The mean intensity of each color channel is computed as:

$$\mu_R = \frac{1}{N}\sum R, \quad \mu_G = \frac{1}{N}\sum G, \quad \mu_B = \frac{1}{N}\sum B$$

Where:

- N is the total number of pixels
- R/G/B are the intensity values for each pixel

The target mean formula we then use is computed as:

$$\mu = \frac{\mu_R + \mu_G + \mu_B}{3}$$

Where:

- μR/μG/μB are the means of each color channel

The gain factors for each channel are then described in formula:

$$a_R = \frac{\mu}{\mu_R}, \quad a_G = \frac{\mu}{\mu_G}, \quad a_B = \frac{\mu}{\mu_B}$$

Where:

- μ is the targeted mean
- μR, μG, μB are the channel means

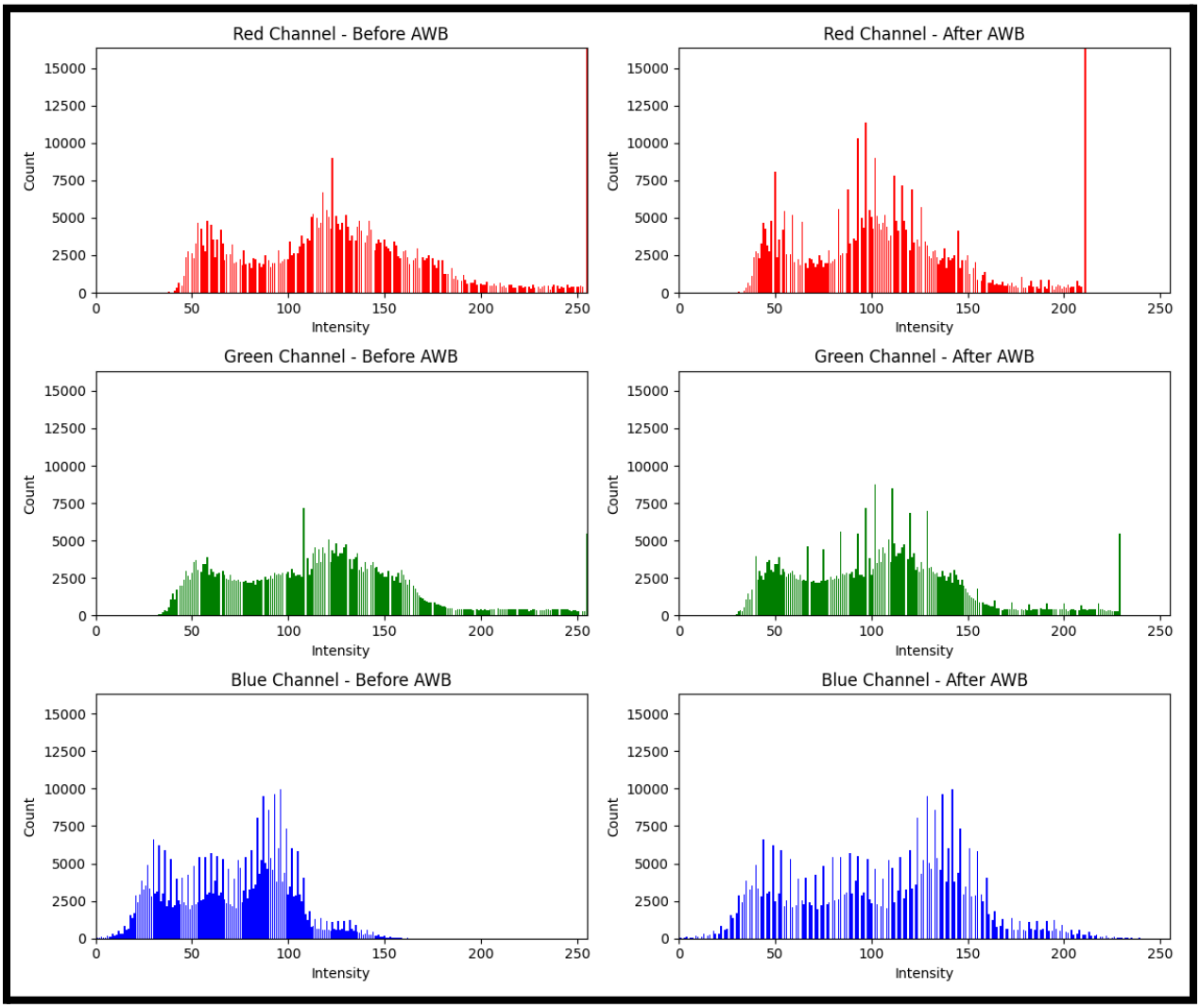
The overall procedure of this method was to first read the RGB image. Then we calculate the mean intensity of each color channel by averaging the pixel values across the entire image. Then we calculate the target gray mean by averaging the means of the 3 color channels. Once we get that information, we plug the target gray mean and the channel mean to get our gain factor. After that, we scale each pixel by the gain factor of their respective channel. We also clip the scaled pixels to have the intensity between 0 to 255 and then we write the white balanced image to the output.

III. Results

Figure 1: sea.png

Figure 2. Sea_awb.png

Before AWB: μR = 126.97, μG = 116.70, μB = 70.996
Targeted Gray Mean: μ = 104.8887
After AWB: μR = 104.9032, μG = 104.8819, μB = 104.8915



IV. Discussion

Histogram Comparison

As we can see in the histograms before AWB, we see that the channels were a lot more spread in their intensity values. For instance, both the red and green histograms highlight the channels shifting toward higher intensity values while the blue channel stays concentrated between 0-175. This is a significant detail to note the color bias we see in Figure 1.

After we ran the AWB technique, we can see that all color channels all share the intensity range with most of them ranging from 0-225. In comparison to the before histogram, the red and green channels shift down their intensities toward slightly lesser values while the blue channel moved toward higher intensities.\

After AWB, the histograms of the red, green, and blue channels become much more aligned. The blue channel shifts toward higher intensities, while the red and green channels shift slightly.

Visual Evaluation

Visually speaking, the initial image had more yellow tones that corrupted the sky, clouds, and even the water to all look more yellow and dark instead of closer to their natural color. After the AWB, the new image shows a more correct blue color for the ocean and sky and a natural white for the clouds. Additionally, the new image still maintains the overall structure and quality the original image had. In fact, with this new color cast, the new image makes it easier to see smaller details such as the sand on the beach and tiny clouds that were originally difficult to spot in the original image.