Problem 1: Basic Image Manipulation

Part a) Cropping and Resizing

1. Face Cropping

Many image processing tools allow for cropping of images. Image cropping is the process of truncating an image to a sub-area of an image.

The process of cropping is relatively straightforward. When given two corner coordinates – i.e. the top left and bottom right coordinates – cropping is simply copying each pixel of the original image over the rectangle area defined by the two corner coordinates. Programmatically, this means copying each pixel from the top left coordinate until the right most coordinate for every row starting from the top coordinate until the bottom. Given the original images of Anna and Rebel below, Figure 1-1-1 and 1-1-2 show the cropped square images.

Further, Figure 1-2-1 and 1-2-2 show the rectangular cropping of the original images.

Figure 1-1-1. Square Cropped Image of Anna

Figure 1-1-2. Square Cropped Image of Rebel

Figure 1-2-1. Rectangle Cropped Image of Anna

Figure 1-2-2. Rectangle Cropped Image of Rebel

2. Face Resizing

Another common feature of image processing tools is resizing of images to a smaller or larger size. The previously cropped square images of Anna and Rebel are of dimension 130x130 and are resized to sizes 100x100, 200x200 and 300x300 in Figures 1-3, 1-4 and 1-5, respectively.

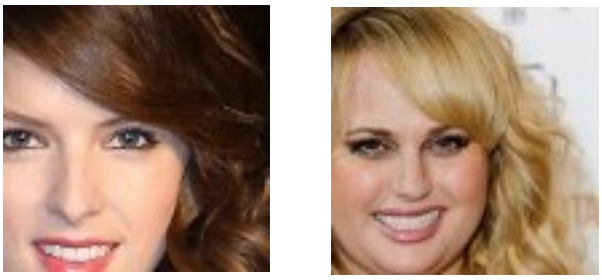


Figure 1-3. 100x100 Resized Images of Anna and Rebel

Figure 1-4. 200x200 Resized Images of Anna and Rebel

There is some quality degradation that is especially noticeable for larger dimensional resizing. This is because bilinear interpolation is an operation that fundamentally loses information. Resizing a 130x130 image to a 200x200 image means that there are 70x70 pixels that are to be 'inferred' from the 130x130 image through interpolation

Figure 1-5. 300x300 Resized Images of Anna and Rebel

Another method of resizing that is more complex with better results is bicubic interpolation. Bicubic interpolation is more computationally intensive but the results are much better than bilinear interpolation.[1]

Part b) Color Space transformation

1. CMY Color Space

CMY stands for Cyan, Magenta and Yellow. It can be viewed of as an 'inverse' of the RGB color spectrum.

$$\begin{cases} C = 1 - R \\ M = 1 - G \\ Y = 1 - B \end{cases}$$

Another way of viewing CMY is that it is a subtractive color scheme – the combination of cyan, magenta and yellow gives white. Whereas RGB is an additive color scheme, such that the combination of red, green and blue gives black. Below are the original images of Clownfish and Octopus, which are decomposed into grayscale representations of cyan, magenta and yellow in figures 1-6-1, 1-6-2 and 1-6-3 for the Clownfish image, respectively and figures 1-7-1, 1-7-2 and 1-7-3 for the Octopus image.

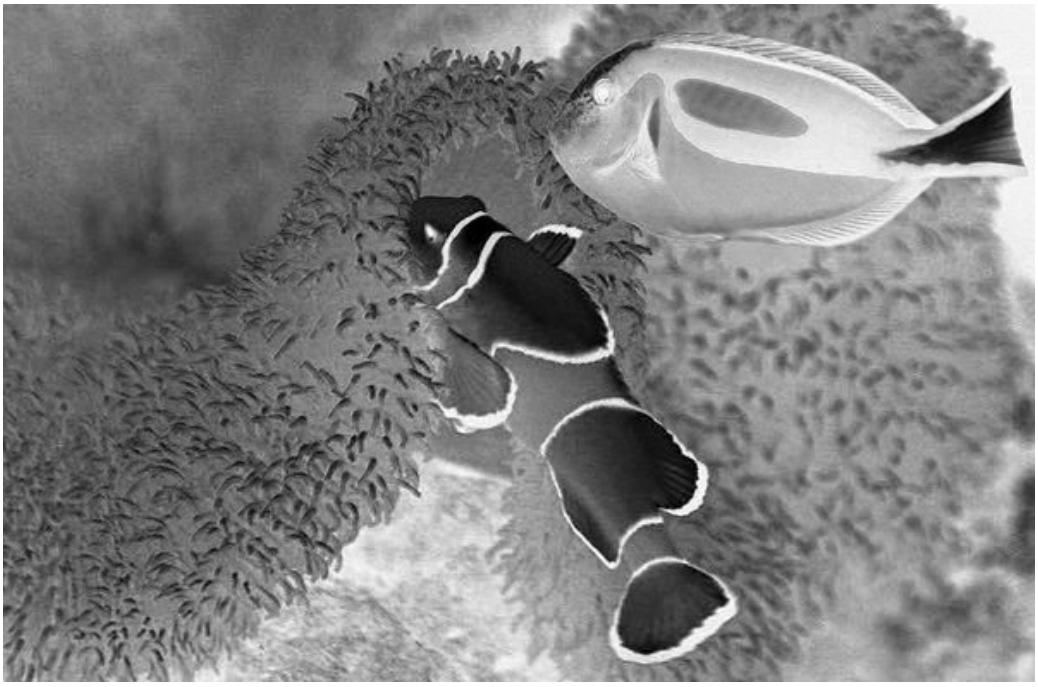


Figure 1-6-1. Cyan Grayscale Representation of Clownfish

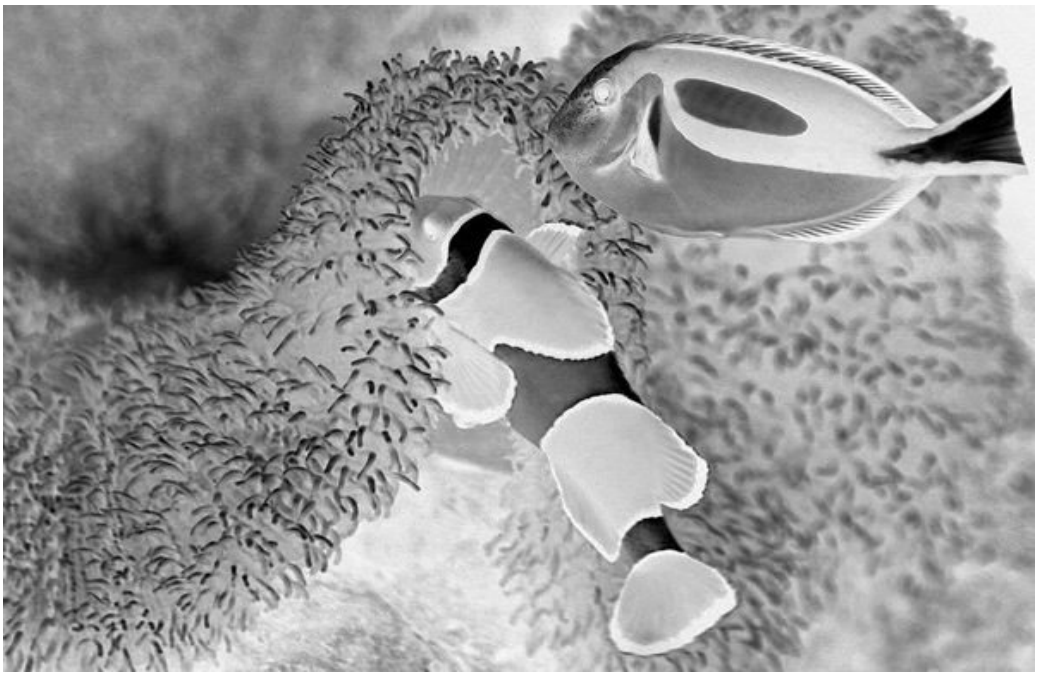


Figure 1-6-2. Magenta Grayscale Representation of Clownfish

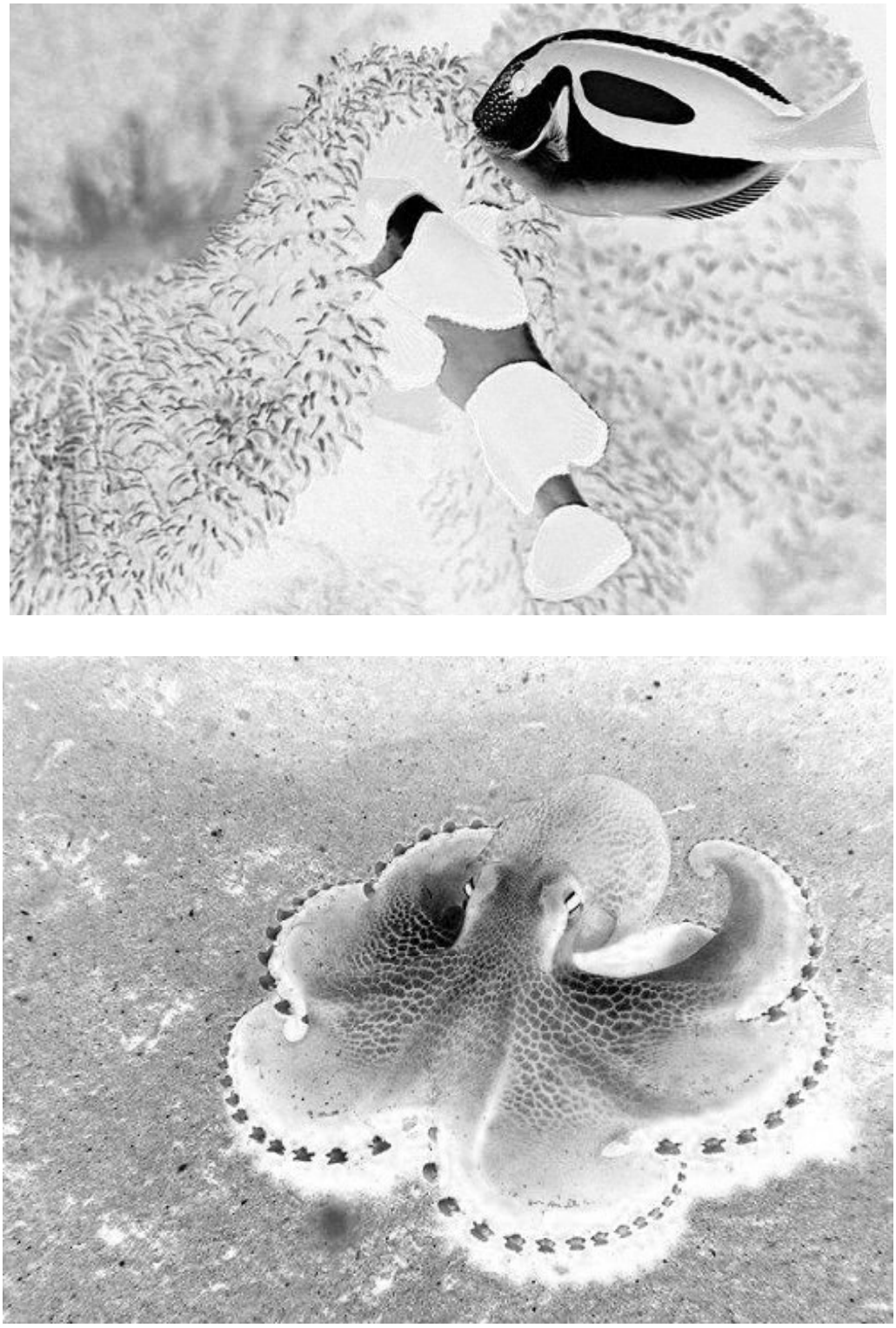


Figure 1-7-1. Cyan Grayscale Representation of Octopus

Note that for white strips on the clownfish for the Clownfish image the CMY decomposition layers have dark pixels for those same areas. A darker intensity means a higher pixel value. Since CMY is subtractive, a high intensity of all CMY colors is equivalent to a resulting white color. A similar pattern can be seen for the octopus dots on its edges.

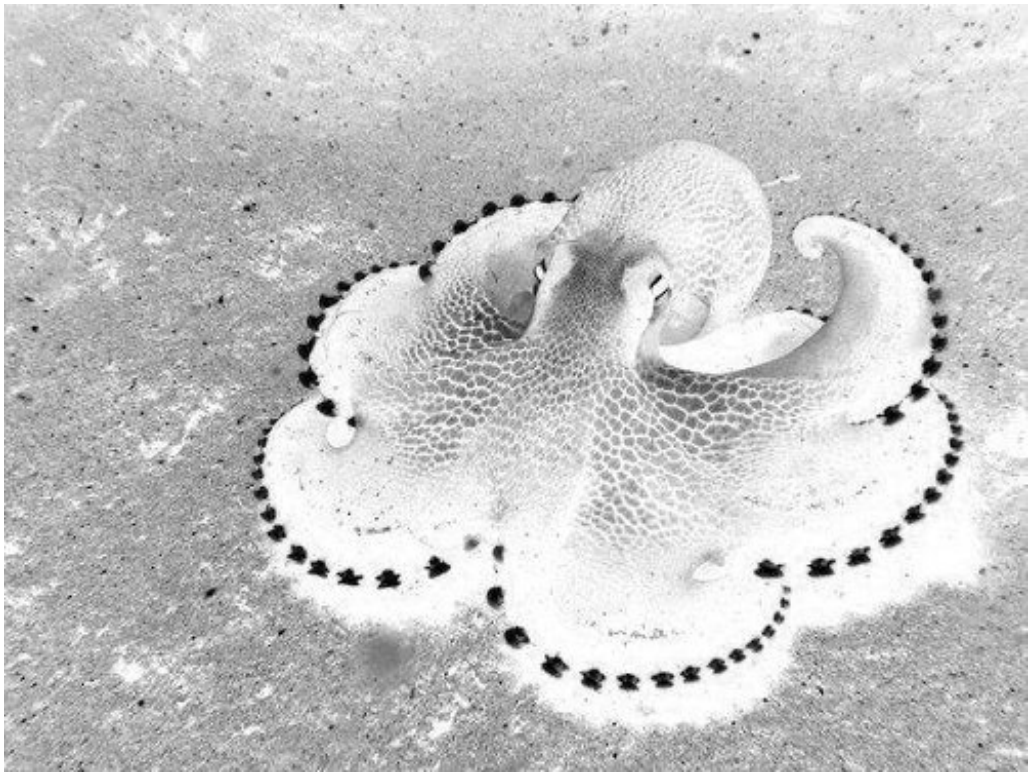


Figure 1-7-2. Magenta Grayscale Representation of Octopus

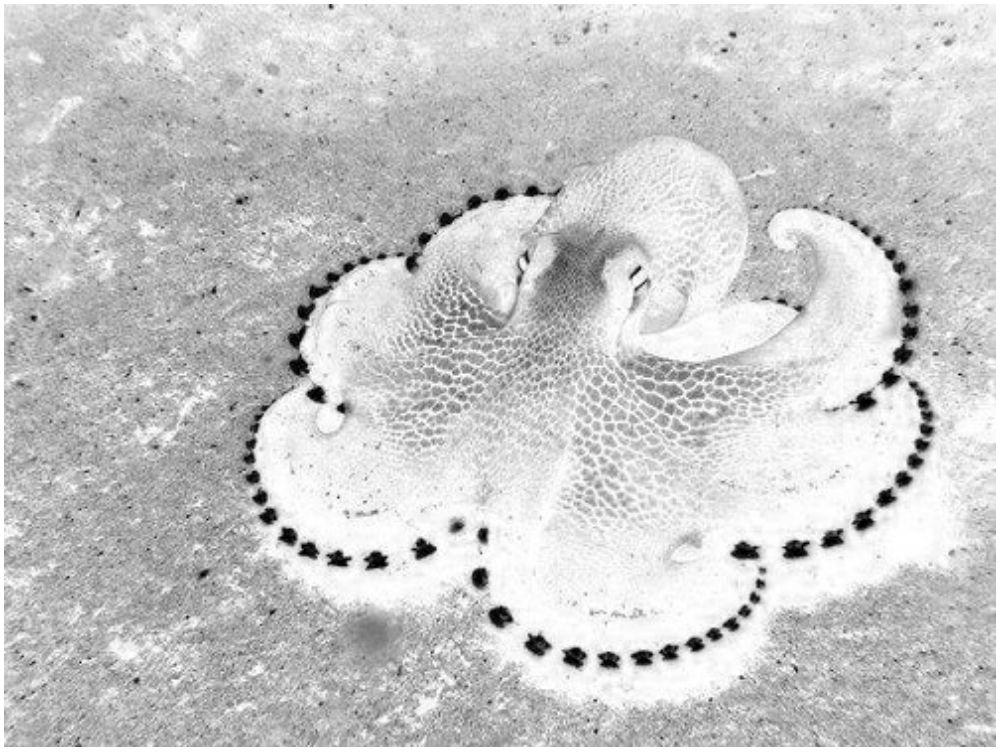


Figure 1-7-3. Yellow Grayscale Representation of Octopus

2. HSL Color Space

HSL stands for Hue Saturation and Lightness, which is a separate color scheme to represent colors. It is meant to be a more intuitive understanding by using a cylindrical coordinate system as opposed to a Cartesian system for RGB and CMY.

Another similar color scheme is HSV, which stands for Hue, Saturation and Value.

A physical understanding of each component of HSL is as follows.

- **Hue** is represented as the angle in the cylindrical coordinate system and represents the actual color to be represented, like red, orange or green.
- **Saturation** is the 'amount' of color or hue present. A larger saturation value is 'more' red or 'more' orange.
- **Lightness** is the intensity of the color where a 0% lightness is equivalent to none of the color or black and 100% lightness is all of the color or white.

The process of converting from RGB space to HSL space is detailed in the equations below.

$$M = \max(R,G,B)$$
$$m = \min(R,G,B)$$
$$C = M - m$$

$$H = \begin{cases} 0 & C = 0 \\ 60\left(\frac{G-B}{C} \bmod 6\right) & M = R \\ 60\left(\frac{B-R}{C} + 2\right) & M = G \\ 60\left(\frac{R-G}{C} + 4\right) & M = B \end{cases}$$

$$L = \frac{M+m}{2}$$

$$S = \begin{cases} 0, & L = 0 \\ \frac{C}{2L}, & 0 < L < 0.5 \\ \frac{C}{2-2L}, & otherwise \end{cases}$$

Below are the two original images: a Turtle and a Jellyfish. These two images are decomposed into three grayscale images representing each HSL values normalized to 0 - 255. Figures 1-8-1,1-8-2, 1-8-3 represent the hue, saturation and lightness for the Turtle image and figure 1-9-1, 1-9-2, 1-9-3 represent the hue, saturation and lightness value for the jellyfish image.

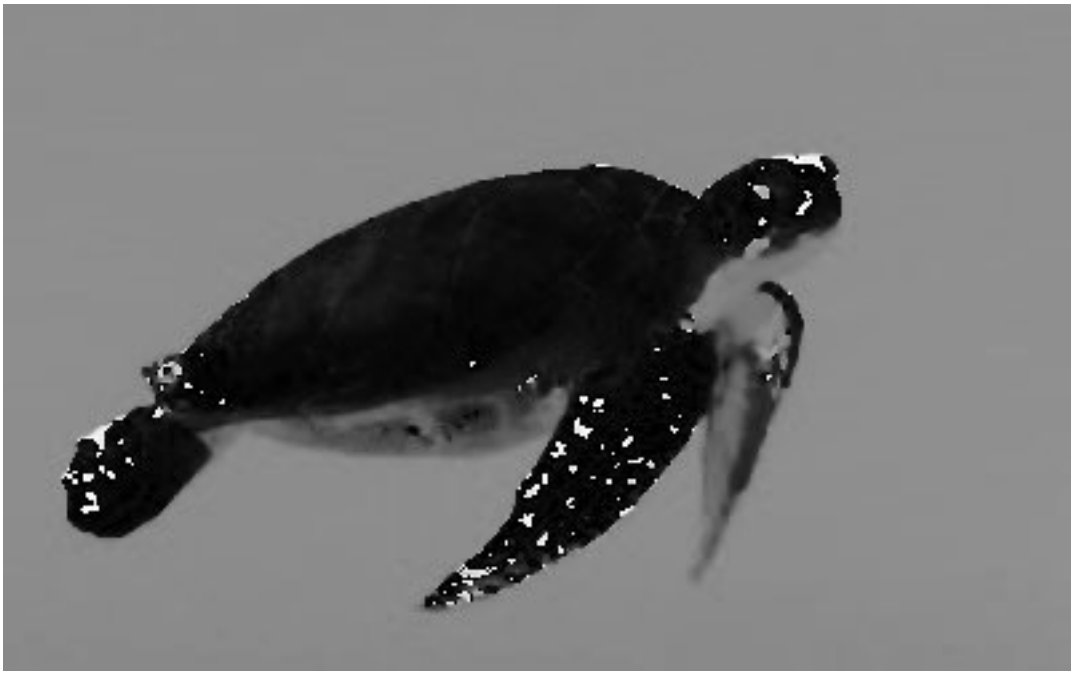


Figure 1-8-1. Hue Grayscale representation of Turtle

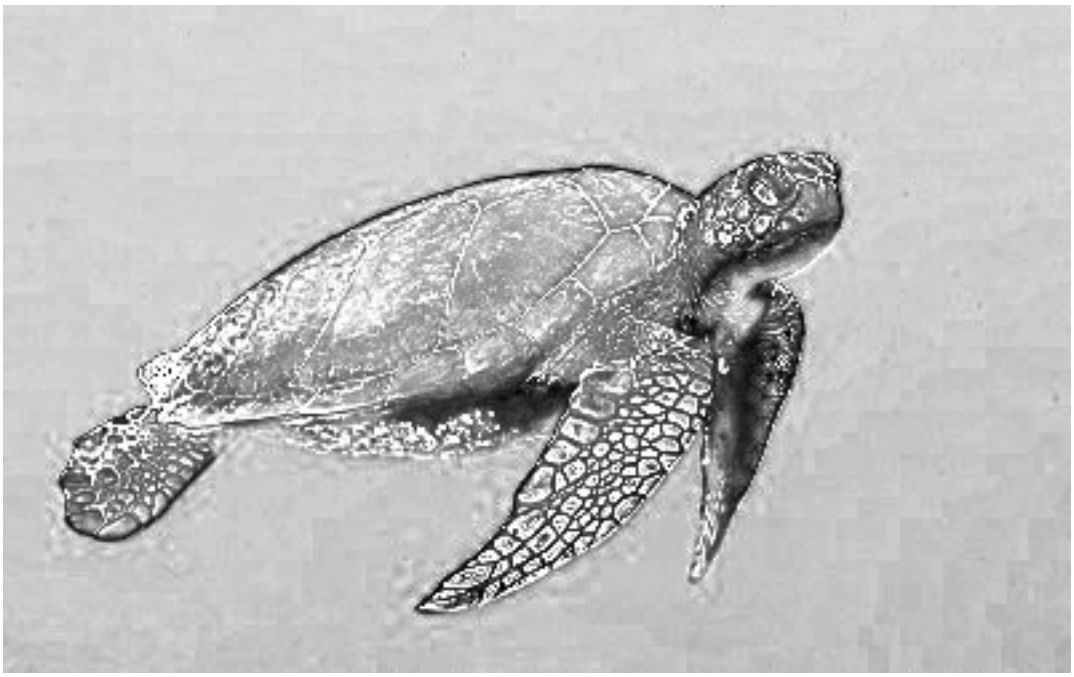


Figure 1-8-2. Saturation Grayscale representation of Turtle

An important observation are the seemingly out-of-place white areas on the hue representation of the Turtle image. However, an important understanding is that this image is a grayscale representation of hue, which is a radial quantity. That is to say, the color spectrum for hue is smooth from 0 to 360, but for grayscale it is obviously not smooth from 255 to 0. The white spots thus represent these values closer to 360 degrees, which are then normalized to values closer to 255 in grayscale. The same can be seen for the Jellyfish images.

Figure 1-8-3. Lightness Grayscale representation of Turtle

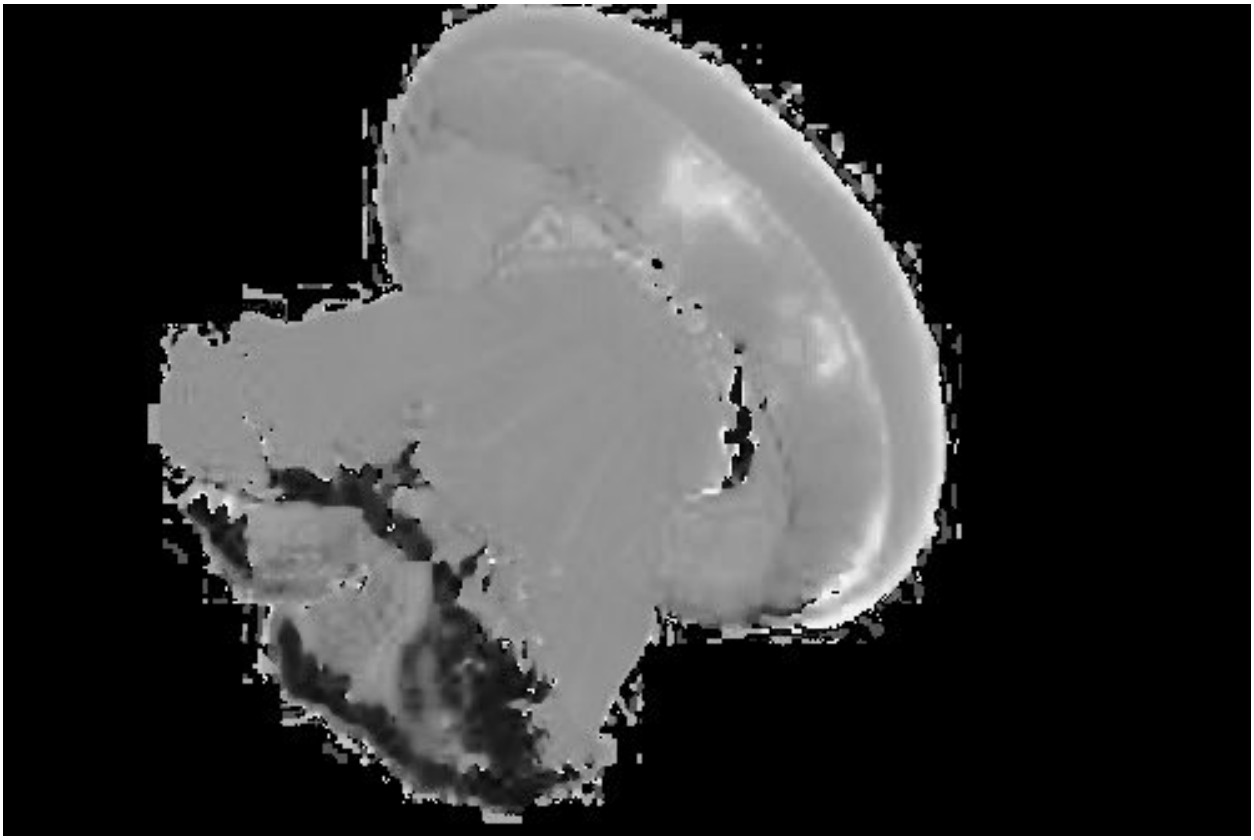


Figure 1-9-1. Hue Grayscale representation of Jellyfish

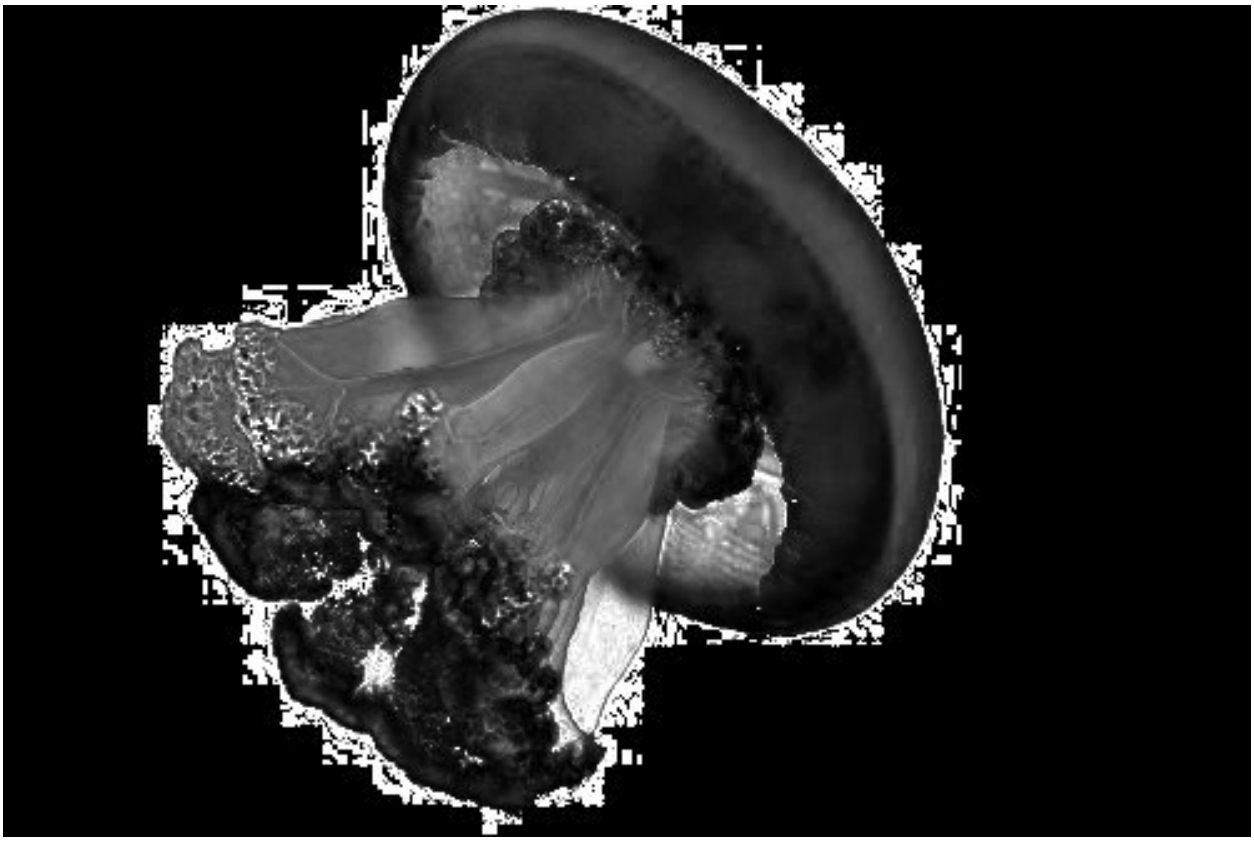


Figure 1-9-2. Saturation Grayscale representation of Jellyfish

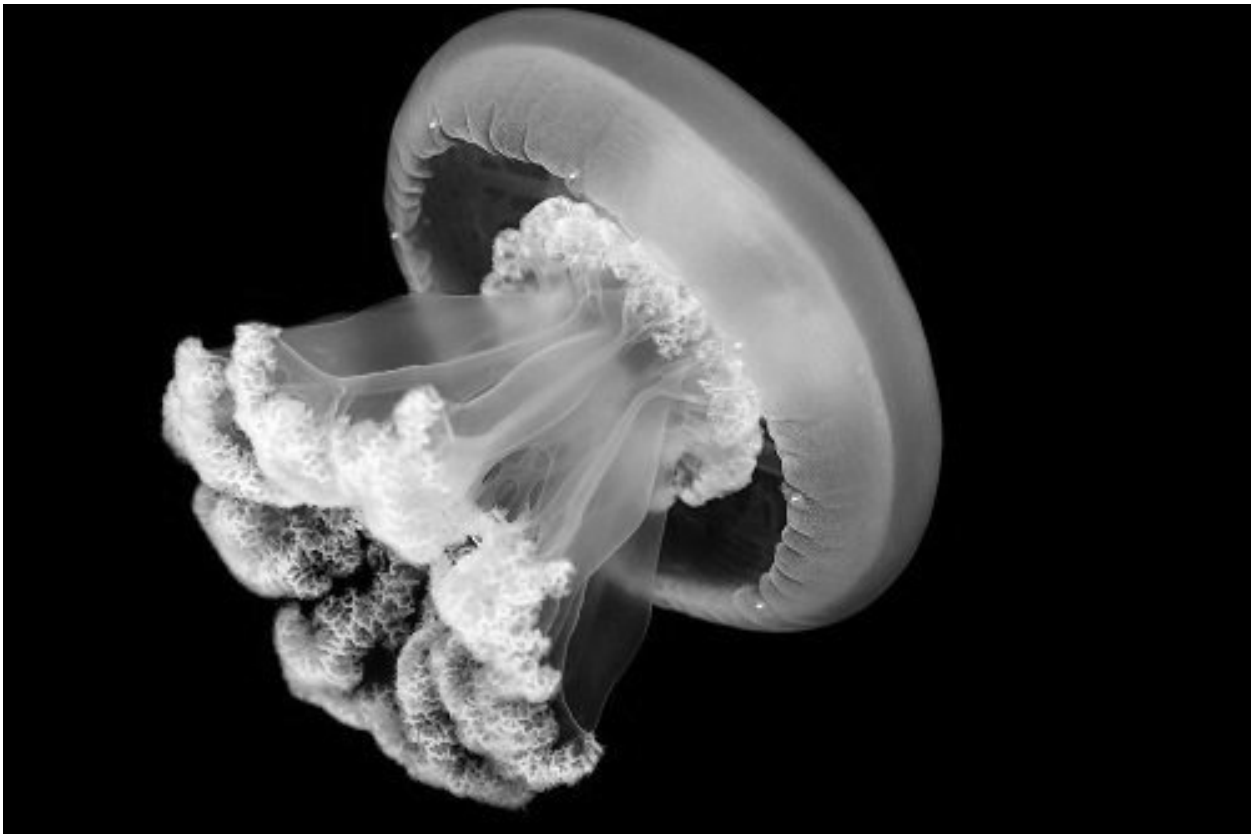


Figure 1-9-3. Lightness Grayscale representation of Jellyfish

One can observe many white spots for the Saturation image for Jellyfish in Figure 1-9-2. The reason for this is because in the equation for converting RGB to the Saturation value, when there are very low but non-zero values for RGB, the saturation value is very high. But when all RGB values are zero, then the conversion expressions similarly yield zero. Closer to the actual Jellyfish in the image, the black pixels start to have non-zero RGB values in the original Jellyfish image and thus represent high values in the saturation values.

Problem 2: Histogram Equalization

Part a) Histogram Equalization for Grayscale Images

Histogram equalization is a process of enhancing contrasts within an image. It involves manipulating a histogram to another histogram that has a more even distribution across more values. This means that a wider range from darker to lighter pixels are represented in the image, and thus a more noticeable contrast. The two original grayscale images that are used are below.

Figure 2-1. Beach_dark and Beach_bright images used for histogram equalization

Two methods were used to achieve histogram equalization.

- **a.** The transfer-function-based histogram equalization method
- **b.** The cumulative-probability-based histogram equalization method

a. Transfer-function-based Histogram Equalization

The transfer-function-based method of histogram equalization is a general method that takes in the pixel value and passes it through a transfer function and outputs a pixel value that allows for a wider range of pixel values. The transfer function used in the report is a full-range linear scaling mapping.

$$G = H(F) = G_{\min} + \left(\frac{G_{\max} - G_{\min}}{F_{\max} - F_{\min}}\right) \cdot (F - F_{\min})$$

Here, G_{min} and G_{max} are the target minimum and maximum pixel values while F_{min} and F_{max} are the actual minimum and maximum pixel values of the image. For the following images, G_{min} and G_{max} are set to be 0 and 255, respectively. F_{min} and F_{max} can be found by observing the image's histogram. From Figure 2-2, one can identify the minimum and maximum pixel values. Observing the min and max pixel values of the images, and setting the target minimum and maximum to 0 and 255 respectively, we have the following transfer functions.

$$G = H(F) = \frac{255}{115} \cdot (PixelVal)$$

$$G = H(F) = \frac{255}{84} \cdot (PixelVal - 171)$$

After linear scaling, the resulting images have a significantly improved contrast as seen in Figure 2-3. The two images are almost indiscernible by inspection alone but differences are noticeable when observing its histograms in Figure 2-4.

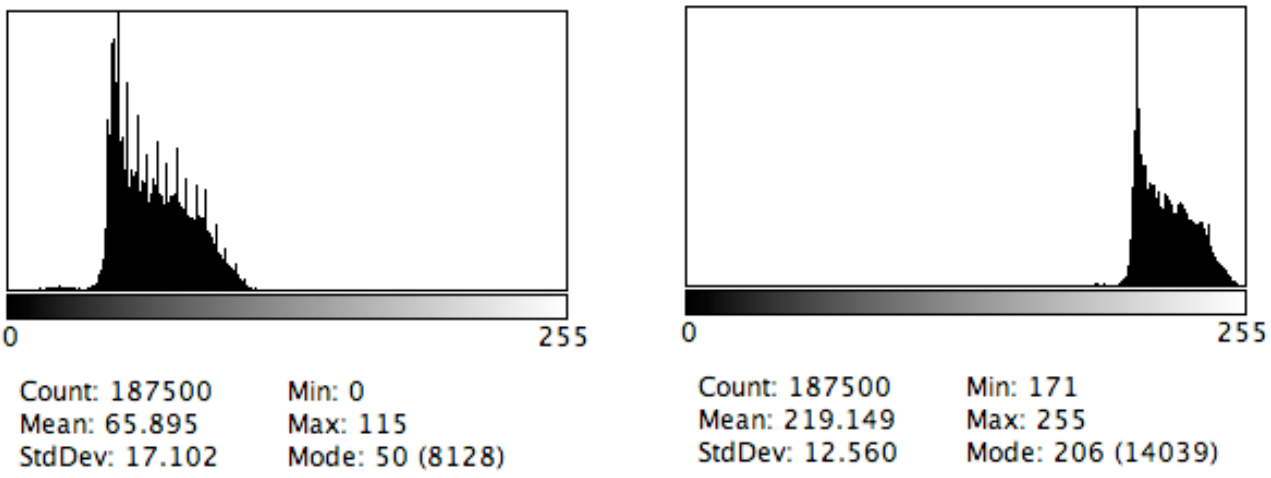


Figure 2-2. Histograms for the Beach_dark and Beach_bright images, respectively

Note that in the histograms after linear scaling, the histograms preserve its general shape but simply has larger 'gaps' between each pixel value. The process is equivalent to having the original histogram stretched to a beginning and end value denoted by G_{min} and G_{max}, or in this case, 0 to 255.

Figure 2-3. Beach_dark and Beach_bright images after linear scaling, respectively

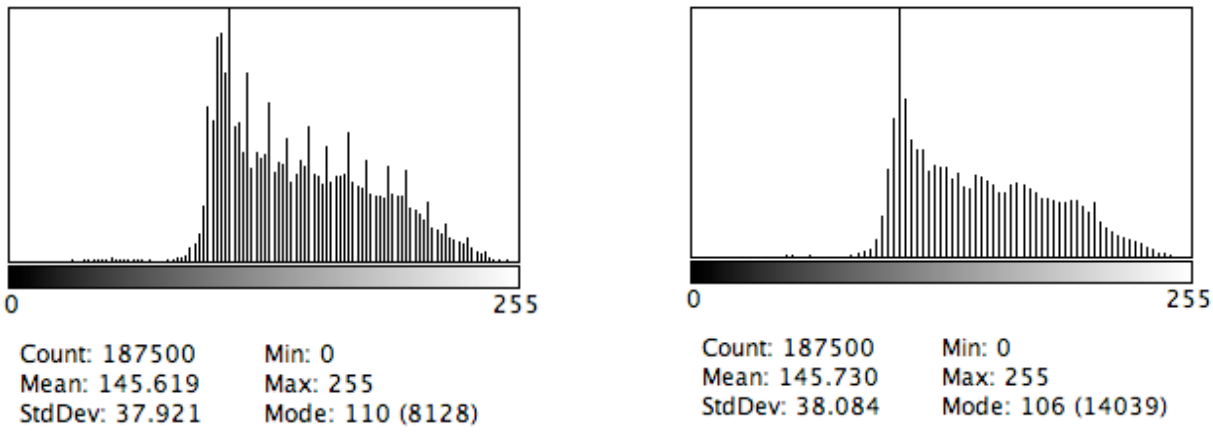


Figure 2-4. Histograms for the Beach_dark and Beach_bright images after linear scaling, respectively

b. Cumulative-probability-based (CPB) Histogram Equalization Method

CPB histogram equalization method is the process of equalizing the number of pixels in every pixel value. In this case, with 187500 pixels and 255 pixel values, this is equivalent to 187500/256 = 732.4 pixels per pixel value. In order to ensure all pixels are considered, the ceiling value is returned and the 'bin size' for each pixel value is 733 pixels. Thus, the original image's pixels lowest 733 pixel values are assigned a pixel value 0. The next 733 pixel values are assigned the pixel value 1 and so on. This can be seen in the histogram in Figure 2-6, where the mode is a value of 0 with 733 pixels. Because the bin size is slightly larger than the total number of pixels in the picture, the last pixel value contains slightly fewer pixels than the other pixel values, which is visible from the histogram in Figure 2-6.

Figure 2-5. Beach_dark and Beach_bright images after CPB, respectively

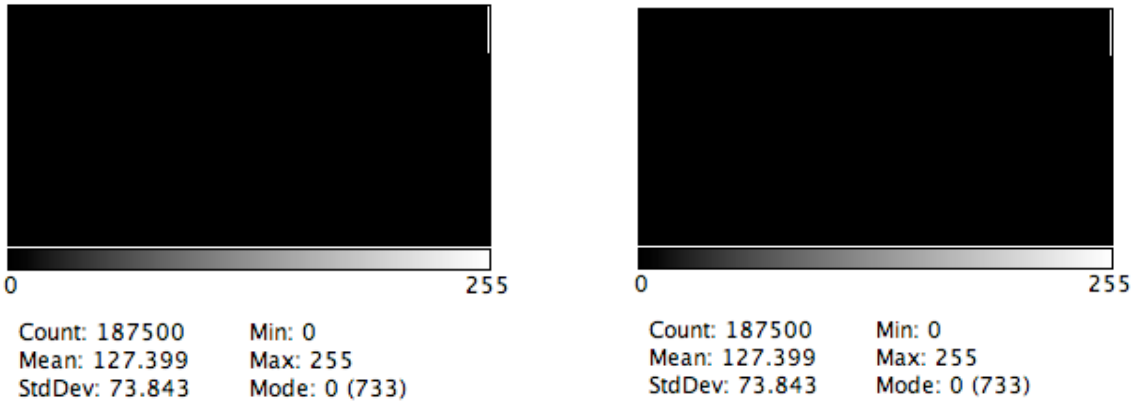


Figure 2-6. Histograms for the Beach_dark and Beach_bright images after CPB Histogram Equalization, respectively

The resulting images after CPB look very similar, with similar contrasting effects. After processing, the images look practically the same, and have identical histograms. The difference between linear scaling as transfer function based equalization and CPB can be seen in the degree of contrast for the resulting images. CPB distributes the same pixel values over a range while linear scaling simply translates all pixels of the same pixel value to a different one. As a result, CPB gives a much stronger contrast, while linear scaling causes contrasts that is based on the original image histogram.

Part b) Histogram Equalization for Color Images

Histogram equalization for color images is the same process as for grayscale images by treating each channel of the color image as a grayscale and performing the same histogram equalization for each channel. Both linear scaling and CPB equalization are used for the color image on the below original Skyline image and illustrated in Figure 2-7 and 2-8.

The original image has a red tint, which is still slightly present after linear scaling in figure 2-7. Analysis of the original histograms and modified histograms in figures 2-9 and 2-10 makes it clear why.

Figure 2-7. Skyline image after linear scaling histogram equalization

Figure 2-8. Skyline image after CPB histogram equalization

Scaling each RGB channel from its respective ranges to the maximum range means that the most prominent pixel values will be translated to a certain pixel value. The red channel has the most frequent value occurring at 177 for the original image, and then translated to 152 for the modified image. Comparing this with the green and blue channels, which have its primary weights at 0, gives the modified image its red tint.

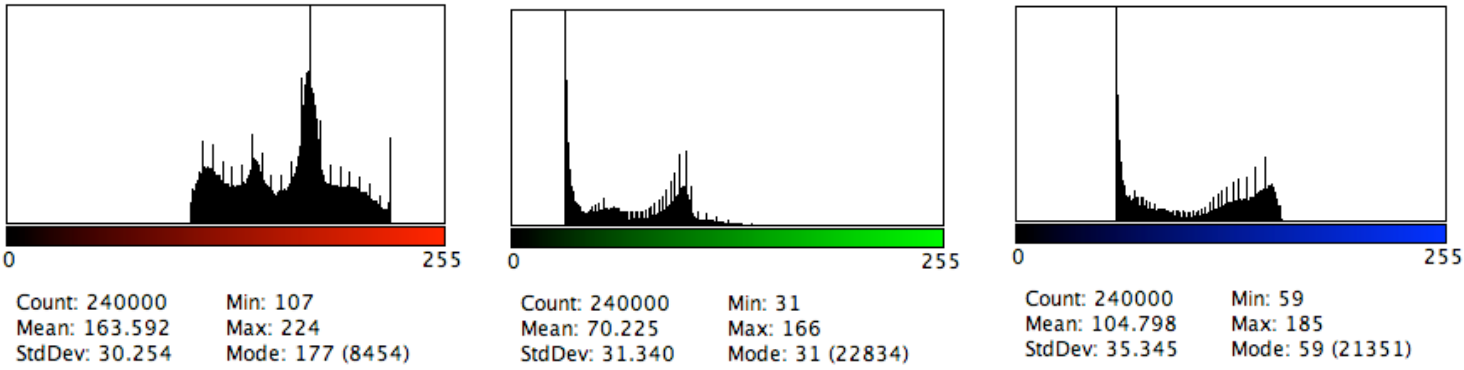


Figure 2-9. Original Skyline image RGB Histograms

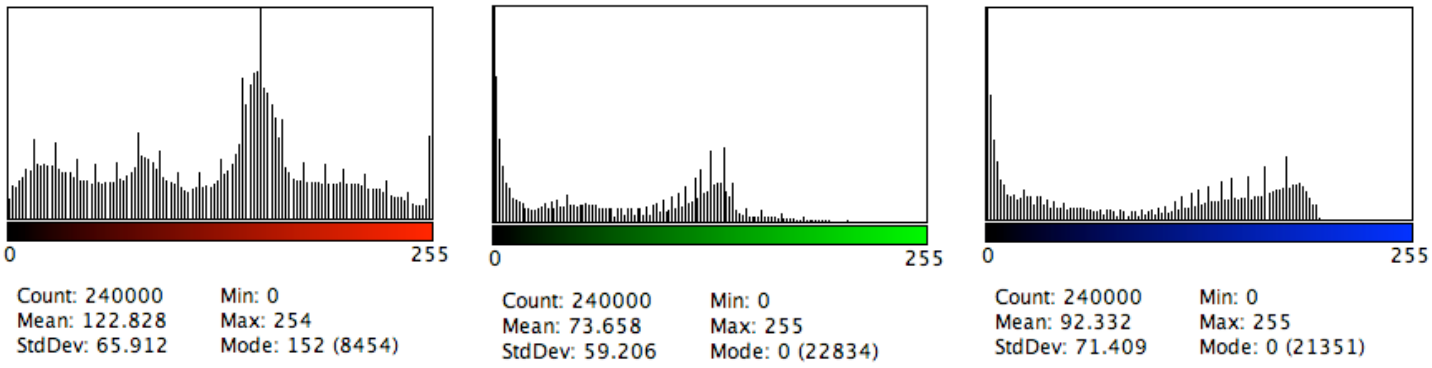


Figure 2-10. Skyline image RGB Histograms after Linear Scaling

In comparison, the CPB modified image in figure 2-8 does not have any color tint because its pixels for each color are distributed evenly across all values (the histograms for each channel look like the histograms in Figure 2-6). As a result, the image has an even representation for every color over the entire spectrum.

Part c) Special Effect via Contrast Manipulation

For this part, instead of distributing the histogram over a uniform distribution, by increasing the frequency of uncommon pixel values and decreasing the frequency of common pixel values, the histogram is to distribute over the same histogram distribution as the Skyline image. As in, modify the image such that its histogram matches the Skyline's histogram as closely as possible. The two original images that are used are below.

The algorithm used to achieve this goal is similar to how CPB equalization is implemented. However, in this case the bin size for each pixel value is not constant. This bin size is defined by a ratio or probability of the total number of pixels. For instance, 8454 pixels have a pixel value of 177. To simulate this histogram, the target images are to have a similar ratio of pixels for a pixel value of 177.

$$P(PixelValue = 177) = \frac{\sum PixelValue = 177}{\sum Pixels} = \frac{8454}{240,000}$$

Do this for all pixel values of the Skyline image and a probability distribution is achieved for all pixel values. More generally,

$$P(PixelValue = c) = \frac{\sum PixelValue = c}{\sum Pixels}, 0 \leqslant c \leqslant 255$$

Thus, for the target image, the bin size for each pixel value is given by the following expression.

$$BinSize|(PixelValue = c) = N_{target} \cdot P(PixelValue = c)$$

From there, the same algorithm is used for CPB equalization. Starting from the lowest pixel values of the target image, these take the pixel value of 0 until the bin size for this pixel value. The next lowest pixel values then take the pixel value of 1 until the bin size for this pixel value is satisfied and so on. Since each relative bin size is exactly the same as the original image, the resulting histogram should look very similar to that of the Skyline image. The resulting images are shown in figure 2-11.

Figure 2-11. Trojan and Park images with Skyline modeled histogram

As can be seen from figure 2-11, the images have a red tint similar to the original Skyline image. Comparing figure 2-12 with figure 2-9 and it can be seen that the resulting histogram shape is practically identical to that of the Skyline's.

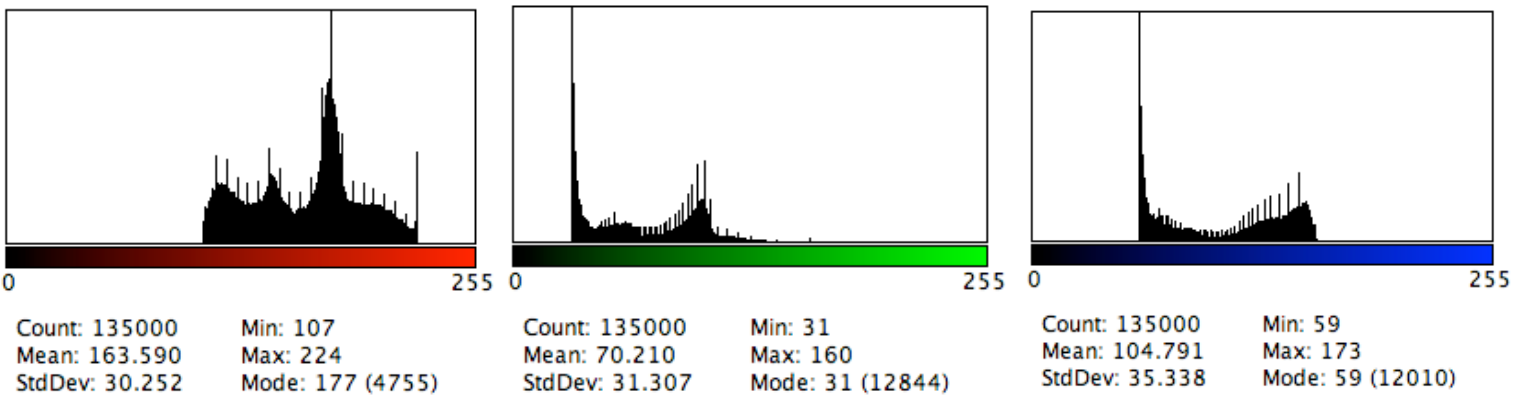


Figure 2-12. Modified Trojan RGB histogram

Part d) Histogram Transform

The purpose of this part is to match the histogram of a specific image to a predefined distribution, which is similar to part c). However, instead of the histogram of an image, this time it is for that of a Gaussian distribution with μ= 125 and σ = 40. The distribution is then truncated at values of 0 and 255 because pixel values outside of this range is meaningless. The equation for a Gaussian distribution is as follows below.

$$p(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

The C++ standard template library comes with a function that allows for realizations of a Gaussian random variable given its mean and standard deviation. With the number of pixels as the number of realizations of the Gaussian random variable and truncated at 0 and 255 then the number of realizations at a specific value is the bin size for that pixel value. Figure 2-13 shows a Gaussian distribution with the same number of realizations as the number of pixels in Student_1 in figure 2-14. After following the same bin filling algorithm as in part a), b) and c), the resulting histogram should match the Gaussian distribution.

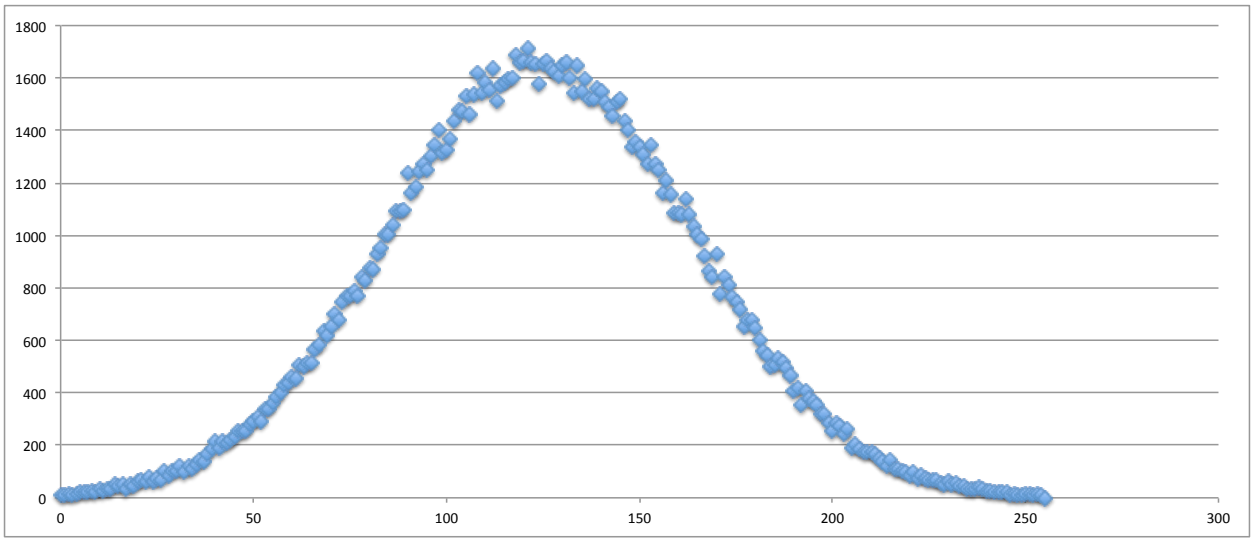


Figure 2-13. Histogram Scatter Plot of 166000 realizations of a Gaussian Distribution with mean, 125 and standard deviation, 40

The original images are shown below in figure 2-14 and the modified images are shown in figure 2-15.

Figure 2-14. Original Student_1 and Student_2 images

Figure 2-15. Modified Student_1 and Student_2 images

If the histogram of the modified Student images followed that of a Gaussian distribution then this results in less intensity on pixels of the extreme sides of grayscale magnitude. Lower and higher pixel values have less presence and pixels of a moderate intensity should have more presence. The histograms of the modified Student images can be seen in figure 2-16. The histograms very clearly follow that of a Gaussian distribution, with a mean of approximately 125 and a standard deviation of 40.

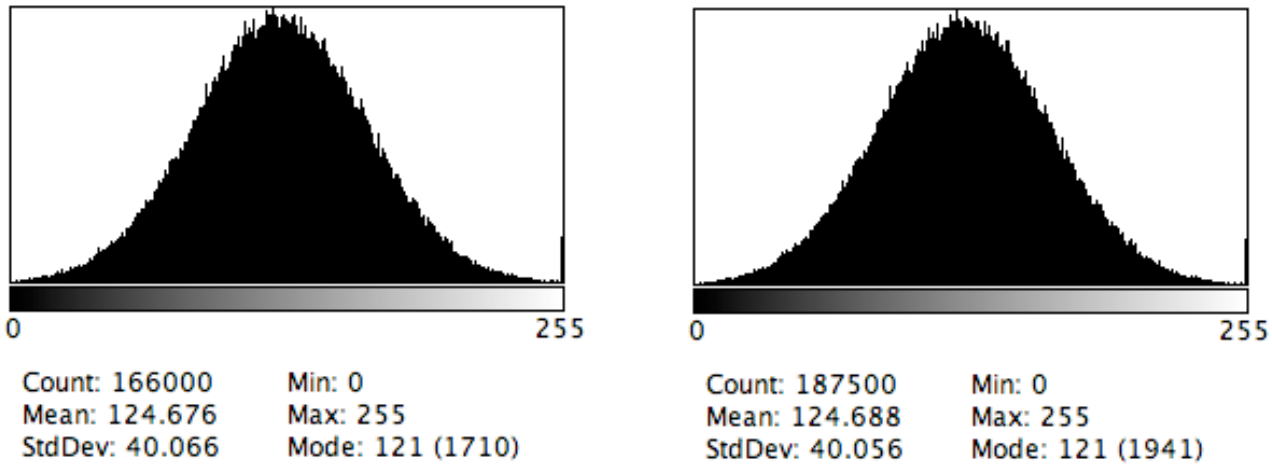


Figure 2-16. Histograms of the Modified Student_1 and Student_2 images

Problem 3. Noise Removal

Part a) Mix Noise in color image

The goal of this problem is to remove as much noise as possible from the noisy image in figure 3-1. It is granted that the noise present in the image is a mixed form of noise. After careful inspection of the images in figure 3-2, it can be deduced that the noise is likely to be a combination of impulse as well as gaussian noise.

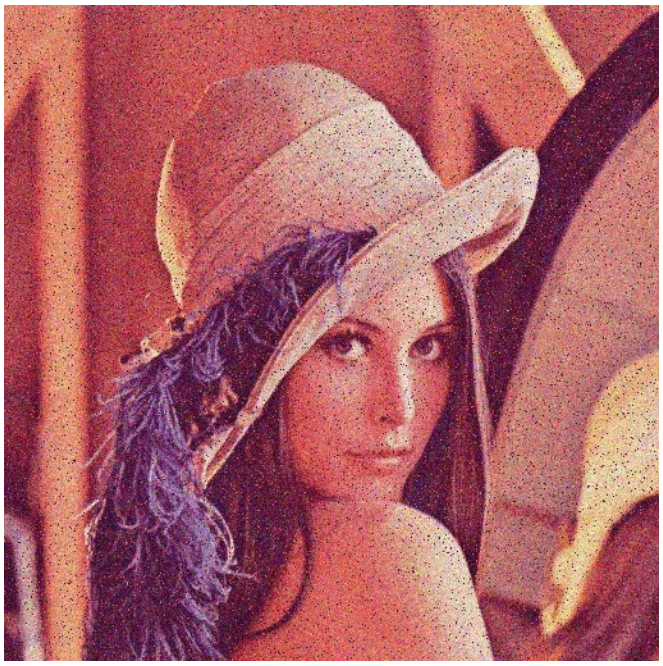


Figure 3-1. Lena_noisy image to be cleaned

Figure 3-2. Grayscale representation of the Lena_noisy image in each of the RGB channels. From left to right: red, green, blue.

Each channel is likely to have the same source of noise. Modeling the noise as an addition of impulse noise and Gaussian noise allows the noise to be treated separately, i.e. use a filter that is more effective for impulse noise and then a filter that is more effective for Gaussian noise. Thus, this form of modeling allows noise filters to be cascaded.

A low-pass filter that is effective for outliers as well as impulse noise is the median filter. This filter considers all values around the center pixel and replaces the center pixel with the median of those pixels.

The other filter to treat Gaussian noise is a noise cleaning mask with values that are empirically chosen. The procedure is as follows. The pixel at the center of the mask is replaced with the dot product of each pixel covered by the mask. Though the values of the mask are empirically chosen, the shape of the noise cleaning mask is similar to that of the Gaussian kernel, i.e. strong weight in the middle and weaker away from the center. The specific mask that is used has the following kernel.

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

While these filters can be used to cascade in any order, the more effective approach is reducing impulse noise before applying a Gaussian noise cleaning mask. The reason for this is because masking filters are more effective with less outliers. Therefore median filters, which reduce outliers and sharp peaks should be applied first before masking/mean filters. The PSNR of different procedures can be seen in the below table. The best result is the one that applies the median filter twice and then a 3x3 noise cleaning mask to smooth over any other Gaussian noise.

Filter Procedure	PSNR
None	18.6008
3x3 Mask	24.9604
Median	25.5156
3x3 Mask -> Median	26.0569
Median -> Median -> 3x3 Mask -> 3x3 Mask	26.5366
Median -> Median	26.5728
Median -> 3x3 Mask	26.6343
Median -> Median -> 3x3 Mask	26.7723

The effect of filter size on performance for the median filter shows that larger filter sizes actually cause a reduction in the performance. With larger filter sizes, more impulse noise is considered and so the noise affects the median filter's performance. The effect of different median filter sizes can be seen below.

Median Filter Size	PSNR
3	25.5156
4	26.0361
5	24.6962
6	23.8548
7	22.8359

The resulting image of applying the median filter twice and then a 3x3 mask can be seen in figure 3-3. Though there is significantly less impulse noise, there is still some Gaussian noise that can be observed in the image. A 3x3 noise cleaning mask is most likely not effective enough in reducing the Gaussian noise. A larger 5x5 Gaussian kernel with heavier weight may have a more significant effect of reducing Gaussian noise and improving the PSNR.

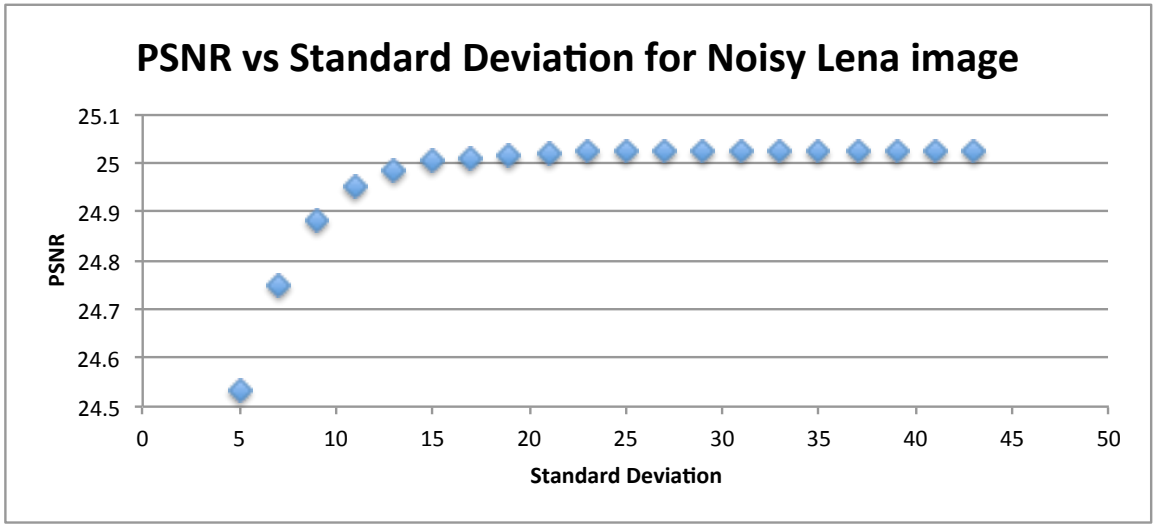
Figure 3-3. Mixed Noise removal result of the image in figure 3-1

Part b) Non-local Means (NLM) Filter

Non-local means is an algorithm that takes advantage of the similarity of between certain regions of an image. For every pixel, there is a neighborhood space of a user-defined radius. Then, iterating through pixels in the image, compute the Euclidean distance between the neighborhood space of the target pixel and the original pixel. This Euclidean distance value is used to calculate the weight of 'similarity' between the target pixel and the original pixel. This process is done for all pixels within a user-defined search space. These weights are then normalized such that each weight is a probability between 0 and 1 and the sum of all the weights is equal to 1. The new pixel value is then the sum of each target pixel in the search space multiplied by the calculated weight.[2]

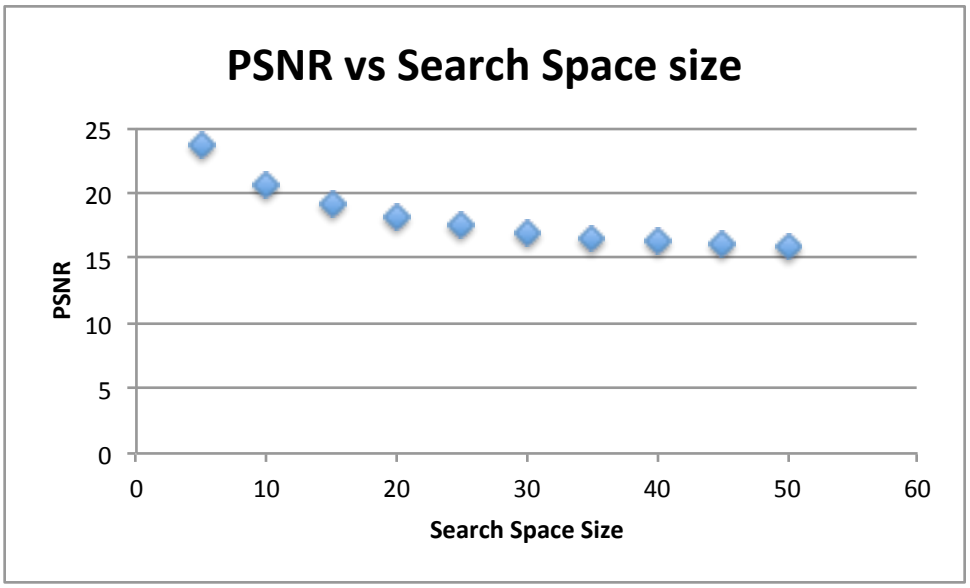
A table of various search space size parameters, filter size as well as the standard deviation is shown below.

Standard Deviation	PSNR
5	24.5315
7	24.7475
9	24.8829
11	24.9513
13	24.9851
15	25.0023
17	25.0114
19	25.0167
21	25.0197
23	25.0219



The standard deviation has a very clear effect in the beginning lower values. But as standard deviation increases, the PSNR approaches a certain value.

Search Space Size	PSNR
5	23.7486
10	20.5892
15	19.1038
20	18.1003
25	17.5071
30	17.0017
35	16.6162
40	16.3268
45	16.1258
50	15.9969



Intuitively, the larger the search space size, the better the performance and therefore the higher the PSNR value. However, from the program implemented, as search space size increases, the PSNR decreases. The most likely reason for this is that the algorithm is incorrectly implemented

Search Space Size	PSNR
1	18.1022
3	18.1003
5	18.1195
7	18.1382
9	18.1518

The search space size has little effect on the PSNR but does appear to show a slight upward trend.

The parameters that seem to give the best result is the following. A search space size of 4, a neighborhood size of 3 and a standard deviation of 17, which gives a PSNR of 25.0113 and the image in figure 3-4.

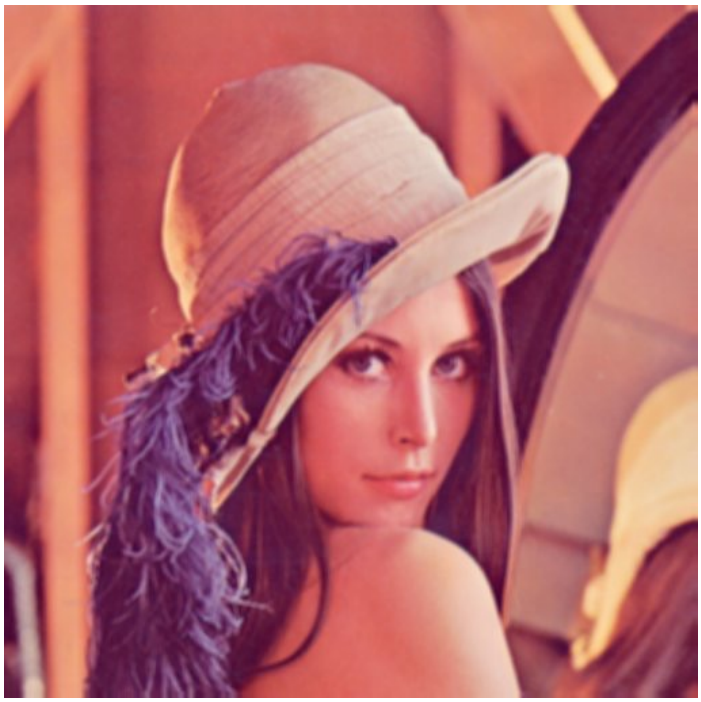


Figure 3-4. Mixed Noise removal result of the image in figure 3-1 using Non-local Means

Interestingly, even though the PSNR is lower than in part a), figure 3-4 shows an image that is subjectively more pleasing, with less visible noise. However, there is more blurring and some smoothing that most likely contributed to the lower PSNR value.

The following figures 3-5 and 3-6 show a series of images that its original form, its noisy noisy form and after NLM algorithm processing and also denoising processing as detailed in part a).

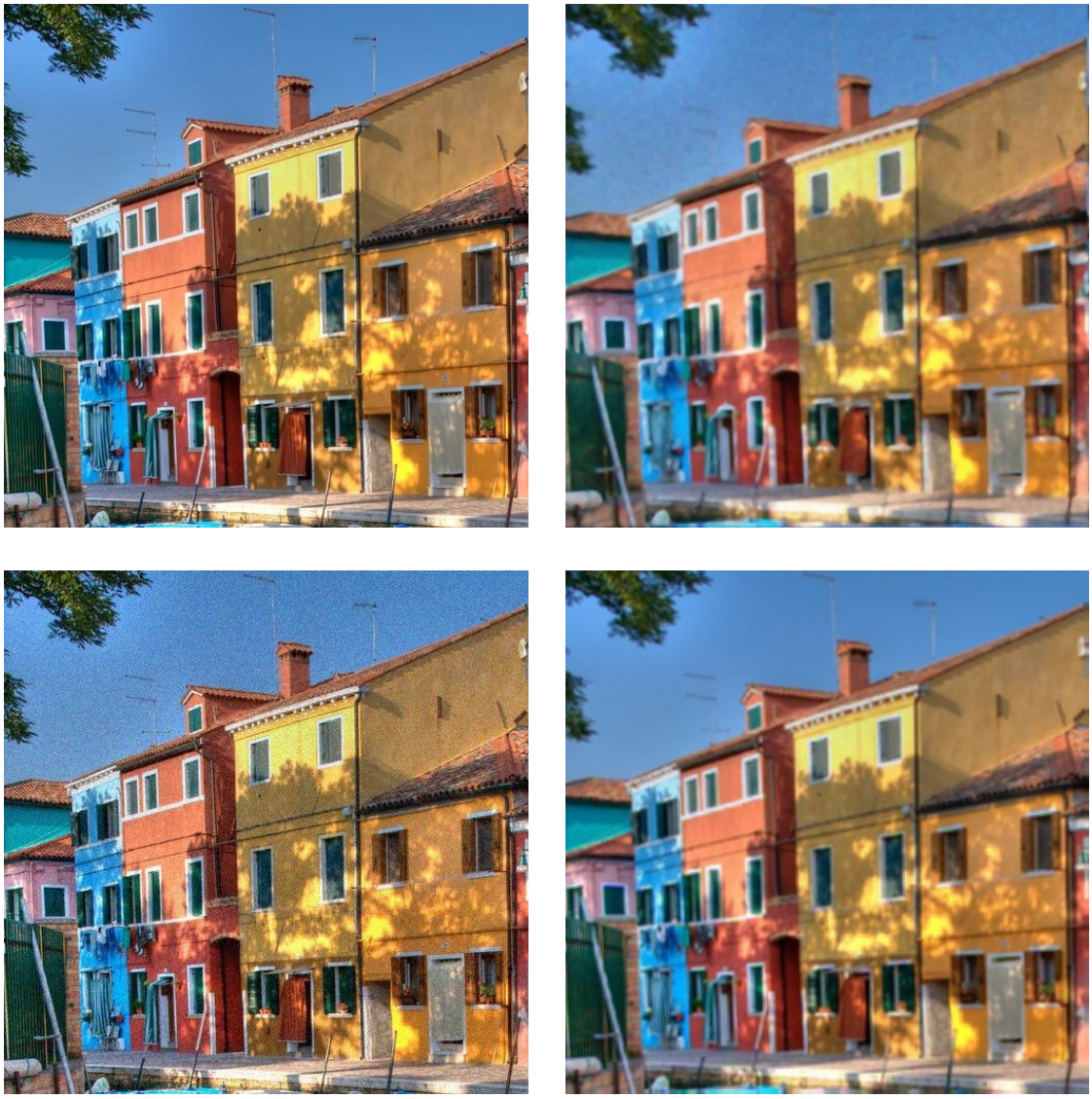


Figure 3-5. Top-left: Original Buildings image, Bottom-left: Noisy Buildings image, Top-right: Noise removal through process described in 3a) Top-left: Noise removal through non-local means processing

Figure 3-6. Top-left: Original Trees image, Bottom-left: Noisy Trees image, Top-right: Noise removal through process described in 3a) Top-left: Noise removal through non-local means processing

Noise is still clearly visible after denoising using the process as detailed in part 3a). The images that are processed using NLM have a subjectively better appearance. Certain smaller details are maintained. For instance, the median filtering and noise cleaning masking of the trees caused some of the thinner branches to be erased. This makes sense because a thin branch would be considered an outlier by the median filter. Meanwhile, the NLM process considers nearby similar-looking branches and gives it a pixel value similar to that. Because of NLM's algorithm of searching over large areas and noting specific similarities between pixels, it would do very well in patterned or textured images, i.e. images that have similar neighborhoods repeated constantly. That way, more weights to different neighborhoods can be used to consider target pixels that are as similar to the original as each other.

Part c) Block Matching and 3-D (BM3D) transform filter

Block matching 3D is a new denoising algorithm invented in 2007 that incorporates spatial processing as well as frequency processing. The algorithm makes use of a Wiener filter which is processing in the frequency domain. It also employs a technique named “hard-thresholding,” which combines similar blocks of the image in its spatial domain.[3]

References

[1]D. Lancaster, “A Review of Some Image Pixel Interpolation Algorithms,” 2007. http://www.tinaja.com/glib/pixintpl.pdf

[2] A. Buades, B. Coll, and J.-M. Morel, “A non-local algorithm for image denoising,” in Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 2. IEEE, 2005, pp. 60–65.

[3] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, “Image denoising by sparse 3-d transform-domain collaborative filtering,” Image Processing, IEEE Transactions on, vol. 16, no. 8, pp. 2080–2095, 2007.