178-Assignment1

Kyle Adrian del Castillo
CMSC 178 B - Image Processing

April 2023

1 Exercise 1

1A Colour Balancing using a Reference Chart

In this exercise, colour balancing is implemented to several images manually. Colour degradation was also applied manually by the student, and a colour balancing function was implemented to correct a simple shifting of RGB channels. Since the assignment provided skeleton code for the functions to be implemented, beginner-friendliness was ensured for introductory level of mastery in MATLAB. Officially, the objectives are as follows:

1. Extract RGB values from a colour-correct reference chart
2. Create a look-up table for color correction later used in scaling "*bad image*" channels
3. Apply the rgb map or the look-up table values to the "*bad image*".

1A.1 Extracting RGB values

With the provided reference chart in Figure 1.1, the function `get_chart_values` extracts a sample from the 24 color patches within the reference card image, providing us with a `(24, 3)` array of RGB values.

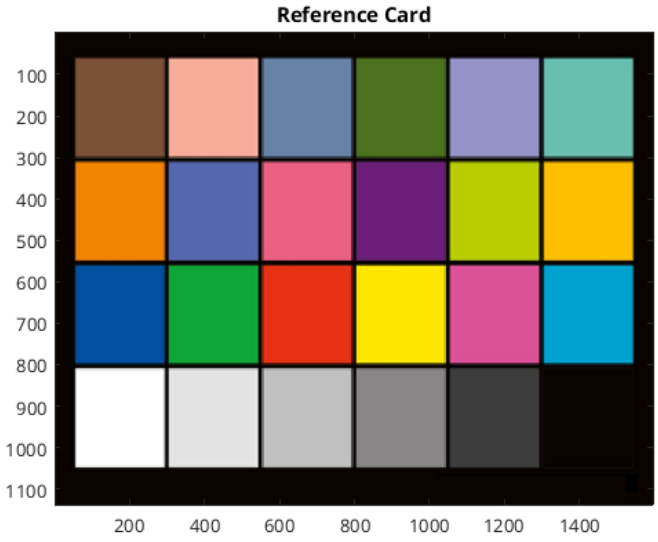


Figure 1: Reference Card image displayed in MATLAB

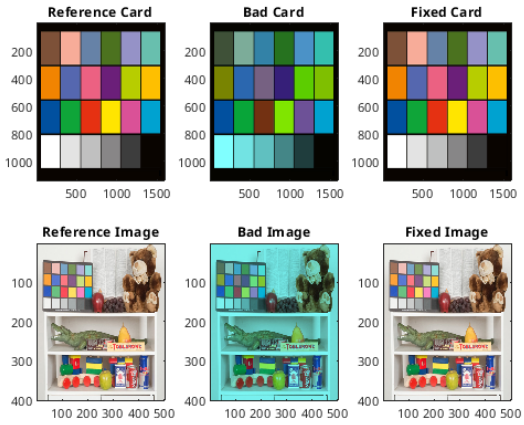
1A.2 Producing a look-up table for color correction

The function `chart_correction` has been modified to use `polyfit()` to guess in-between unknown points by fitting the values of each channel to a linear regression polynomial from two RGB channel sources. We then use `polyval()` to extract the fitted values of each channel and populate our RGB lookup table. The student notes that while `interp1()` was suggested as an alternative to `polyfit/polyval`, a working solution was never achieved and was met with errors.

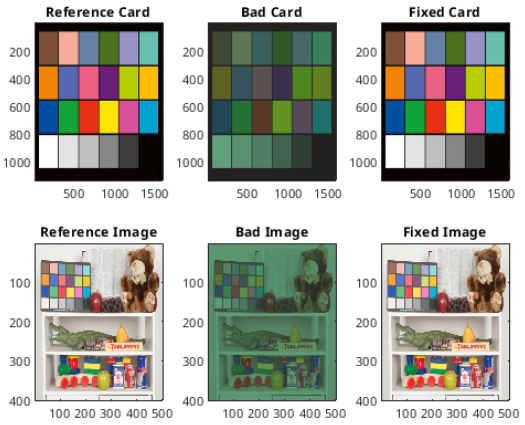
1A.3 Applying the RGB map

With a color correction lookup table from the `chart_correction` script, we can now work on mapping the contents of the lookup table to our *"bad image"*. Initially, the student worked with the provided tests from the `chart_test` script as seen in figure 2 and 3. Additional testing with other images with combined channel scaling and skewing was successfully attempted. Figure 2a shows a 50% reduction in red tones, and 2b shows a maximum and minimum range skew in the RGB channel.

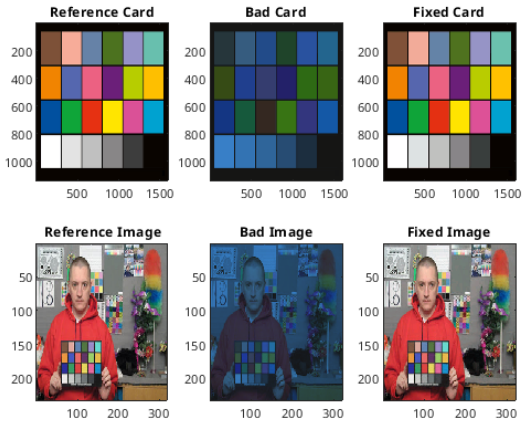
Because the bad images and bad test cards are produced with a predictably simple skweing and scaling of RGB channels, most of the results are easily restored to original through the `polyfit/polyval` method.



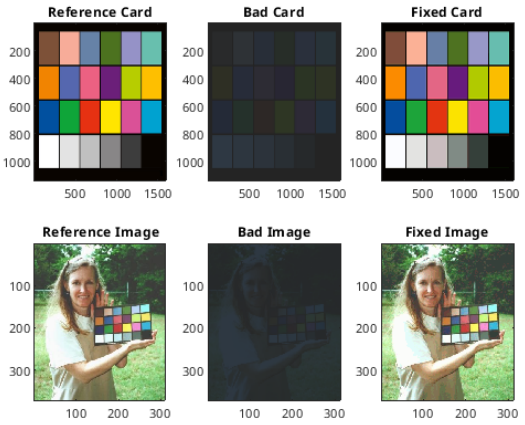
(a) First test in `chart_test.m`



(b) Second test in `chart_test.m`



(c) Third test using `color_person1.jpg`



(d) Fourth test using `color_person2.jpg`

Figure 2: Testing the color balancing capability of the `chart_correction` script.

1B Image Contrast Enhancement

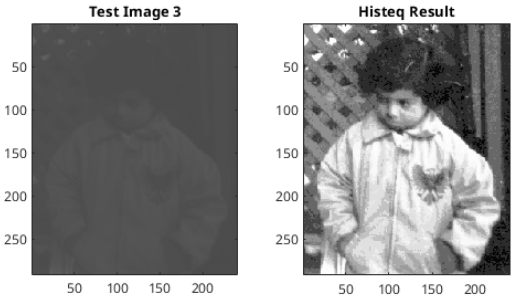
In this exercise, histogram equalization is undertaken, wherein the goal of the student is to reproduce a matlab script that can adjust the contrast of a grey-scale image and improve overall quality. Histogram equalization is a contranst enhancement technique wherein the intensity of the pixels would be evenly allocated to a histogram.

Following the steps outlined in the lecture notes as well as in the assignment document itself, the student implemented a `histeq_contrast` script that makes use of the cumulative distribution function (CDF) of the histogram of the image, and then uses the CDF to create the equalization map m. Results in testing are shown in the figures 3 and 4 within the next page.

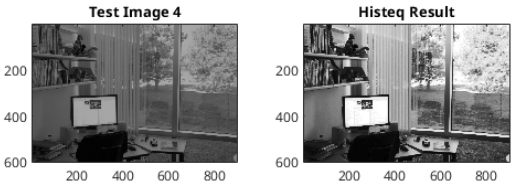
The images shown have manually damaged contrasts with manual scaling. The darker values of Figure 4d resulted in a reasonably sharper equalization. Histogram equalization performs remarkably well when enhancing dark gray levels, but may reduce or ruin the contrast in other parts of the image.

(a) First test in `histeq_test.m`

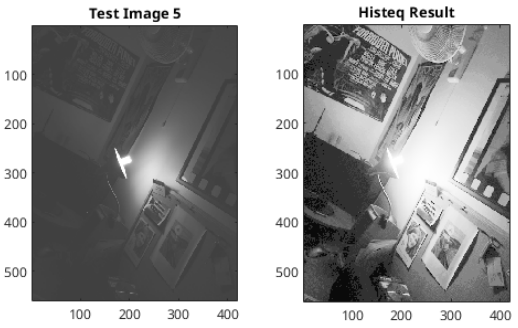
(b) Second test in `histeq_test.m`



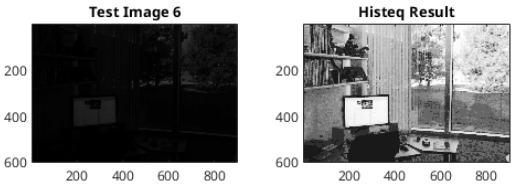
(c) Third test in `histeq_test.m`



(d) Fourth test in `histeq_test.m`



(e) Fifth test in `histeq_test.jpg`



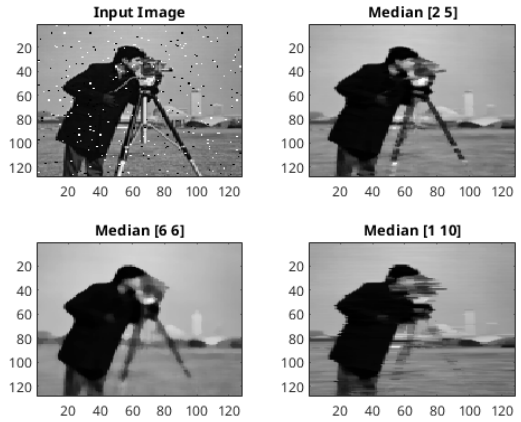
(f) Sixth test in `histeq_test.jpg`

Figure 3: Tests in `histeq_test`

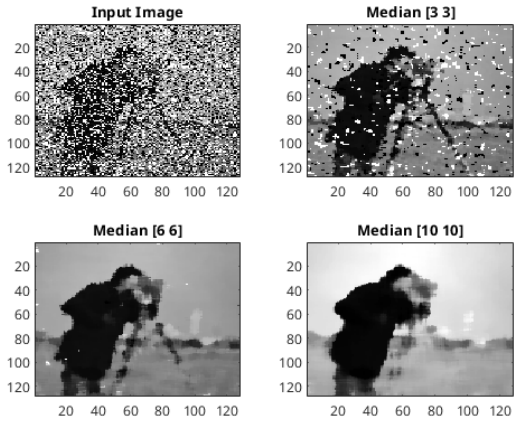
The test images in Figure 3 have manually damaged contrasts, resulting in a darkening effect on each of the images. While results are effectively similar, it is interesting to note that the lighter damaging as demonstrated by Figure 3e results in a histogram equalization that is somewhat too bright. Figure 3f is also worth mentioning, as it largely demonstrates how well histogram equalization performs for severely darkened images.

1C A Simple Median Filter

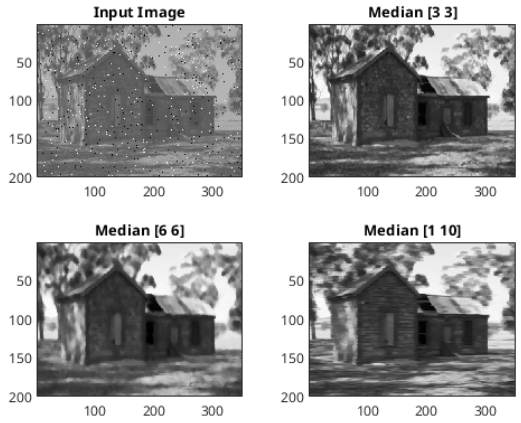
In this exercise, a simple median filter is implemented by the student without using built-in functions. The `median_filter` script takes three parameters: the image, M and N. The parameters M and N determine the size of the neighborhood pixels extracted and sorted for a median value. To handle pixels on the edge of the image, the input image is padded with zeroes. The median value is then assigned to a pixel in the output image, and this process is done for all the pixels. This is done for each pixel in the image. After filtering, the output image is converted back to the same class as the input image.



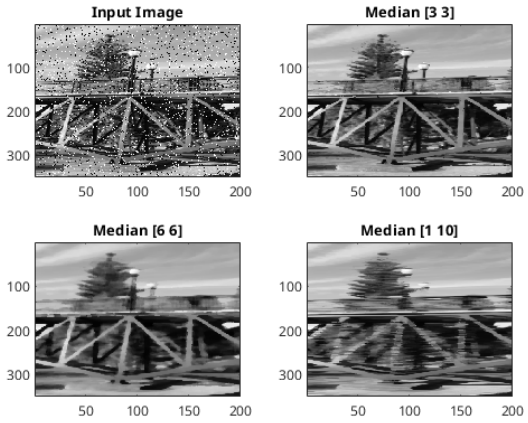
(a) First test in `median_test.m`



(b) Second test in `median_test.m`



(c) Third test in `median_test.m`



(d) Fourth test in `median_test.m`

Figure 4: Tests in `median_test`

As seen in Figure 4b, high density salt and pepper noise highly distorts the image. A Median filter with patch dimensions 10x10 is applied, but leftover artifacts of noise is still present.

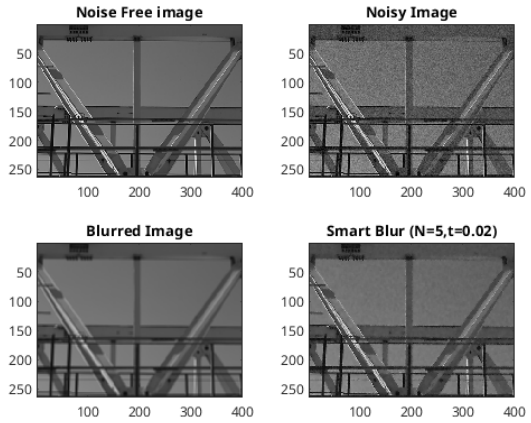
Question: Under what noise conditions does the median filter fail?

The median filter starts to fail with the increasing density of noise, most especially when the filter can barely extract neighborhood pixels from which it can normalize the image (because it has become all noise). Increasing the patch size for the filter may remove the salt and pepper noise due to the larger area to gather the median from, at the trade-off that the image becomes barely recognizable. Aside from this, the median filter fails on other types of noise, and is only good at removing salt and pepper noise.

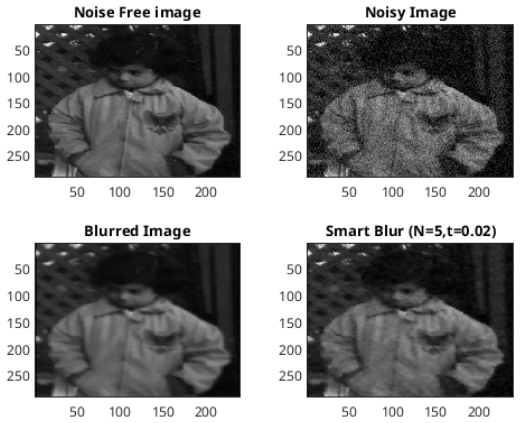
1D A "Smart" Edge Preserving Noise Filter

This exercise involves devising a "smart" solution to attempt to preserve edge details where possible in a noisy image, while following the step-by-step guide provided by the assignment manual. While it looked daunting with all the equations at first, it was surprisingly straightforward to implement the steps involved in creating the "Smart" edge preserving filter. The `smart_blur` script attempts to preserve edge details while reducing the effects of noise in an image by adjusting the amount of blurring applied to different parts of the image based on their gradient magnitude. The algorithm works by applying an NxN averaging filter to the image, then calculating the image gradients using Sobel filters. The gradient magnitude image is used to determine a weighting function for each pixel, which is used to construct a weighted combination of the blurred image and the original image.

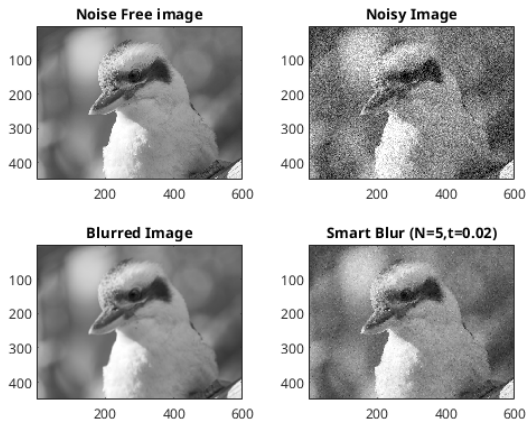
Testing results show small variation in values N for filter dimensions and t for tolerance, along with trying out stronger noise values.



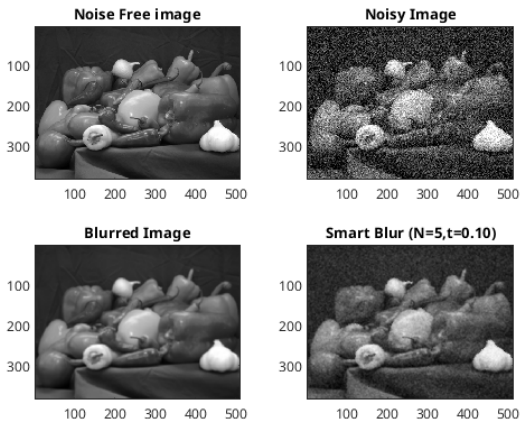
(a) First test in `smart_blur_test.m`



(b) Second test in `smart_blur_test.m`



(c) Third test in `smart_blur_test.m`, increased noise



(d) Fourth test in `smart_blur_test.m`, increased noise

Figure 5: Tests in `smart_blur_test`

The values of N and t control the amount of blurring applied to different parts of the image. Increasing N will result in a smoother image, while decreasing N will preserve more details but may not remove as much noise. Increasing t will increase the proportion of the original image used in the final result, while decreasing t will increase the proportion of the blurred image used in the final result. When N is too small, the resulting image may have residual noise artifacts, particularly in areas with high frequency content. When N is too large, the image may be overly blurred, and details such as edges may be lost. When t is too small, the resulting image may have visible blur artifacts, particularly around edges. When t is too large, the resulting image may have residual noise artifacts.

In general, the optimal values of N and t will depend on the specific image and the amount and type of noise present. It is necessary to experiment with different values of N and t to find the best settings for a particular image.

1E Written Questions

1. You have been given a set of imagery from a 256 x 256 pixel video surveillance camera to analyze. However, to work out roughly how far away objects of known size (e.g., people) are from the camera in the footage you need to determine the field of view of the sensor. Using a tape measure as a guide you are able to estimate that a doorway of 2 meters height appears to span around 32 pixels in the imagery when viewed from 10 meters away.

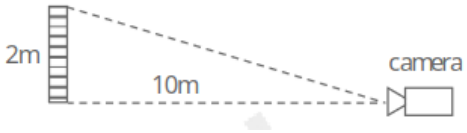


Figure 6: problem diagram

Q: Given the above, what is the likely field of view of the camera? (you may assume the vertical and horizontal fields of view are the same)

Given:
$h = 2m$, $d = 10m$, $p = 32px$, $n = 256px$, $f^0 = ?$

Solution:

$$h \approx pd\frac{2}{n}\tan(\frac{f^0}{2}\frac{\pi}{180})$$
$$2m = 32px \cdot 10m \cdot \frac{2}{256px}\tan(\frac{f^0}{2}\frac{\pi}{180})$$
$$2m = 2.5m \cdot (\frac{f^0}{2}\frac{\pi}{180})$$

Divide both sides with 2.5m and tan()

$$\tan^{-1}(0.8) = \frac{f^0}{2}\frac{\pi}{180}$$

Divide both sides with π and multiply both sides with 2 and 180

$$\frac{2 \cdot 180}{\pi}\tan^{-1}(0.8) = f^0$$

Simplify,

$$f^0 = 77.32$$

Q: How far away would a person of height 1.75 meters be if they appeared as a region of height 8 pixels in this imagery?

Given:
$h = 1.75m$, $d = ?m$, $p = 8px$, $n = 256px$, $f^0 = 77.32$

Solution:

$$h \approx pd\frac{2}{n}\tan(\frac{f^0}{2}\frac{\pi}{180})$$
$$1.75m = 8px \cdot d \cdot \frac{2}{256px}\tan(\frac{77.32}{2}\frac{\pi}{180})$$
$$1.75m = d \cdot 0.05000034303$$
$$\frac{1.75m}{0.05000034303} = d$$

$$\mathbf{d = 35m}$$

2. Carefully explain how a 'median' and 'alpha trimmed mean' filter work and describe under what circumstances they are useful (illustrate if required).

   Median filter is effective in removing salt and pepper noise while preserving the image's edges. It works by sliding a window or an N x M patch over the image, calculating and sorting the median value of the pixel values within that window. The median value is then assigned to a corresponding coordinate of an output image (when applying a median filter over an image, you loop through all the coordinates).

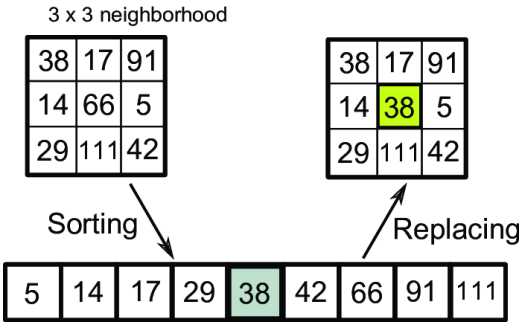


Figure 7: median filter illustration

The alpha trimmed mean filter is similar to the median filter but with an additional *trimming*, reducing the outliers in a noisy image. A filtering window is used to acquire the neighborhood of a pixel of interest. The extracted values are sorted, then the min and max values are trimmed. How much outliers are trimmed can be changed and are usually implemented via a parameter. A sample is shown below, with the trimming parameter = 1.

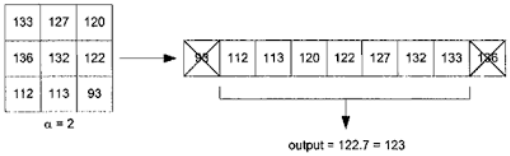
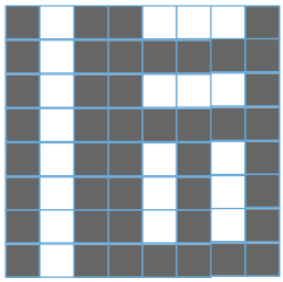


Figure 8: alpha trimmed mean filter illustration

Q: Without using a computer, what would be the result of applying a 3x3 and 5x5 median filter to the following simple image? (you may assume that white=1, black=0, and that all values outside the image boundaries shown here are also black ie. 0)



(a) Original simple image

0	1	0	0	1	1	1	0
0	1	0	0	0	0	0	0
0	1	0	0	1	1	1	0
0	1	0	0	0	0	0	0
0	1	0	0	1	0	1	0
0	1	0	0	1	0	1	0
0	1	0	0	1	0	1	0
0	1	0	0	0	0	0	0

(b) simple image to 1 and 0 representation

0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(c) 3x3 median filter applied

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(d) 5x5 median filter applied

Figure 9: manually applying a median filter

Q: Approximately, what would the result be if we instead applied a 3x3 alpha trimmed mean filter with d=3?

0	0	0	0	0	0	0	0
0	0	0	0	0.33	1	0.33	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0.67	0	0
0	0	0	0	0	0.33	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0.33	0	0
0	0	0	0	0	0	0	0

with $d = 3$, the top and bottom 3 values within a 3 x 3 neighborhood of each pixel will be trimmed.