Image Processing - Assignment 1

Kelvin Likollari

1. Tone mapping & Linearization

The original Ferrari image has been loaded and shown using the `imread` and `imshow` MATLAB functions respectively. The result of applying these operations is depicted below:

Figure 1: Original Ferrari Image

The following two images show the difference between the original and the linearized Ferrari image.

Figure 2: Original Ferrari

Figure 3: Linearized Ferrari

Specifically, the original image is an image that has been encoded using gamma correction, whereas, on the other hand, the linearized image is an image that has every element of it inverted in order to get rid of the gamma correction. Using linearization, the pixel values are restored, and the gamma correction is removed. By using linearization, the illumination of the image is enhanced, leading to a more bright image, but at the same time increasing the possibility of image artifacts. To derive those images, first, the image's pixels are normalized so as to have values in the [0, 1] interval. This conversion is performed so as to be able to apply gamma correction and obtain proper pictures. Next, the inverse gamma correction procedure is applied. This shall aid in enhancing the brightness of the Ferrari image. For the record, the dot in this operation denotes the fact that this operation shall be applied to every pixel value of the image.

```
gamma = 2.2
linear_image = double(img) / 255;
linear_image = linear_image .^ (1/gamma);
```

The next step is to apply some brightness increase and contrast enhancement. As the exercise description states, those operations can be applied simply by a linear multiplication and an exponential function respectively. Since the description does not state any numbers, the choice of the numbers has been done arbitrarily. The results are shown below:

Figure 4: Bright Ferrari

Figure 5: Contrasted Ferrari

2. Color Correction

In this section, a white balance method is implemented in order to an inputted image so that objects that are supposed to be gray, appear gray in the image. Effectively this can be regarded as color correction. The goal is to define the per-channel gain, which can yield the aforementioned color correction. There are two techniques acquired to find this gain, namely **Pixel-based correction** and **Gray-World assumption**. The following section contains detailed commentary and results, derived after applying a pixel-based

correction.

The script starts off by loading and demonstrating the original image. Following, a conversion of this image to the linear space and a gamma-correction operation to every pixel of the image is applied in order to obtain the proper results, necessary for the next steps. Next, the ginput() function is called and stored in a variable. This function is particularly helpful as when the image is loaded, it applies a cursor on top of it and lets the user express a pixel preference on which the color correction operations shall be applied. A modified version of ginput() has been chosen (ginput(1)), which allows us to choose one pixel of the image on which we'd like the color correction operations to be performed, compared to the original one which allows us to select multiple pixels. After the user has expressed a pixel preference, the coordinates of the pixels have been rounded to the nearest integer using the round() math function and not the ceil() or floor() functions, as on frequent occasions a different pixel might be chosen when applying them. After that follows the calculation of the average value of the R, G, and B channels of the selected pixel. The mean() function has been acquired for this use. Effectively, the mean is passed the 3D matrix (`linear_image`), which contains the 3 different RGB values, which in turn contains a 2D vector containing the (x, y) coordinates of our chosen image pixel. Effectively this line gets the average value of the RBG channels at the selected image pixel. After that, the color of the neighboring pixels should be interpolated in order to perform correct pixel color correction. The average color of the neighboring pixels is taken into account. The average function is called twice, so as to compute the average of the neighboring pixels. The reason it is called twice is that the first mean is supposed to be applied to the rows of the matrix, while the second one, is to the columns. Calling the average function just once shall return a scalar, and not a vector, as we intend. Following this, we use indexing to get the means in the 1st and 2nd dimensions and transform the matrix into a row vector. The gain for each channel is computed by applying a division to all the elements of the gray value with the channel means. In the last steps, to see the actual pixel correction, every pixel in the (`linear_image`) is multiplied by the gain value of each RBG channel. In the end, after applying the gain to each color channel, we should not forget to apply inverse gamma correction, so as to get the correct and aesthetically pleasing visual output.

The concept is pretty much identical for **Gray-World assumption**. The difference between **Pixel Correction** and **Gray-World assumption** is that when using pixel correction, depends on the image pixel the user will choose, whereas, as the name itself states, gray world states that on average the color of the image is gray.

The images generated are the following:

Figure 6: White Balance

Figure 7: Pixel Image

Figure 8: Gray-World

As seen from the image, the pixel-based correction provided slightly better results than the gray world. The reason why pixel-based correction performs better is that it performs the correction on each pixel separately, taking into account its color and the neighbor pixel color values. The precision and the accuracy of the computation and the correction are increased when dealing with one pixel at a time, unlike the gray-world assumption, which is more general, and considers on average that the color is gray. The number of operations applied to the gray-world assumption is significantly less than the one applied to the pixel-based correction, hence the poorer image quality.

3. Histograms

The histogram for the Ferrari image for each of its 3 channels is the following:

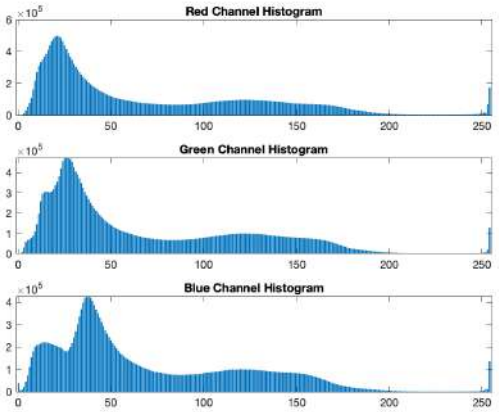

Figure 9: Histograms

A possible comment that can be made here is regarding the number of bins chosen. We know that the histogram represents the number of pixels illuminated and to what extent. That illumination varies from 0 for total black to 255 for total white. Hence, we have 256 different levels of brightness. We can express those 256 different brightness levels as a power of 2, meaning $2^8 = 256$. This is ideal for our 8-bit images.

4. Histogram Equalization

(i) Histogram equalization is a widely-used technique, acquired to manipulate images. Effectively pixel intensities across the image are redistributed so that the contrast can be improved. However, in the scenario in which the given image is a compressed image, this might cause some unwanted artifacts on a plethora of occasions. Compression effectively means *resize to smaller*, and this yields a loss of image pixels and their distribution in the image itself. This affects at the same time the histogram equalization of the compressed image, yielding severe image artifacts. Even though histogram equalization is supposed to make the images more aesthetically pleasing to the viewer, if compressed, it may result in information (regarding pixels) loss and in image distortions.

(ii) After equalization, the histogram is supposed to have its pixels more evenly distributed across its entire length. The pixel intensities shall be redistributed across the whole span of the image, resulting in each intensity level having more or less the same number of pixels, yielding a greater contrast compared to the original image, and more aesthetically pleasing to the viewer.

The following image depicts the enhanced Ferrari image after having applied global histogram equalization to it.

Figure 10: Global Histogram Equalization

For the local histogram equalization scenario instead, 2 window sizes have been selected, namely window sizes of 10 and 100. For window sizes of 64, 128, and 256, check the respective MATLAB file and the images generated from it. The results are shown below.

Figure 11: Local Histeq wSize = 10

Figure 12: Local Histeq wSize = 100

As seen from the images depicted above, global histogram equalization performs better than local histogram equalization. In particular, global histogram equalization aims to redistribute the image as a whole and not do local optimizations in some particular pixel area of the image. Essentially, global histogram equalization gives out more *image-contrast*, unlike the local one which technically the images generated are identical to the original provided image. In global histogram equalization, the pixel values are trying to be spread across the whole image span, unlike the local one which operates on small pixel regions of alternating pixel intensity. The results might not be ideal, as compared to the global one. Moreover, in local histogram equalization, as the window size increases, the contrast increases too, but the downside of this scenario is that the computation becomes expensive, and the image sizes are of a relatively high size. Therefore it is noticeable that the global histogram equalization is simpler to implement and at the same time produces better and more efficient results.

Regarding the locally adaptive method, it is also very efficient in performing histogram equalization. The newly generated image is significantly enhanced, meaning the shadows are lighter and the contrast is increased. We can control how much contrast we want by toggling the value of the *contrast* variable.

Figure 13: Enhanced image

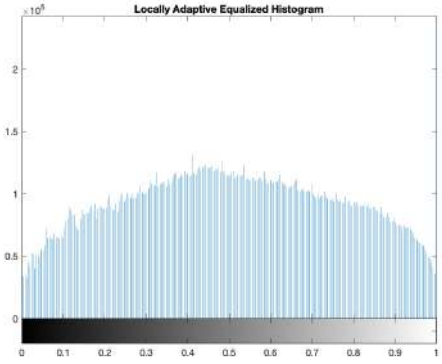

Figure 14: Locally Adaptive Histogram

The benefits of this method among others typically include significantly enhanced images, and efficient results but on the other hand, it comes with some disadvantages such as expensive computation and a significant increase in image file sizes. Note that the specific selection of the contrast variable has been selected as 0.15 for testing reasons. You may try to toggle its value to obtain different enhancement and contrast results.

5. Manual Histogram Equalization

The formula for the standard histogram equalization is:

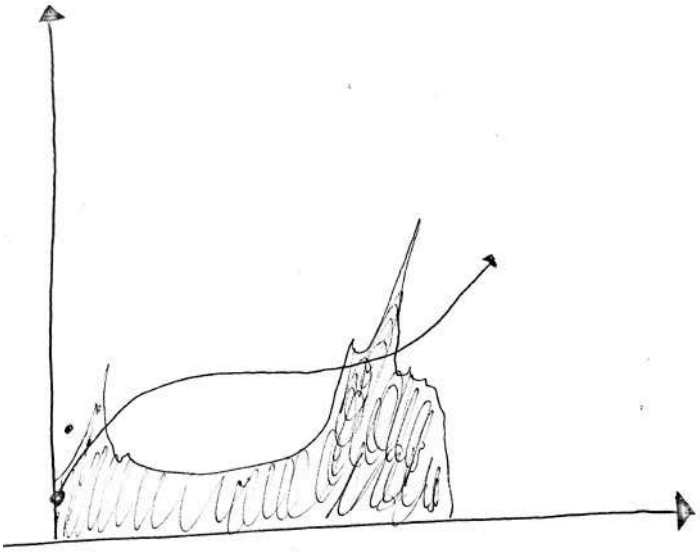

Figure 15: Exercise 1.5

6. Thresholding and Matting

In this section, we are filtering the Ferrari image based on its background color and we are obtaining a mask. This mask is being used to essentially replace this image with the cat image. Although the two images are not suitable for these kinds of operations, the results are still visible. The threshold value can be toggled appropriately depending on how much you want the cat image to appear over the Ferrari one. The mask is applied to the Ferrari image, and values outside the mask are colored black. Following, the background pixel values are inverted, which is useful as we can see then which pixel values are inverted and be able to place our new image there. However, the newly added image should be resized so that it matches the original image's size. Lastly, the inverted mask is applied to the background image so as to extract the background Apply the inverted mask to the background image to extract the background of it. In the last step, to obtain the final image, the 2 images are added together.

The image generated is the following:

Figure 16: Thresholding and Matting

7. How many bits is enough?

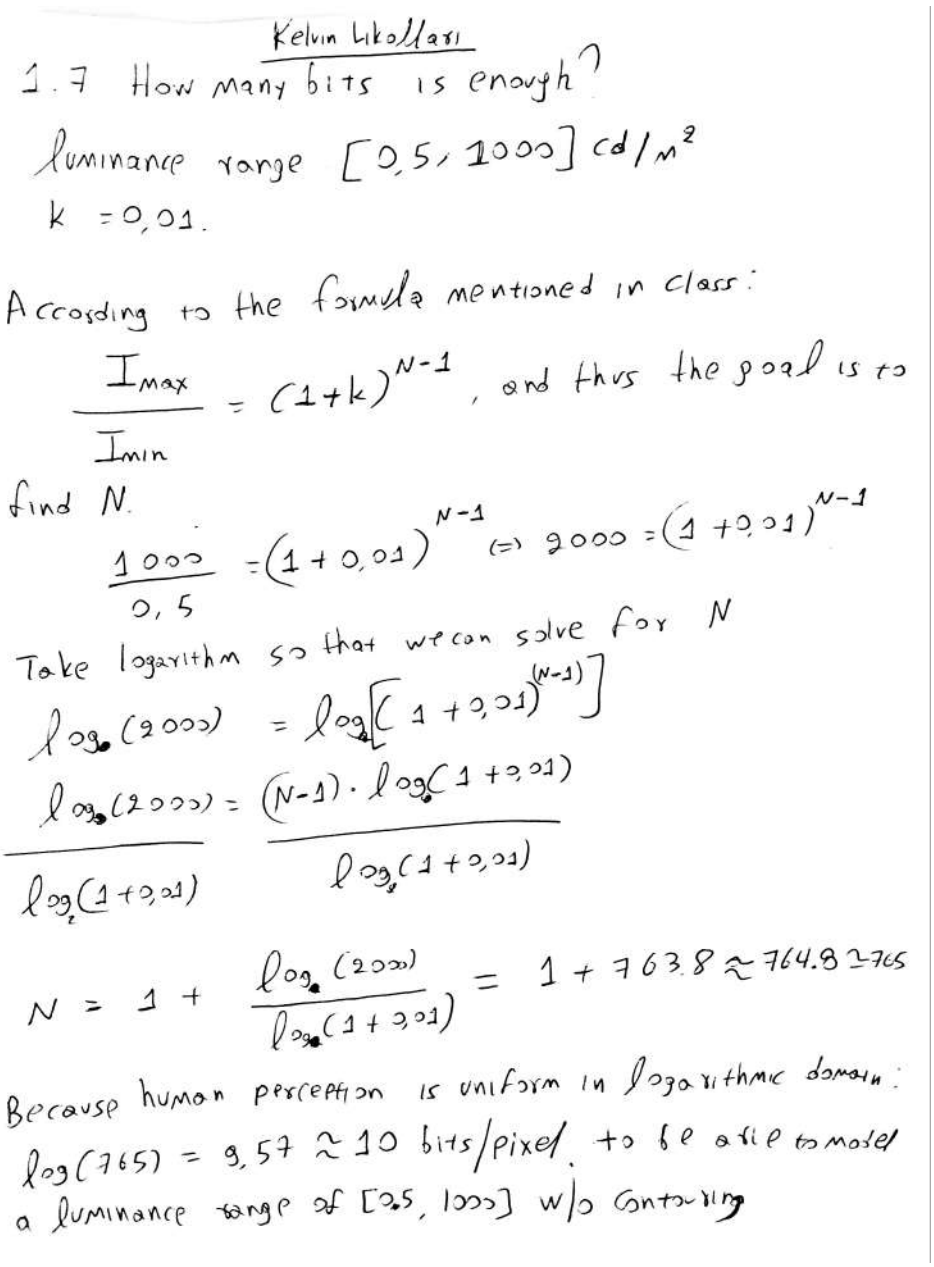

Figure 17: Exercise 1.7

Therefore, 10 bits/pixel is needed to be able to store the grayscale version of the gradient image.

8. Bonus: use your own pictures

For the bonus part of the assignment, I choose to apply some color enhancement, rotations, and contrast enhancement and finally average the pixels of 2 images taken at relatively the same place. The results are depicted below.

Figure 18: Contrast

Figure 19: Rotated Image

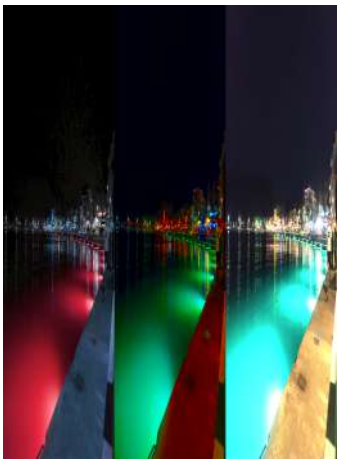

Figure 20: Color

Lastly, I searched through my gallery and managed to find 2 relatively high-quality pictures taken at relatively the same place. The results of averaging the pixels of those 2 images are shown below. What is worth noting is a bit of distortion and artifacts in the images as they are not taken in exactly the same position. If they actually were, then the distortion and artifacts would have been at minimal levels.

Figure 21: Averaging Images

More image processing operations have been added to the *bonus.m* file.