Introduction to Digital Image Processing

Homework 1

October 15, 2020

Student:	Ivan Arevalo
Perm Number:	5613567
Email:	ifa@ucsb.com

Department of Electrical and Computer Engineering, UCSB

Problem 1: Image and video data requirements

a) *Calculate the data size (in bits) of the following:*

$$N_x \times N_Y \times N_Z \times N_{Time} \times N_{Channels} \times Bitdepth$$

- 1024×1024 color (RGB) image with 8 bits per pixel per color channel

$$1024 \times 1024 \times 1 \times 1 \times 3 \times 8 \text{ bits} = 25,165,824 \text{ bits}$$

- $512 \times 1024 \times 64$ microscopy volume, single channel, with intensity quantization to 4096 gray levels

$$512 \times 1024 \times 64 \times 1 \times 1 \times (\log_2 4096 = 12\text{bits}) = 402,653,184 \text{ bits}$$

- 1920×1080 color (RGB) video at 30 frames per second, at 8 bits per pixel per channel, running for 10 seconds.

$$1920 \times 1080 \times 1 \times 30 \times 3 \times 8 \text{ bits} \times 10 \text{ seconds} = 14,929,920,000 \text{ bits}$$

b) *How long will it take to transmit the above video sequence using a mobile phone whose transmission speed is* 2.5×10^5 *bits/s?*

$$\frac{14,929,920,000}{2.5 \times 10^5 \text{bits/s}} = 59719.68s = 16.5888h$$

c) *Finally, consider this video running continuously (not restricted to 10 seconds), what transmission speed in bits/s would be required of your transmission channel to enable you to communicate this video in real time (without delay)?*

$$1920 \times 1080 \times 1 \times 30 \times 3 \times 8 \text{ bits} \times 1 \text{ second} = 1,492,992,000 \text{ bits/s} + \text{some margin}$$

Problem 2: Contrast stretching with paper and pencil: Consider the following image

$$\begin{bmatrix} 20 & 35 & 11 \\ 51 & 9 & 23 \\ 23 & 11 & 60 \end{bmatrix}$$

a) *Find a linear pixel value transformation of the form* $g(r) = ar + b$*, or if you prefer the linear form* $g(r) = a(r - b)$*, where* r *stands for the pixel value, such that when it is applied to all pixels in the image, the resulting image will have dynamic range of* $[0, 255]$*. Recall that this simply means that the lowest value is 0, and the highest is 255. Please explain your approach before jumping into nitty gritty calculations.*

We first want to find the dynamic range of our image which is between $[9, 60]$. Next, we want to push the lower bound of our dynamic range to 0. We do this by subtracting each pixel value by this lower bound. Last, we want to scale the resulting dynamic range so our upper bound is at 255.

$$g(r) = r - 9 \quad \rightarrow \text{dynamic range: [0, 51]}$$
$$g(r) = \frac{255}{51}(r - 9) \quad \rightarrow \text{dynamic range: [0, 255]}$$

b) *Find the output image after applying the linear transformation.*

$$\frac{255}{51}\left(\begin{bmatrix}20 & 35 & 11\\ 51 & 9 & 23\\ 23 & 11 & 60\end{bmatrix} - 9\right) = \begin{bmatrix}55 & 130 & 10\\ 210 & 0 & 70\\ 70 & 10 & 255\end{bmatrix}$$

c) *Suppose now that you wanted to apply a linear transformation that achieves full dynamic range as before, but at the same time creates a negative image (which means that the lowest value pixel before transformation is mapped to 255, and the highest pre-transformation value is mapped to 0.) Find this linear transformation. Again, first explain your approach before crunching numbers.*

Once we have applied the linear transformation necessary to maximize the dynamic range of the image as shown above, we can now apply a second linear transformation to create a negative image. We can do this by subtracting our image matrix from a same size matrix with all values equal to 255.

$$\begin{bmatrix}255 & 255 & 255\\ 255 & 255 & 255\\ 255 & 255 & 255\end{bmatrix} - \begin{bmatrix}55 & 130 & 10\\ 210 & 0 & 70\\ 70 & 10 & 255\end{bmatrix} = \begin{bmatrix}200 & 125 & 245\\ 45 & 255 & 185\\ 185 & 245 & 0\end{bmatrix}$$

d) *This problem setting was contrived to avoid unnecessary trouble. In real life, which is not often as friendly as your ECE 178 homework, you will typically find that your approach results in non-integer values for the pixels. A natural solution would be to simply round the pixel values to the nearest integer. Explain why the overall transformation will then not be strictly linear, but only "approximately linear".*

A function is a linear transformation if it satisfies the following two conditions:

$$f(\mathbf{u}+\mathbf{v}) = f(\mathbf{u}) + f(\mathbf{v})$$
$$f(c\mathbf{u}) = cf(\mathbf{u})$$

If we round the pixel values to the nearest integer, then these two conditions won't be satisfied. We can say the transformation is approximately linear because the perturbation from the left and right side of each condition will be small.

Problem 3: Optics

You were hired to install a fixed camera at appropriate distance from the base of the Eiffel tower in Paris, which is 300 meters tall. The camera lens has focal length of 20 mm and the custom made CCD sensor active area is 21 mm (height) × 16 mm (width), with 4200 × 3200 pixels. (We assume "portrait" setting.)

a) *How far from the base of the tower do you need to position it so that the tower is about 3150 pixels tall on the array?*

Given thin lens law (eq. 1) and the magnification law(eq. 2), we can solve for the distance from the base of the tower to the lens in order for the tower to be about 3150 pixels tall on the array.

$$\frac{1}{d_o} + \frac{1}{d_I} = \frac{1}{f} \tag{1}$$

$$M = \frac{y_I}{y_O} = -\frac{d_I}{d_O} \tag{2}$$

Solving from lens law (eq. 1), we can solve for d_I in terms of d_O and f.

$$\begin{aligned}
\frac{f}{d_o} + \frac{f}{d_I} &= 1 \\
\frac{f}{d_I} &= 1 - \frac{f}{d_O} \\
\frac{f}{d_I} &= \frac{d_O - f}{d_O} \\
d_I &= f\frac{d_O}{d_O - f}
\end{aligned}$$

We can observe that for $d_O >> f$, d_I is approximately equal to f. Plugging back into equation 2, we can solve for d_O.

$$\begin{aligned}
M = -\frac{d_I}{d_O} &= -\frac{f d_O}{d_O(d_O - f)} \\
M = -\frac{d_I}{d_O} &= -\frac{f}{d_O - f} \\
d_O &= f - \frac{f}{M} \\
M = \frac{y_I}{y_O} &= \frac{(\frac{3150}{4200})21mm}{300m} = 5.25 \times 10^{-5} \\
d_O &= 20mm - \frac{20mm}{5.25 \times 10^{-5}} = -380.932m \text{ wrt lens}
\end{aligned}$$

b) *The camera will also serve as a security camera, and given the pandemic, it has recently been tasked with verifying that social distancing of 2 meters is maintained by all visitors. This is done by checking the number of pixels of separation be- tween persons standing at the base of the tower, as they appear on the sensor (i.e., on the image). How many pixels of separation would be required so as to not trigger the alarm?*

From equation 2, we can derive the image distance that corresponds to 2m and find the number of pixels that represent this distance.

$$\begin{aligned}
M &= \frac{y_I}{y_O} \\
y_I &= M y_O = 5.25 \times 10^{-5} \times 2m = 1.05 \times 10^{-4} m \\
1.05 \times 10^{-4} m &\times \frac{3200 pixels}{16mm} = 21 pixels
\end{aligned}$$

Problem 4: Image Contrast Stretching

Let us now revisit the problem of contrast stretching but with a real image. Consider the image posted with this homework: peppers poor contrast.png

a) *Stretching the image to full dynamic range: Write a program that finds and performs a linear transformation of the form discussed in Problem 2 (either g(r) = ar + b, or if you prefer the form g(r) = a(r - b), where r stands for the pixel value) such that when it is applied to all pixels in the image, the resulting image will have dynamic range of [0, 255]. Word of caution: If working with matlab, images are loaded as uint8 by default when using imread. Cast the images to appropriate data type (e.g., double precision) before performing the contrast stretching, and then back to uint8 before saving the output image. Save and submit the resulting image in format: YourName- HW1-P4A.png*

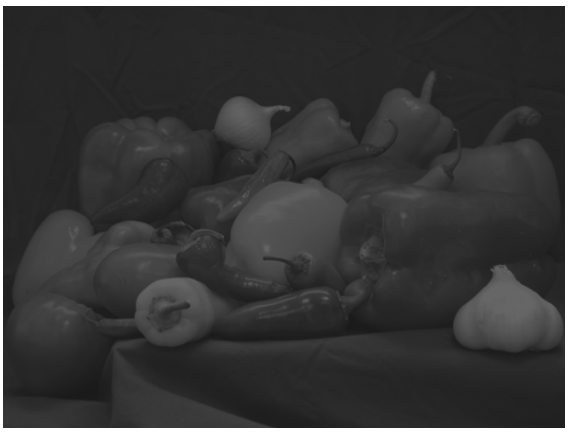

Figure 1: Original Image

Figure 2: Contrast Stretched Image

b) *Contrast stretching in the negative direction: apply another linear transformation to the original image but this time, the transformation should map the minimum pixel value in the image to 255 and maximum to 0. (Again, watch out for data type conversions when loading, transforming, and saving the image). Save and submit the resulting image in format: YourName-HW1-P4B.png*

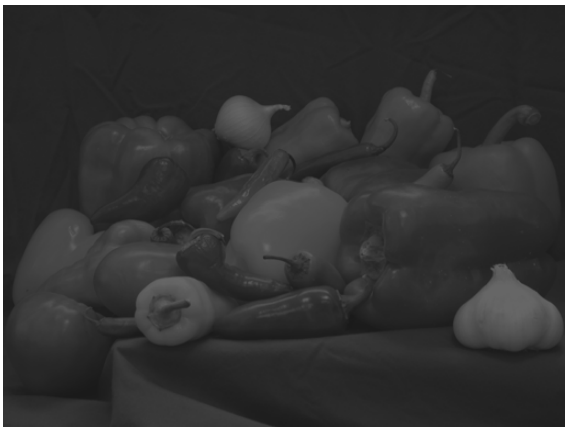

Figure 3: Original Image

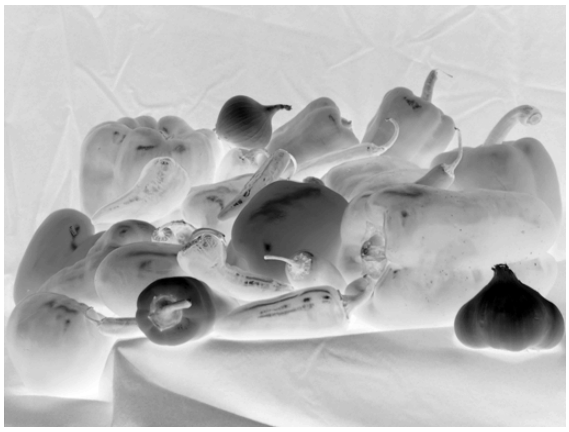

Figure 4: Negative Contrast Stretched Image

c) *Quantify the rounding error: referring to your response to Problem 2d, which concerned the need to round pixel values of contrast-stretched images to the nearest integer. For parts (a) and (b) of this problem, compute the root-mean-squared error between the images before and after this rounding operation.*

The RMSE for the contrast stretched image was 0.28746762803081816.

The RMSE for the negative contrast stretched image was 0.2874676280308208

Code

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

# Generate contrast stretched image
peppers_img = Image.open("peppers_poor_constrast.png", )
peppers_img_pxl = np.asarray(peppers_img)
im_min = np.min(peppers_img_pxl)
im_max = np.max(peppers_img_pxl)
contrast_stretch_image = np.asarray((255 / (im_max - im_min)) * (peppers_img_pxl -
    im_min))
approx_contrast_stretch_image = np.round(contrast_stretch_image).astype(np.uint8)
plt.imshow(approx_contrast_stretch_image, cmap='gray')
plt.imsave('IvanArevalo-HW1-P4A.png', approx_contrast_stretch_image, cmap='gray')

# Generate negative of contrast stretched image
negative_im = 255 - contrast_stretch_image
aprox_negative_im = 255 - approx_contrast_stretch_image
plt.imshow(negative_im, cmap='gray')
plt.imsave('IvanArevalo-HW1-P4B.png', aprox_negative_im, cmap='gray')

# Calculate rounding error
error_contrast = np.sqrt(np.mean(np.square(contrast_stretch_image -
    approx_contrast_stretch_image)))
print("The RMSE for the contrast stretched image was {}".format(error_contrast))
error_negative = np.sqrt(np.mean(np.square(negative_im - aprox_negative_im)))
print("The RMSE for the negative contrast stretched image was {}".format(error_negative))
```