Introduction

We had been tasked with creating a program that can execute various functions for processing images. The tasks focused on the 4 main topics: histogram specification and matching, gaussian and Laplacian pyramid construction, application for image blending, and final filtering of an image in the frequency domain. In this report I plan to outline the results of my implementation and discuss my findings from translating the theory to practical implementation.

Design Approach

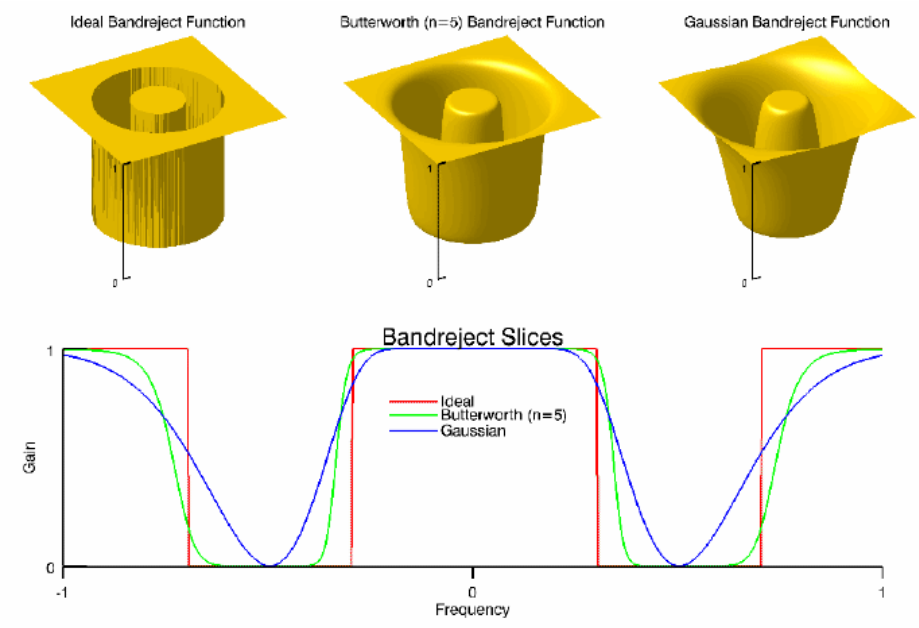
For my approach I decided to implement my function in a class. I originally did this so that I could store my global variables in a safe manner. As I implemented more of the functions from the problem, I found that the class was more useful as a library of functions rather than a module with functions because a library of functions allows a user to import the class and have access to the independent functions and the variables required for each were independent.

For my histogram specification function, I broke it up into three parts. One helper function generated the desired histogram for transforming our image and another helped find the closest matching intensity relating to the image. These functions were then called by my main "hist_match(img)" function which takes in an image as a parameter, generates its intensity histogram, then converts it to an equalized histogram with a normal distribution. The program also equalizes the desired histogram providing us with the inverse function we used to match the given image's histogram.

For my gaussian and Laplacian pyramid function and my image blending function I followed a similar technique. Each function takes in an image and the max height of the desired pyramid to be constructed or used. If the user does not provide a max height, the function will reduce until one of the axes is of size 1. This uses a for loop set to the max height to recursively process the image, append it to a python list, and down sample the image to be processed again and to the smaller size. For the Laplacian pyramid function, this process requires creating a blur image from the original image and subtracting the blur to create an image containing only the edges and sharp details. The gaussian pyramid function can down-sample the image or blur the image and then down-sample it, resulting in similar effects. The image blending function makes use of both the gaussian and the Laplacian pyramids. We generate the Laplacian image for our two images and a gaussian pyramid for our region image which determines which pixels will come from which region. We then go through our Laplacian images and reconstruct it using pixels from the selected region. Once completed, we start from the smaller image, recursively sample the image, and add the sharpness from our blended Laplacian. This results in an image with sharp edges of both images and a well-blended blur where sharpness is not necessary.

For my frequency filter function, I made use of OpenCV's filter function to help apply my kernel/mask to my image. My function focuses more on constructing the kernel that is applied to my image while accounting for padding to prevent wraparound error. To generate my kernel,

I used the function provided for gaussian band reject filtering, looped through each pixel in my kernel, and assigned it the value related to that pixel which depended on its distance from the center and the parameters passed through. The C_not parameter centers the band reject filter at a certain frequency while the W parameter controls the width of the gap between the band. I have attached an image below for clarity.



Program Interface:

mt= Midterm()

Image1 = cv2.imread("path/to/image1")
new_image_p1 = mt.hist_match(image1)

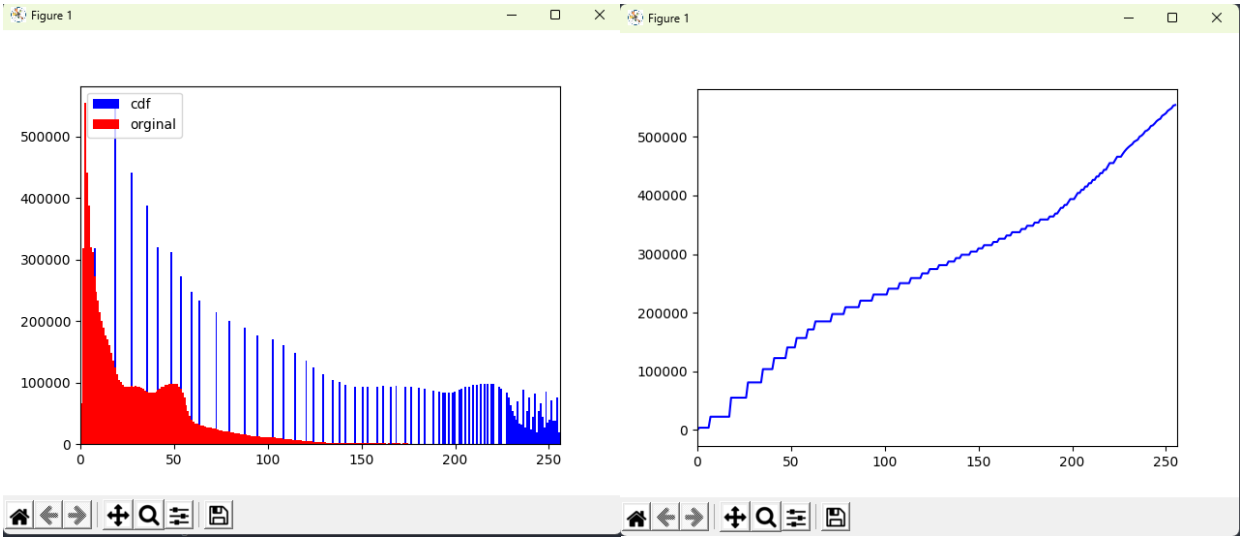
max_level = 3
new_gp_list_p2 = mt.gaussian_pyramid(image1, max_level)
new_lp_list_p2 = mt.laplacian_pyramid(image1, max_level)
image2 = cv2.imread("path/to/image2")
new_image_p2 = mt.image_blend(image1, image2, max_level)

P = image.shape[0]+padding
Q = image.shape[1]+padding
C_n = 200
W = 11500
new_image_p3 = mt.hist_match(image1, P, Q, C_n, W)

Experimental Results P1

Original & Filtered Image

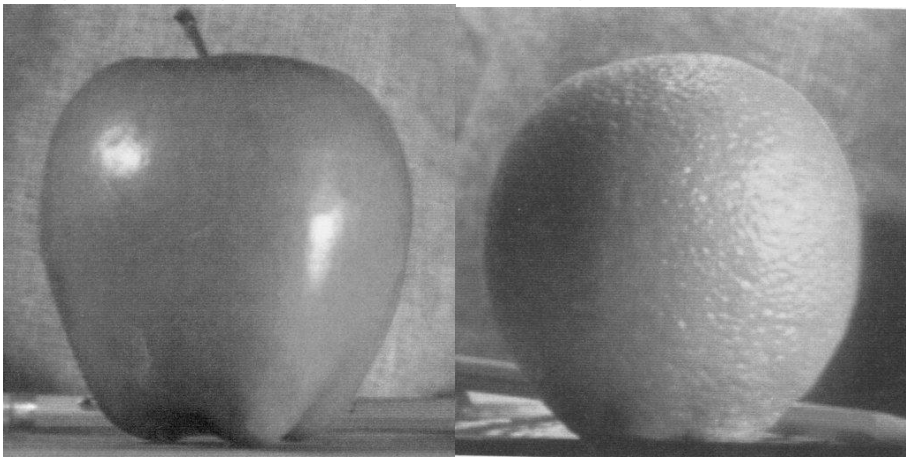
Original & Filtered Image Histogram, and Intensity Transformation function



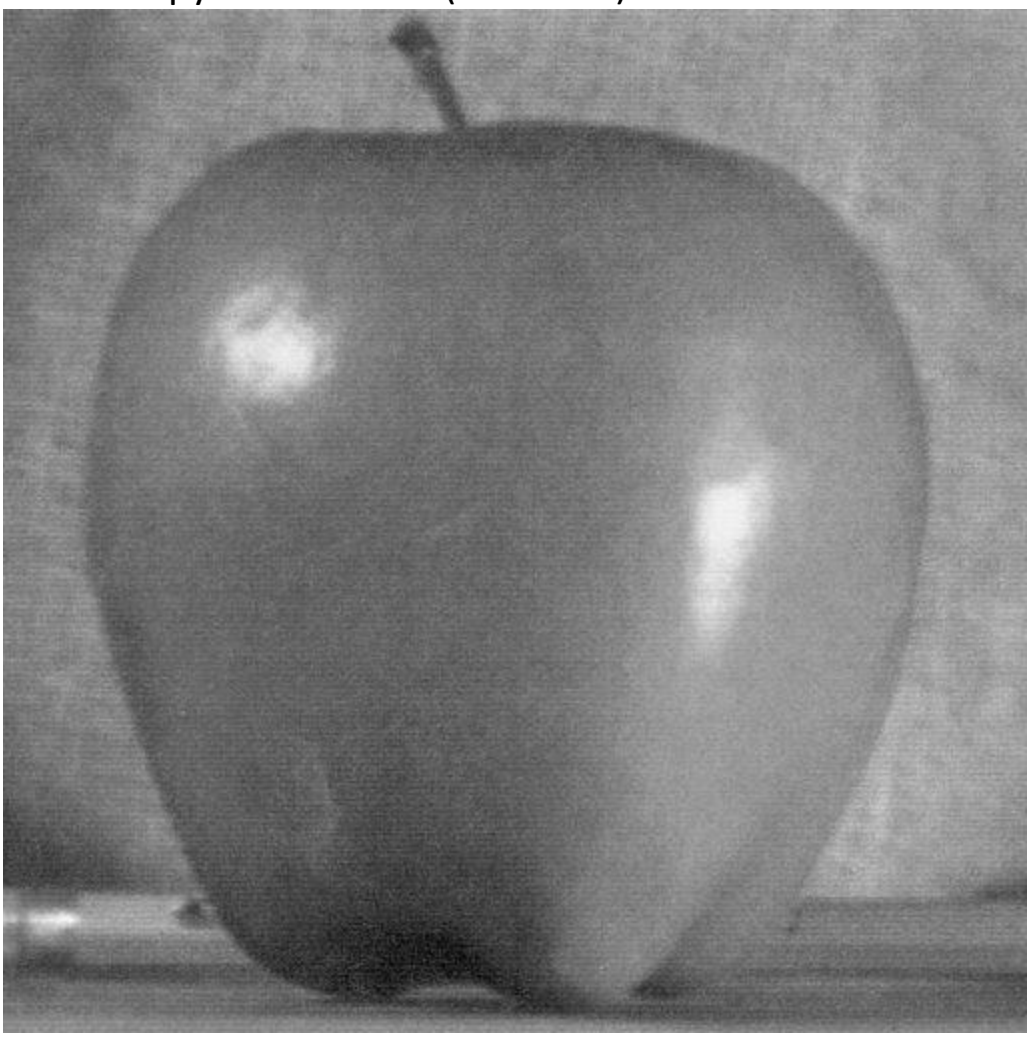
I believe the results from problem 1 were successful. From the from the Original and filtered image histograms it is clear to see how the histogram matching function uses the desired histogram to match and stretch top the image. The result led to a more balanced spread of intensities while also finding the next closest intensity for pixels some pixels the 250 range of intensity values.

Experimental Results P2

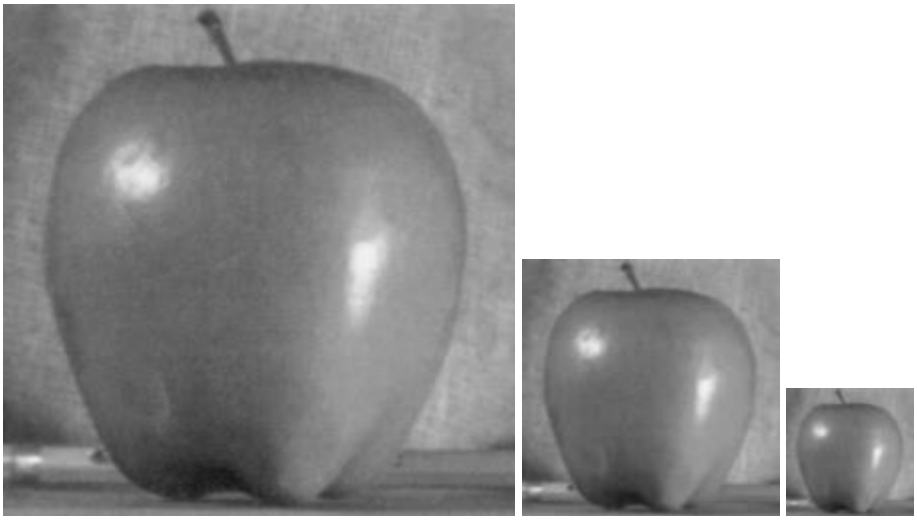
Original Images of Apple & Orange



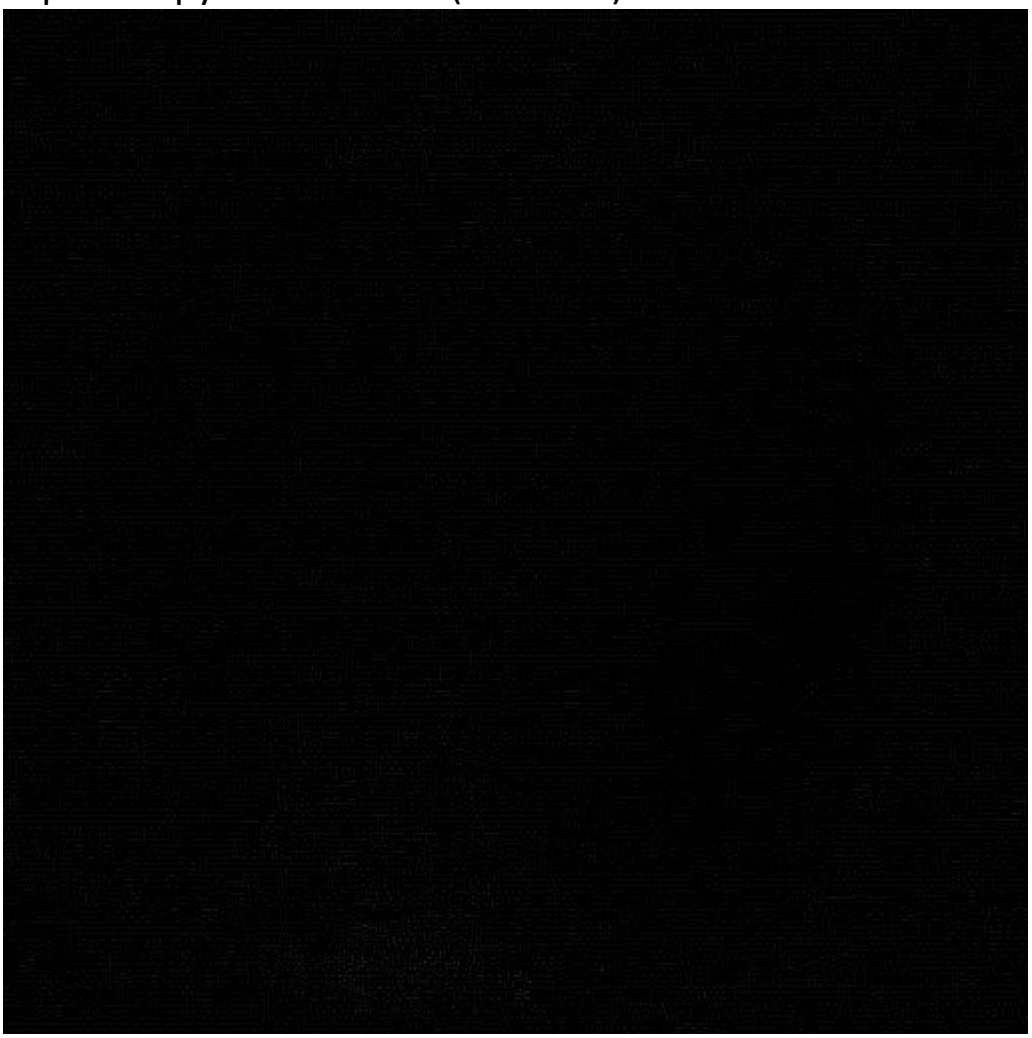
Gaussian pyramid level 0 (512x 512)



Gaussian pyramid level 1-3 (256x256, 128x128, 64x64)



Laplacian pyramid level 0 (512x512)



Laplacian pyramid level 1-3 (256x256, 128x128, 64x64)

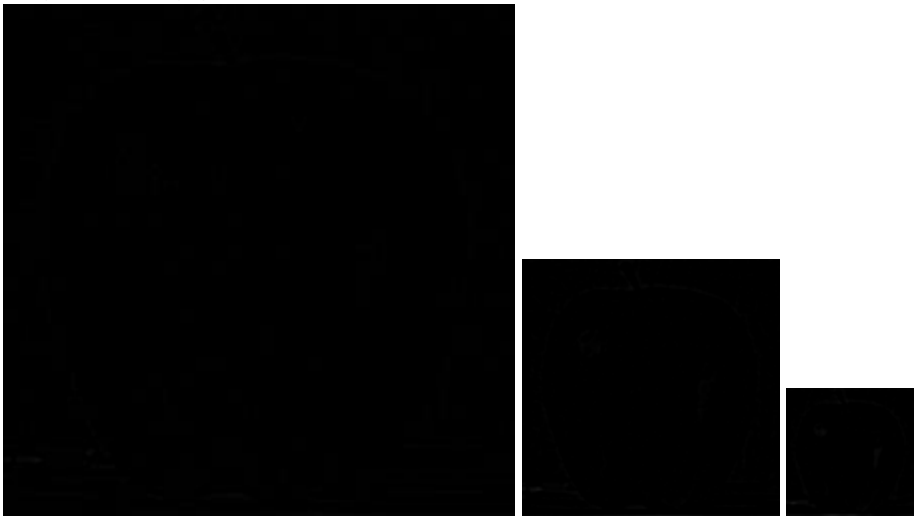
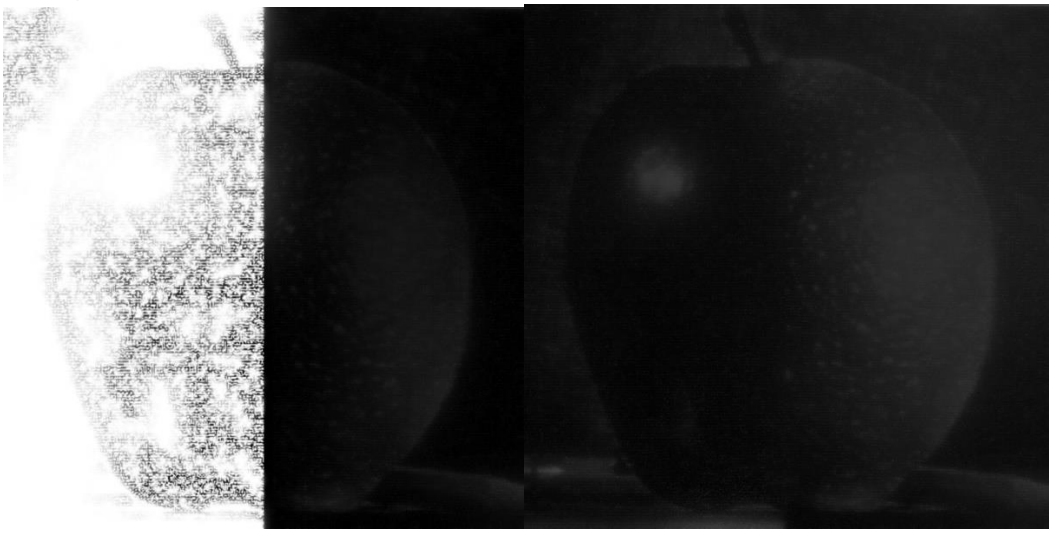


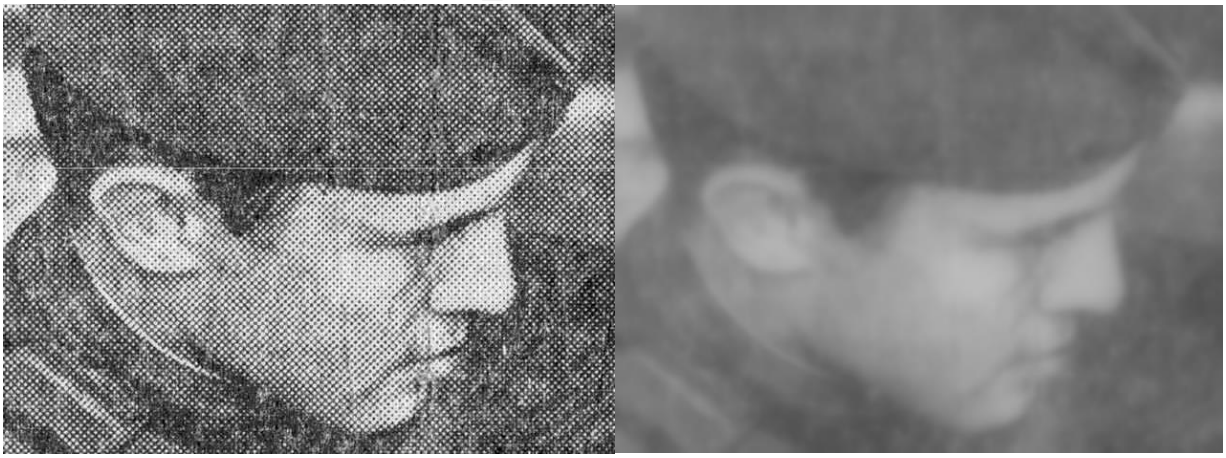
Image Blend Results



I believe that the results from the gaussian and Laplacian pyramid were successful. While it is hard to tell directly from looking at the images, I was able to conform their behavior through first ensuring the dimensions of images at higher levels of the pyramid were have that of their previous image. This was satisfactory for the gaussian pyramid as this would be is most visible display of its effect as the blurred image keeps its quality due to its reduced size. For the Laplacian this was trickier as the information contains the sharpness of the image and noise can also be considered sharp ness in an image, so I was unsure on whether I was looking at pure noise due to a bug in my program or the proper image. I was able to confirm this was correct by apply my Laplacian function in my image blend function those if the information being stored was just pour noise the blended image would not be recognizable. From my image blend results I could tell that the napoleon was working as the details hand blended correctly and both images sharp features where still clearly distinguishable. One thing worth noting for the implementation of the image blend is that using the mask worked best when selecting which region should be viable based off the mask compared to trying to multiple the images by a mask for its inverse as this added unwanted brightness ruining the effect.

Experimental Results P3

Original & Filtered Image (C_not = 200, W=11500)



I believe that my frequency filtering was successful in removing unwanted artifacts however after processing the image last some detail and became slightly blurred. This could be fixed with a high pass filter to help add more sharpness to the image.

Conclusion

In conclusion, all 3 problems highlighted powerful theories for image enhancement and manipulation but require more nuance in real-life application. I believe that my results, while not perfect, accurately demonstrated the behavior and effects of each function.