EE 5620 Project 03

I. Cody Hume and Corban Boyd

October 22, 2022

Abstract
This document provides the purpose, methods, and results of the project 03 assignment for EE 5620. The code written and the images of the results are shown. Each task given will detail how the results were found and what methods were used to find it.

Introduction

The purpose of this project is to become more proficient with spatial domain image enhancement techniques as presented in Chapter 3 of the textbook, and how they can be combined. Mainly, this will include the mask program and how it can be applied to a Laplacian, Sobel operator, and smoothing techniques.

Methods

This section will outline how each part of the process was completed. In addition, the report questions and discussion prompts will be addressed here.

The first task of the project was to create a masking program that accepted two parameters: the image that will be masked and an array of any odd size (3x3, 5x5, 7x7, etc) called,

$$ip\_mask(A, maskArray)$$

This task is the foundation to what the rest of the project will do. This function takes in two parameters being the image, A, and maskArray which is to be set by the user. The method as to which the code was built began by taking the given image and getting the size of it. It will then create a new matrix which will be used as the final matrix using the original size of the image. The next step is to pad the image with zeros. This was done by taking the dimensions of the maskArray and subtracting one and dividing it by two. This equation will work for any odd number mask given. Since the zero padding needs to be around the perimeter of the image, this value is multiplied by two to give it padding on all sides. Next, a new matrix is created with all zeros of that size and the original image is put into that matrix using a double for loop. The last part of the code is the "meat" of the function. The given mask array is multiplied by a section of the exact same size of the array. For example, a given 3x3 mask will multiply by a 3x3 section of the padded image. Then the total of all pixels in that matrix is summed and put into the final matrix created in the beginning. Using a double for loop, that value is calculated for each pixel and finally imaged in the end. This function was tested using test images given on the course website and verified on MATLAB. Images here will not be shown in the results according to the instructions on the project description. The purpose of this step is to create a mask function that will be used for the rest of the project. This step achieved its purpose very well as it always was used to enhance the image of the skeleton. In addition, it can be used in the future as it is a very versatile function. Refer to code listing 1 for code.

Now that the foundation of the project is created, the next task was replicating image enhancement steps in Figure 3.63 (b) – (h). To do this, a MATLAB function was created called,

$$topLevel(A)$$

This takes in an image, A, as it parameter, which for this project is the image of the skeleton. The first task in creating the steps was to first image the original picture with no enhancement for reference. Next, the Laplacian of the image was needed. Referencing the lecture slides from class as well as the TA Bruce, this step was to simply create a new matrix with the given values and run them through the masking function created before. The matrix is as follows,

```
laplacianMatrix = [1 1 1;
                   1 -8 1
                   1 1 1];
```

Once this has run through the masking function, the image was scaled purely for seeing the result. This was done by taking the original image minus the lowest value in that image and dividing that by the maximum value in the image. To image the result better, the minimum and maximum values were done twice. Then this result was imaged. The actual step was to then subtract the Laplacian with no normalization from the original image to sharpen it. This result was imaged. The intended purpose of this step was to get the edges of the image and add it to the original to sharpen the image. It achieved its purpose well as the resulting image did have sharpened edges. Refer to code listing 2 for code.

The next task was to perform a Sobel product of the previous result. The interesting thing about Sobel operators is that both the horizontal and the vertical direction are needed to ensure the correct output. This ultimately means that two matrixes are needed to perform this operation. Once calculated using the masking program, the magnitude of both results are taken and finally added together to get the final result. This result was then imaged. The two matrixes are shown below,

```
sobelMatrixH = [-1 -2 -1;
                 0 0 0;
                 1 2 1];
```

```
sobelMatrixV = [-1 -0 1;
                -2 0 2;
                -1 0 1];
```

The next part of process is to smooth the image with a 5x5 mask. This task is very simple as it requires a mask of all ones. So once the previous image with a 5x5 mask was run through the masking program, it outputted an image. A smoothing mask requires that all of the values must be averaged which was done by simply dividing each pixel value by the mask size squared, or in this case, 25. The result was then imaged. The purpose of this step was to get edge detection on the image similar to the Laplacian. It achieved its purpose well as it highlighted the horizontal and vertical edges well. Refer to code listing 2 for code.

The next step in this process is to take the product of the Laplacian image and the newly smoothed Sobel image. A simple multiplication operation was used but the values of the image resulted out of the range of 0-255. Using the exact same process as the Laplacian, the image was scaled from 0-1 and finally multiplied by 255 to make the image visible. Then the result was imaged. The purpose of this step is to combine the two enhanced images and create a visible combination of the two. It achieved its purpose well since the result produced a clear image that will be late used. Refer to code listing 2 for code.

The next task was very straightforward. Take the original image and add it to the previous result (product of Laplacian and smoothed image.) This result was imaged. This step is intended to add the original image to the newly detailed image from the previous step. It achieved its purpose well as it more clearly represented the textbook result. Refer to code listing 2 for code.

The final task for this process was to use power law transformations to enhance the image to see the skin of the skeleton. This was done using the following equation,

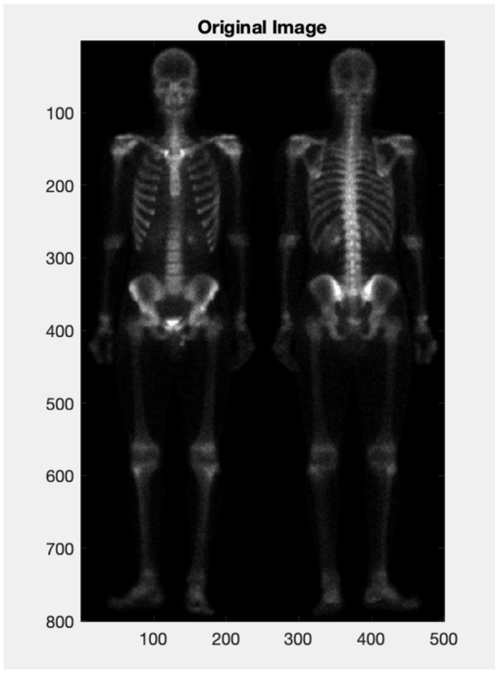
$$s = c * r^{(\gamma)}$$

Where s is the result, c is a constant, r is the image, and γ is gamma. The constant chosen was at a value of 1 and gamma was chosen to be 1.185. This value was chosen by experimenting with many different values and finding the best-looking result. Finally, the result was imaged. The purpose of this step is to enhance the image so as to see the skin and the bones of the skeleton represented in the textbook. This step achieved its purpose well as it clearly showed what was needed. Refer to code listing 2 for code.

Overall, all the steps taken together greatly enhanced the image. Each step was very crucial in highlighting certain features that eventually was brought out at the end of all the steps. Many lessons were learned from the project. The first being that a solid masking program will make each of the steps work fluently and produce an accurate result. The most important lesson was to ask for help. Little hiccups in the project can be easily completed by simply asking for help.

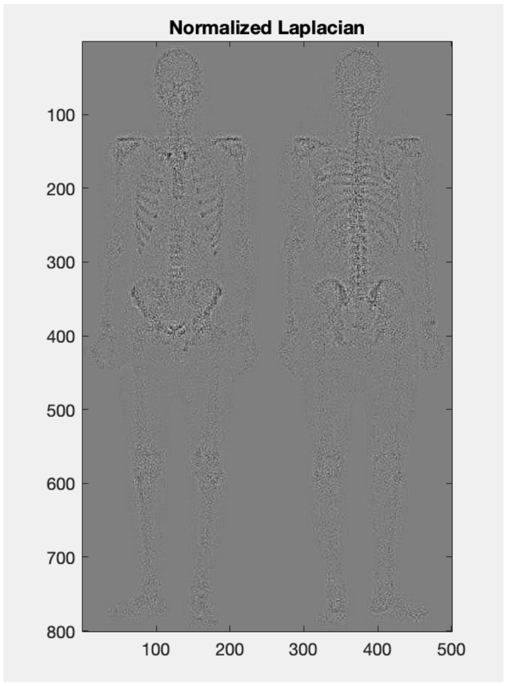
Results

This next section will outline all of the results described in the methods above. The original image is displayed,

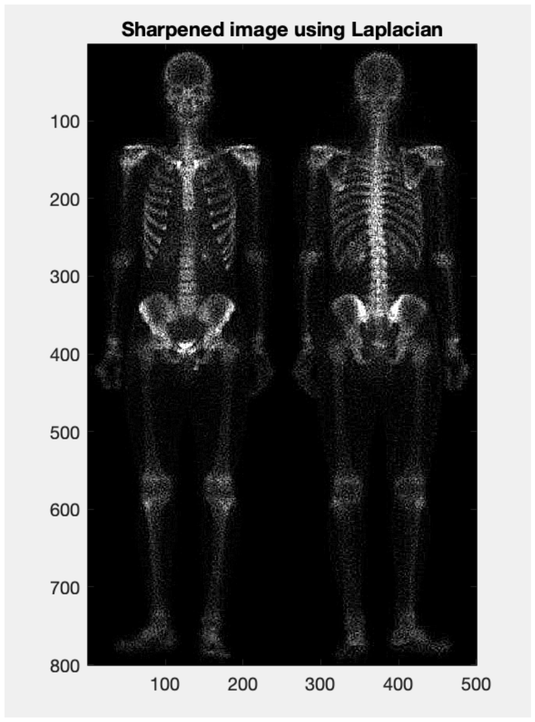


1. Original image with no enhancement

Image using Laplacian that has been normalized and image sharpened with the Laplacian with no modifications,

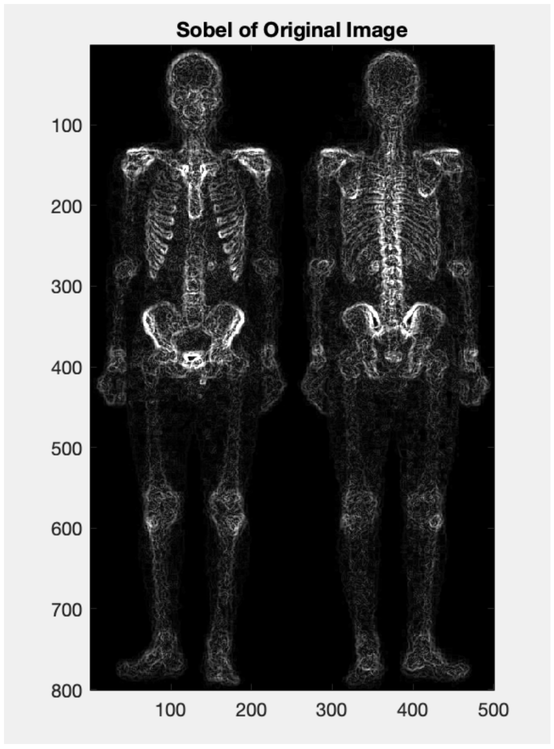


2. Laplacian normalized



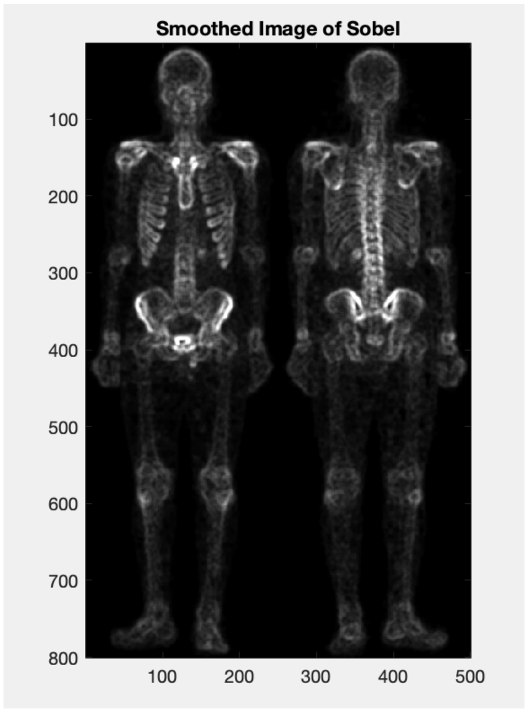
3. Sharpened image by adding original image to the Laplacian

Sobel image using vertical and horizontal masks,



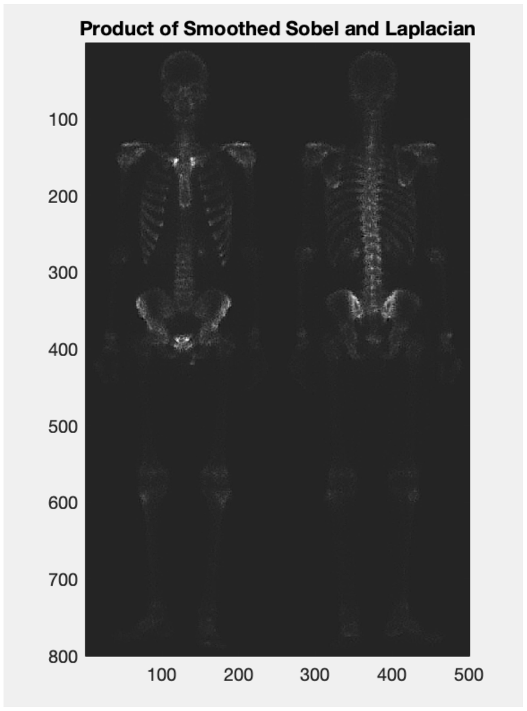
4. Sobel using horizontal and vertical masks

Smoothed image of the Sobel result using a 5x5 mask of ones,



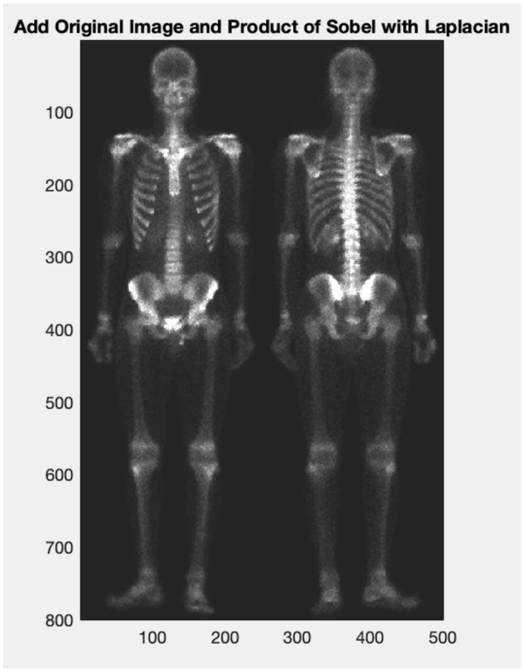
5. Sobel image smoothed by a 5x5 matrix

Product of the smoothed image and the Laplacian,



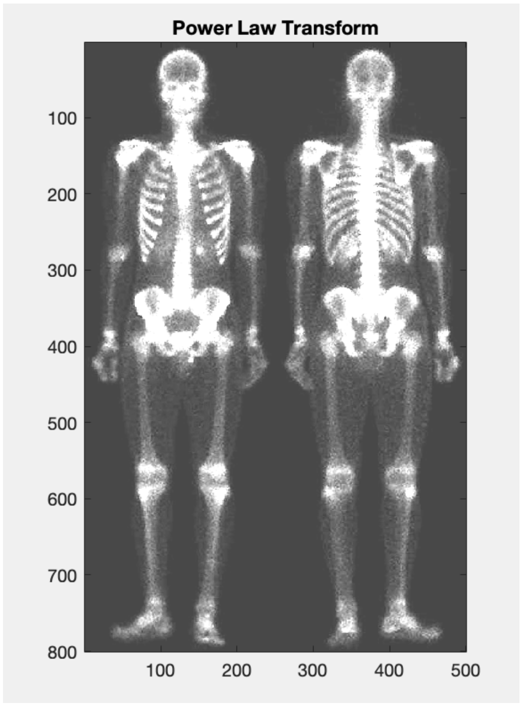
6. Product of the smoothed image and the Laplacian

Image of the original added to the product of the smoothed image and Laplacian,



7. Image of original image added to the product

Power law transform the result of figure 7,



8. Power law transform of figure 7

Conclusion

The objective of this project was to become more proficient with spatial domain image enhancement techniques as presented in Chapter 3 of the textbook, and how they can be combined. This task was completed through the enhancement of the skeleton image. Many steps were taken to complete this including Laplacian, Sobel, smoothing, products, and much scaling. The methods to complete these tasks were outlined and described in detail. These steps were all important as to completing figures 3.63 (b) – (h) from the textbook and slides. The objectives were all met in this project. This project taught many lessons which will be very helpful in the upcoming projects and the masking function is a very helpful tool in image processing. Simply changing the mask and doing some scaling can drastically change an image and highlight parts that could otherwise not be seen from the original. Overall, this project was fun and enforced the image processing techniques that were outlined in class.

A Code Listings

Listing 1: Masking function – ip_mask(A, maskArray)

```
function finalImage = ip_mask(A, maskArray)
% This function will take any image and a given array as its parameters and
% create a mask that will alter the image. It will run through every pixel
% determined by the size of the mask and averaging them and place that
% value in a new array.
% This function can be called above using the parameters

A = double(A);

[h, w] = size(A);

% This matrix will be where the computed values will be stored.

finalImage = ones(h);

% Grabbing the size of the given array. Only the height is being used so a
% tilda is being for the width

[maskH, ~] = size(maskArray);

% To create a padded image, I created an entirely new maxtrix with zeros
% and filled in the matrix with the original image.

% numPads takes the size of the desired array and subracts 1 then divides
% by two to find the rows and columns needed to pad the image.
numPads = (maskH-1)/2;

% This is used to calculate the number of pads for both sides of the image.
totalNumber = (2*numPads)+h;

% Creates matrix of zeros based on the new padding values found above.
paddedImage = zeros(totalNumber);

% This double for loop adds the original image into the new matrix allowing
% space for the zero padding on the edge
for i=1:h
    for j=1:w
        paddedImage(i+numPads,j+numPads) = A(i,j);
    end
end

% This for loop does multiple steps. First, the filter array (maskArray x
 maskArray) of all
% ones is multipled with a maskArray x maskArray matrix of the newly padded
 image. Since the
% image is already padded, the matrix can start at 1,1. The matrix of
% paddedImage "shifts" across the row 1 pixel at a time. It is then
% averaged by taking the sum of all the values within and dividing it by
% the total (mask array size^2). Finally, the averagedPixels is stored
% within the new matrix finalImage which is what is displayed at the end.

for i=1:h
```

```
    for j=1:w
        matrixMultiplied = (maskArray.*paddedImage(i:(maskH+(i-1)), j:(maskH +
 (j-1))));
        sumOfMatrix = sum(matrixMultiplied, 'all');
        finalImage(i,j) = sumOfMatrix;
    end
end

finalImage = finalImage(1:h, 1:w);

end
```

Listing 2: Top Level Function – topLevel(A)

```
function topLevel(A)
% This function contains all of the code that will form the steps needed to
% image the skeleton according to the textbook. Each step is detailed in
% the comments above the code.
% To run the code call the above function with an image as it's parameter.

L = 256;

A = double(A);

[h,w] = size(A);

% This is simply the original image.

figure(1);
image(A);
title("Original Image");
axis ij
axis equal
axis tight
colormap(gray(L))

% Laplacian data below. The new matrix for the laplacian is below with the
% diganol values. The laplacian is first shown scaled just for reference
% and because the project description requires it. To find the scaled
% laplacian, the equation takes the minimum of the laplacian and divides it
% by the max value of the new minimum found. Finally it all gets multiplied
% by the scale of 0-255 The actual laplacian is used by subracting it from
% the original image which is actually being used.

laplacianMatrix = [1 1 1;
                   1 -8 1
                   1 1 1];

scaledImageLaplacian = ip_mask(A, laplacianMatrix);

laplacianMin = scaledImageLaplacian - min(min(scaledImageLaplacian));

normalizedLaplacian = 255*(laplacianMin ./max((max(laplacianMin))));

figure(2);
image(normalizedLaplacian);
title("Normalized Laplacian");
axis ij
axis equal
axis tight
colormap(gray(L))

finalImageLaplacian = ip_mask(A, laplacianMatrix);
finalImageLaplacian = A - finalImageLaplacian;

figure(3);
```

```
image(finalImageLaplacian);
title("Sharpened image using Laplacian");
axis ij
axis equal
axis tight
colormap(gray(L))

% Sobel data below. This is a multi-step process which involes running the
% vertical sobel mask and taking the magnitude and then taking the
% horizontal sobel mask and taking the magnitude, and take the diagonal
% sobel masks as well doing the same process. Finally, all of those
% results are added together and finally imaged.

sobelMatrixH = [-1 -2 -1;
                0  0  0;
                1  2  1];

sobelMatrixV = [-1 -0  1;
                -2  0  2;
                -1  0  1];

sobelImageH = abs(ip_mask(A, sobelMatrixH));

sobelImageV = abs(ip_mask(A, sobelMatrixV));

finalSobelImage = sobelImageH + sobelImageV;

figure(4);
image(finalSobelImage);
title("Sobel of Original Image");
axis ij
axis equal
axis tight
colormap(gray(L))

% Smoothing data below. The matrix is a 5x5 of ones which will smooth the
% image once applied to the ip_mask function. Then to get around the
% averaging problem, the result was divided by 5^2. Finally the result is
% imaged.

smoothingMatrix = ones(5);

finalImageSmoothing = ip_mask(finalSobelImage, smoothingMatrix);

finalImageSmoothing = finalImageSmoothing*(1/25);

figure(5);
image(finalImageSmoothing);
title("Smoothed Image of Sobel");
axis ij
axis equal
axis tight
colormap(gray(L))
```

```
% The next step is to take the product of laplacian and the newly smoothed
% sobel image. But this needs to be scaled so the same process was used for
% this as it was for the laplacian. Find the min and max values of the
% product and scale them from 0 to 1. Then multiply that by 255 to get it
% back in the correct range. Finally, image the matrix.

product = (finalImageSmoothing.*finalImageLaplacian);

minProduct = min(min(product));

maxProduct = max(max(product));

finalProduct = (255*((product - minProduct)./(maxProduct)));

figure(6);
image(finalProduct);
title("Product of Smoothed Sobel and Laplacian");
axis ij
axis equal
axis tight
colormap(gray(L))

% The next step is to take the original image and add it to the result of
% the laplacian and newly smoothed sobel image.

sharpenedOriginalProduct = A + finalProduct;

figure(7);
image(sharpenedOriginalProduct);
title("Add Original Image and Product of Sobel with Laplacian");
axis ij
axis equal
axis tight
colormap(gray(L))

% The final step is to apply the power law transform. The equationm to use
% a power law transform or gamma correction is s = cr^(gamma) which is used
% below. The constant c =1 and gamma = 1.185. I found that this value closely
% represents what the figures in the textbook show (skin and bones).
% Finally the final result is imaged.

sharpenedOriginalProduct = double(sharpenedOriginalProduct);

for i=1:h
    for j=1:w
        finalPowerLaw(i,j) = 1*(sharpenedOriginalProduct(i,j)^1.185);
    end
end

finalPowerLaw = abs(finalPowerLaw);

figure(8);
image(finalPowerLaw);
title("Power Law Transform");
```

```
axis ij
axis equal
axis tight
colormap(gray(L))

end
```