Abu Dhabi University

CEN 464 - Digital Signal Processing

Lab Report 3

Applying Advanced Filters to Image & Object Detection using MATLAB

Authors:
Muhammad Obaidullah 1030313

Supervisor:
Dr. Mohammed Assad Ghazal

Section 1

June 9, 2014

Contents

Abstract

In this lab we were introduced to Image Processing Toolbox provided in MATLAB. We were exposed to simple image cropping, various types filtering and applying 2 dimensional Fast Fourier Transform on an image.

1 Objectives of the lab

- Load and display an image from hard disk.
- Distinguish between imtools and imshow and their features.
- Separate the color channels of an RGB signal and discuss it.
- Cut a part of the image.
- Convert an image from color to gray level.
- Use non-overlapping block processing on an image. Discuss the effect of changing the block size.
- Use overlapping block processing on an image. Discuss the effect of changing the block size.
- Study the effect of low-pass filtering in the time and frequency domain. Use the plot of H(w) for an the low-pass filtered image to prove it is corresponding to a low-pass filter.
- Use Sobel and Prewit operators to high-pass filter an image.
- Use Canny Edge detection and compare it with Sobel's edge detection.
- Use gray-level thresholding with morphological operations to extract the cameraman from his image.

2 List of equipment used

- A Computer.
- MATLAB.
- Image Processing Toolbox.

3 Load and display an image from hard disk.

To read a image from hard-drive, the following command is written:

```
1 x = imread("C:\Users\Muhammad\Desktop\image.png");
```

4 Distinguish between imtools and imshow and their features.

To show the image, we can do one of the following commands:

```
imshow(x);
```

As the name suggests, this command only just displays the image in a window with several basic options including saving the file as image into the hard-disk.
OR

```
imtool(x);
```

This command provide more tools and options to the user by providing several additional options such as zoom and detailed pixel under cursor information. When the cursor is moved on the picture, the detailed information about the pixel under the cursor such as Red value, green value, blue value, x position, and y position.

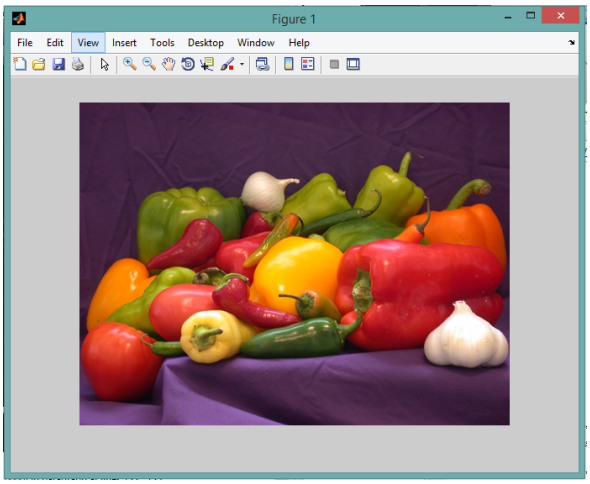

Figure 1: The image which is displayed by using the imshow() command.

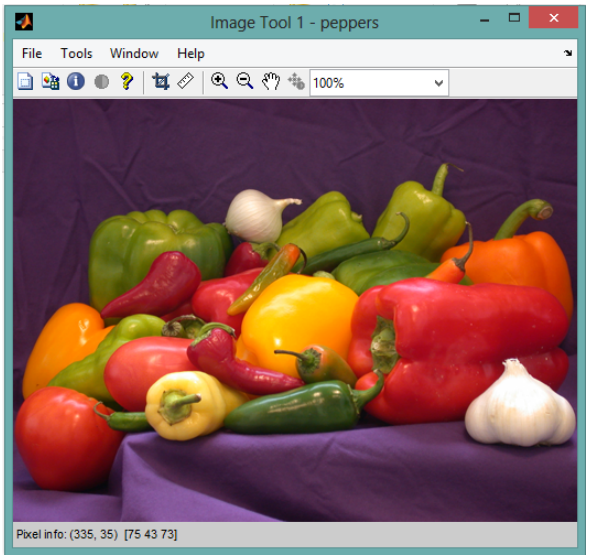

Figure 2: The image which is displayed by using the imtool() command.

5 Separate the color channels of an RGB image and discuss it

Let's select a color channel of an image in MATLAB:

```
% read the image into MATLAB
peppers = imread('peppers.png');
% select all of the x,y but select red channel
redpeppers = peppers(:,:,1);
% Show the red peppers by advanced imtool
imtool(redpeppers)
% select all of the x,y but select blue channel
```

```
bluepeppers = peppers(:,:,3);
% Show the blue peppers by advanced imtool
imtool(bluepeppers)
```

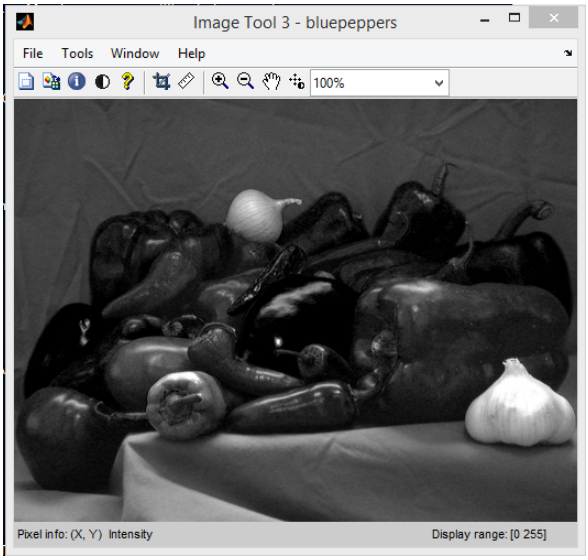

Figure 3: The blue channel of the peppers image. The image is majorly black because the peppers image does not contain a lot of blue pixels.

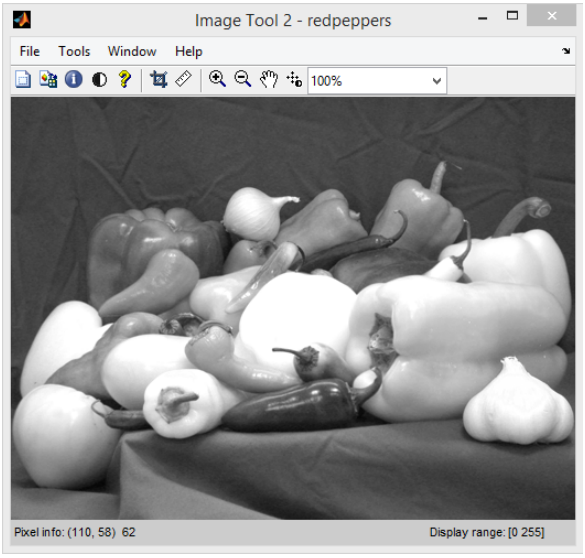

Figure 4: The red channel of the peppers image. As we can notice that there are many white areas. That is because the image is largely red.

6 Cut a part of the image

If we think of an image as a matrix, we can select some elements of the matrix and copy them into a new and smaller matrix. This is known as cropping an image. All computer image editing software use this technique behind the scenes. What a user essentially does is to select the corners of bounding box by drawing a rectangle on the image and then the software selects the pixels inside that rectangle, copies them onto a new matrix and displays it.

Let's do this cropping for ourselves in MATLAB:

```
% read the image into MATLAB
peppers = imread('peppers.png');
% select part of the x,y but select all of RGB
garlic = peppers(230:314,407:511,:);
% Show the garlic by advanced imtool
imtool(garlic)
```

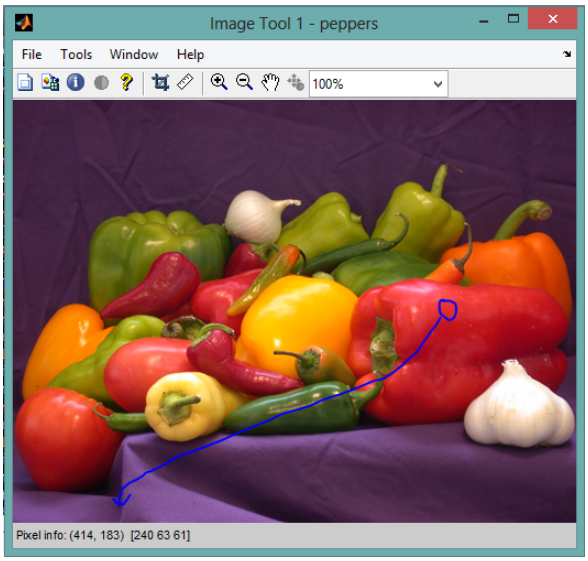

Figure 5: The original peppers image shown by imtool() command.

Figure 6: The cropped garlic from the original image

7 Convert an image from color to gray level

Any digital image is composed of pixels spread in two dimensional space. Any one of the pixel is represented by its x and y positions on the screen. In gray-level image, a pixel has a value determining how close to black or white color a pixel is. If the pixel values are unsigned integer 8-bits, then value 255 is the whitest and value 0 is blackest. Any value in between is a shade of gray. On the contrary, a color image is composed of pixels which often have three values, the red value, the green value, and the blue value. These values indicate how close to the primary color the pixel is. For example, if a pixel has R = 255 G = 0 B = 0 values then it means the pixel is entirely red.

There are some image formats which also allow an additional value for each pixel to be stored in an image. This is the alpha value, which determines the transparency of the pixel. A value of 255 indicates that the pixel is completely opaque while the 0 value indicates that the pixel is completely transparent. This fourth alpha channel is crucial in many image applications and visual effects because it allows dynamics of the background and sense of reality.

For an image to be processed by MATLAB, it is often recommended to convert the pixel value from unsigned integer to double and map every value from 0-255 to 0-1. This allows the image matrix to become double so that is image operations and processing is done on it, it can candle decimal values. At the end of image processing, the image is converted back to 0-255 range so that the real world displays can handle them.

We can load some images from the MATLAB library and convert a RGB color image into a gray-level image by typing the following code:

```
peppers = imread('peppers.png');
imtool(peppers);
pepgray = rgb2gray(peppers);
```

```
imtool(pepgray);
```

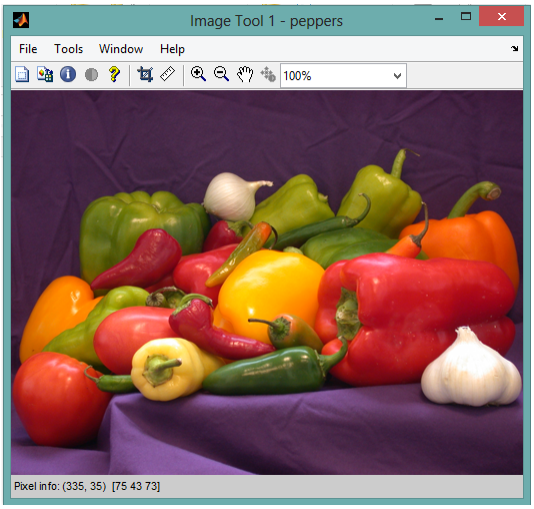

Figure 7: RGB image

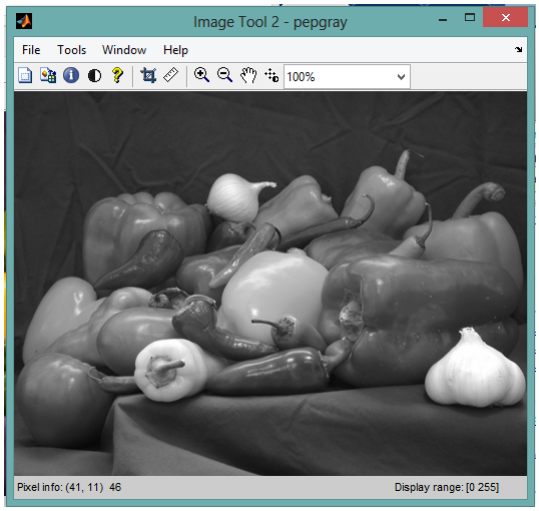

Figure 8: Gray-level image

8 Doing block processing on an image

Block processing is very powerful technique in image processing as it allows the program to focus and process a part of an image at a time. If we think of about block processing, it is kind of like doing convolution in 2 dimensional space. There are two kinds of block processing we can do.

- Non-overlapping block processing

The function we are going to use is blockproc(A, [M,N], FUN). This function takes a matrix A as an input argument and applies the function FUN on the matrix A with the sample size of M,N. The size of sample can be thought of as a brush size.

So first let's make an averager function to apply to a block:

```
function y = averager(x)
  % This function simply finds a 2 dimensional average of the matrix supplied to it
    y = mean2(x);
end
```

Let's make a function to pass:

```
% This function takes a block structure and averages the pixels around it to produce
    the value at the center
fun = @(block_struct) averager(block_struct.data(:,:));
```

Now, let's apply this function on the peppers image

```
% Read image into MATLAB
peppers = imread('peppers.png');
% Convert the peppers image into gray-level image
peppers_gray = rgb2gray(peppers);
% Show gray-level image
imtool(peppers_gray);
% Apply block processing with brush size of 3,3
result_image = blockproc(peppers_gray,[3,3],fun);
% Convert the resulting image from double to unsigned integer 8-bits
result_image = uint8(result_image);
% Show the resulting image in imtool window
imtool(result_image);
```

Figure 9: The original gray-level peppers.png image.

Figure 10: The block processed gray-level peppers.png image.

Let's make another function to pass:

```
% This function takes a matrix and returns a pixel at the center of the matrix
function y = sampler(x)
    %imshow(x);
    [rows,cols] = size(x);
    rowinmiddle = ceil(rows/2);
    colinmiddle = ceil(cols/2);
    y = x(rowinmiddle,colinmiddle);
end
```

Now, let's apply this function on the peppers image

```
% Read image into MATLAB
peppers = imread('peppers.png');
% Convert the peppers image into gray-level image
peppers_gray = rgb2gray(peppers);
% Show gray-level image
imtool(peppers_gray);
% Apply block processing with brush size of 3,3
processed_image = blkproc(peppers_gray,[3 3],@sampler);
% Show the resulting image in imtool window
imtool(processed_image);
```

Figure 11: The original gray-level peppers.png image.

Figure 12: The block processed gray-level peppers.png image.

This block processing results in a much sharper result than the averager block process. This is because the average or mean is kind of low-pass filtering of the image.

Non-overlapping block operation As we can notice from the block processed image, the resulting image is much smaller and has less pixel information. This is because of the fact that a single block results in a single pixel on the resulting image. The bigger the sample size, the smaller is the resulting image.

9 Use overlapping block processing (nlfiltering) on an image

Let's apply sampler function on the peppers image using non-local filter

```
% Read image into MATLAB
peppers = imread('peppers.png');
% Convert the peppers image into gray-level image
peppers_gray = rgb2gray(peppers);
% Show gray-level image
imtool(peppers_gray);
% Apply non-local filtering with brush size of 20,20
processed_image = nlfilter(peppers_gray,[20 20],@sampler);
% Show the resulting image in imtool window
imtool(uint8(processed_image));
```

Figure 13: The original gray-level peppers.png image.

Figure 14: The non-local filter processed gray-level peppers.png image.

We can immediately conclude that the images are not changed, that is because the function sampler returns the pixel at the center back to the filter and the filter places the pixel on the same position again.

Let's apply averager function on the peppers image using non-local filter

```
% Read image into MATLAB
peppers = imread('peppers.png');
% Convert the peppers image into gray-level image
peppers_gray = rgb2gray(peppers);
% Show gray-level image
imtool(peppers_gray);
% Apply non-local filtering with brush size of 20,20
processed_image = nlfilter(peppers_gray,[20 20],@averager);
% Show the resulting image in imtool window
imtool(uint8(processed_image));
```

Figure 15: The original gray-level peppers.png image.

Figure 16: The non-local filter processed gray-level peppers.png image.

We can immediately conclude that the image is blurred and there are no sudden changes of color. It is because of the fact that the averager is kind of low-pass filter and allows small changes of colors to pass while smoothing or lowering the amplitude of the high frequency regions.

10 Low Pass Filtering

```
% Read the cameraman image from the hard drive
cameraman = imread('cameraman.tif');
% Define the h_n for the low pass filter
h_n_low_pass = [1/9 1/9 1/9; 1/9 1/9 1/9; 1/9 1/9 1/9];
% Apply the low pass filter on the image and save the result in output
% array
output = imfilter(cameraman,h_n);
% Show the filtered image using the imtool
imtool(output);
```

Figure 17: The original gray-level cameraman.tif image.

Figure 18: The low pass filter processed cameraman.tif image.

As we can see, the low pass filter destroyed the picture quality and did not allow sudden changes of pixel values. It kind of averages the pixel value based on the surrounding pixels making the color transitions smooth.

Now let's take this image into frequency domain to find out whether the higher frequencies are really filtered off.

```
% Read the cameraman image from the hard drive
cameraman = imread('cameraman.tif');
% create a new figure
figure;
% Convert the input image to double for easy processing
input = im2double(cameraman);
% Normalize the image for frequency domain
img_in = fftshift(input);
% Take the input image to the frequency domain
F = fft2(img_in);
% Show the grayscale version of the fft
imagesc(100*log(1+abs(fftshift(F)))); colormap(gray);
```

```
% Put the title
title('Original image magnitude spectrum');

% Define the h_n for the low pass filter
h_n_low_pass = [1/9 1/9 1/9; 1/9 1/9 1/9; 1/9 1/9 1/9];
% Apply the low pass filter on the image and save the result in output
% array
output = imfilter(input,h_n);
% Show the filtered image using the imtool
imtool(output);
% Convert the output image to double for easy processing
output = im2double(output);
% Normalize the image for frequency domain
out_img = fftshift(output);
% Take the output image to the frequency domain
G = fft2(out_img);
% create a new figure
figure;
% Show the grayscale version of the fft
imagesc(100*log(1+abs(fftshift(G)))); colormap(gray);
% Put the title
title('Low pass filtered image magnitude spectrum');
```

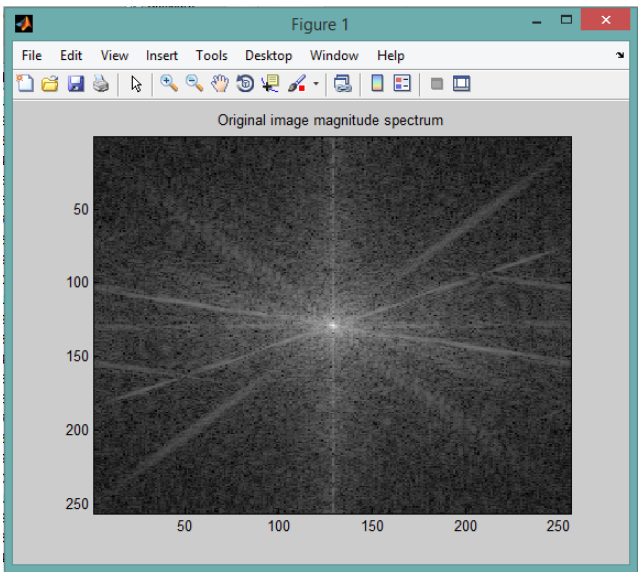

Figure 19: 2D frequency domain plot of the original image.

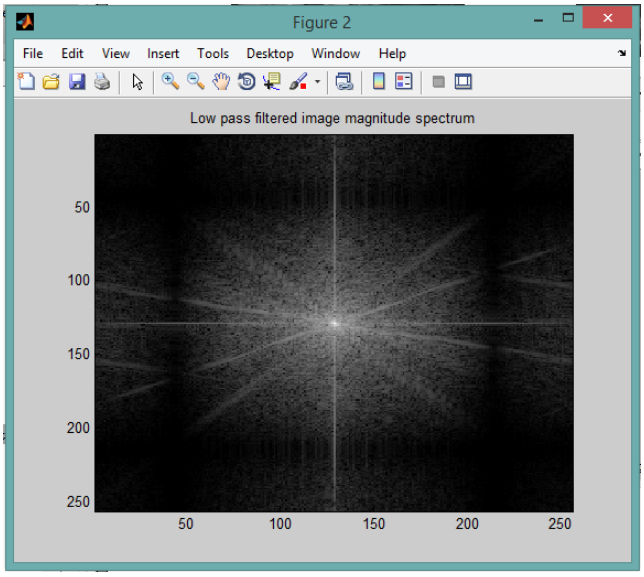

Figure 20: 2D frequency domain plot of the low pass filtered image.

The further away we go from the center, the frequency gets higher. Figure 20 clearly gets darker on as we move away from the center. This refers to the fact that higher frequency components have being filtered out/removed.

11 Sobel and Prewit operators for high-pass filtering

11.1 Sobel Filter

Sobel filter is a very popular vertical edge pass, horizontal edge attenuate filter. It is pre-defined in MATLAB. Matlab returnsa sobel matrix filter by typing the command $fspecial('sobel')$

$$h = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

```
% Read the cameraman image from the hard drive
cameraman = imread('cameraman.tif');
% create a new figure
figure;
% Convert the input image to double for easy processing
input = im2double(cameraman);
% Normalize the image for frequency domain
img_in = fftshift(input);
% Take the input image to the frequency domain
F = fft2(img_in);
% Show the grayscale version of the fft
imagesc(100*log(1+abs(fftshift(F)))); colormap(gray);
% Put the title
title('Original image magnitude spectrum');

% Define the h_n for the low pass filter
h_n_sobel = fspecial('sobel');
% Apply the low pass filter on the image and save the result in output array
output = imfilter(input,h_n_sobel);
% Show the filtered image using the imtool
imtool(output);
% Convert the output image to double for easy processing
output = im2double(output);
% Normalize the image for frequency domain
out_img = fftshift(output);
% Take the output image to the frequency domain
G = fft2(out_img);
% create a new figure
figure;
% Show the grayscale version of the fft
imagesc(100*log(1+abs(fftshift(G)))); colormap(gray);
% Put the title
title('Sobel filtered image magnitude spectrum');
```

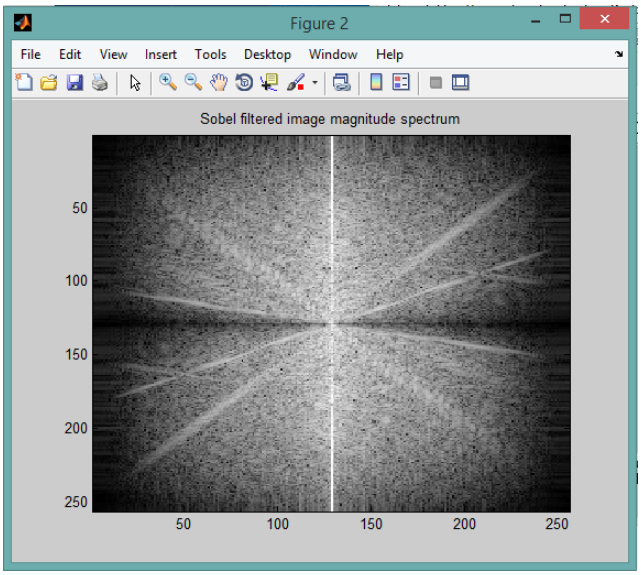

Figure 21: 2D frequency domain plot of the sobel filtered image.

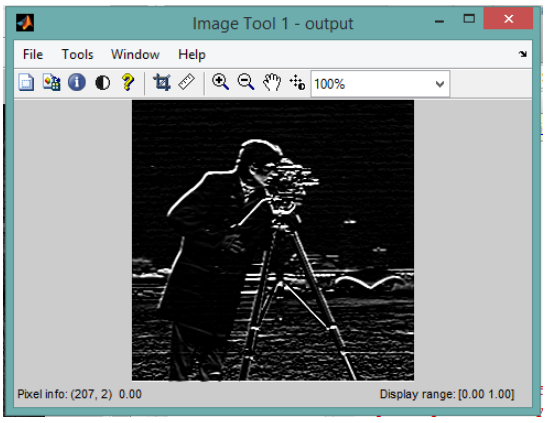

Figure 22: Sobel filtered image. Horizontal edges are ignored and vertical edges are amplified to appear more bright. This filter is usually used in edge detection algorithms as we can see it has detected the man's outline and mountain edges.

It can clearly be seen that the frequency with horizontal phase were attenuated and the vertical frequencies were amplified.

11.2 Prewitt Filter

Prewitt filter is a very popular vertical edge pass, horizontal edge attenuate filter. It is pre-defined in MATLAB. Matlab returnsa sobel matrix filter by typing the command $fspecial('prewitt')$

$$h = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

```
% Read the cameraman image from the hard drive
cameraman = imread('cameraman.tif');
% create a new figure
figure;
% Convert the input image to double for easy processing
input = im2double(cameraman);
% Normalize the image for frequency domain
img_in = fftshift(input);
% Take the input image to the frequency domain
F = fft2(img_in);
% Show the grayscale version of the fft
imagesc(100*log(1+abs(fftshift(F)))); colormap(gray);
% Put the title
title('Original image magnitude spectrum');

% Define the h_n for the low pass filter
h_n_prewitt = fspecial('prewitt');
% Apply the prewitt filter on the image and save the result in output array
output = imfilter(input,h_n_prewitt);
% Show the filtered image using the imtool
```

```
imtool(output);
% Convert the output image to double for easy processing
output = im2double(output);
% Normalize the image for frequency domain
out_img = fftshift(output);
% Take the output image to the frequency domain
G = fft2(out_img);
% create a new figure
figure;
% Show the grayscale version of the fft
imagesc(100*log(1+abs(fftshift(G)))); colormap(gray);
% Put the title
title('Prewitt filtered image magnitude spectrum');
```

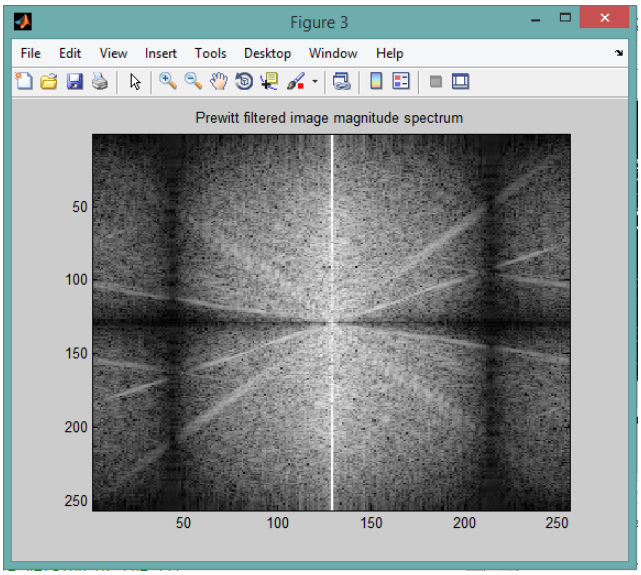

Figure 23: 2D frequency domain plot of the prewitt filtered image.

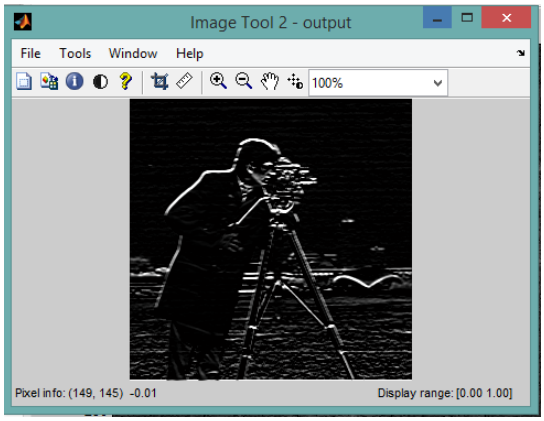

Figure 24: Prewitt filtered image. Horizontal edges are ignored. This filter is usually used in edge detection algorithms as we can see it has detected the man's outline and mountain edges.

It can clearly be seen that the frequency with horizontal phase were attenuated and the vertical frequencies were amplified.

Difference between the Sobel and Prewitt filters Prewitt filter is a basic version of Sobel filter where the coefficients are not modified. Whereas, in sobel filter, the coefficients can be modified to detect more or less edges.

$$h_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad h_2 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad h_3 = \begin{bmatrix} 1 & 5 & 1 \\ 0 & 0 & 0 \\ -1 & -5 & -1 \end{bmatrix} \quad h_4 = \begin{bmatrix} 1 & 0.5 & 1 \\ 0 & 0 & 0 \\ -1 & -0.5 & -1 \end{bmatrix}$$

12 Canny Edge detection

Canny edge detection works on multi-stage algorithm to come up with all the edges in the image. The intensity of the filter can be controlled with a parameter. Matlab has a function $output = edge(input,'canny')$; which applies canny edge detection algorithm to an image and returns a edge image.

```
% Read the cameraman image from the hard drive
cameraman = imread('cameraman.tif');
% create a new figure
figure;
% Convert the input image to double for easy processing
input = im2double(cameraman);
% Normalize the image for frequency domain
img_in = fftshift(input);
% Take the input image to the frequency domain
F = fft2(img_in);
% Show the grayscale version of the fft
imagesc(100*log(1+abs(fftshift(F)))); colormap(gray);
% Put the title
title('Original image magnitude spectrum');

output = edge(input,'canny');
% Show the filtered image using the imtool
imtool(output);
% Convert the output image to double for easy processing
output = im2double(output);
% Normalize the image for frequency domain
out_img = fftshift(output);
% Take the output image to the frequency domain
G = fft2(out_img);
% create a new figure
figure;
% Show the grayscale version of the fft
imagesc(100*log(1+abs(fftshift(G)))); colormap(gray);
% Put the title
title('Canny edge setection image magnitude spectrum');
```

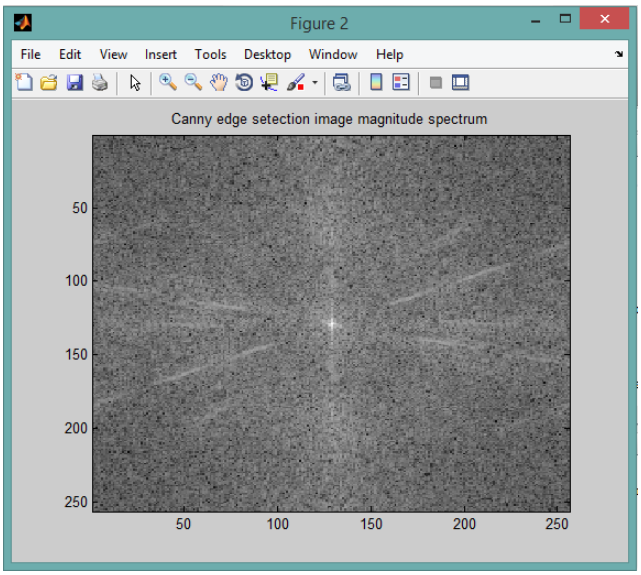

Figure 25: 2D frequency domain plot of the canny edge detection image. As expected all the higher frequency components(edges) are amplified.

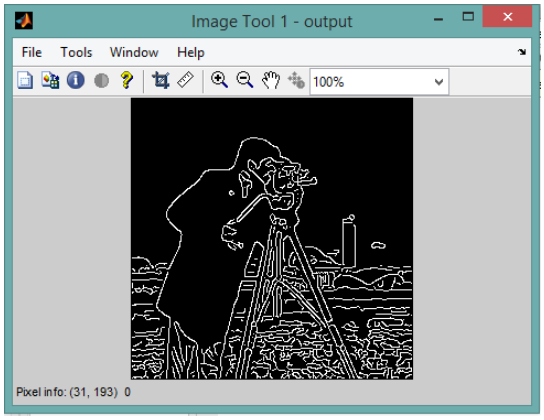

Figure 26: Output image from canny edge detection algorithm. Almost all the edges are detected.

Difference between the Canny edge detection and Sobel edge detection Sobel edge filter focuses on a particular frequency phase while the canny edge detection first generates the frequency phase map and applies corresponding phase shifted filter to the image. Sobel filter will not be able to detect edges for which it is not designed for while canny edge detection will be able to detect edges regardless of the direction of frequency.

13 Gray-Level thresholding with morphological operations

Object detection is a huge field in image processing. We will practice one of the algorithm to detect the image. This algorithm converts the image into logical values of either a completely black pixel or a completely white pixel. Then this image is inverted because the object we are trying to detect is black but we want it as white and the background to be black. The person is then filled with the disk of size we specify. This can also be called as brush size. Once the image is filled, there is a slight problem. There are multiple regions where the brush has filled with white pixels. Now how can we choose one single area and deleted others. This is done by fetching the properties for each of the region and calculating the area. If the area is larger than our minimum requirement, we allow the region to appear on the screen.

```
function y = SegmentLargeBlobs(filename,p)
% Reads the passed file from the given path
x = imread(filename);
% Converts the image into logical value type. Either completely black or completely
    white
b = im2bw(x);
% Takes the inverse of the image to get the man white and the background
% black
b = 1-b;
% Creates a matrix which has ones in the shape defined and size
```

```
se = strel('disk',5);
% Tries to fit maximum number of se matrices inside the cameraman
w = imopen(b,se);
% get number of blobs
r = regionprops(bwlabel(w));
y = w;
% Get number of rows and columns of the matrix
[rows,cols] = size(y);
% Calculate the area of the imopen
S = rows*cols;
% Repeat for each blob region
for i = 1:length(r),
    if r(i).Area/S < p,
        % Draw a bounding box for blob regionwhich satisfies our area requirement
        BB = r(i).BoundingBox;
        TopLeftX = round(BB(1));
        TopLeftY = round(BB(2));
        Width = round(BB(3));
        Height = round(BB(4));
        y(TopLeftY:TopLeftY+Height,TopLeftX:TopLeftX+Width) = 0;
    end
end
```

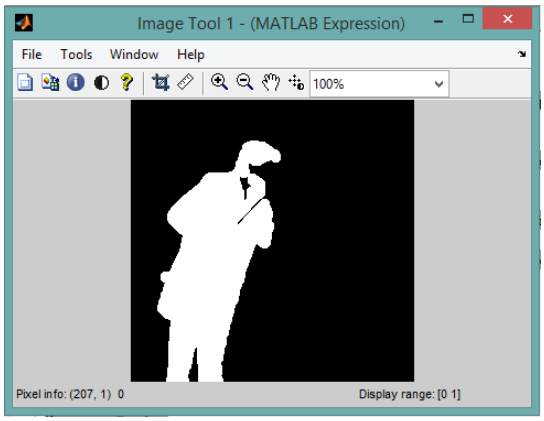

Figure 27: The figure shows the result produced when the parameter passed is 0.201. The following is the command passed.

```
imtool(SegmentLargeBlobs('cameraman.tif',0.201));
```

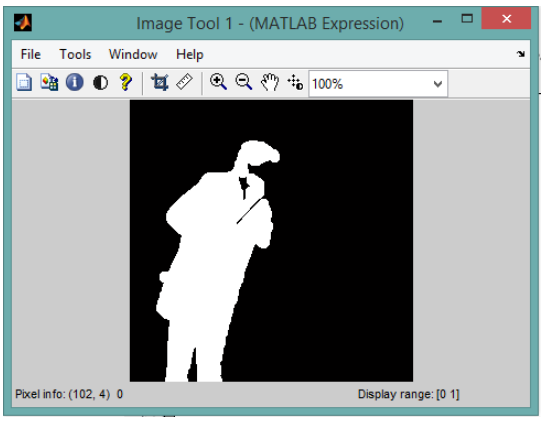

Figure 28: The figure shows the result produced when the parameter passed is 0.02. The following is the command passed.

```
imtool(SegmentLargeBlobs('cameraman.tif',0.02));
```

13.1 Final Steps

The final steps to extract the man from the image is to multiply the mask obtained from the image processing with the original image.

```
Get_Blobby_Image = SegmentLargeBlobs('cameraman.tif',0.201);
cameraman_original = imread('cameraman.tif');
```

```
cameraman_double = im2double(cameraman_original);
Get_Blobby_Image = Get_Blobby_Image(1:256,1:256)
Resulting_Image = Get_Blobby_Image .* cameraman_double;
imtool(Resulting_Image+(1-Get_Blobby_Image));
```

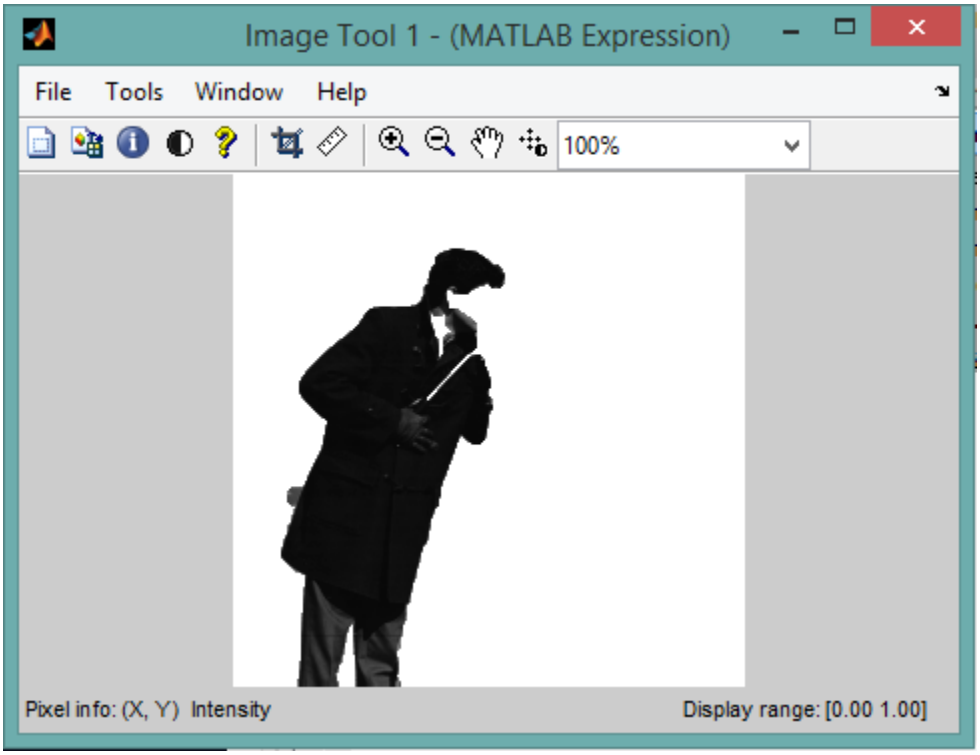

Figure 29: The final man extracted from the image with the background made white.

References

[1] Mathworks. *Image Processing Toolbox*. [Electronic]. Available: `http://www.mathworks.com/products/datasheets/pdf/image-processing-toolbox.pdf` [2 June 2014].

[2] Dr.Tim Morris. *Image Processing with MATLAB*. Page 2. Available: `http://studentnet.cs.manchester.ac.uk/ugt/COMP27112/doc/matlab.pdf` [2 June 2014].

[3] Huidan Cao & Yuming Shen. *Application of MATLAB image processing technology in sewage monitoring system* [Electronic]. Available: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5274144&isnumber=5273983` [2 June 2013].