CS 484 Image Analysis
Spring 2019
HW1 Report

Name : Ünsal Öztürk
ID: 21601565

Q1-

The following is the implementation for the dilation operation in MATLAB that does not make use of the “logical” types. The implementation is located in dilation.m.

```
% dilation: if structuring element hits, then 1.
function out = dilation(image, str_elem)
% pad the image with zeroes to avoid dimension loss
padx = floor(size(str_elem,1)/2);
pady = floor(size(str_elem,2)/2);
[N,M] = size(image);
out = zeros(size(image));

padded_image = zeros(N+2*padx, M+2*pady);
padded_image(padx+1:N+padx, pady+1:M+pady) = image;

% loop over the pixels of the image
for i = padx+1: padx+N
    for j = pady+1:pady+M
        img_seg = padded_image(i-padx:i+padx, j-pady:j+pady);
        % logical intersection is analogous to element-wise multiplication
        intersection = img_seg .* str_elem;
        % if the structuring element "hits", then at least one element
        % in intersection is non-zero.
        if sum(sum(intersection)) > 0
            out(i-padx,j-pady) = 1;
        end
    end
end
```

The following is the implementation for the erosion operation in MATLAB that does not make use of the “logical” types. The implementation is located in erosion.m.

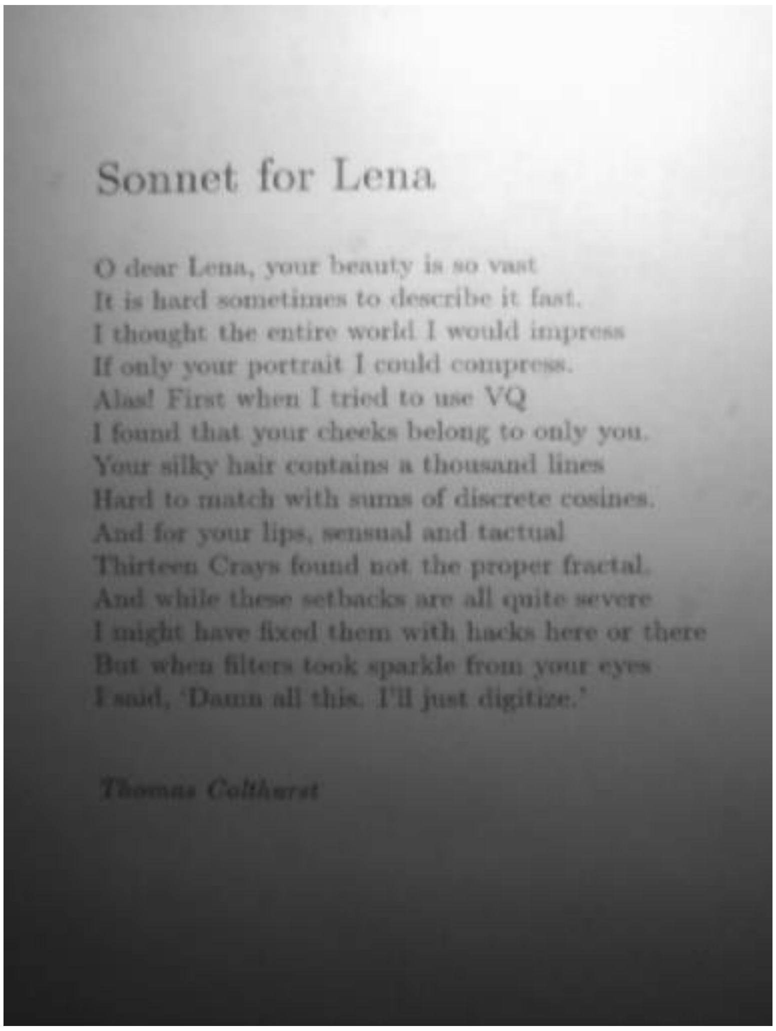
```
% erosion: if structuring element fits, then 1
function out = erosion(image, str_elem)
% pad the image with zeroes to avoid dimension loss
padx = floor(size(str_elem,1)/2);
pady = floor(size(str_elem,2)/2);
[N,M] = size(image);

out = zeros(size(image));

padded_image = zeros(N+2*padx, M+2*pady);
padded_image(padx+1:N+padx, pady+1:M+pady) = image;

% loop over the pixels of the image
for i = padx+1: padx+N
    for j = pady+1:pady+M
        img_seg = padded_image(i-padx:i+padx, j-pady:j+pady);
        intersection = img_seg .* str_elem;
        % if the structuring element "fits", then the logical and of the
        % structuring element and the image segment should be equal to the
        % structuring element itself.
        if isequal(str_elem,intersection)
            out(i-padx,j-pady) = 1;
        end
    end
end
```

The definitions for these operations were taken from:
https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm

Q2 -

Recovering the letters from the image in their exact form was quite challenging, and I must say that the final version is barely legible.

The first idea to recover the letters from the image is to make sure that the background color is actually static throughout the image. Initially, as seen below, the image is illuminated by a light source somewhere in the top right corner of the image. I assumed that there is a point light source somewhere at a fixed height from the image, with some intensity value. I assumed that the paper in the picture is planar and rigid, and I modeled the point light as in a 3D scene and removed the lighting as much as possible to extract albedo information from the picture, given that most of the reflections on the paper are diffuse. Here are the results of brightness normalization.

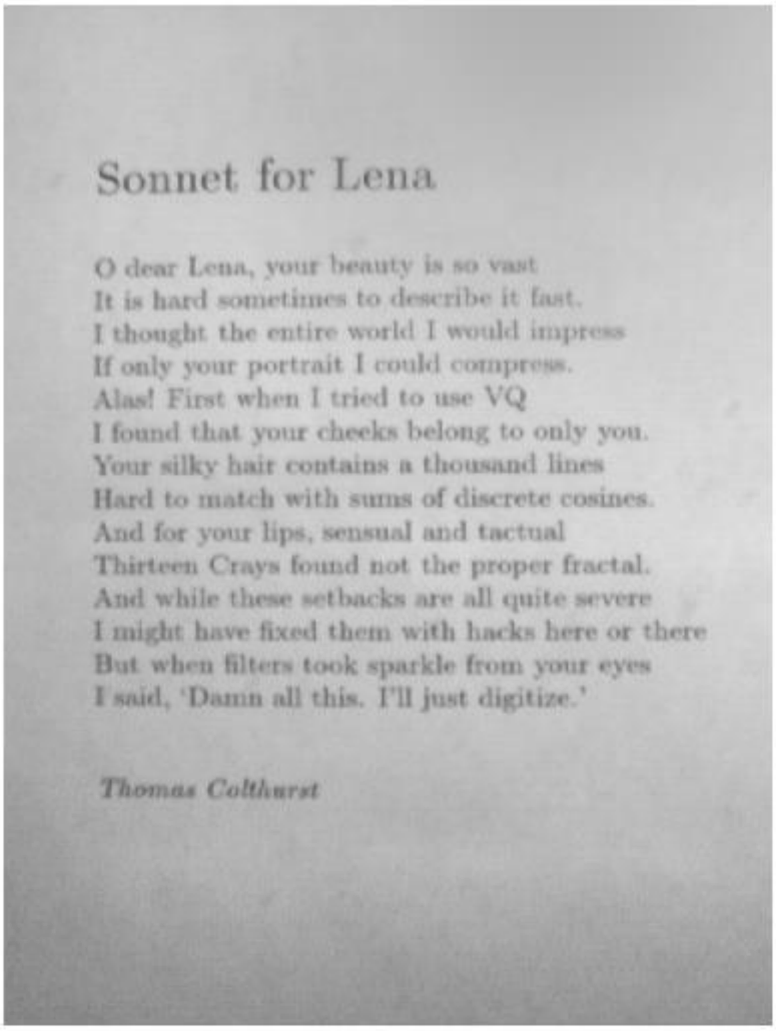


Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

Figure – Original Image



Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

Figure – Image with normalized lighting

The model assumes that the lighting is in the form

$$L = k\frac{I}{d^2}cos\theta$$

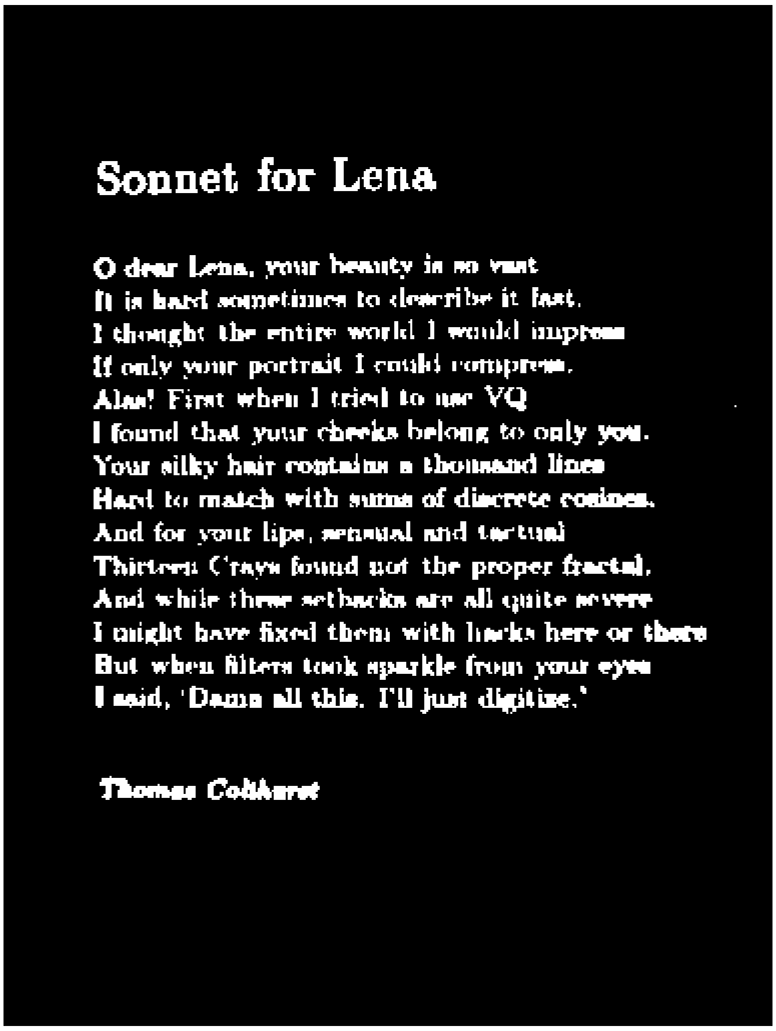
where k is linearly scaled version of the albedo of the texture, I is the radial intensity of the light, and $cos\theta$ is the cosine of the angle between the incident angle and the surface normal. It normalizes the lighting by setting the value of the pixel to k, which is a scaled version of the albedo.

The main drawback of this approach is that there may be multiple non-point or non-spherical light sources in the scene. Therefore not all lighting has been normalized; however the results are significantly better than the initial image.

The next step is to threshold the image using adaptive thresholding, which computes a local threshold given a patch of a grayscale image. Before thresholding, I did the arithmetic pixel-wise luminance transformation of

$$p \to e^{-e^{p}}$$

which greatly amplifies the contrast between black and white colors, for easier thresholding. After this transformation, I threshold the image using adaptive thresholding with a sensitivity parameter of 0.48.



Sonnet for Lena

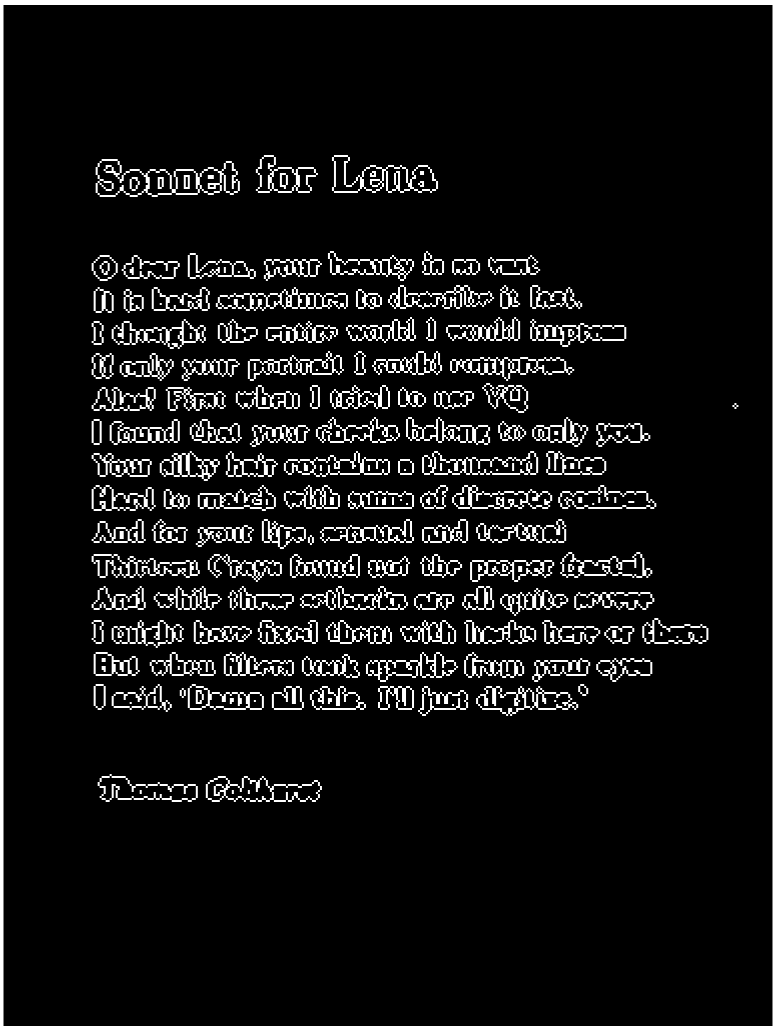
O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

Figure: Thresholded image

Even though this image looks fine, we are required to use morphological operators to further enhance the legibility of the image. For this purpose, I decided to use the classic dilated – image for “edge” extraction.

The following is the dilated – image response with a diamond structuring element of size 3x3.



Sonnet for Lena

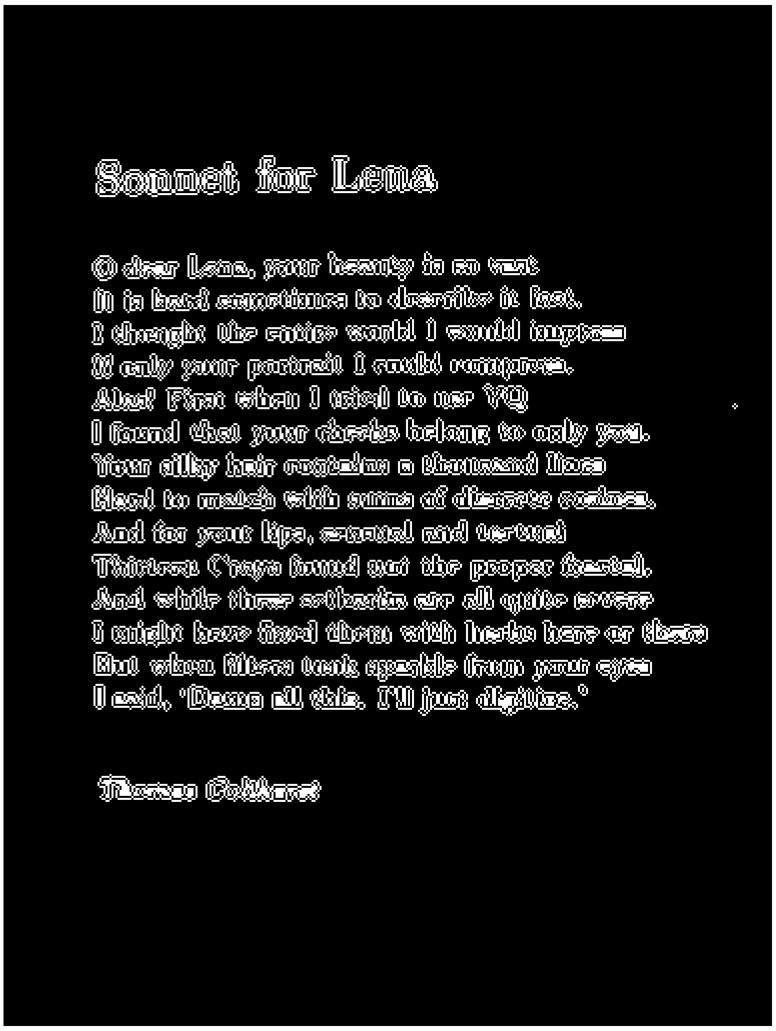
O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

Figure – Dilated image – Original Image

It is possible to add the erosion of the image to this version to make it further look like a legible version. The following image is dilated image + eroded image – original image.

[This space was intentionally left blank]



Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

Figure – Final Image

The script that reproduces the images is located in main.m.

Difficulties encountered: Letters are too thin to process with morphological operators. Therefore, any version of this image based on morphological operations is somewhat more inferior to the original thresholded image. Computing an opening or closing ruins the legibility of the image most of the time. The final image is more of a patchwork: it tries to capture the main distinctive features of letters; however, the result isn't the most satisfactory. The following page has the script for the reproduction of these results.

```
clear
clc
% read in image
image = im2double(imread('sonnet.png'));
imout = image;

% Obvious stuff: there is a light source in the upper right corner of the
% image. Assume a point light at (x,y,z), with radial intensity I. Play
% with the parameters and undo the lighting so that the lighting on the
% bottom of the image is similar to the lighting at the top right corner.
% The model is L = I/d^2*cos(theta)*albedo/(2*pi), assuming the surface is
% perfectly diffuse. We are interested in the albedo of the surface. The
% ink will have a blacker albedo whereas the paper will have a whiter
% albedo. Solving the equation, one has albedo = L/I*d^2/cos(theta)*2*pi.
% One can directly omit 2*pi and incorporate it into I for practical
% purposes. Now, one should play with the parameters x,y,z and I below to
% obtain an image composed purely of the albedo of the surface.
%
% DRAWBACKS: There may be more than one light source in the scene. The
% reflections might not be perfectly diffuse.

x = 275;
y = -20;
z = 375;
C = 200000;

for i=1:size(imout,1)
    for j=1:size(imout,2)
        d2_plane = (i-y)^2+(j-x)^2;
        d2_space = d2_plane + z^2;
        cosTheta = z/d2_space^0.5;
        imout(i,j) = image(i,j) * d2_space / C / cosTheta;
    end
end

figure
imshow(imout);
imout = exp(-exp(imout));
threshold = adaptthresh(imout, 0.48);
imbit = imbinarize(imout, threshold);
figure
imshow(imbit);

str_elem = [0,1,0;
            1,1,1;
            0,1,0;];

dilated = dilation(imbit,str_elem);

figure
imshow(dilated - imbit);
eroded = erosion(imbit,str_elem);
final = (dilated-imbit+eroded);

figure
imshow(final);
```

Q3 –

The implementation is located in convolution.m.

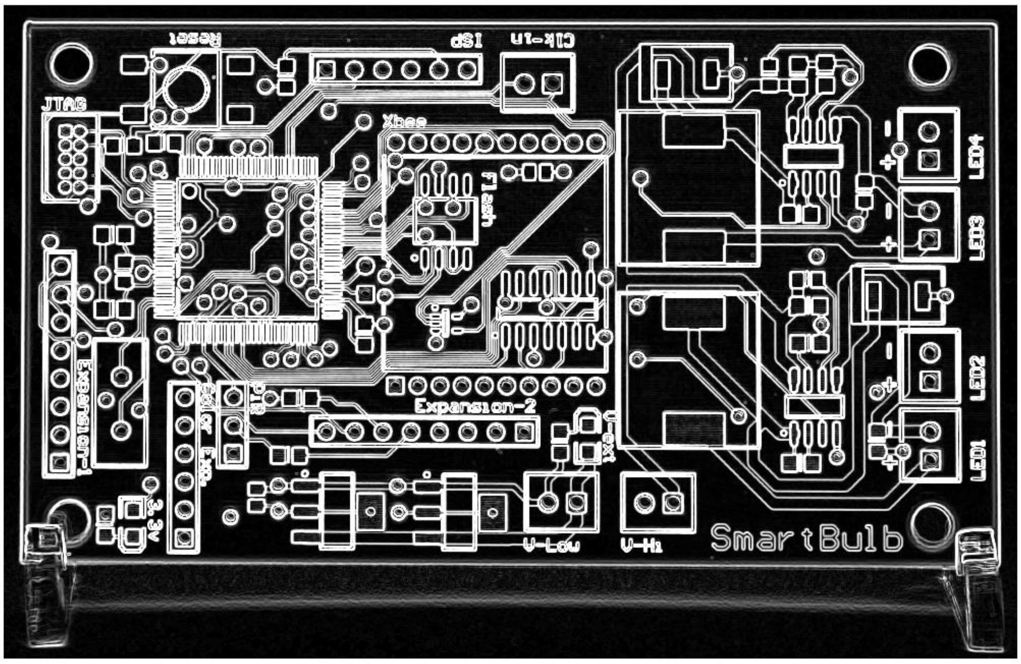
It quite literally computes $\sum\sum K(i,j)I(x-i,y-j)$ over all i,j.

```
function cnv = convolution(filter, image)
% flip the filter for convolution
filter_flipped = flip(flip(filter,1),2);
% pad the image with zeroes to avoid dimension loss
padr = floor(size(filter_flipped,1) / 2);
padc = floor(size(filter_flipped,2) / 2);
cnv = zeros(size(image));
zero_pad_img = pad(image, padr, padc);
% convolve the image
for i =  1 + padr:size(zero_pad_img,1) - padr
    for j = 1 + padc:size(zero_pad_img,2) - padc
        img_seg = zero_pad_img(i-padr:i+padr,j-padc:j+padc);
        dt = sum(img_seg .* filter_flipped);
        cnv(i - padr,j - padc) = sum(dt, 2);
    end
end
```

The script to obtain the output image is located in convolution_example.m.
The script is also included here:

```
im = imread('pcb.png');
im = im2double(rgb2gray(im));

sobelx = [-1,0,1;-2,0,2;-1,0,1];
sobely = [-1,0,1;-2,0,2;-1,0,1]';

Ix = convolution(sobelx, im);
Iy = convolution(sobely, im);

grad = (Ix.^2 + Iy.^2).^(1/2);

imshow(grad);
```

The following is the output image:



Q4 –
To do frequency domain filtering, I used the built-in functions in MATLAB for taking the forward and backward discrete Fourier transform of the image. The following is the code for filtering an image (located in freqfilt.m):

```
function f = freqfilt(filter, image)
% take fft of image
fftimage = fft2(image);
% first pad the filter, then take its fft
fftfilter = zeros(size(image));
fftfilter(1:size(filter, 1), 1: size(filter, 2)) = filter;
fftfilter = fft2(fftfilter);
% convolution theorem
freq_filtered = fftimage .* fftfilter;
f = ifft2(freq_filtered);
```

The steps to do frequency domain filtering is as follows. First, the discrete Fourier transform of the image is computed using the built-in fft2(img) command. Similarly, the same command will be called to compute the DFT of the kernel. However, for frequency domain filtering to work, the kernel should be of the same size as the image itself. Therefore, the three lines of code after taking the DFT of the image using fft2(img), first pad the kernel such that it has the same size as the image, and such that the kernel is placed to the top left corner of the matrix, and then computes its DFT. Now that we have the DFT of the image and the padded kernel, we use the convolution theorem, i.e.

$$F\{f * g\} = F\{f\}F\{g\}$$

The line of code beginning with freq_filtered applies the convolution theorem: To compute the convolution in the spatial domain, it multiplies the frequency responses of the image and the kernel, element-wise. After everything has been successfully computed, it takes the inverse discrete Fourier transform using the ifft2(freq) command provided by MATLAB.

To display the power spectrum, I wrote a separate one-liner script, located in freqresponse.m:

```
function f = freqresponse(image)
f = abs(fftshift(fft2(image)));
```

This simply returns the magnitude of the frequency response of an image such that the DC component is centered in the middle of the image.

Unfortunately, the filter I used to filter the image is a size 21 Gaussian filter with standard deviation 6, and therefore the spatial response of the filter is quite small.

Here are the outputs of the script provided in freq_example.m.

Script:

```
clear
clc
im = imread('pcb.png');
im = im2double(rgb2gray(im));
%display image
figure
imshow(im);
lpf = fspecial('gaussian', 21, 6);
%display fourier response of lpf(gaussian)
figure
fftfilter = zeros(size(im));
fftfilter(1:size(lpf, 1), 1: size(lpf, 2)) = lpf;
fftfilter = fft2(fftfilter);
imshow(abs(fftshift(fftfilter)));

%display frequency response of log log image
figure
```

```
imshow(log(abs(log(freqresponse(im)))));

im = freqfilt(lpf, im);
%display blurred image
figure
imshow(im);

%display lpf(gaussian) in spatial domain
figure
imshow(lpf);

%display log of image frequency response
figure
imshow(log(freqresponse(im)));
```

The images are:

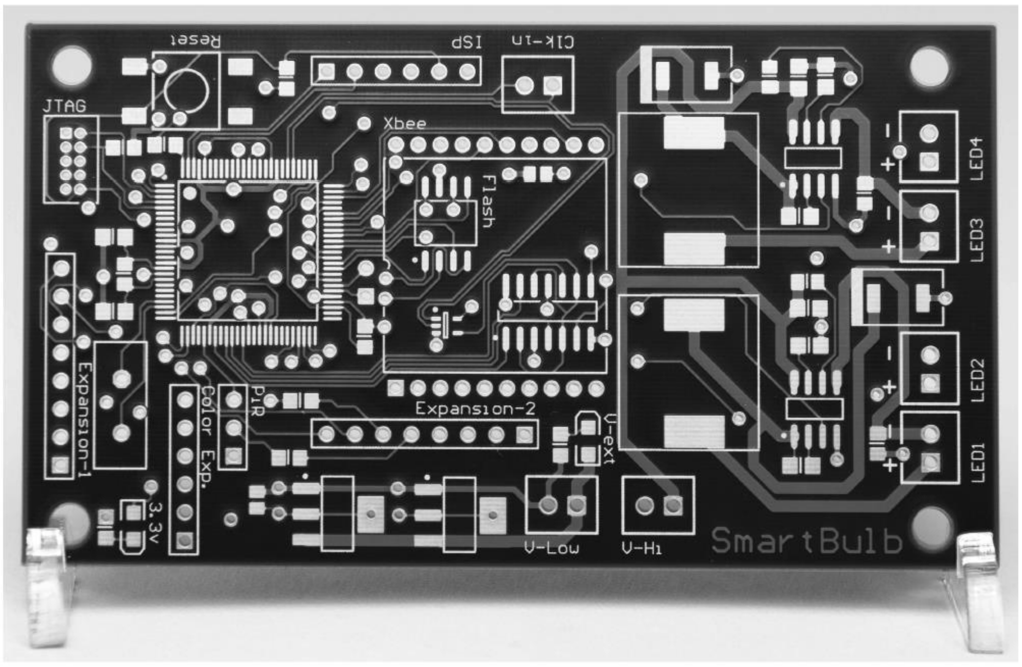


Figure – Original Image

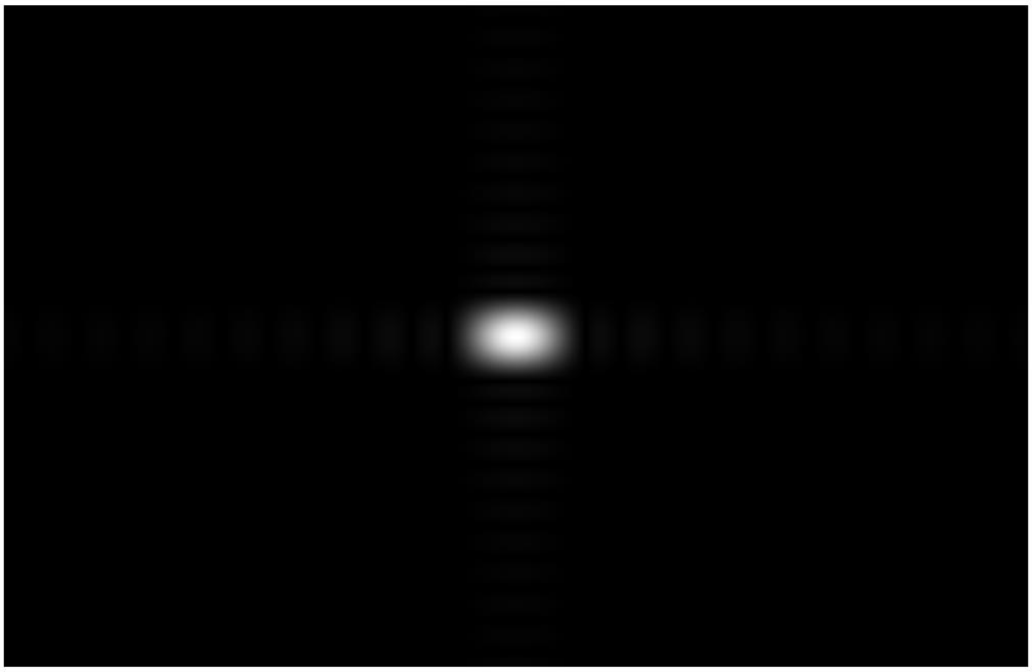


Figure – Gaussian Filter Frequency Domain Response

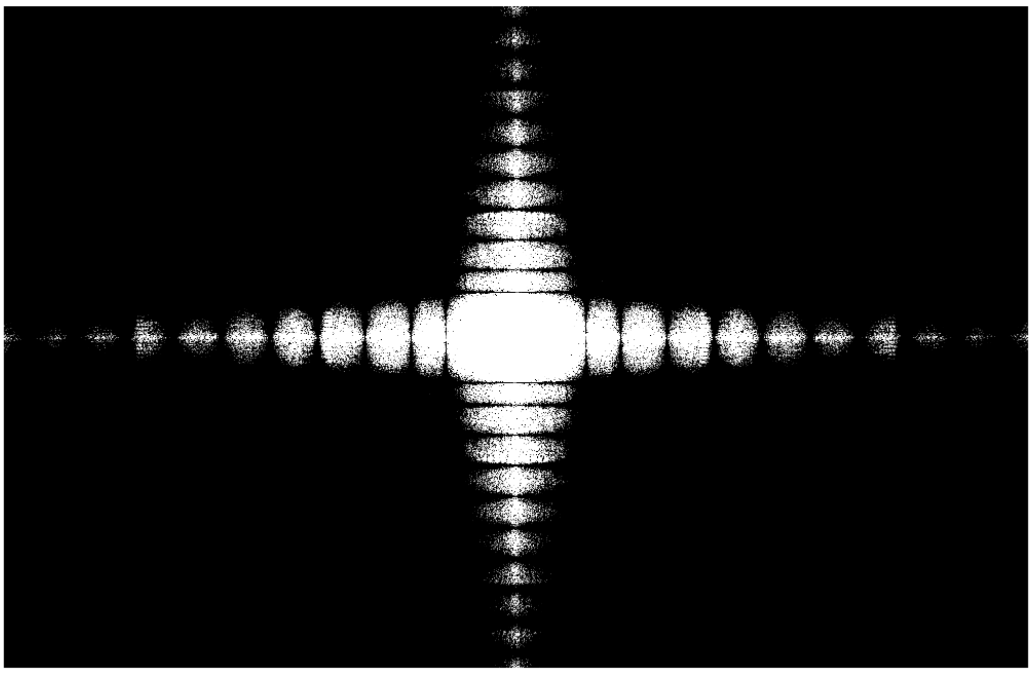


Figure – Frequency response of the Filtered Image

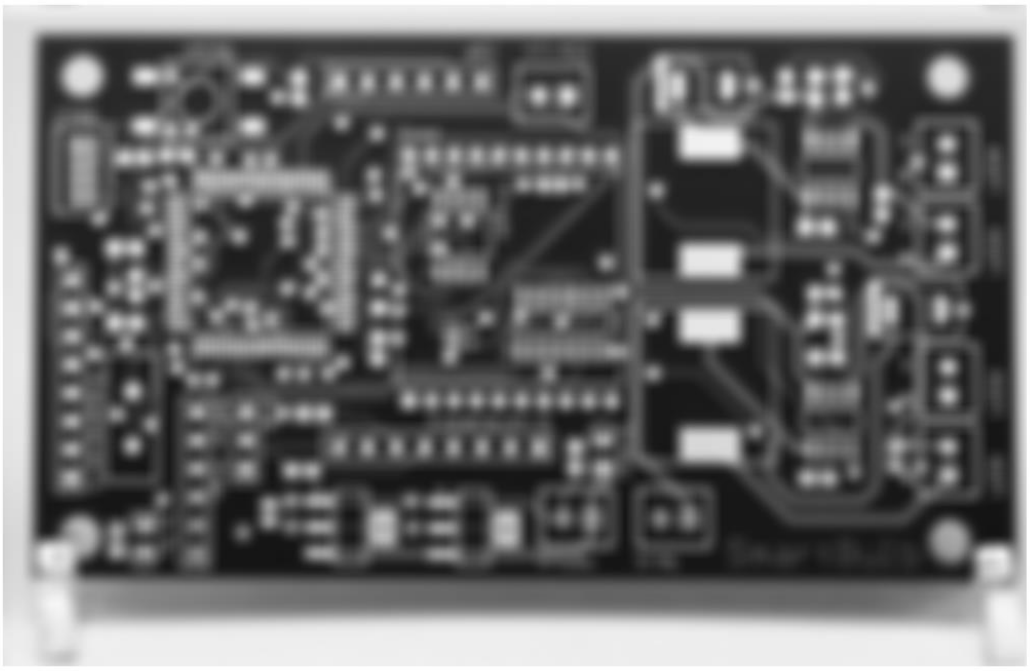


Figure – Blurred Image via filtering with Gaussian, size 21 and standard deviation 6

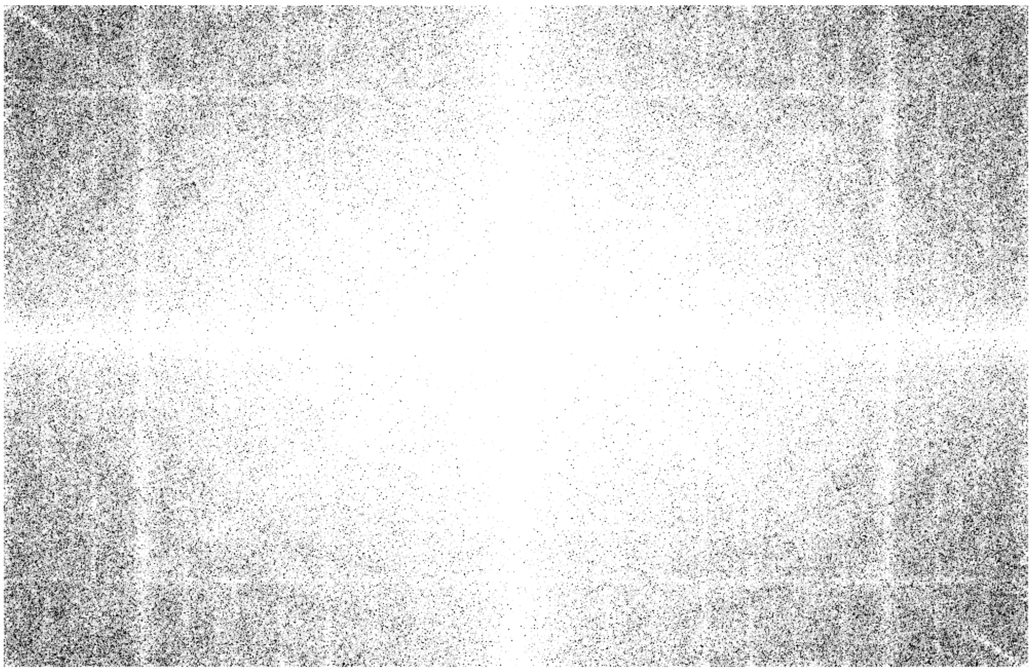


Figure – Log log frequency response of the original image

Figure – Spatial Filter. Rather small to display it here.

Image Source: https://pngio.com/images/png-111786.html