Frequency Domain Filtering

Sobel Filter, Gaussian Lowpass Filter, Butterworth Notch Filter

11712616 王仲华

I. Sobel Filter

1. Introduction

Sobel Filter is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasizing edges. The Sobel operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation that it produces is relatively crude, in particular for high-frequency variations in the image.

2. Algorithm Description

The sobel operator is a 3 by 3 matrix, which can be represented in vertical and horizontal.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

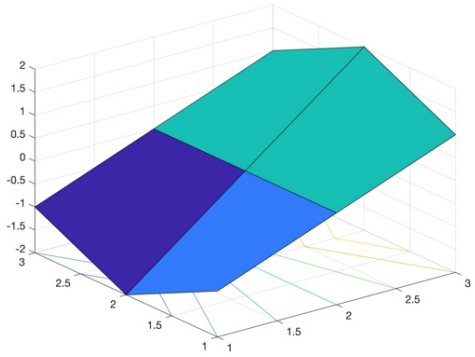

Figure 1: Stereoscopic Representation of Sobel Filter

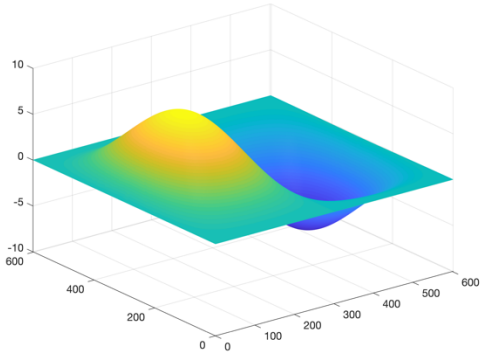

Figure 2: Sobel Filter in Frequency domain

These are convolved with the original image to calculate approximations of the derivatives.

The first step we have to do is to expand the original to a specific size using function.

$$M = m + n - 1$$

Then we add zeros to the image to get the new image. Then for every pixel in the image we multiply the operator by dividing the new image into 3 by 3 sub image. Multiplying two operator we can get two new matrix, then we make a sum for each matrix and calculate the magnitude of the two value, then we can generate the new image. Also we can generate the image in frequency domain instead of in spatial domain. Since we do convolution in spatial domain go generate the image, we can make multiplication in frequency domain. And the result generated should be the same.

In frequency domain, in order to avoid error in the step of Fourier Transform, we also have to expand both image and sobel operator to the same size using the same method. After expanding we'll start to generate the sobel mask. The first step is also expanding the sobel operator to the size the same as the image by adding to zero. Then we multiply $(-1)^{x+y}$ for every point in the operator to center the frequency domain filter. Then by doing Fourier Transform we get the operator in frequency domain. Set the real part of the operator to 0. At last we have to multiply $(-1)^{u+v}$ again, for we multiply $(-1)^{x+y}$ in the previous step, we'll need to shift the filter back. After generating the sobel filter in frequency domain, we multiply the Fourier Transform of the original image with sobel filter and then do inverse Fourier Transform to get the enhanced picture. The result in spatial domain and in frequency domain should be the same.

3. Pseudo Code

i. Spatial Domain

- *Initialize the image and the sobel operator in vertical and horizontal.*
- *Expand the original image. For every pixel point in the new image, apply the multiplication to it and its neighbor in both vertical and horizontal.*
- *Calculate the magnitude using the value in step 2. Generate the new image using the magnitude.*

ii. Frequency Domain

- *Initialize the image and the sobel operator in vertical and horizontal.*
- *Expand the original image and the sobel operator to the same size.*
- *Multiply $(-1)^{x+y}$ with every point of the sobel operator. Do DFT. Set the real part of the sobel filter to 0. Then multiply $(-1)^{u+v}$ again.*
- *Multiply the sobel operator with the image in frequency domain. Then do inverse Fourier Transform to get the new image.*

4. Results and Analysis

Figure 3: Enhanced in spatial domain

Figure 4: Enhanced in frequency domain

As is shown in the figure, after using the sobel mask to the image, the edge of the components in the original image is kept and the other part of the image is reduced, which enhanced profile of the components.

5. Matlab Code

```
function [OutputImage1,OutputImage2] = Sobel_11712616(file_name)
% OutputImage1 is process in spatial domain, outputImage2 is process in
% frequency domain.

image_name = file_name;
image = im2double(imread(image_name));
gx = [-1 0 1; -2 0 2; -1 0 1];
gy = [-1 -2 -1; 0 0 0; 1 2 1];
step = 1;

left = zeros(size(image,2),step*2);
right = zeros(step*2,size(image,2)+step*2);
image = [image,left;right];

mag = zeros(size(image,1),size(image,2));
for i = 1:size(image,1)-2
   for j = 1:size(image,2)-2
      s1 = sum(sum(gx.*image(i:i+2,j:j+2)));
      s2 = sum(sum(gy.*image(i:i+2,j:j+2)));

      mag(i+1,j+1) = sqrt(s1.^2+s2.^2);
   end
end
OutputImage1 = mag;

corner1 = zeros(300,300);
corner2 = zeros(300,299);
```

```
corner3 = zeros(299,299);
cent1 = zeros(3,300);
cent2 = zeros(3,299);
gpx = [corner1,cent1',corner2;cent1,gx,cent2;corner2',cent2',corner3];
gpy = [corner1,cent1',corner2;cent1,gy,cent2;corner2',cent2',corner3];

for i = 1:size(gpx,1)
   for j = 1:size(gpx,2)
      gpx(i,j) = ((-1)^(i+j))*gpx(i,j);
      gpy(i,j) = ((-1)^(i+j))*gpy(i,j);
   end
end

gpx1 = fft2(gpx);
gpy1 = fft2(gpy);

gpx2 = zeros(size(gpx1));
gpy2 = zeros(size(gpy1));

for i = 1:size(gpx2,1)
   for j = 1:size(gpx2,2)
      gpx2(i,j) = imag(gpx1(i,j))*sqrt(-1);
   end
end

for i = 1:size(gpy2,1)
   for j = 1:size(gpy2,2)
      gpy2(i,j) = imag(gpy1(i,j))*sqrt(-1);
   end
end

for i = 1:size(gpx2,1)
   for j = 1:size(gpx2,2)
      gpx2(i,j) = ((-1)^(i+j))*gpx2(i,j);
      gpy2(i,j) = ((-1)^(i+j))*gpy2(i,j);
   end
end

new_image = fft2(image).*gpx2;
new_image2 = abs(ifft2(new_image)*8);
OutputImage2 = new_image2;
end
```

II. Gaussian Lowpass Filter

1. Introduction

A Gaussian filter is a filter whose impulse response is a Gaussian function.

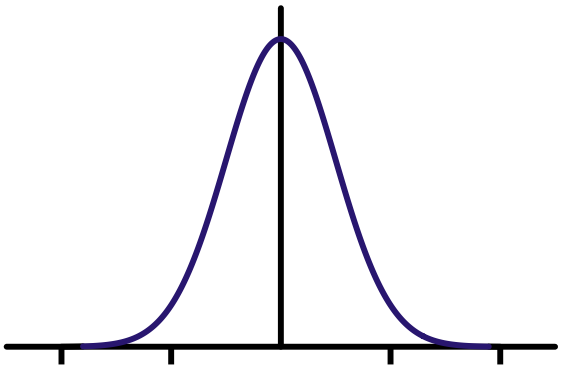

Figure 5: Shape of the impulse response of a typical Gaussian Filter

Gaussian filters have the properties of having no overshoot to a step function input while minimizing the rise and fall time. This behavior is closely connected to the fact that the Gaussian filter has the minimum possible group delay. It is considered the ideal time domain filter. Mathematically, a Gaussian filter modifies the input signal by convolution with a Gaussian function.

2. Algorithm Description

As in the first part, the first part we have to do is to expand the image. In this part, we just simply expand the picture to fourth of its original size and add zeros to the other position.

The second step we have to do is to generate a gaussian lowpass filter. The size of the filter should be the same as the expanded picture. We define a constant D_0 and formula

$$D(u,v) = \left[\left(u - \frac{P}{2}\right)^2 + \left(v - \frac{Q}{2}\right)^2\right]^{\frac{1}{2}}$$

The formula is to represent the distance from point (u,v) in the frequency domain to the center of the frequency rectangle. Then using the formula

$$\mathrm{H(u,v)} = e^{-D^2(u,v)/2D_0}$$

We can generate a gaussian lowpass filter in two dimension. And by changing the value of D_0, we can change the lowpass filter with different bandwidth.

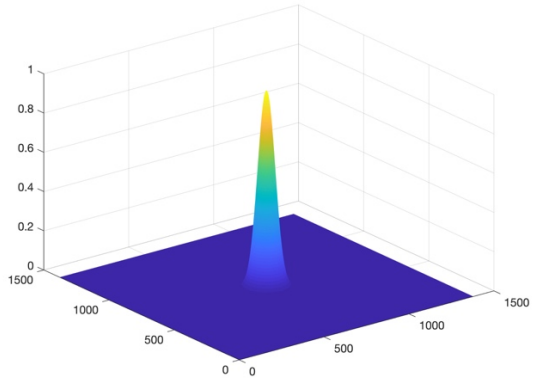

Figure 6: Gaussian Lowpass Filter with $D_0 = 40$

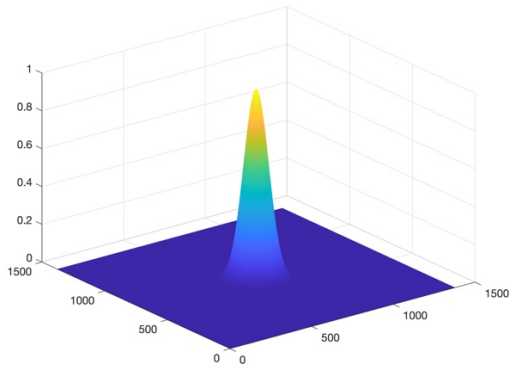

Figure 7: Gaussian Lowpass Filter with $D_0 = 60$

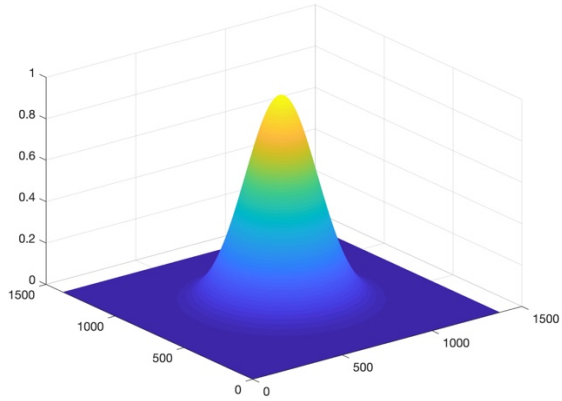

Figure 8: Gaussian Lowpass Filter with $D_0 = 160$

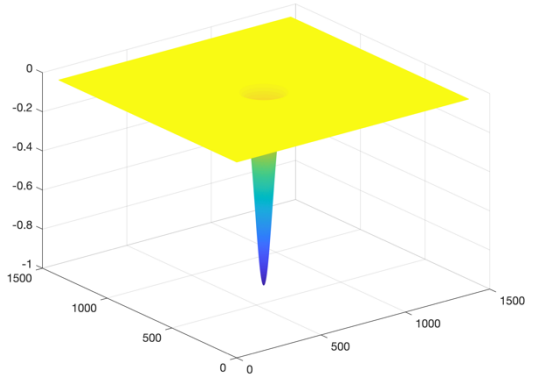

Figure 9: Gaussian Highpass Filter with $D_0 = 40$

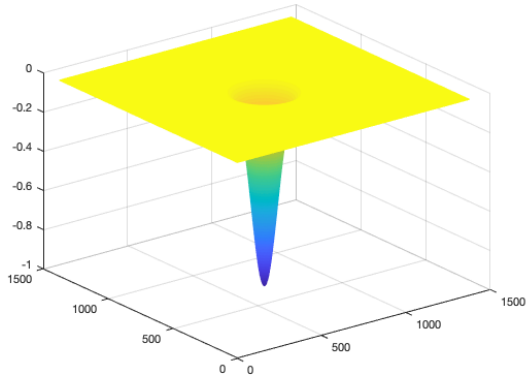

Figure 10: Gaussian Highpass Filter with $D_0 = 60$

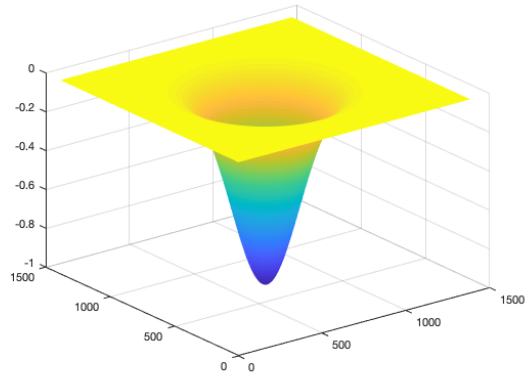

Figure 11: Gaussian Highpass Filter with $D_0 = 160$

Then we multiply the Fourier transform result of the original picture with the lowpass filter. By using different lowpass filter, the picture we enhanced could have different effect. That is because different high frequency part is band by the filter. By doing inverse Fourier transform we can get the enhanced image.

3. Pseudo Code

- *Expand the original image to fourth of its original image.*
- *Using the formula to generate the gaussian lowpass filter that has the same size with the same size as the expanded image.*
- *Transfer the expanded image to frequency domain, multiply the image with the filter.*
- *Do inverse fourier transform to the multiplication result to get the enhanced image.*

4. Results and Analysis

Figure 12: Lowpass enhanced result with $D_0 = 40$

Figure 13: Lowpass enhanced result with $D_0 = 60$

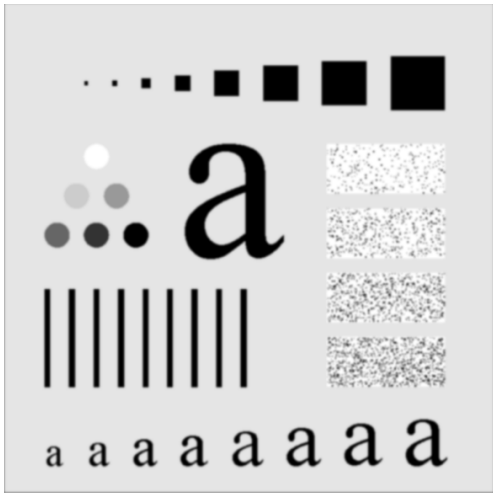

Figure 14: Lowpass enhanced result with $D_0 = 160$

As the increment of D_0, the result become much clearer, this is because with the increase of D_0, the high-frequency part of the image is kept, and the information that the image can present becomes more.

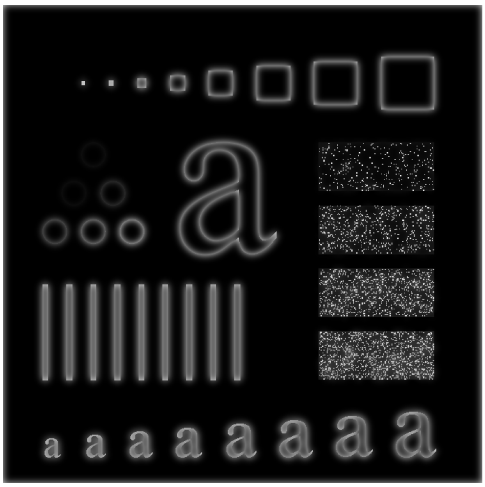

Figure 15: Highpass enhanced result with $D_0 = 40$

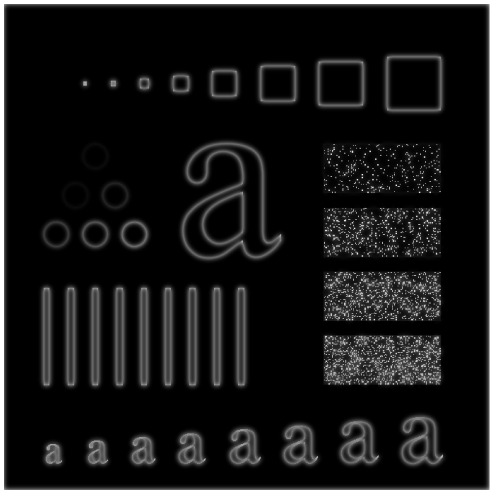

Figure 16: Highpass enhanced result with $D_0 = 60$

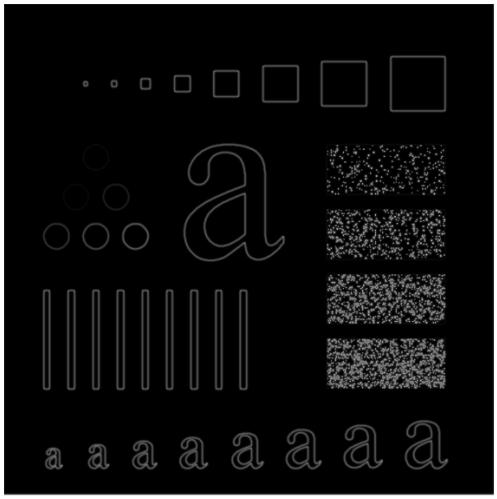

Figure 17: Highpass enhanced result with $D_0 = 160$

In the highpass band result we can see that the low frequency part of the image is limited and the rest of the enhanced image is the high frequency part, most of which are the edges of the components of the original image.

5. Matlab Code

```
function [OutputImage] = Gaussian_11712616(file_name,D0)

image = im2double(imread(file_name));

P = 2*size(image,1);
Q = 2*size(image,2);
corner = zeros(size(image,1),size(image,2));
image = [image,corner;corner,corner];

u = (1:P)';
v = (1:Q);
D = sqrt((u-P/2).^2+(v-Q/2).^2);
H = exp((-D.^2)./(2*D0^2));

new_image = fftshift(fft2(image)).*H;
new_image2 = ifft2(new_image);

new_image2 = new_image2(1:size(image,1)/2,1:size(image,2)/2);

OutputImage = new_image2;
end
```

III. Butterworth Notch Filter

1. Introduction

The Butterworth filter is a type of signal processing filter designed to have a frequency response as flat as possible in the passband. It is also referred to as a maximally flat magnitude filter. The frequency response of the Butterworth filter is maximally flat (i.e. has no ripples) in the passband and rolls off towards zero in the stopband.

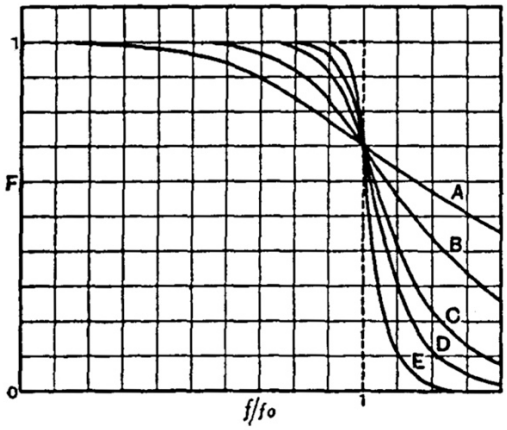

Figure 18: The frequency response plot of Butterworth Filter

2. Algorithm Description

The key point of the algorithm is to generate the notch filter that can band the specialized part in frequency domain to reduce moiré effect. Unlike the lowpass filter, the notch filter is a kind of highpass filter that has different bandwidth and band position. We can use the formula below.

$$H_{NR}(u,v) = \prod_{k=1}^{m}\left[\frac{1}{1+[D_{0k}/D_k(u,v)]^{2n}}\right]\left[\frac{1}{1+[D_{0k}/D_{-k}(u,v)]^{2n}}\right]$$

$$D_k(u,v) = \left[(u - M/2 - u_k)^2 + (v - N/2 - v_k)^2\right]$$

$$D_{-k}(u,v) = \left[(u - M/2 + u_k)^2 + (v - N/2 + v_k)^2\right]$$

Choosing the value of n and D_{0k}, we can generate the notch filter according to different u,v values. And D_{0k} represents the bandwidth or the radius of the bandstop width, different u_k, v_k values represents the position of the bandstop part according to the frequency center.

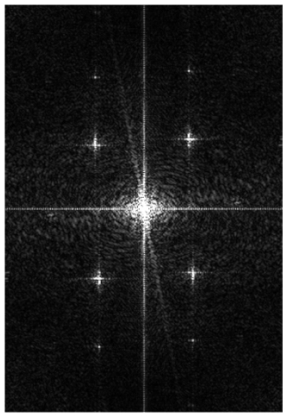

Figure 19: Original image in frequency domain

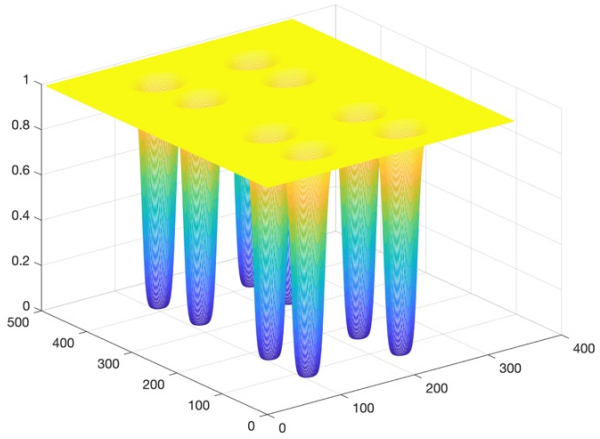

Figure 20: Butterworth Notch Filter designed for the image

From the formula we can found that the stopband of the filter is central symmetric by the center of the frequency domain. Also the image in frequency domain is central symmetric, so we can change the u_k and v_k value to change the position of the band of the filter according to different image. Then we can multiply the filter with the fourier transform result and by doing inverse fourier transform, we can get the enhanced image.

3. Pseudo Code

- *Expand the image to fourth of its original size.*
- *Define the value of D_{0k} and n, define the value of u_k and v_k according to the image result in frequency domain. Generate the notch filter using the formula.*
- *Multiply the filter with the image in frequency domain. Do inverse fourier transform to get the enhanced image.*

4. Results and Analysis

Figure 21: Enhanced image using notch filter

After multiplying the notch filter, the moiré effect in the original image is reduced.

5. Matlab Code

```
function [OutputImage] = Butterworth_11712616(file_name)

image = im2double(imread(file_name));

M = 2*size(image,1);
N = 2*size(image,2);
corner = zeros(size(image,1),size(image,2));
image2 = [image,corner;corner,corner];

f = fftshift(fft2(image2));

D0 = 20;
n = 4;
H_NR = 1;

k = 4;
u = (1:M)';
v = (1:N);
uk = [76,160,86,170];
vk = [60,60,-55,-60];

for i = 1:k
    Dk1 = sqrt((u-M./2-uk(i)).^2+(v-N./2-vk(i)).^2);
    Dk2 = sqrt((u-M./2+uk(i)).^2+(v-N./2+vk(i)).^2);
    H_NR = H_NR.*((1./(1+(D0./Dk1).^(2*n))).*(1./(1+(D0./Dk2).^(2*n))));
end

new_image = f.*H_NR;

new_image2 = abs(ifft2(new_image));
new_image2 = new_image2(1:size(image,1),1:size(image,2));
OutputImage = new_image2;
end
```