MSPL Week 2 何柏昇

Denoise

Mean Filter

平滑降噪，原理主要是計算該點周圍的平均值，來達到降低雜訊，但也會使圖片模糊化。

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Median Filter

取其與周圍的鄰近，找出中位數並以其取代該點，其優點在於不會使圖片過度模糊化。

$$\text{mid}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Gaussian Filter

減少圖像雜訊以及降低細節層次，其視覺效果就像是經過一個半透明屏幕在觀察圖像。少了很多顆粒感，但也變模糊了。

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

$$G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

```
// Denoise: Mean, Median, Gaussian
#include <stdio.h>
#include <ctime>
#include <cmath>
#include "opencv2/opencv.hpp"

using namespace std;
using namespace cv;

//Mean Filter
void meanFilter(Mat &src, Mat &dst, int kernel)
{
    if (!src.data) return;

    //At pixel of image
    for (int y = 0; y < src.rows; y++)
    {
        for (int x = 0; x < src.cols; x++)
        {
            int sum;
            if ((y-kernel >= 0)&&(x-kernel >= 0)&&(y+kernel <
src.rows)&&(x+kernel < src.cols))
            {
                // Sum pixels in the kernel size
                for (int k1 = 0; k1 < (2*kernel+1); k1++)
                    for (int k2 = 0; k2 < (2*kernel+1); k2++)
                        sum += src.at<uchar>(y-kernel+k1, x-kernel+k2);
            // Calculate the mean value
            sum /= ((2*kernel+1)*(2*kernel+1));
            dst.at<uchar>(y,x)=sum;
            }
        }
    }
}

//Median Filter
void medianFilter(Mat &src, Mat &dst, int kernel)
{
```

```
    if (!src.data) return;

    //At pixel of image
    for (int y = 0; y < src.rows; y++)
    {
        for (int x = 0; x < src.cols; x++)
        {
            int tmp[(2*kernel+1)*(2*kernel+1)];
            if ((y-kernel >= 0)&&(x-kernel >= 0)&&(y+kernel <
src.rows)&&(x+kernel < src.cols))
            {
                // List pixels in the kernel size
                for (int k1 = 0; k1 < (2*kernel+1); k1++)
                    for (int k2 = 0; k2 < (2*kernel+1); k2++)
                        tmp[k1 * (2*kernel+1) + k2] = src.at<uchar>(y-
kernel+k1, x-kernel+k2);
            // Finde the median value
            sort(tmp,tmp+((2*kernel+1)*(2*kernel+1)));
            dst.at<uchar>(y, x)=tmp[((2*kernel+1)*(2*kernel+1)-1)/2];
            }
        }
    }
}

// Gaussian Filter

void gaussianFilter(Mat &src, Mat &dst, int kernel, double sigma=
0.84089642)
{
    // Initialise values
    double GKernel[2*kernel+1][2*kernel+1];
    double r, s=2.0 * sigma * sigma;
    double sum=0.0;

    // Generate the gaussian kernel
    for (int y = 0; y < (2*kernel+1); y++)
    {
        for (int x = 0; x < (2*kernel+1); x++)
```

```
        {
            r = ((x-kernel)*(x-kernel)+(y-kernel)*(y-kernel));
            GKernel[y][x] = (exp(-r/s)) / (M_PI * s);
            sum += GKernel[y][x];
        }
    }

    // Normalize the kernel
    for (int y = 0; y < (2*kernel+1); y++)
        for (int x = 0; x < (2*kernel+1); x++)
            {GKernel[y][x] /= sum;
            //cout << GKernel[y][x]<<endl;
            }

    // Convolution image with GKernel
    for (int y = 0; y < src.rows; y++)
    {
        for (int x = 0; x < src.cols; x++)
        {
            if ((y-kernel >= 0)&&(x-kernel >= 0)&&(y+kernel <
src.rows)&&(x+kernel < src.cols))
            {
                // Convolution
                double sum = 0;
                for (int k1 = 0; k1 < (2*kernel+1); k1++)
                    for (int k2 = 0; k2 < (2*kernel+1); k2++)
                        sum += (double)src.at<uchar>(y-kernel+k1, x-
kernel+k2)*GKernel[k1][k2];
                dst.at<uchar>(y,x)=sum;
            }
        }
    }

}

// Salt the image
void salt(Mat &image, int num)
{
```

```
    if (!image.data) return;
    int x, y;
    srand(time(NULL));
    for (int i = 0; i < num; i++)
    {
        x = rand() % image.rows;
        y = rand() % image.cols;
        image.at<uchar>(y,x) = 255;
    }
}

int main(int argc, char** argv)
{
    // Read the image
    Mat image = imread("../images/lena.png",0);

    // Add the salt noise to image
    Mat saltImage = image.clone();
    salt(saltImage, 5000);

    // Setting the kernel size
    if ( argc != 2 || (atoi(argv[1])%2) == 0 || atoi(argv[1]) > image.rows
|| atoi(argv[1]) > image.cols)
    {
        printf("usage: ./denoise <kernel size(3 ,5 ,7, ..., 2N+1)>\n");
        return -1;
    }
    int kernel;
    kernel =  (atoi(argv[1])-1)/2;

    // Mean Filter
    Mat meanImage = image.clone();
    meanFilter(saltImage, meanImage, kernel);

    // Median Filter
    Mat medianImage = image.clone();
    medianFilter(saltImage, medianImage, kernel);
```

```
    // Gaussian Filter
    Mat gaussianImage = image.clone();
    gaussianFilter(saltImage, gaussianImage, kernel);


    // Show images
    imshow("Original",image);
    imshow("Salt",saltImage);
    imshow("Mean",meanImage);
    imshow("Median",medianImage);
    imshow("Gaussian",gaussianImage);
    waitKey();

    // Save images
    imwrite("../images/salt.png", saltImage);
    imwrite("../images/mean.png", meanImage);
    imwrite("../images/median.png", medianImage);
    imwrite("../images/gaussian.png", gaussianImage);

    return 0;
}
```

Original

Salt

Mean

Median

Gaussian

Edge Detection

Sobel

分別計算：

1. 偏 x 方向的G_x
2. 偏 y 方向的 G_y
3. 求絕對值，壓縮到[0, 255]區間

$$G(x,y) = G_x + G_y$$

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Laplacian

一階差分：

$$f'(x) = f(x) - f(x-1)$$

二階差分：

$$f'(x) = \big(f(x+1) - f(x)\big) - (f(x) - f(x-1)$$
$$= f(x-1) - 2f(x) + f(x+1)$$

0	1	0
1	−4	1
0	1	0

二維:

$$f'(x,y) = -4f(x,y) + f(x-1,y) + f(x+1,y) + f(x,y-1) + f(x,y+1)$$

1	1	1
1	−8	1
1	1	1

−1	2	−1
2	−4	2
−1	2	−1

Canny

1.高斯濾波器平滑圖像。

2.一階差分偏導計算梯度值和方向➔Sobel。

3.對梯度值不是極大值的地方進行抑制。

4.用雙閾值連接圖上的聯通點。

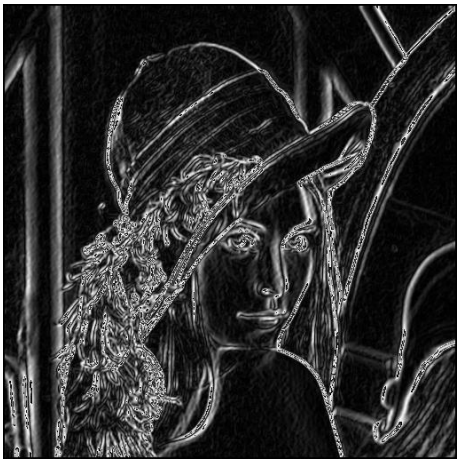
```cpp
// Edge Detect: Sobel, Laplacian
#include <stdio.h>
#include <cmath>
#include "opencv2/opencv.hpp"

using namespace std;
using namespace cv;

void sobelKernel(Mat &src, Mat &dst, int kernel, int sita)
{
    // Initialise values
    int size= 2 * kernel + 1;
    double Gx[size][size], Gy[size][size], G[size][size];

    // Generate sobel kernel
    for (int y = 0; y < size; y++)
    {
        for (int x = 0; x < size; x++)
        {
            if (y != kernel || x != kernel)
            {
                // Gx f(x,y) dot (1,0)
                Gx[y][x] = (double)(x-kernel) / (double)((x-kernel)*(x-
kernel) + (y-kernel)*(y-kernel));
                // Gy f(x,y) dot (0,1)
                Gy[y][x] = (double)(y-kernel) / (double)((x-kernel)*(x-
kernel) + (y-kernel)*(y-kernel));
            }
            else
                Gx[y][x]=Gy[y][x]=0;
            // G = (Gx*cosa + Gy*sina)/2
            G[y][x] = (Gx[y][x] * cos(sita) + Gy[y][x] * sin(sita));
        }
    }

    // Convolution image with soble kernel
    for (int y = 0; y < src.rows; y++)
    {
```

```
        for (int x = 0; x < src.cols; x++)
        {
            if ((y-kernel >= 0)&&(x-kernel >= 0)&&(y+kernel <
src.rows)&&(x+kernel < src.cols))
            {
                // Convolution
                double sum = 0;
                for (int k1 = 0; k1 < size; k1++)
                    for (int k2 = 0; k2 < size; k2++)
                        sum += src.at<uchar>(y-kernel+k1, x-kernel+k2) *
G[k1][k2];
                dst.at<uchar>(y,x)=abs(sum);
            }
            else
                dst.at<uchar>(y,x)=0;
        }
    }
}

// Combine two images to one
void addImage(Mat &src1, Mat &src2,Mat &dst)
{
    for (int y = 0; y < dst.rows; y++)
        for (int x = 0; x < dst.cols; x++)
            dst.at<uchar>(y,x)=sqrt(pow(src1.at<uchar>(y,x),2) +
pow(src2.at<uchar>(y,x),2));
}

// Laplacian  Filter
void laplacianFilter(Mat &src, Mat &dst, int kernel,double sigma = 84089642)
{
    // Initialise values
    int size = 2 * kernel + 1;
    double LKernel[size][size];

    // Generate sobel kernel
    for (int y = 0; y < size; y++)
    {
```

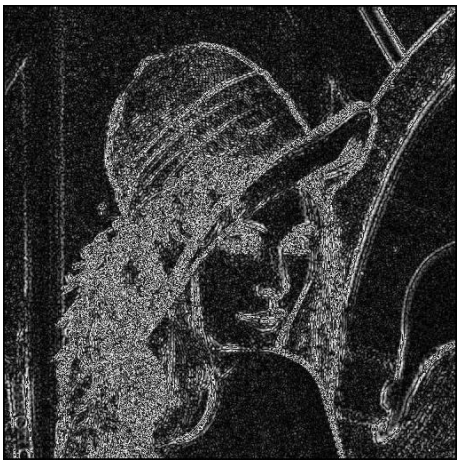
```
        for (int x = 0; x < size; x++)
        {
            if (y != kernel || x != kernel)
                LKernel[y][x] = -1;
            else
                LKernel[y][x] = size * size - 1;
        }
    }

    // Convolution image with GKernel
    for (int y = 0; y < src.rows; y++)
    {
        for (int x = 0; x < src.cols; x++)
        {
            if ((y-kernel >= 0)&&(x-kernel >= 0)&&(y+kernel <
src.rows)&&(x+kernel < src.cols))
            {
                // Convolution
                double sum = 0;
                for (int k1 = 0; k1 < size; k1++)
                    for (int k2 = 0; k2 < size; k2++)
                        sum += (double)src.at<uchar>(y-kernel+k1, x-
kernel+k2)*LKernel[k1][k2];
                dst.at<uchar>(y,x)=abs(sum);
            }
            else
                dst.at<uchar>(y,x) = 0;
        }
    }
}


int main(int argc, char** argv)
{

    // Read the image
    Mat image = imread("../images/lena.png",0);

    // Setting the kernel size
```

```
    if ( argc != 2 || (atoi(argv[1])%2) == 0 || atoi(argv[1]) > image.rows
|| atoi(argv[1]) > image.cols)
    {
        printf("usage: ./edgeDetect <kernel size(3 ,5 ,7, ..., 2N+1)>\n");
        return -1;
    }
    int kernel;
    kernel =  (atoi(argv[1])-1)/2;

    // Sobel
    Mat sobelImageX = image.clone(), sobelImageY = image.clone(), sobelImage
= image.clone();
    sobelKernel(image, sobelImageX, kernel, (0*M_PI/180));
    sobelKernel(image, sobelImageY, kernel, (90*M_PI/180));
    addImage(sobelImageX, sobelImageY, sobelImage);

    // Laplacian of Gaussian
    Mat LaplacianImage = image.clone();
    laplacianFilter(image,LaplacianImage,kernel);

    // Show images
    imshow("Original",image);
    imshow("Sobel",sobelImage);
    imshow("Laplacian", LaplacianImage);

    waitKey();

    // Save images
    imwrite("../images/sobel.png", sobelImage);
    imwrite("../images/sobelX.png", sobelImageX);
    imwrite("../images/sobelY.png", sobelImageY);
    imwrite("../images/laplacian.png", LaplacianImage);

}
```

Original



Sobel



Laplacian

```
/* Edge Detect: Canny
    * Step 1: Apply a Gaussian blur
    * Step 2: Find edge gradient strength and direction
    * Step 3: Trace along the edges
    * Step 4: Suppress non-maximum edges
*/
#include <stdio.h>
#include <unistd.h>
#include <cmath>
#include "opencv2/opencv.hpp"

using namespace std;
using namespace cv;

void gaussianFilter(Mat &src, Mat &dst, int kernel, double sigma= 1.4)
{
    // Initialise values
    int size = 2 * kernel + 1;
    double GKernel[size][size];
    double r = 0, s=2.0 * sigma * sigma, sum = 0;

    // Gaussian Kernel
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            int y = i -kernel, x = j - kernel;
            r = pow(x,2) + pow(y,2);
            double normal = 1 / (s * M_PI);
            GKernel[y][x] = exp(-(r) / s) * normal;
            sum +=  GKernel[y][x];
        }
    }

    // Normalize the kernel
    for (int y = 0; y < size; y++)
        for (int x = 0; x < size; x++)
            GKernel[y][x] /= sum;
```

```
    // Convolution image with GKernel
    for (int y = 0; y < src.rows; y++)
    {
        for (int x = 0; x < src.cols; x++)
        {
            if ((y-kernel >= 0)&&(x-kernel >= 0)&&(y+kernel < 
src.rows)&&(x+kernel < src.cols))
            {
                // Convolution
                sum = 0;
                for (int k1 = 0; k1 < (size); k1++)
                    for (int k2 = 0; k2 < (size); k2++)
                        sum += (double)src.at<uchar>(y-kernel+k1, x-
kernel+k2)*GKernel[k1][k2];
                dst.at<uchar>(y,x)=sum;
            }
        }
    }
}

void sobelFilter(Mat &src, Mat &dst, Mat &dst2,int kernel)
{
    // Initialise values
    int size= 2 * kernel + 1;
    double Gx[size][size], Gy[size][size];
    double angle[src.rows][src.cols];
    double G[src.rows][src.cols];
    double Gmax=0.0;
    double sum = 0.0, sumX = 0.0, sumY = 0.0;
    for (int i = 0; i < src.rows; i++)
    {
        for (int j = 0; j < src.cols; j++)
        {
            angle[j][i] = 0.0;
            G[j][i] = 0.0;
        }
    }
```

```
    // Sobel kernel
    for (int y = 0; y < size; y++)
    {
        for (int x = 0; x < size; x++)
        {
            if (y != kernel || x != kernel)
            {
                // Gx f(x,y) dot (1,0)
                Gx[y][x] = (double)(x-kernel) / (double)((x-kernel)*(x-
kernel) + (y-kernel)*(y-kernel));
                // Gy f(x,y) dot (0,1)
                Gy[y][x] = (double)(y-kernel) / (double)((x-kernel)*(x-
kernel) + (y-kernel)*(y-kernel));
            }
            else
                Gx[y][x]=Gy[y][x]=0;
        }
    }

    // Convolution image with Gx and Gy
    for (int y = 0; y < src.rows; y++)
    {
        for (int x = 0; x < src.cols; x++)
        {
            if ((y-kernel > 0)&&(x-kernel > 0)&&(y+kernel <
src.rows)&&(x+kernel < src.cols))
            {
                // Convolution
                sumX = 0, sumY=0;
                for (int k1 = 0; k1 < (size); k1++)
                    for (int k2 = 0; k2 < (size); k2++)
                    {
                        sumX += src.at<uchar>(y-kernel+k1, x-
kernel+k2)*Gx[k1][k2];
                        sumY += src.at<uchar>(y-kernel+k1, x-
kernel+k2)*Gy[k1][k2];
                    }
```

```
                G[y][x] = sqrt(pow(sumX,2) + pow(sumY,2));
                if (G[y][x] > Gmax)
                    Gmax = G[y][x];
                angle[y][x] = (atan2(sumX,sumY) / M_PI) * 180.0;
                if (angle[y][x] < 0)
                    angle[y][x] += 180;
            }
        }
    }

    // Normalize
    for (int y = 0; y <  src.rows; y++)
        for (int x = 0; x <  src.cols; x++)
                dst.at<uchar>(y,x) = (uchar)(G[y][x]  * (255/Gmax));



    // Non Max Suppression
    for (int y = 0; y < src.rows; y++)
    {
        for (int x = 0; x < src.cols; x++)
        {
            if ((y-kernel >= 0)&&(x-kernel >= 0)&&(y+kernel <
src.rows)&&(x+kernel < src.cols))
            {
                int q = 255, r = 255;
                if ((0 <= angle[y][x] < 22.5) || (157.5 <= angle[y][x] <=
180))
                {
                    q = dst.at<uchar>(y,x+1);
                    r = dst.at<uchar>(y,x-1);
                }
                else if ((22.5 <= angle[y][x]) && (angle[y][x] < 67.5))
                {
                    q = dst.at<uchar>(y+1,x-1);
                    r = dst.at<uchar>(y-1,x+1);
                }
                else if ((67.5 <= angle[y][x]) && (angle[y][x] < 112.5))
```

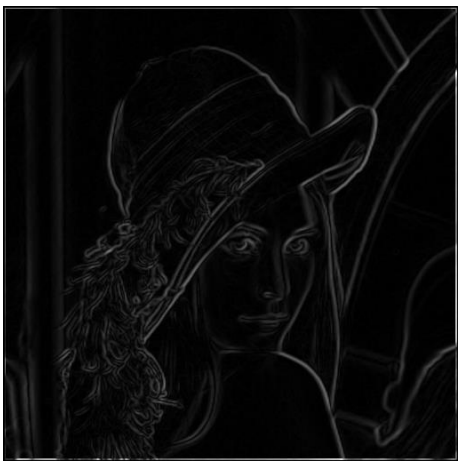
```
                {
                    q = dst.at<uchar>(y+1,x);
                    r = dst.at<uchar>(y-1,x);
                }
                else if ((112.5 <= angle[y][x]) && (angle[y][x] < 157.5))
                {
                    q = dst.at<uchar>(y-1,x-1);
                    r = dst.at<uchar>(y+1,x+1);
                }

                if ((dst.at<uchar>(y,x) >= q) && (dst.at<uchar>(y,x) >= r))
                    dst2.at<uchar>(y,x) = dst.at<uchar>(y,x);
                else
                    dst2.at<uchar>(y,x) = 0;
            }
            else
                dst2.at<uchar>(y,x) = 0;
        }
    }
}

// double thershold
void db_thershold(Mat &src, Mat &dst, int kernel,int low, int high)
{
    int G[src.rows][src.cols];
    int size = 2 * kernel + 1;
    for (int y = 0; y < src.rows; y++)
    {
        for (int x = 0; x < src.cols; x++)
        {
            if ((y-kernel >= 0)&&(x-kernel >= 0)&&(y+kernel <
src.rows)&&(x+kernel < src.cols))
            {
                if (src.at<uchar>(y,x)<low) G[y][x] = 0;
                else if ((src.at<uchar>(y,x)>low) &&
(src.at<uchar>(y,x)<high))
                {
                    bool edge = false;
```

```
                    for (int k1 = 0; k1 < (size); k1++)
                    {
                        for (int k2 = 0; k2 < (size); k2++)
                        {
                            if (src.at<uchar>(y-kernel+k1, x-kernel+k2)>high)
                                {
                                    edge = true;
                                    break;
                                }
                        }
                    }
                    if (!edge) G[y][x] = 0;
                }
                dst.at<uchar>(y,x) = G[y][x];
            }
        }
    }
}

int main(int argc, char** argv)
{
    // Read the image
    Mat image = imread("../images/lena.png",0);
    Mat gaussianImage = image.clone();
    Mat sobelImage = image.clone();
    Mat suppressionImage = image.clone();
    Mat dbImage = suppressionImage.clone();

    // Gaussian
    int kernel;
    printf("Enter kernel size of gaussian filter(3 ,5 ,7, ..., 2N+1): ");
    scanf("%d",&kernel);
    if ((kernel%2) == 0 || kernel > image.rows || kernel > image.cols ||
kernel == 1)
        {
            printf("Error value!!\n");
            return -1;
        }
```

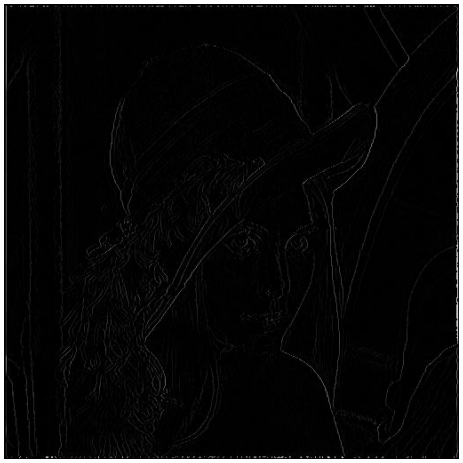
```
    kernel =  (kernel-1)/2;
    gaussianFilter(image, gaussianImage, kernel);

    // Sobel
    printf("Enter kernel size of sobel filter(3 ,5 ,7, ..., 2N+1): ");
    scanf("%d",&kernel);
    if ((kernel%2) == 0 || kernel > image.rows || kernel > image.cols ||
kernel == 1)
        {
            printf("Error value!!\n");
            return -1;
        }
    kernel =  (kernel-1)/2;
    sobelFilter(gaussianImage, sobelImage, suppressionImage, kernel);

    // Double thershold
    printf("Enter kernel size of double thershold(3 ,5 ,7, ..., 2N+1): ");
    scanf("%d",&kernel);
    if ((kernel%2) == 0 || kernel > image.rows || kernel > image.cols ||
kernel == 1)
        {
            printf("Error value!!\n");
            return -1;
        }
    kernel =  (kernel-1)/2;
    int low=0, high=0;
    printf("Enter low and high value for double thershold: ");
    scanf("%d %d",&low, &high);
    if (low > high || high > 254 || low < 0)
        {
            printf("Error value!!%d %d\n",low ,high);
            return -1;
        }
    db_thershold(suppressionImage, dbImage, kernel, low, high);

    // Show images
    /*imshow("Original",image);
    imshow("Gaussian", gaussianImage);
```

```
    imshow("Sobel", sobelImage);
    imshow("Suppression", suppressionImage);
    imshow("Double thershold", dbImage);
    waitKey();*/

    // Save images
    imwrite("../images/1-gaussian.png", gaussianImage);
    imwrite("../images/2-sobel.png", sobelImage);
    imwrite("../images/3-suppression.png", suppressionImage);
    imwrite("../images/4-dbThershold.png", dbImage);

    return 0;
}
```

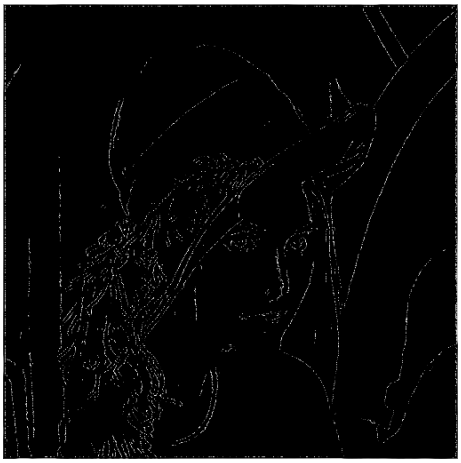
1-Gaussian



2-Sobel



3-Supperssion



4-Double thershold

Reference

1. https://blog.csdn.net/qinghuaci666/article/details/81737624?spm=1001.2101.3001.6661.1&utm_medium=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1-81737624-blog-7297736.pc_relevant_default&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1-81737624-blog-7297736.pc_relevant_default&utm_relevant_index=1
2. http://www.librow.com/articles/article-1
3. https://www.geeksforgeeks.org/gaussian-filter-generation-c/
4. https://blog.csdn.net/rocky_shared_image/article/details/7238796
5. https://medium.com/@bob800530/python-gaussian-filter-%E6%A6%82%E5%BF%B5%E8%88%87%E5%AF%A6%E4%BD%9C-676aac52ea17
6. https://www.twblogs.net/a/5b7ac2042b7177392c96c153
7. https://stackoverflow.com/questions/9567882/sobel-filter-kernel-of-large-size
8. https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html
9. https://medium.com/%E9%9B%BB%E8%85%A6%E8%A6%96%E8%A6%BA/%E9%82%8A%E7%B7%A3%E5%81%B5%E6%B8%AC-%E6%8B%89%E6%99%AE%E6%8B%89%E6%96%AF%E7%AE%97%E5%AD%90-laplacian-operator-ea877f1945a0
10. https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html
11. https://medium.com/@bob800530/opencv-%E5%AF%A6%E4%BD%9C%E9%82%8A%E7%B7%A3%E5%81%B5%E6%B8%AC-canny%E6%BC%94%E7%AE%97%E6%B3%95-d6e0b92c0aa3

Github

MSPL/ week2