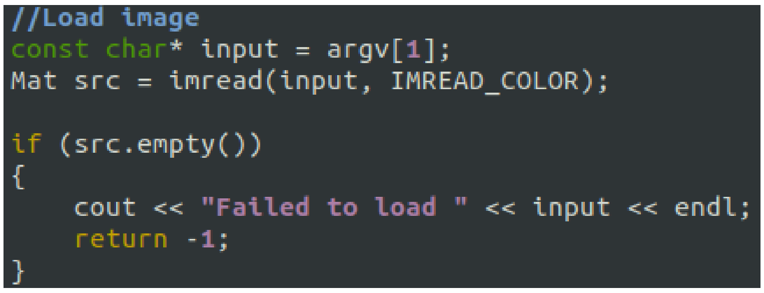
Computer vision HW1

Work flow:

input image -> transform gray -> Gaussian Blur -> Canny Edge detection
-> Hough Transform

input image:

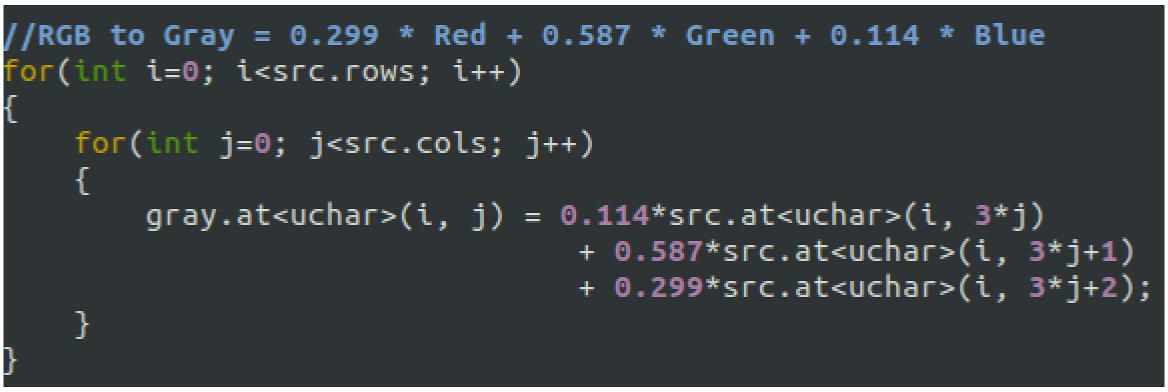
use library of opencv “imread”
and set the sharpenkernel



```
//Load image
const char* input = argv[1];
Mat src = imread(input, IMREAD_COLOR);

if (src.empty())
{
    cout << "Failed to load " << input << endl;
    return -1;
}
```

transform gray:

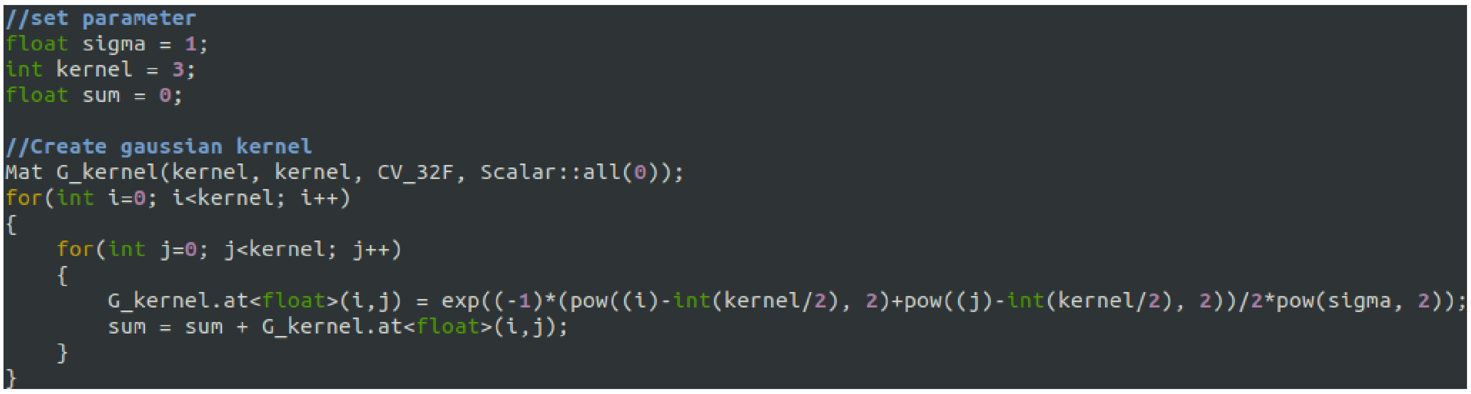
transform formula: Gray = R*0.299 + G*0.587 + B*0.114
Calculation and put output in Mat gray



```
//RGB to Gray = 0.299 * Red + 0.587 * Green + 0.114 * Blue
for(int i=0; i<src.rows; i++)
{
    for(int j=0; j<src.cols; j++)
    {
        gray.at<uchar>(i, j) = 0.114*src.at<uchar>(i, 3*j)
                                + 0.587*src.at<uchar>(i, 3*j+1)
                                + 0.299*src.at<uchar>(i, 3*j+2);
    }
}
```

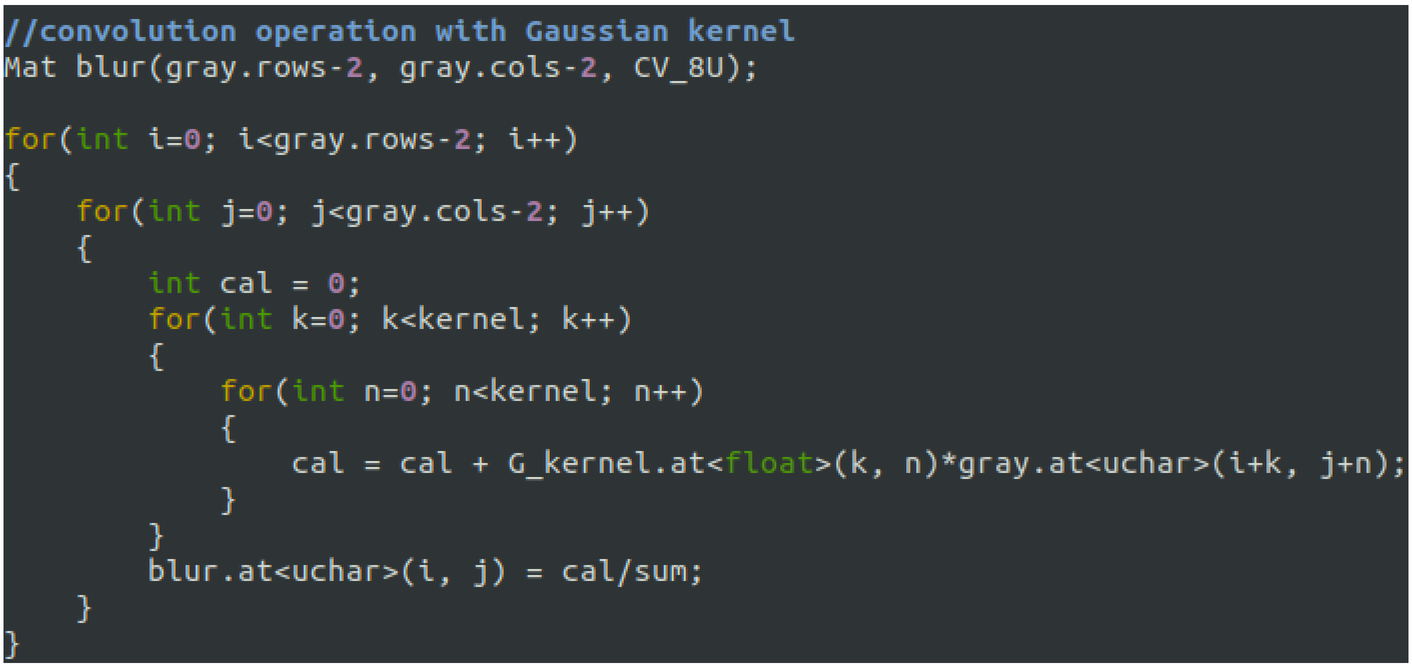
Gaussian Blur:

transform formula: exp(-(x^2+y^2)/2σ^2)



```
//set parameter
float sigma = 1;
int kernel = 3;
float sum = 0;

//Create gaussian kernel
Mat G_kernel(kernel, kernel, CV_32F, Scalar::all(0));
for(int i=0; i<kernel; i++)
{
    for(int j=0; j<kernel; j++)
    {
        G_kernel.at<float>(i,j) = exp((-1)*(pow((i)-int(kernel/2), 2)+pow((j)-int(kernel/2), 2))/2*pow(sigma, 2));
        sum = sum + G_kernel.at<float>(i,j);
    }
}
```

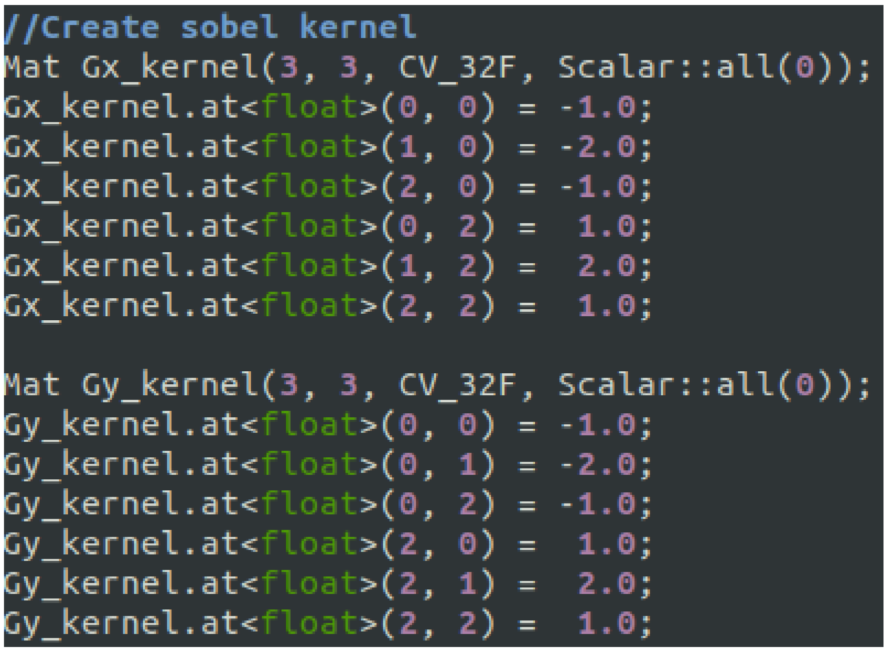
calculation convolution with gaussian kernel and put output in Mat blur



```
//convolution operation with Gaussian kernel
Mat blur(gray.rows-2, gray.cols-2, CV_8U);

for(int i=0; i<gray.rows-2; i++)
{
    for(int j=0; j<gray.cols-2; j++)
    {
        int cal = 0;
        for(int k=0; k<kernel; k++)
        {
            for(int n=0; n<kernel; n++)
            {
                cal = cal + G_kernel.at<float>(k, n)*gray.at<uchar>(i+k, j+n);
            }
        }
        blur.at<uchar>(i, j) = cal/sum;
    }
}
```

Canny Edge detection :

after gaussian blur, I do about canny



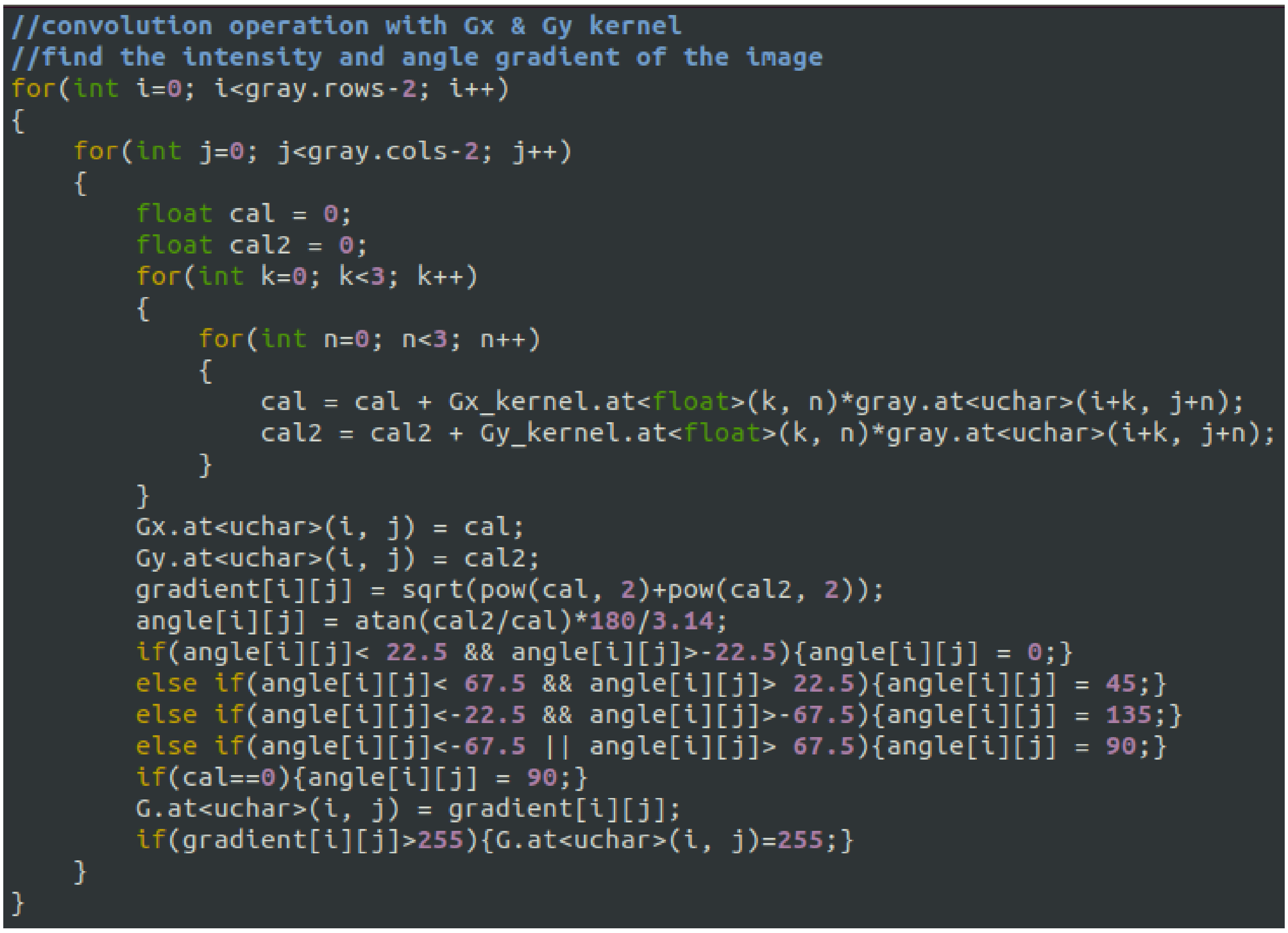
```
//Create sobel kernel
Mat Gx_kernel(3, 3, CV_32F, Scalar::all(0));
Gx_kernel.at<float>(0, 0) = -1.0;
Gx_kernel.at<float>(1, 0) = -2.0;
Gx_kernel.at<float>(2, 0) = -1.0;
Gx_kernel.at<float>(0, 2) =  1.0;
Gx_kernel.at<float>(1, 2) =  2.0;
Gx_kernel.at<float>(2, 2) =  1.0;

Mat Gy_kernel(3, 3, CV_32F, Scalar::all(0));
Gy_kernel.at<float>(0, 0) = -1.0;
Gy_kernel.at<float>(0, 1) = -2.0;
Gy_kernel.at<float>(0, 2) = -1.0;
Gy_kernel.at<float>(2, 0) =  1.0;
Gy_kernel.at<float>(2, 1) =  2.0;
Gy_kernel.at<float>(2, 2) =  1.0;
```

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

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

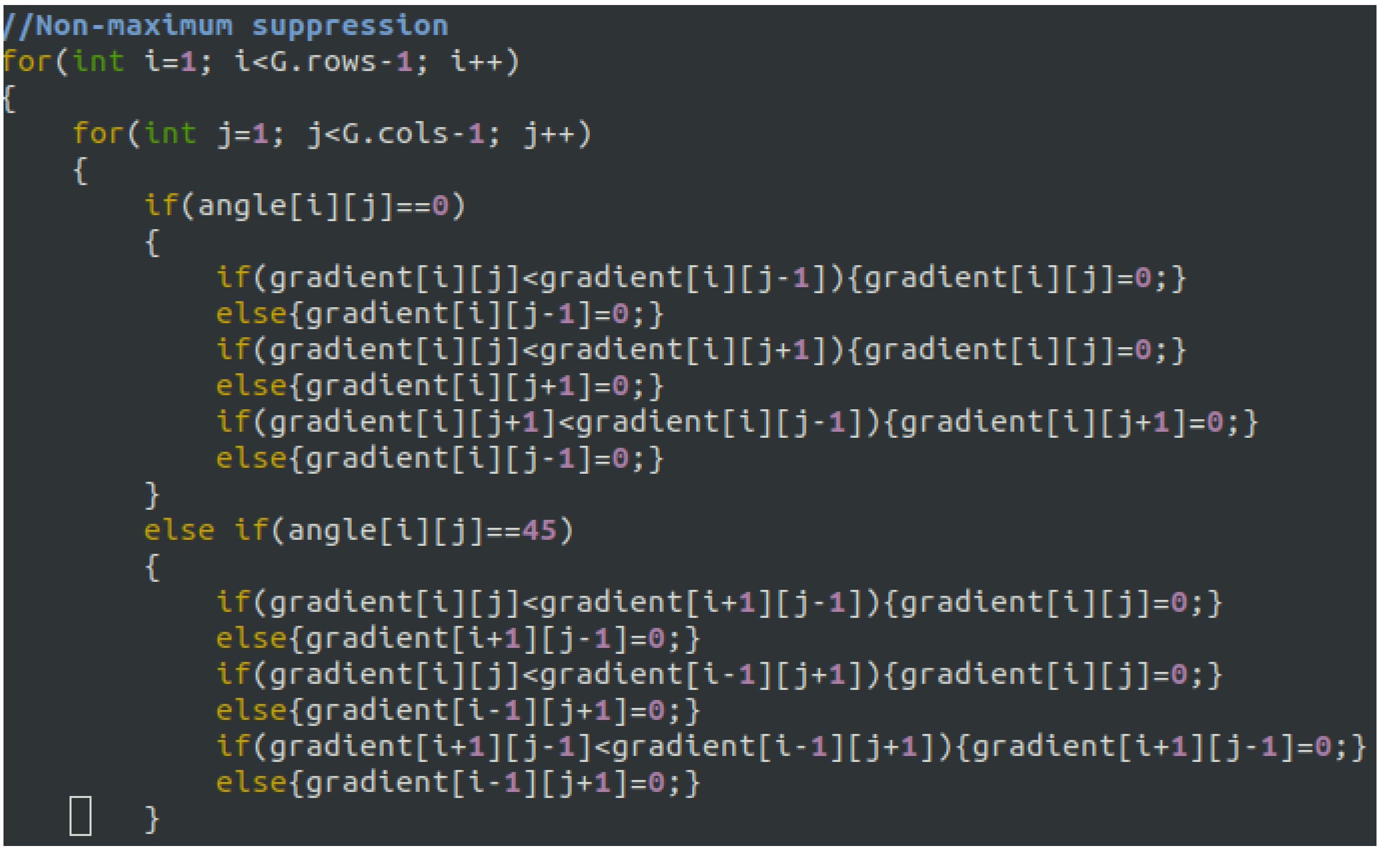
Find the intensity gradient of the image

convolution operation with Gx & Gy kernel

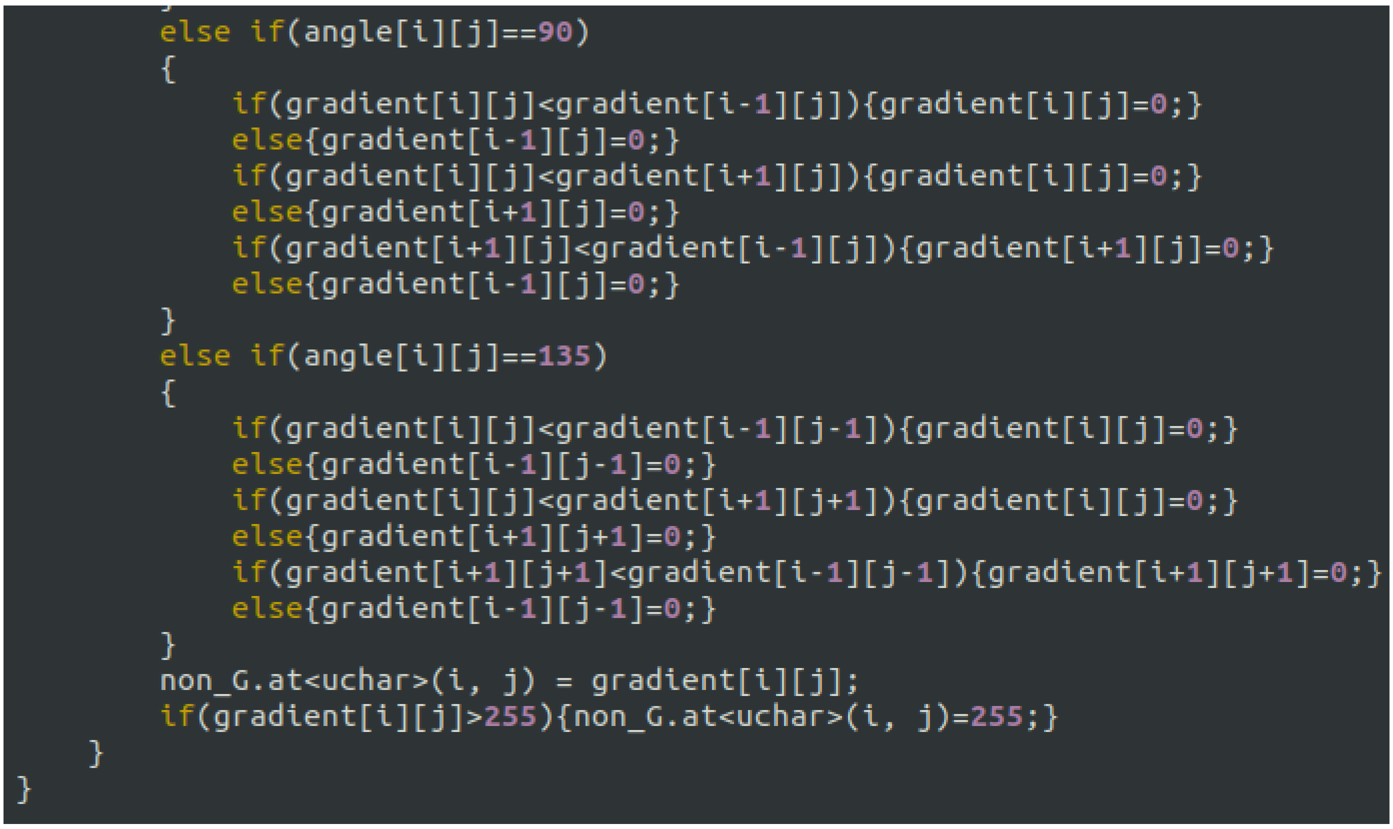


```
//convolution operation with Gx & Gy kernel
//find the intensity and angle gradient of the image
for(int i=0; i<gray.rows-2; i++)
{
    for(int j=0; j<gray.cols-2; j++)
    {
        float cal = 0;
        float cal2 = 0;
        for(int k=0; k<3; k++)
        {
            for(int n=0; n<3; n++)
            {
                cal = cal + Gx_kernel.at<float>(k, n)*gray.at<uchar>(i+k, j+n);
                cal2 = cal2 + Gy_kernel.at<float>(k, n)*gray.at<uchar>(i+k, j+n);
            }
        }
        Gx.at<uchar>(i, j) = cal;
        Gy.at<uchar>(i, j) = cal2;
        gradient[i][j] = sqrt(pow(cal, 2)+pow(cal2, 2));
        angle[i][j] = atan(cal2/cal)*180/3.14;
        if(angle[i][j]< 22.5 && angle[i][j]>-22.5){angle[i][j] = 0;}
        else if(angle[i][j]< 67.5 && angle[i][j]> 22.5){angle[i][j] = 45;}
        else if(angle[i][j]<-22.5 && angle[i][j]>-67.5){angle[i][j] = 135;}
        else if(angle[i][j]<-67.5 || angle[i][j]> 67.5){angle[i][j] = 90;}
        if(cal==0){angle[i][j] = 90;}
        G.at<uchar>(i, j) = gradient[i][j];
        if(gradient[i][j]>255){G.at<uchar>(i, j)=255;}
    }
}
```

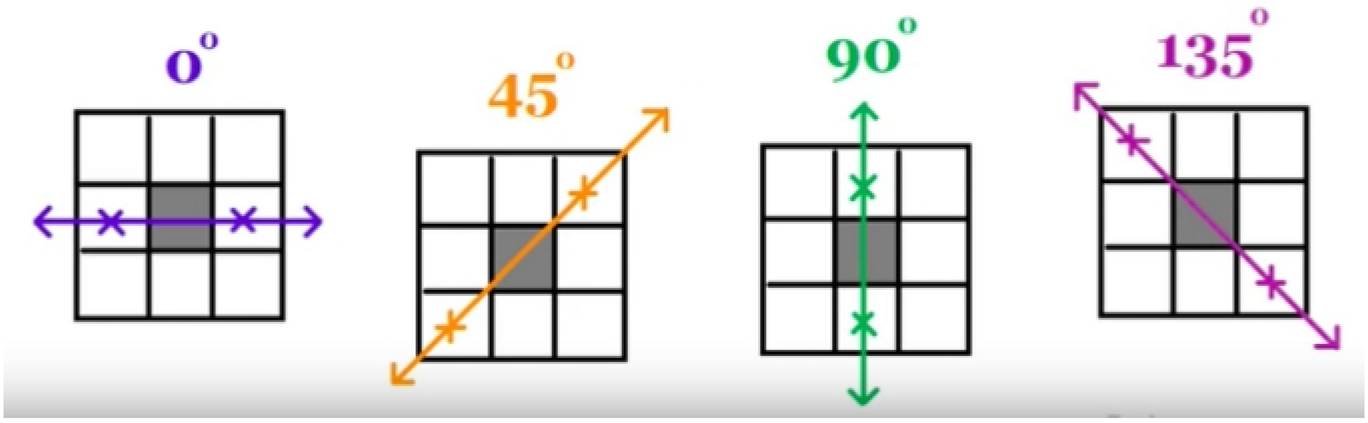
Non-maximum suppression



```
//Non-maximum suppression
for(int i=1; i<G.rows-1; i++)
{
    for(int j=1; j<G.cols-1; j++)
    {
        if(angle[i][j]==0)
        {
            if(gradient[i][j]<gradient[i][j-1]){gradient[i][j]=0;}
            else{gradient[i][j-1]=0;}
            if(gradient[i][j]<gradient[i][j+1]){gradient[i][j]=0;}
            else{gradient[i][j+1]=0;}
            if(gradient[i][j+1]<gradient[i][j-1]){gradient[i][j+1]=0;}
            else{gradient[i][j-1]=0;}
        }
        else if(angle[i][j]==45)
        {
            if(gradient[i][j]<gradient[i+1][j-1]){gradient[i][j]=0;}
            else{gradient[i+1][j-1]=0;}
            if(gradient[i][j]<gradient[i-1][j+1]){gradient[i][j]=0;}
            else{gradient[i-1][j+1]=0;}
            if(gradient[i+1][j-1]<gradient[i-1][j+1]){gradient[i+1][j-1]=0;}
            else{gradient[i-1][j+1]=0;}
        }
```


```
        else if(angle[i][j]==90)
        {
            if(gradient[i][j]<gradient[i-1][j]){gradient[i][j]=0;}
            else{gradient[i-1][j]=0;}
            if(gradient[i][j]<gradient[i+1][j]){gradient[i][j]=0;}
            else{gradient[i+1][j]=0;}
            if(gradient[i+1][j]<gradient[i-1][j]){gradient[i+1][j]=0;}
            else{gradient[i-1][j]=0;}
        }
        else if(angle[i][j]==135)
        {
            if(gradient[i][j]<gradient[i-1][j-1]){gradient[i][j]=0;}
            else{gradient[i-1][j-1]=0;}
            if(gradient[i][j]<gradient[i+1][j+1]){gradient[i][j]=0;}
            else{gradient[i+1][j+1]=0;}
            if(gradient[i+1][j+1]<gradient[i-1][j-1]){gradient[i+1][j+1]=0;}
            else{gradient[i-1][j-1]=0;}
        }
        non_G.at<uchar>(i, j) = gradient[i][j];
        if(gradient[i][j]>255){non_G.at<uchar>(i, j)=255;}
    }
}
```

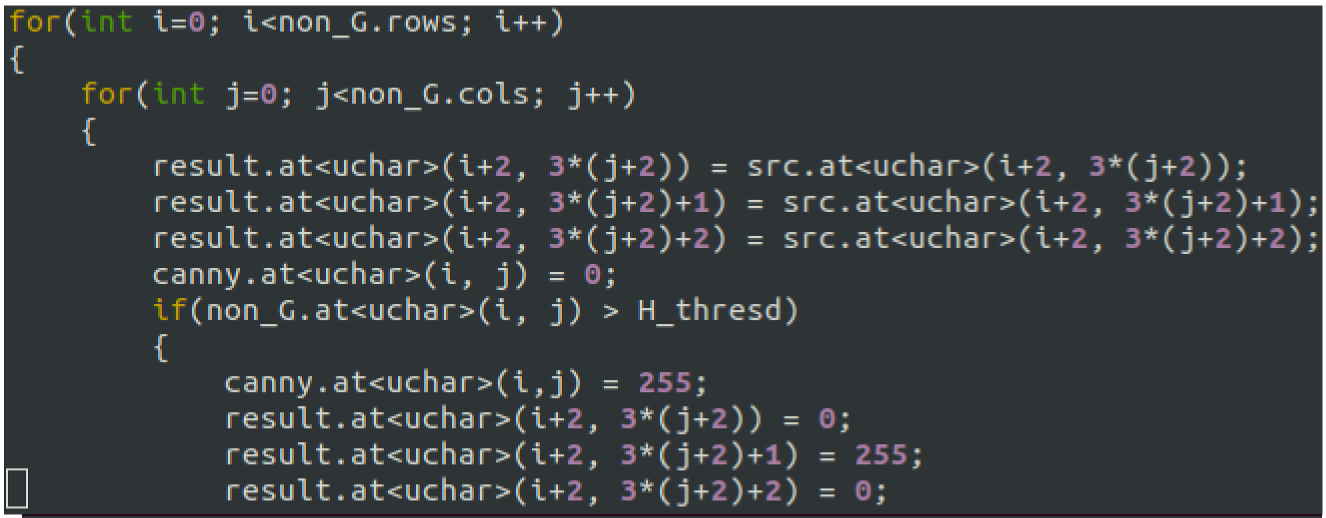
Find the direction of maximum gradient
In order to simplify the calculation, the direction of the gradient is roughly divided into four types, 0 degrees, 45 degrees, 90 degrees, and 135 degrees.

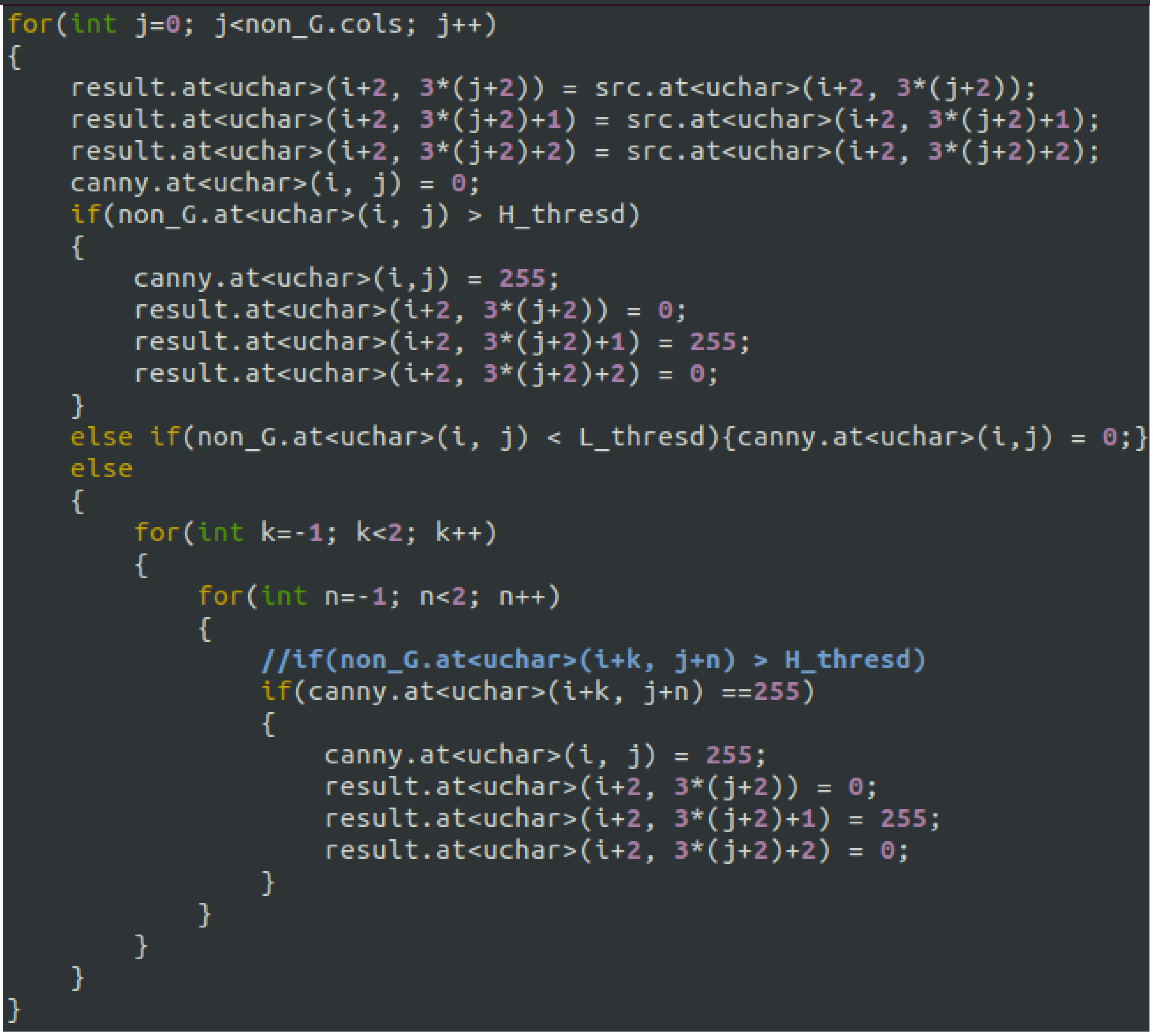


It has been determined that the gradient direction is 45 degrees, and the point in the matrix with the highest intensity in the 45-degree direction is gradually searched for, and the remaining points in the 45-degree direction are all returned to 0

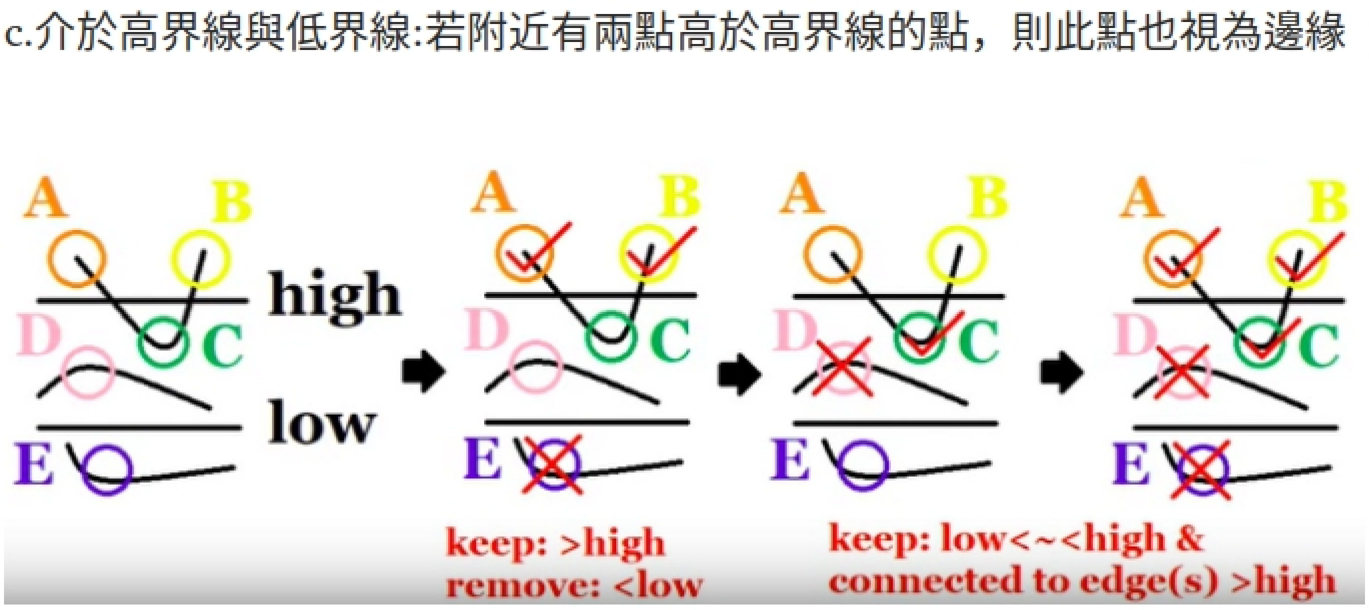
$$\begin{bmatrix} 0.5 & 0.9 & 1 \\ 0 & 0.3 & 0.7 \\ 0.9 & 0 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 0.5 & 0.9 & 1 \\ 0 & 0 & 0.7 \\ 0 & 0 & 0 \end{bmatrix}$$

Connect Weak Edge & draw on source image


```
for(int i=0; i<non_G.rows; i++)
{
    for(int j=0; j<non_G.cols; j++)
    {
        result.at<uchar>(i+2, 3*(j+2)) = src.at<uchar>(i+2, 3*(j+2));
        result.at<uchar>(i+2, 3*(j+2)+1) = src.at<uchar>(i+2, 3*(j+2)+1);
        result.at<uchar>(i+2, 3*(j+2)+2) = src.at<uchar>(i+2, 3*(j+2)+2);
        canny.at<uchar>(i, j) = 0;
        if(non_G.at<uchar>(i, j) > H_thresd)
        {
            canny.at<uchar>(i,j) = 255;
            result.at<uchar>(i+2, 3*(j+2)) = 0;
            result.at<uchar>(i+2, 3*(j+2)+1) = 255;
            result.at<uchar>(i+2, 3*(j+2)+2) = 0;
```

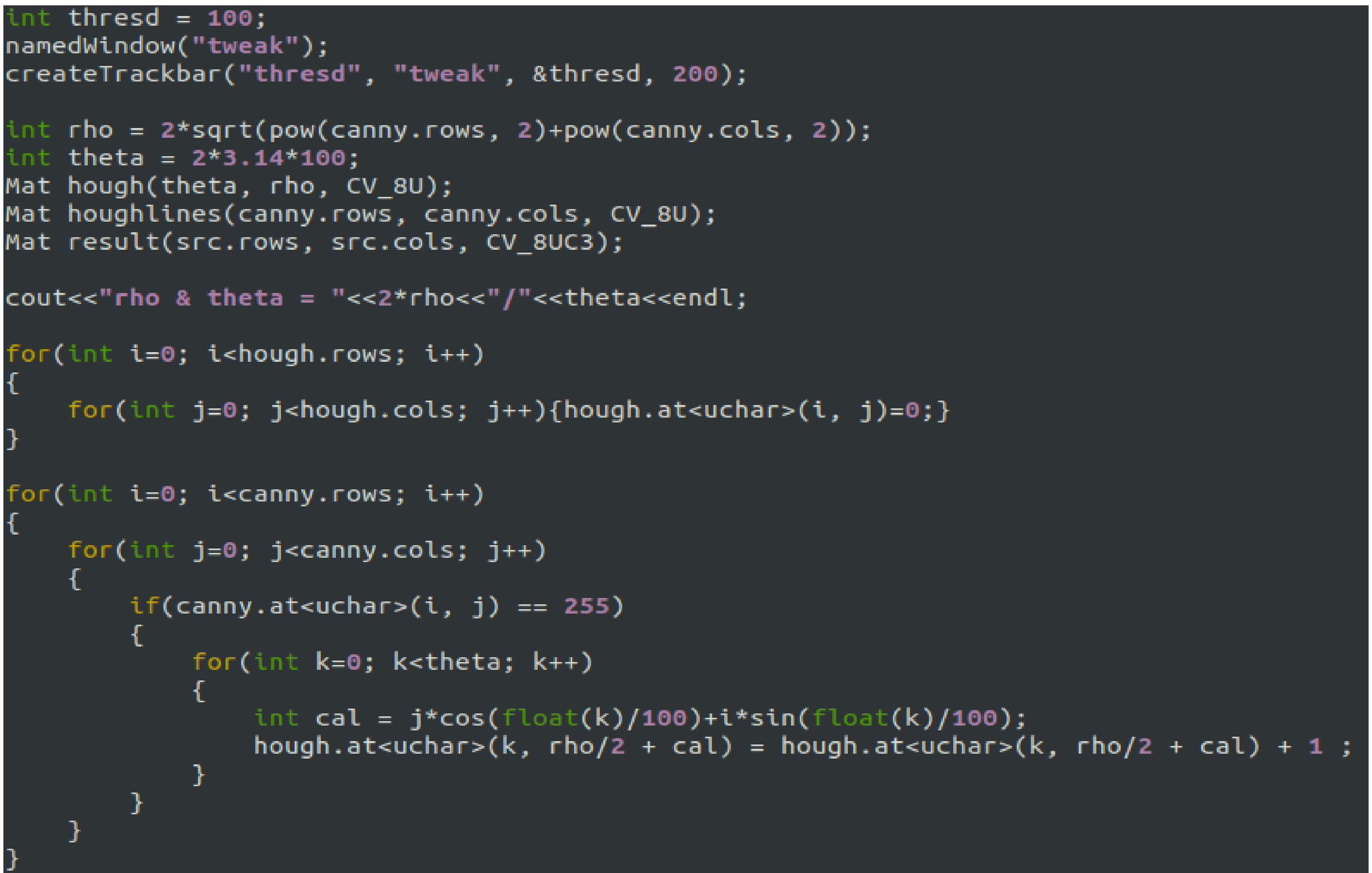

```
for(int j=0; j<non_G.cols; j++)
{
    result.at<uchar>(i+2, 3*(j+2)) = src.at<uchar>(i+2, 3*(j+2));
    result.at<uchar>(i+2, 3*(j+2)+1) = src.at<uchar>(i+2, 3*(j+2)+1);
    result.at<uchar>(i+2, 3*(j+2)+2) = src.at<uchar>(i+2, 3*(j+2)+2);
    canny.at<uchar>(i, j) = 0;
    if(non_G.at<uchar>(i, j) > H_thresd)
    {
        canny.at<uchar>(i,j) = 255;
        result.at<uchar>(i+2, 3*(j+2)) = 0;
        result.at<uchar>(i+2, 3*(j+2)+1) = 255;
        result.at<uchar>(i+2, 3*(j+2)+2) = 0;
    }
    else if(non_G.at<uchar>(i, j) < L_thresd){canny.at<uchar>(i,j) = 0;}
    else
    {
        for(int k=-1; k<2; k++)
        {
            for(int n=-1; n<2; n++)
            {
                //if(non_G.at<uchar>(i+k, j+n) > H_thresd)
                if(canny.at<uchar>(i+k, j+n) ==255)
                {
                    canny.at<uchar>(i, j) = 255;
                    result.at<uchar>(i+2, 3*(j+2)) = 0;
                    result.at<uchar>(i+2, 3*(j+2)+1) = 255;
                    result.at<uchar>(i+2, 3*(j+2)+2) = 0;
                }
            }
        }
    }
}
```

Connect the lines through Connect Weak Edge, set the high and low boundaries, and follow this mechanism to find the edges.
a. Above the high boundary: it must be the edge
b. Below the low line: it must not be the edge
c. Between the high boundary and the low boundary: if there are two nearby points higher than the high boundary, this point is also regarded as an edge


c.介於高界線與低界線:若附近有兩點高於高界線的點，則此點也視為邊緣

Hough Transform:

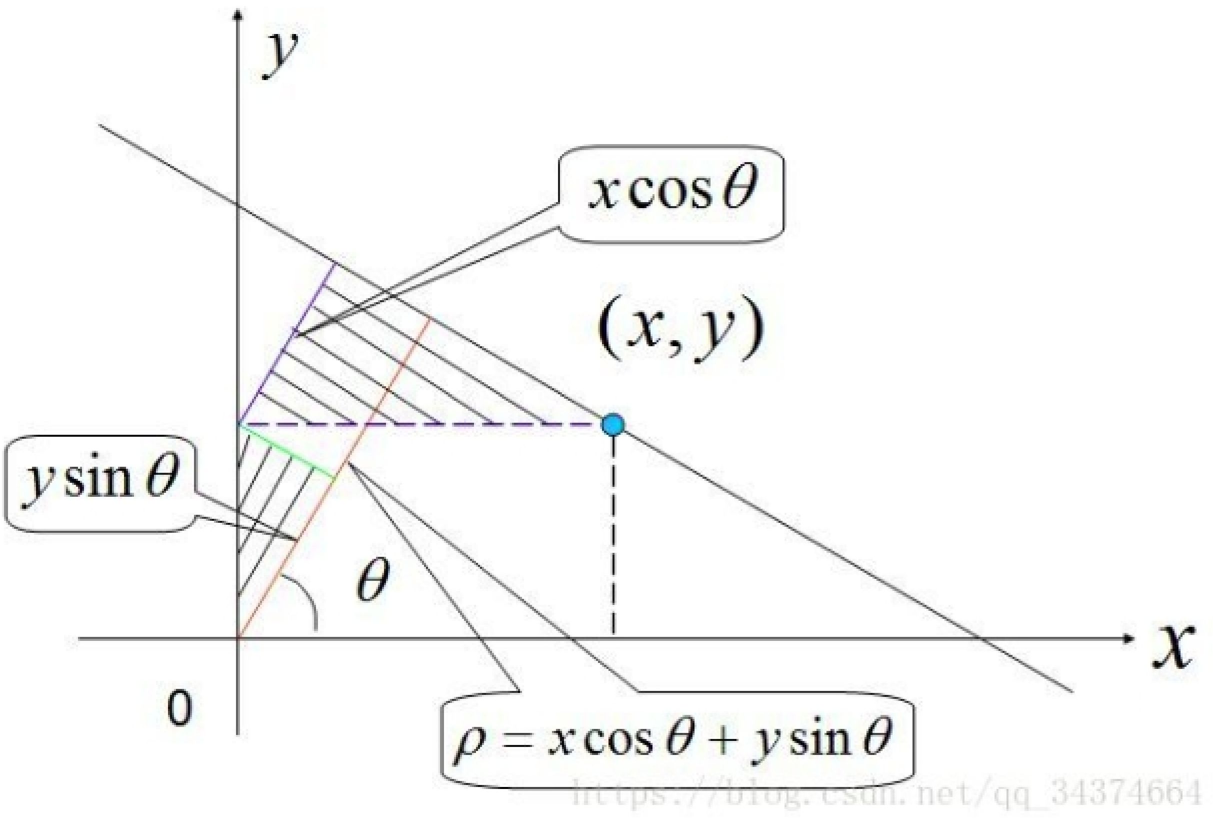
after canny, we can do Houghlines


```
int thresd = 100;
namedWindow("tweak");
createTrackbar("thresd", "tweak", &thresd, 200);

int rho = 2*sqrt(pow(canny.rows, 2)+pow(canny.cols, 2));
int theta = 2*3.14*100;
Mat hough(theta, rho, CV_8U);
Mat houghlines(canny.rows, canny.cols, CV_8U);
Mat result(src.rows, src.cols, CV_8UC3);

cout<<"rho & theta = "<<2*rho<<"/"<<theta<<endl;

for(int i=0; i<hough.rows; i++)
{
    for(int j=0; j<hough.cols; j++){hough.at<uchar>(i, j)=0;}
}

for(int i=0; i<canny.rows; i++)
{
    for(int j=0; j<canny.cols; j++)
    {
        if(canny.at<uchar>(i, j) == 255)
        {
            for(int k=0; k<theta; k++)
            {
                int cal = j*cos(float(k)/100)+i*sin(float(k)/100);
                hough.at<uchar>(k, rho/2 + cal) = hough.at<uchar>(k, rho/2 + cal) + 1 ;
            }
        }
    }
}
```

$$p = x\cos\theta + y\sin\theta$$

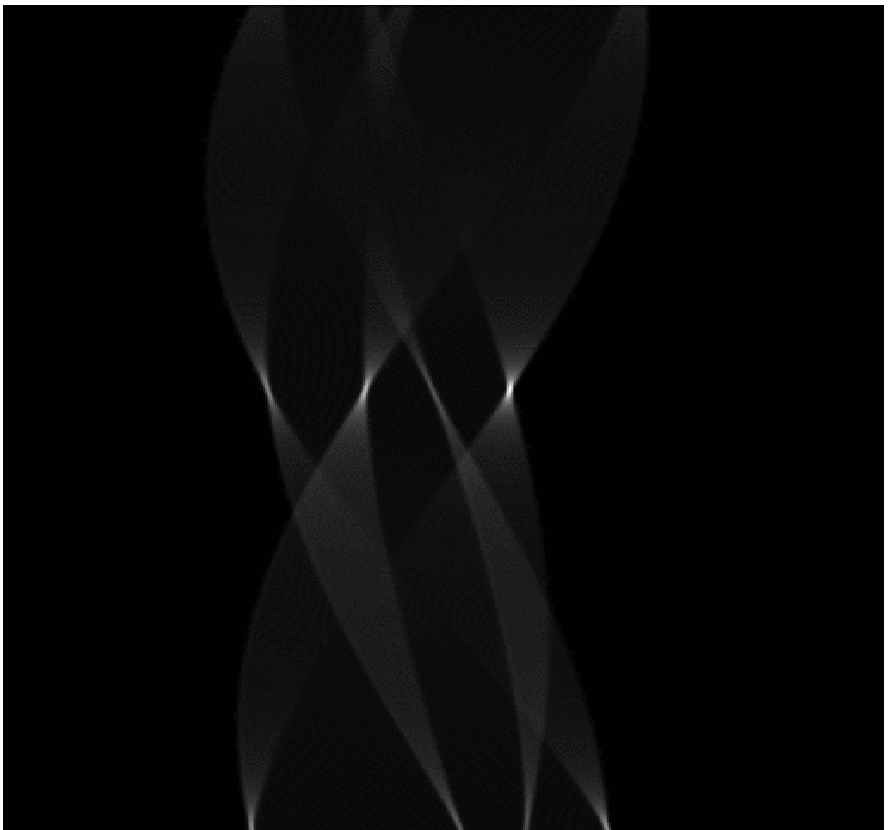
transform formula:

$$p=x\cos\theta+y\sin\theta$$



y
$x\cos\theta$
(x, y)
$y\sin\theta$
θ
x
0
$\rho = x\cos\theta + y\sin\theta$
https://blog.csdn.net/qq_34374664

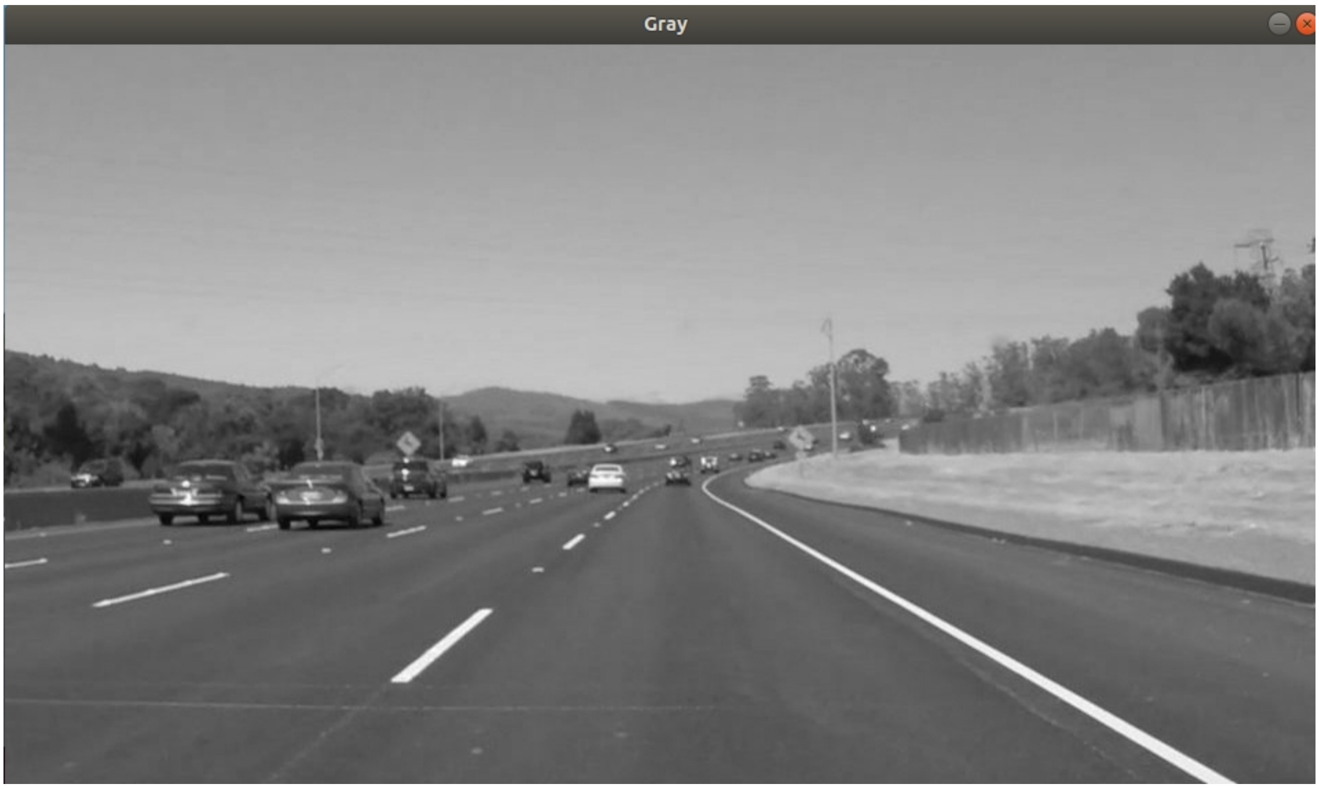
We convert the spatial coordinates of the image to Hough space like this

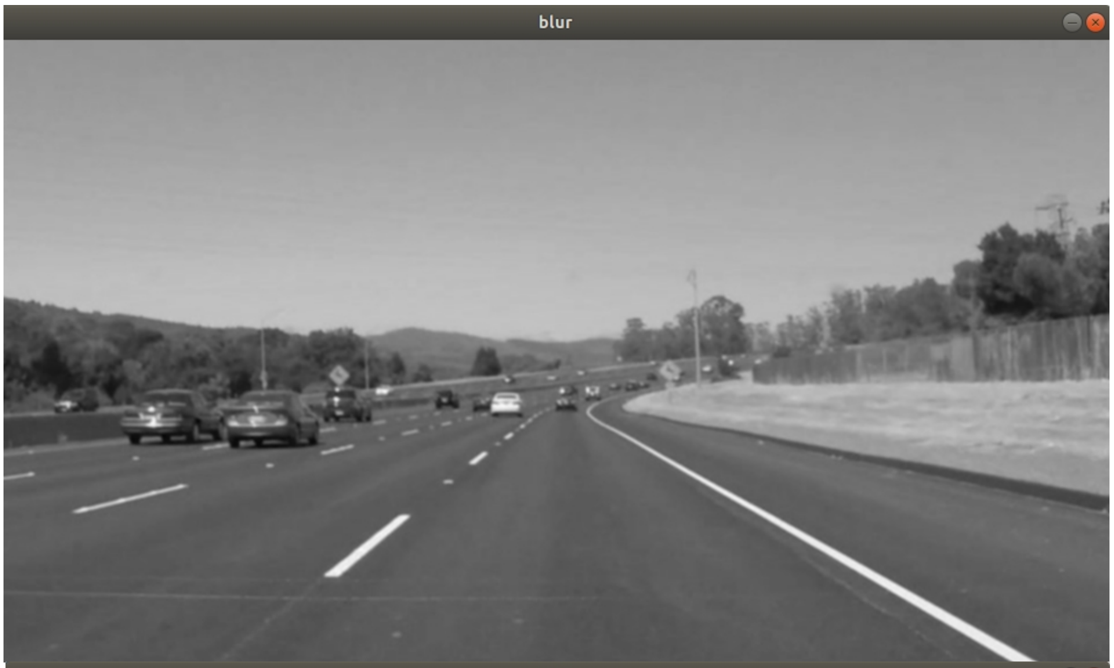


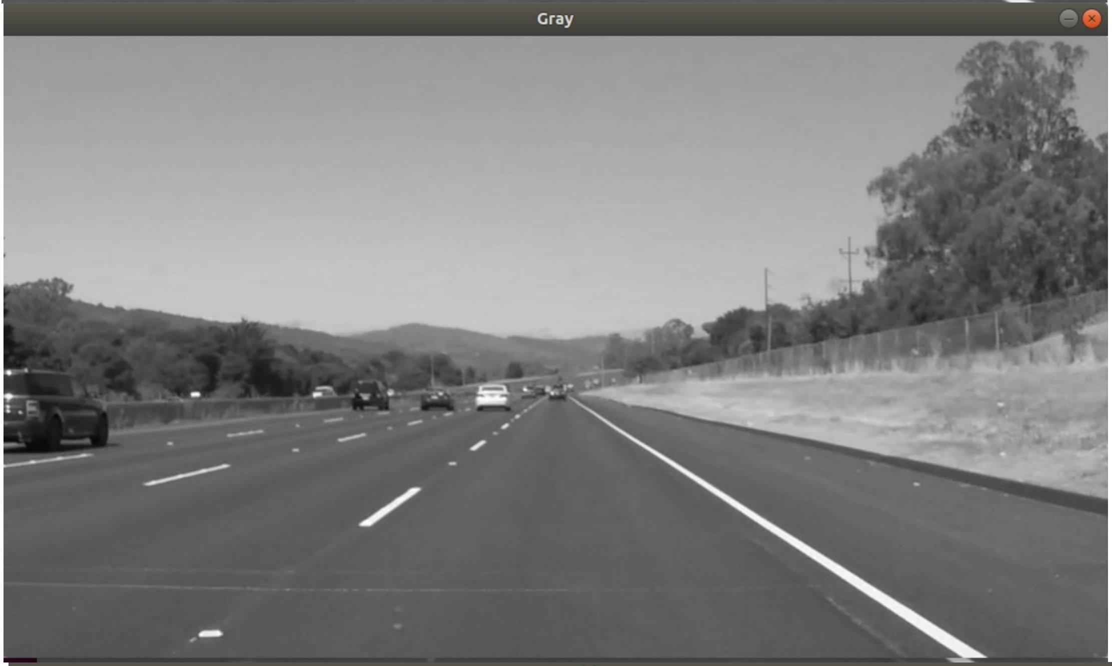
Then we calculate how many Hough space curves will pass through the graph and accumulate
And set the threshold to filter out the cumulative number below the threshold
Finally, the value above the threshold is converted back to the parameter space of the image

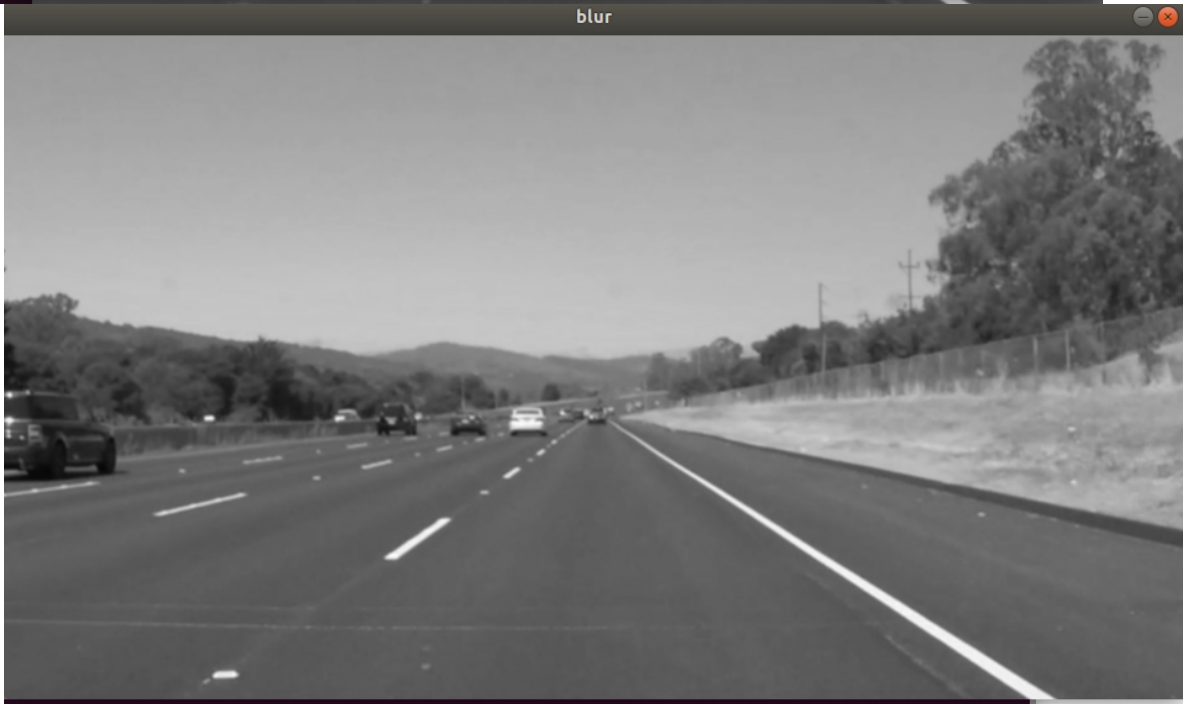
```
for(int i=0; i<hough.rows; i++)
{
    for(int j=0; j<hough.cols; j++){
        if(hough.at<uchar>(i, j) > thresd)
        {
            for(int k=0; k<houghlines.cols; k++)
            {
                int cal = ((j-rho/2)-k*cos(float(i)/100))/sin(float(i)/100);
                if(cal < houghlines.rows && cal>0)
                {
                    houghlines.at<uchar>(cal, k) = 255;
                    result.at<uchar>(cal+2, 3*(k+2)) = 0;
                    result.at<uchar>(cal+2, 3*(k+2)+1) = 255;
                    result.at<uchar>(cal+2, 3*(k+2)+2) = 0;
                }
            }
        }
    }
}
```

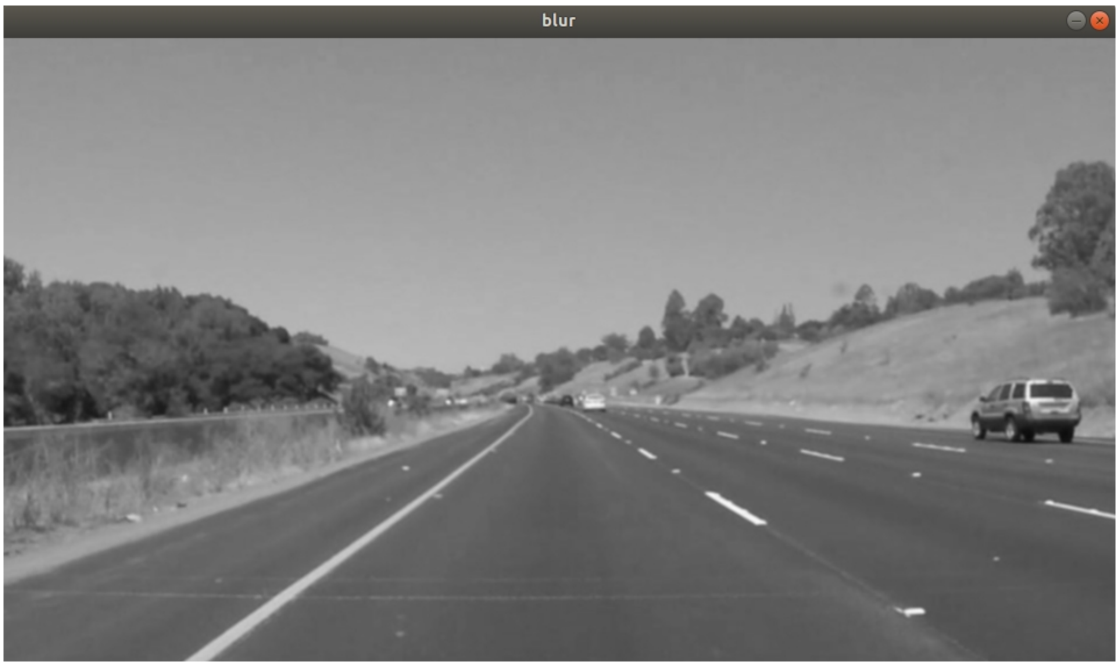
result mat :
Gaussian Blur :

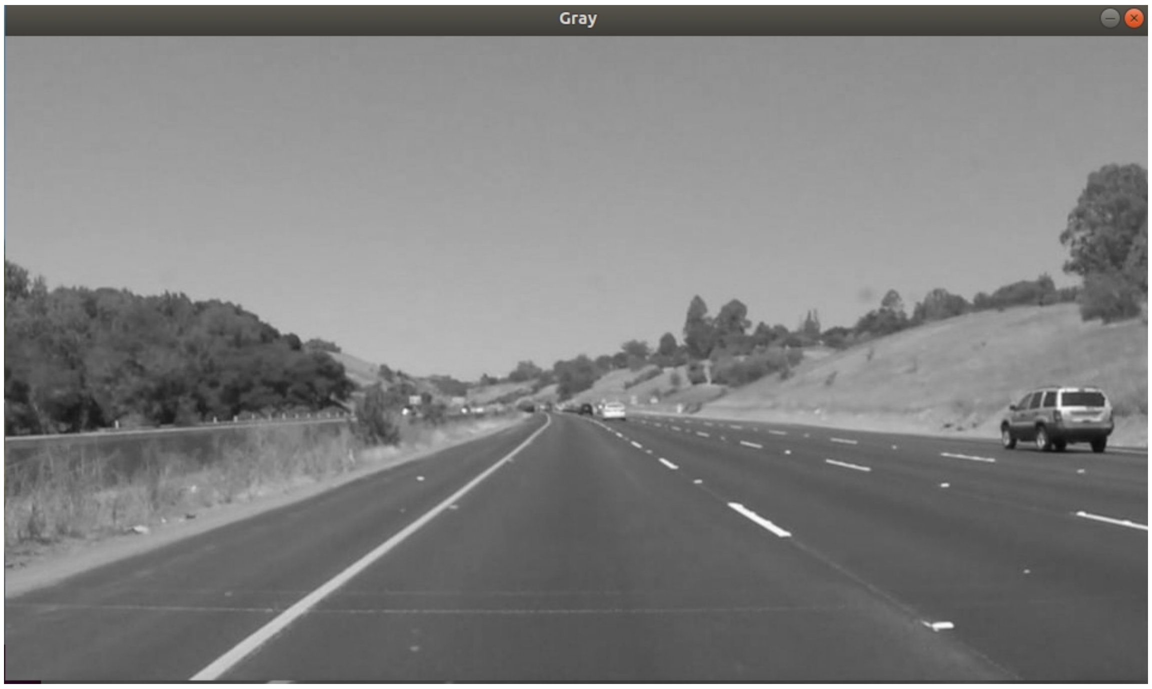


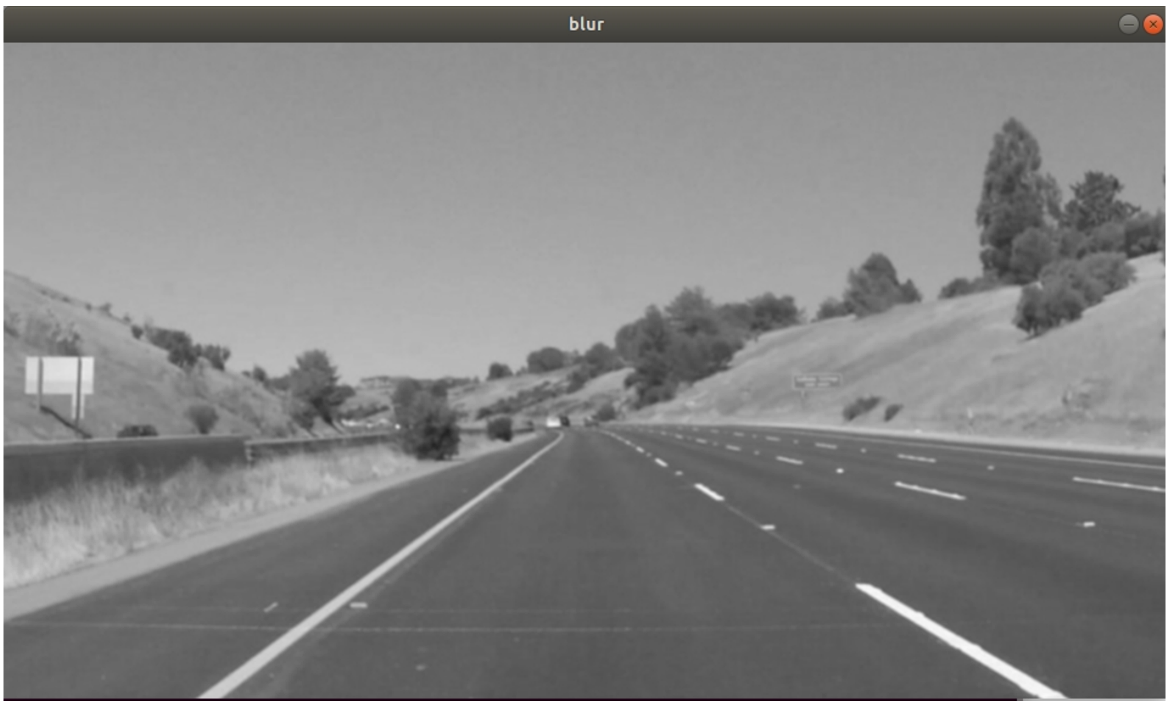

blur

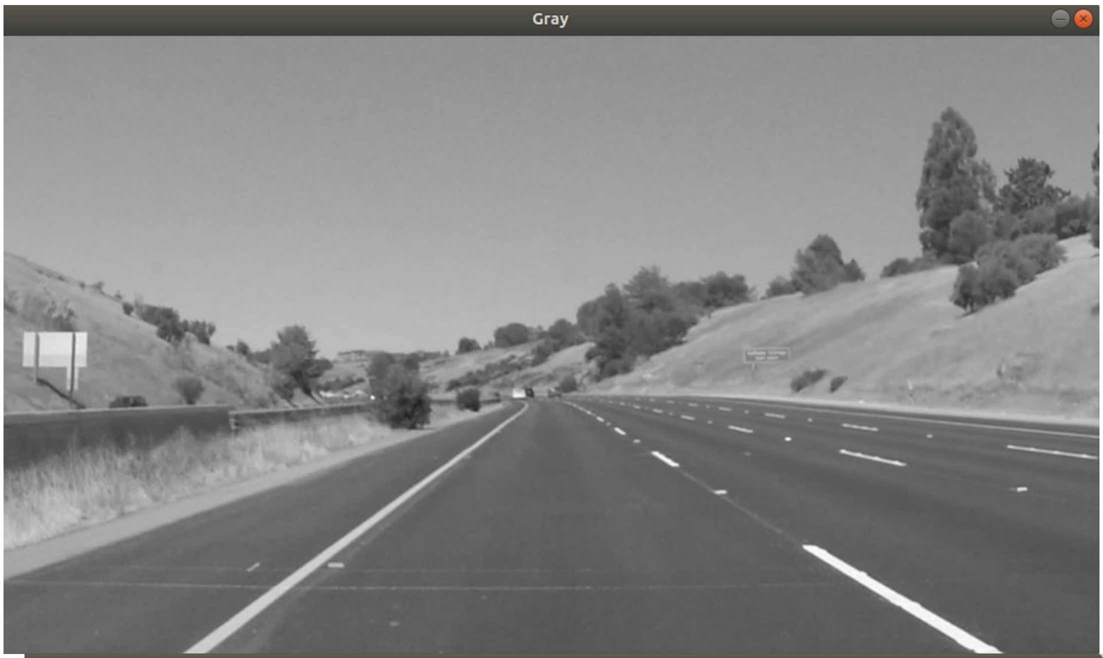
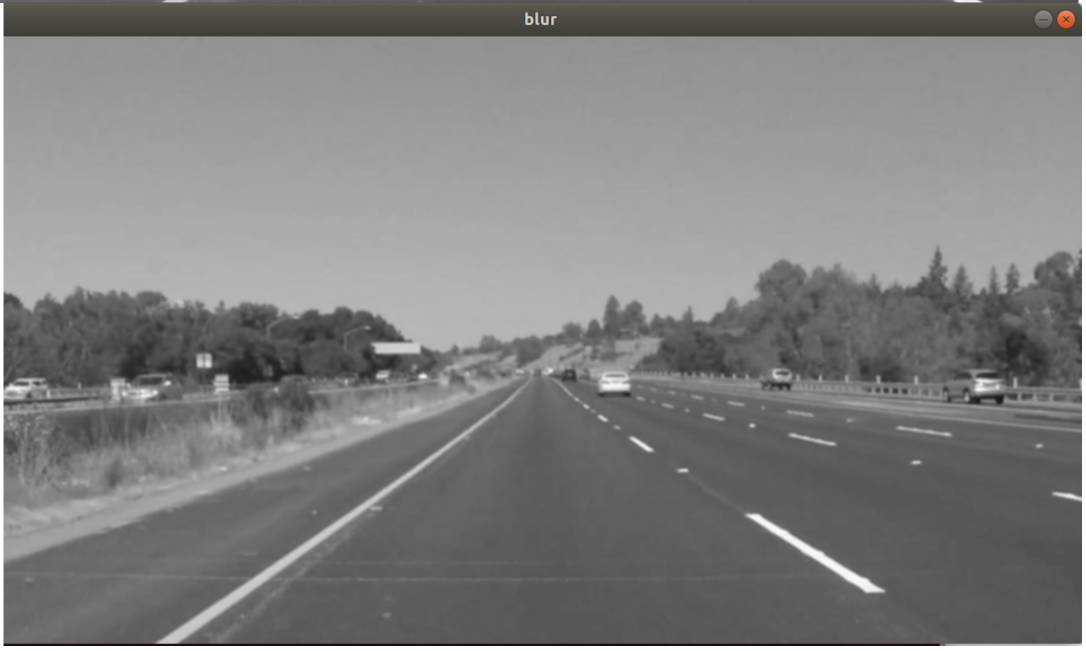
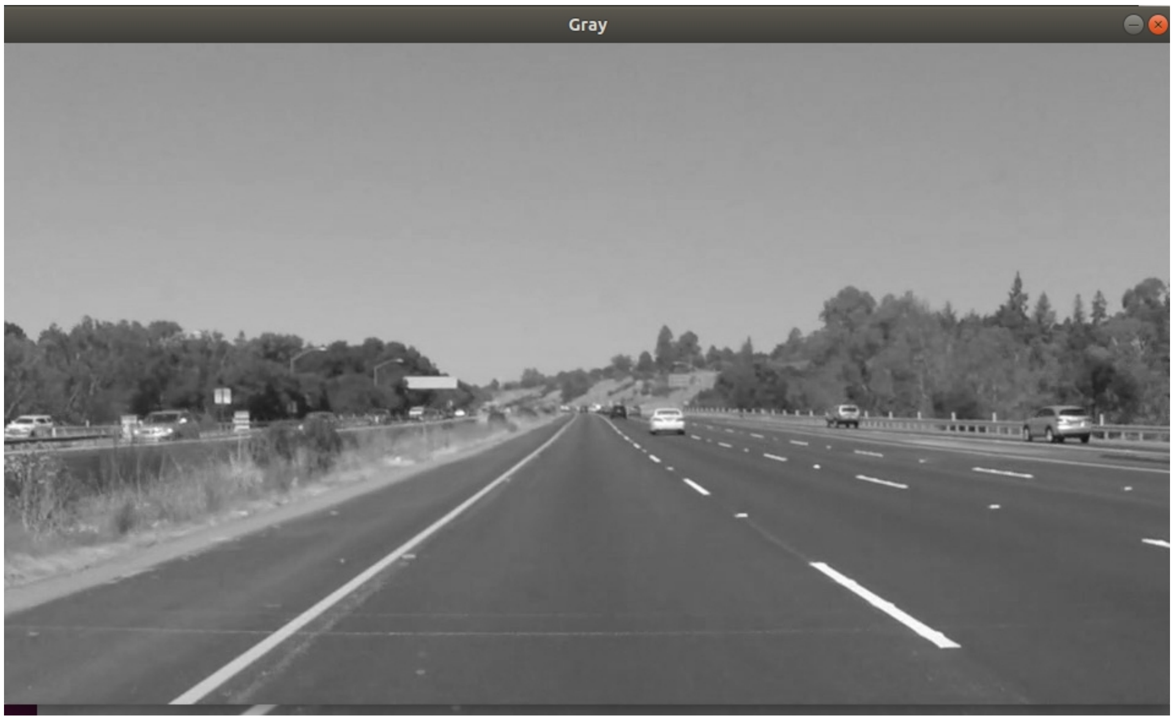

Gray

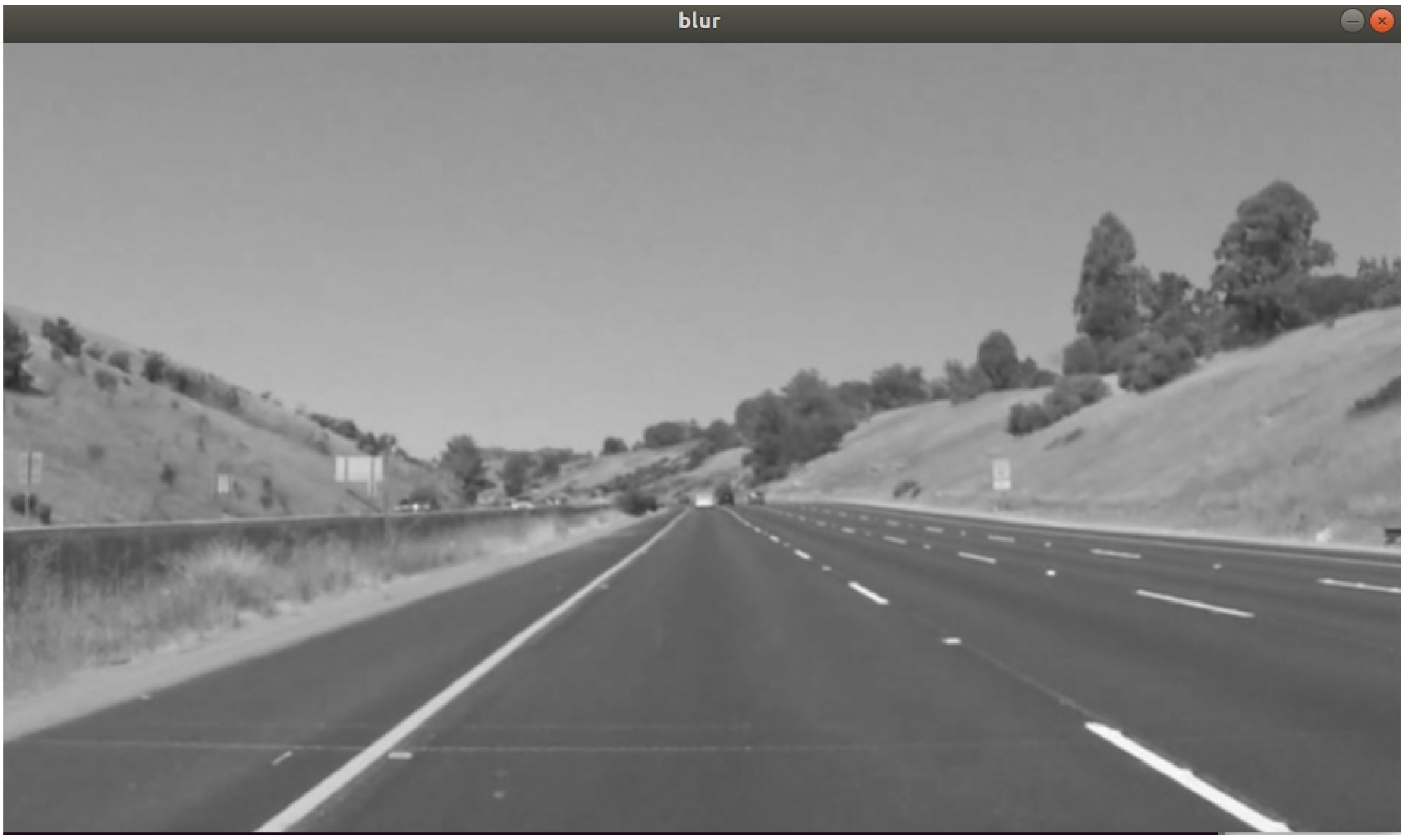

blur

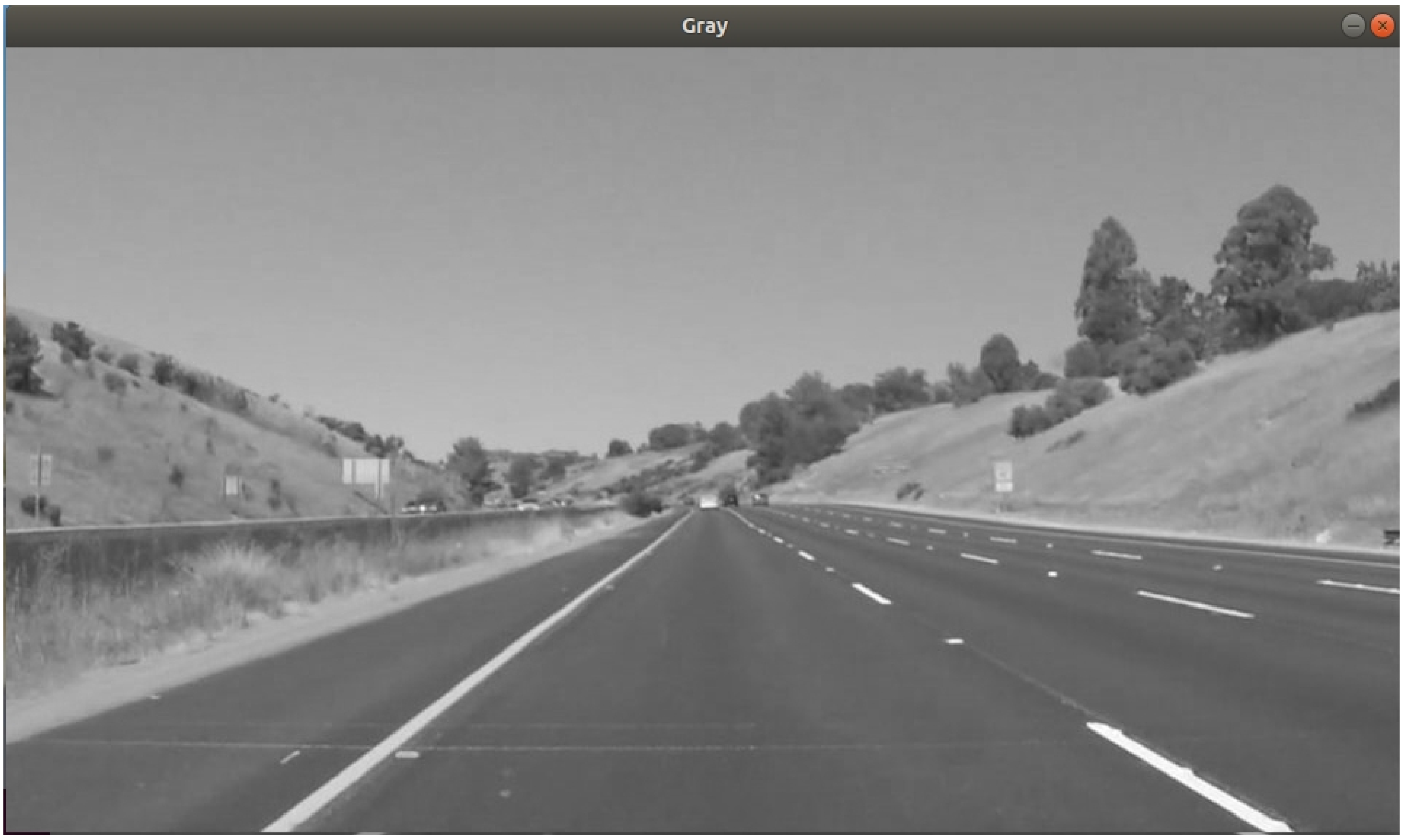

blur


Gray

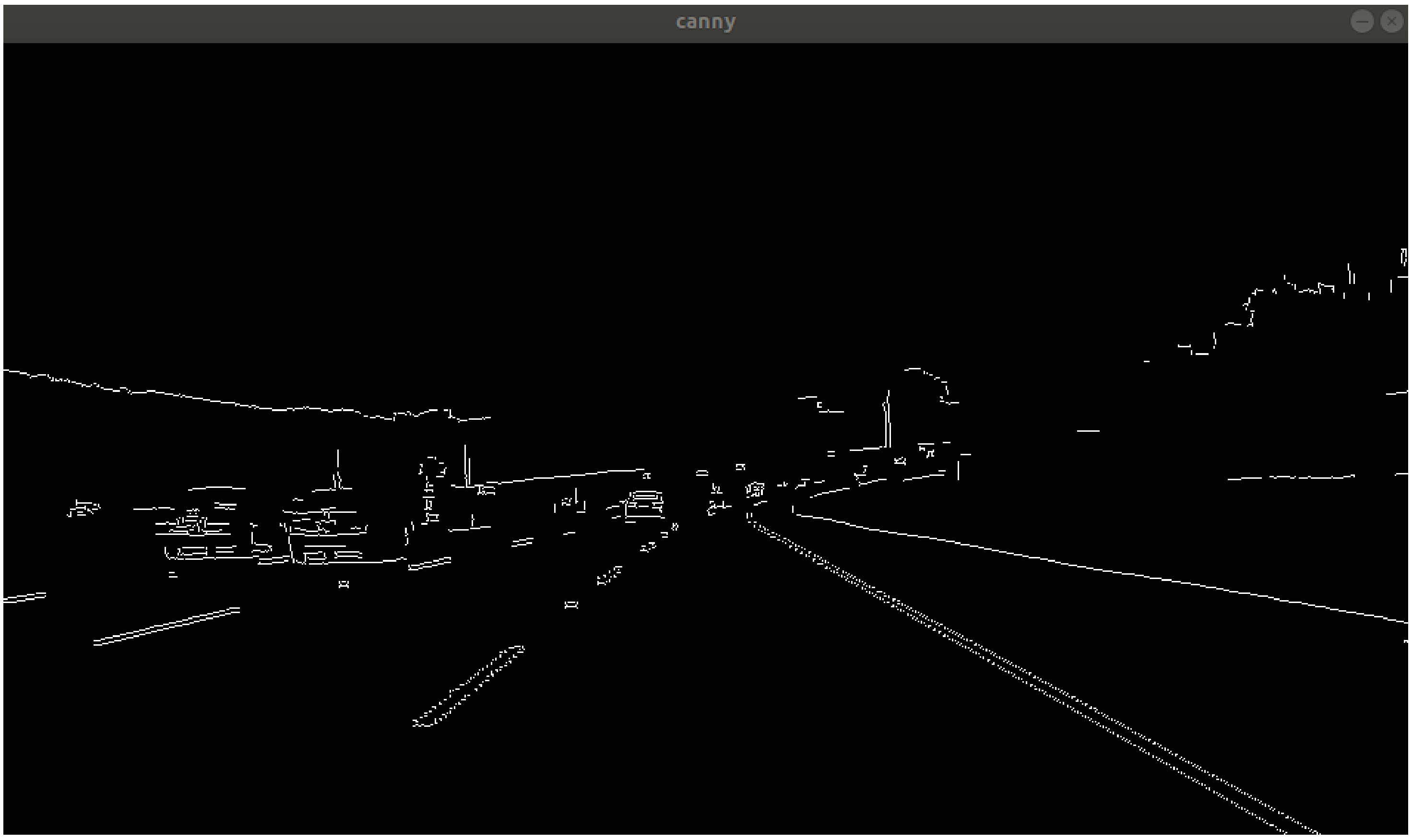

blur

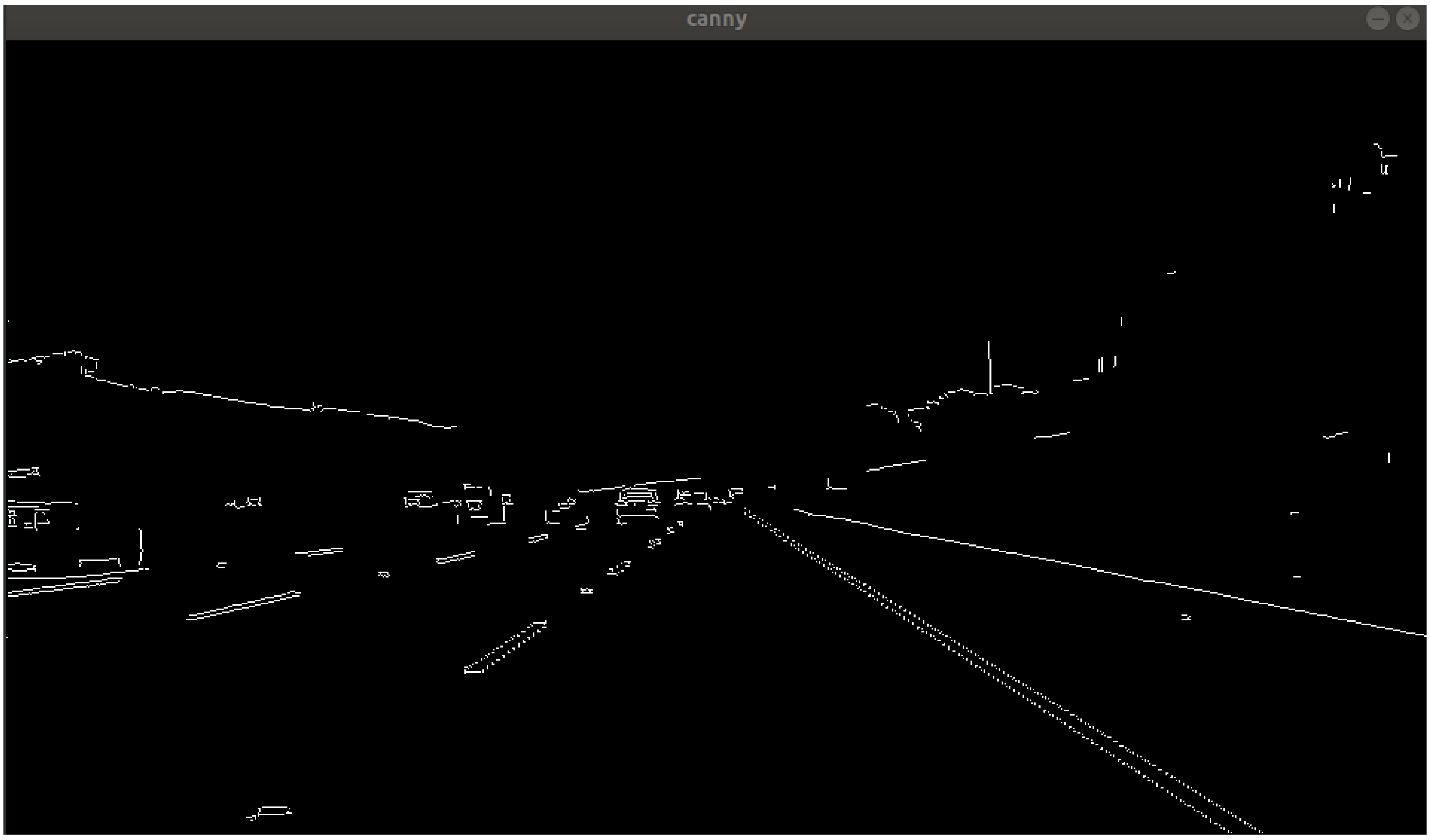

Gray

blur

Gray

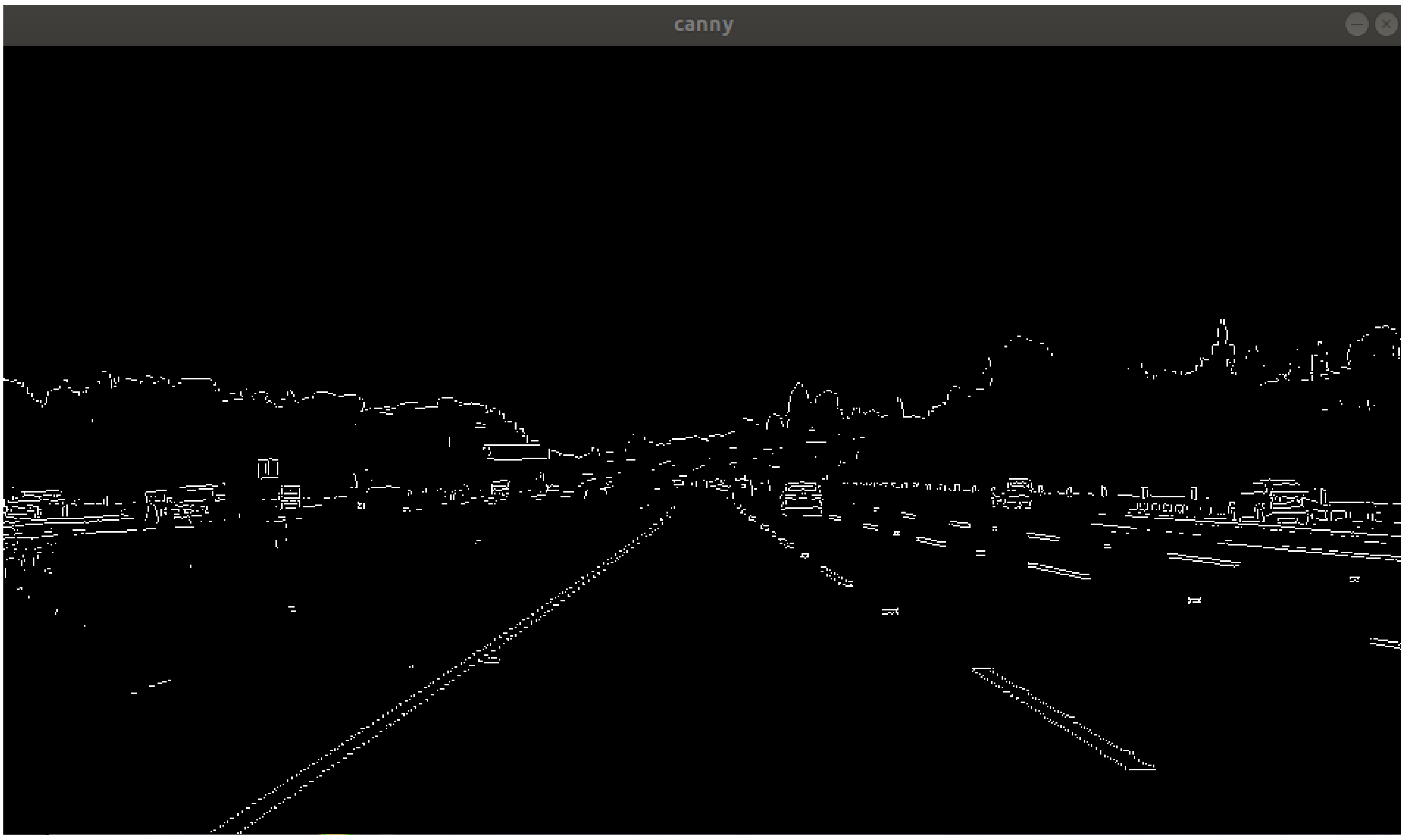

blur

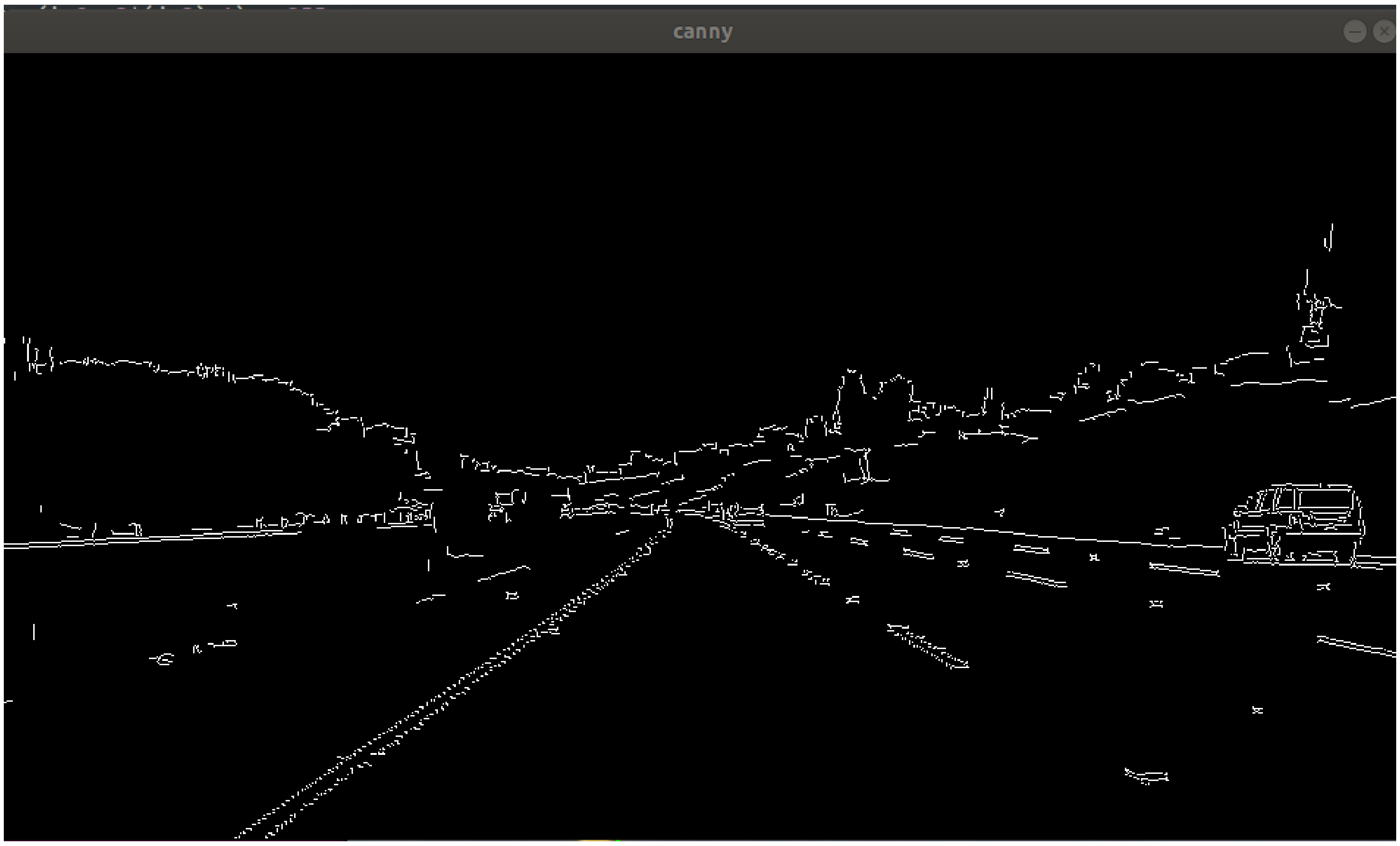

Gray

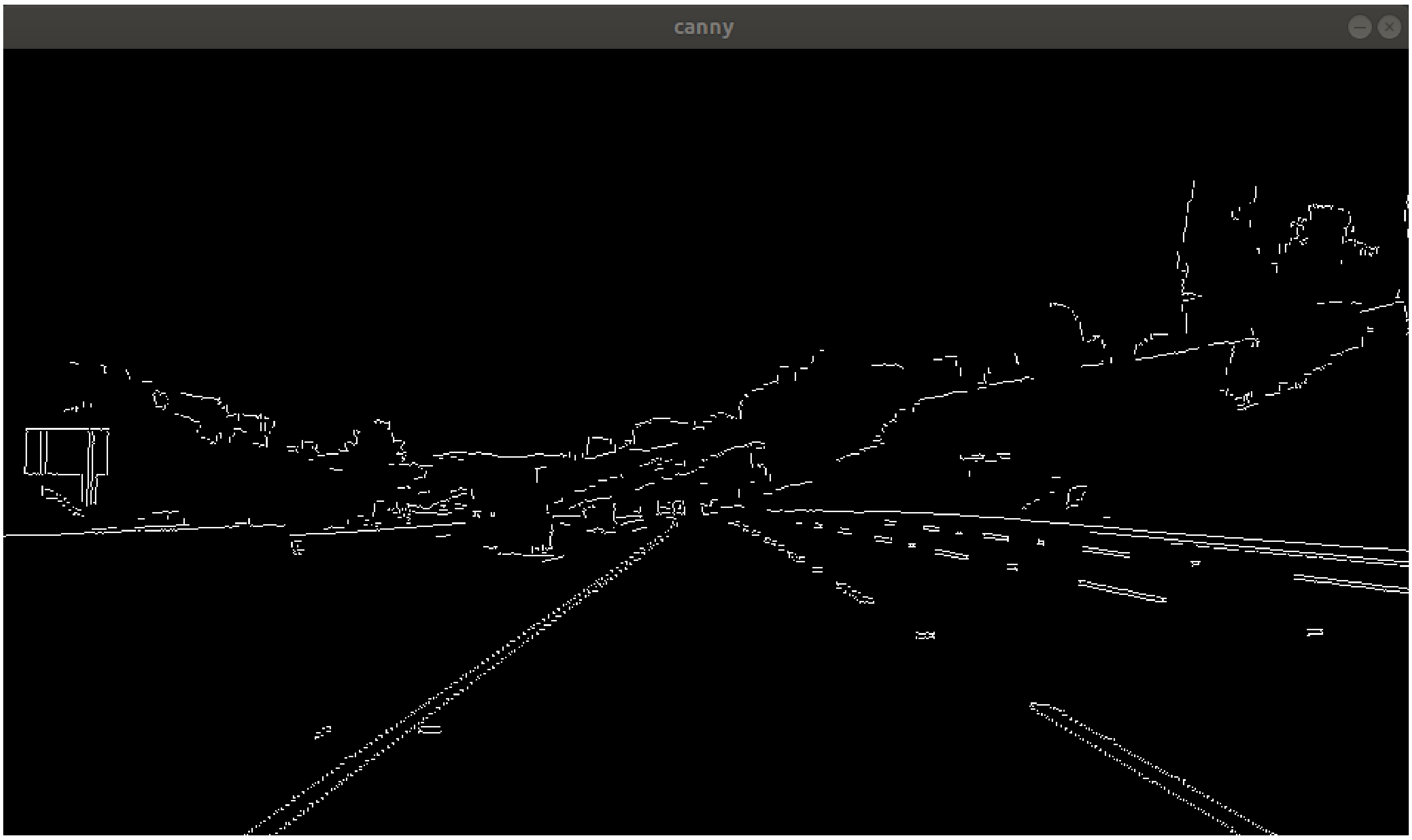
Canny Edge detection

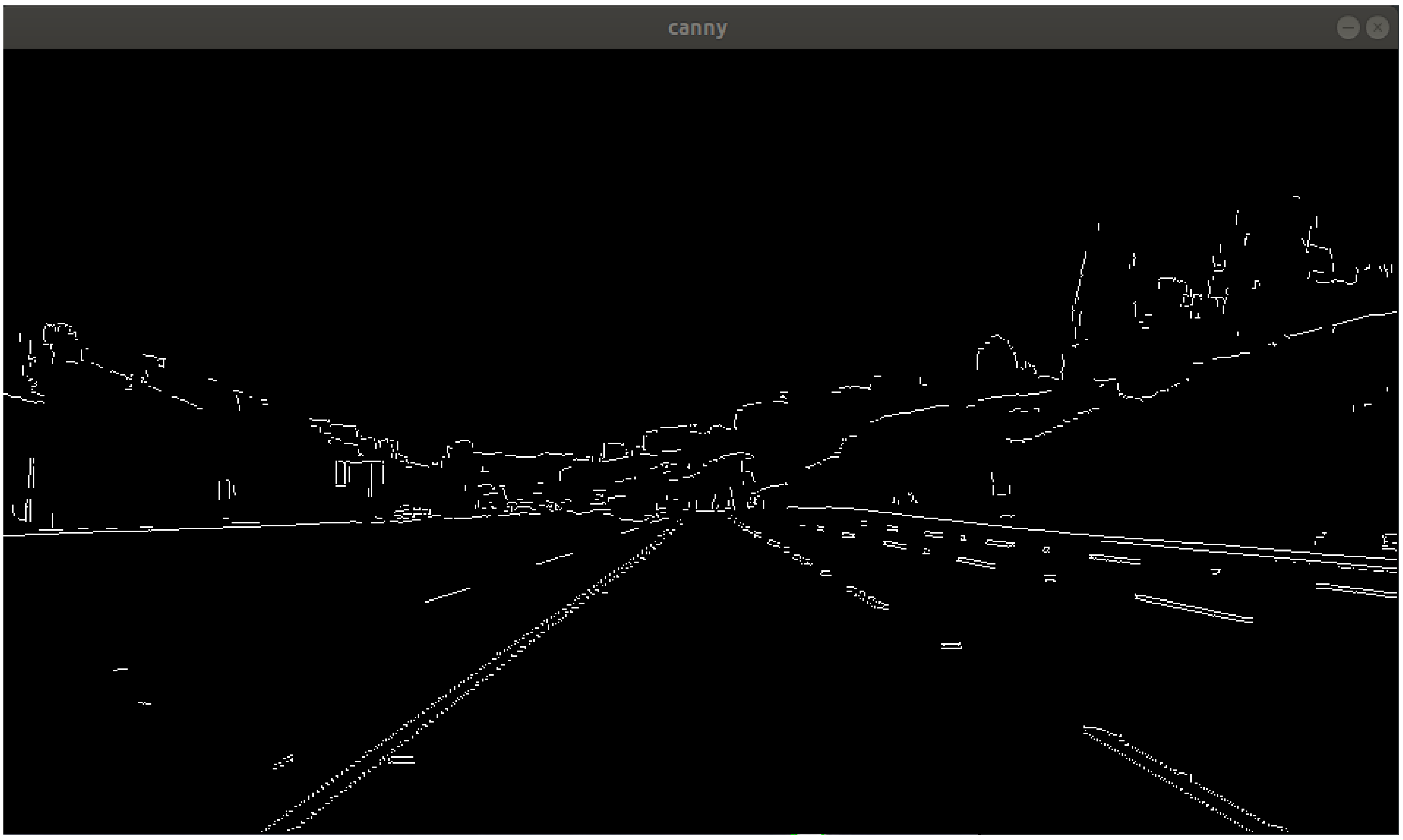




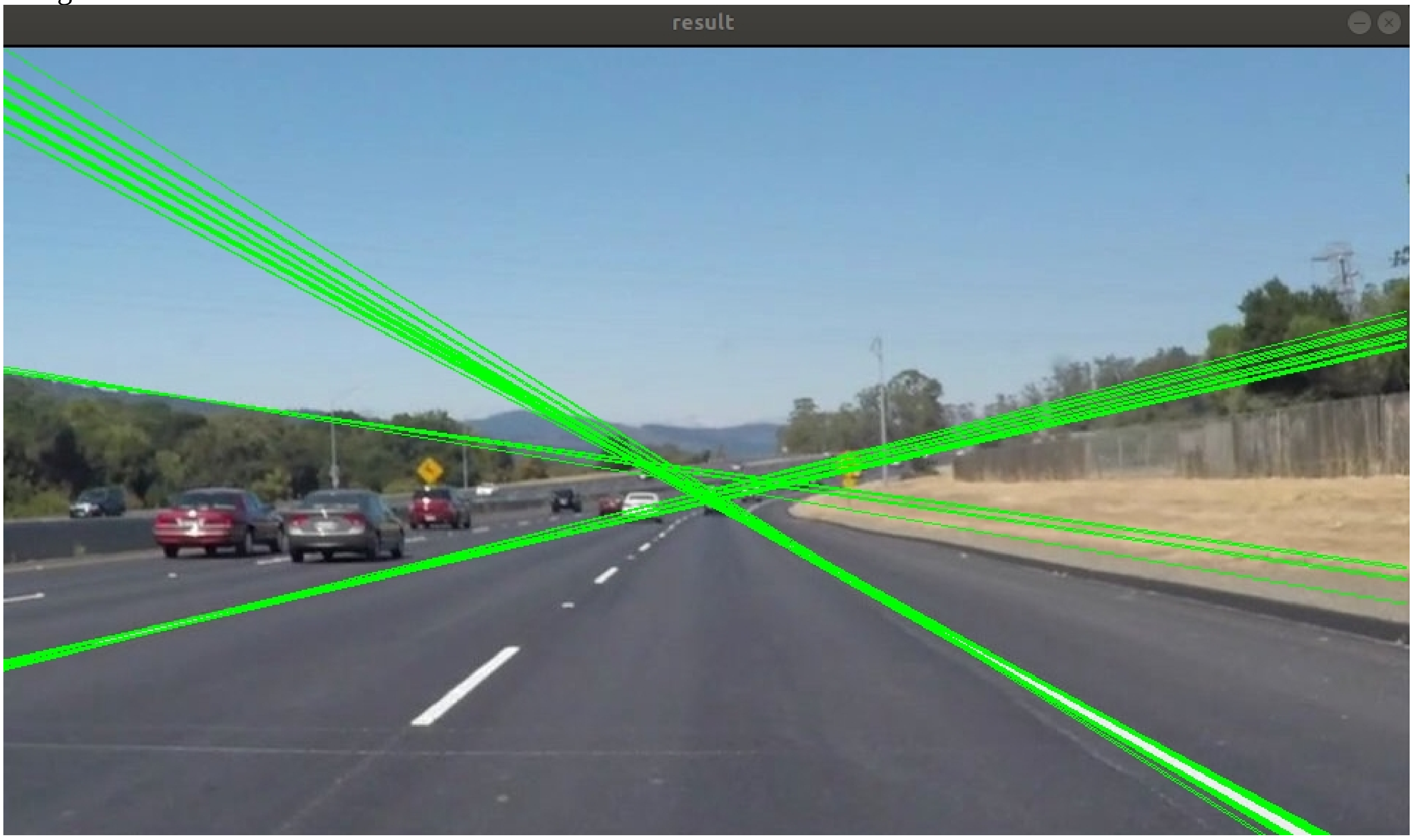

canny

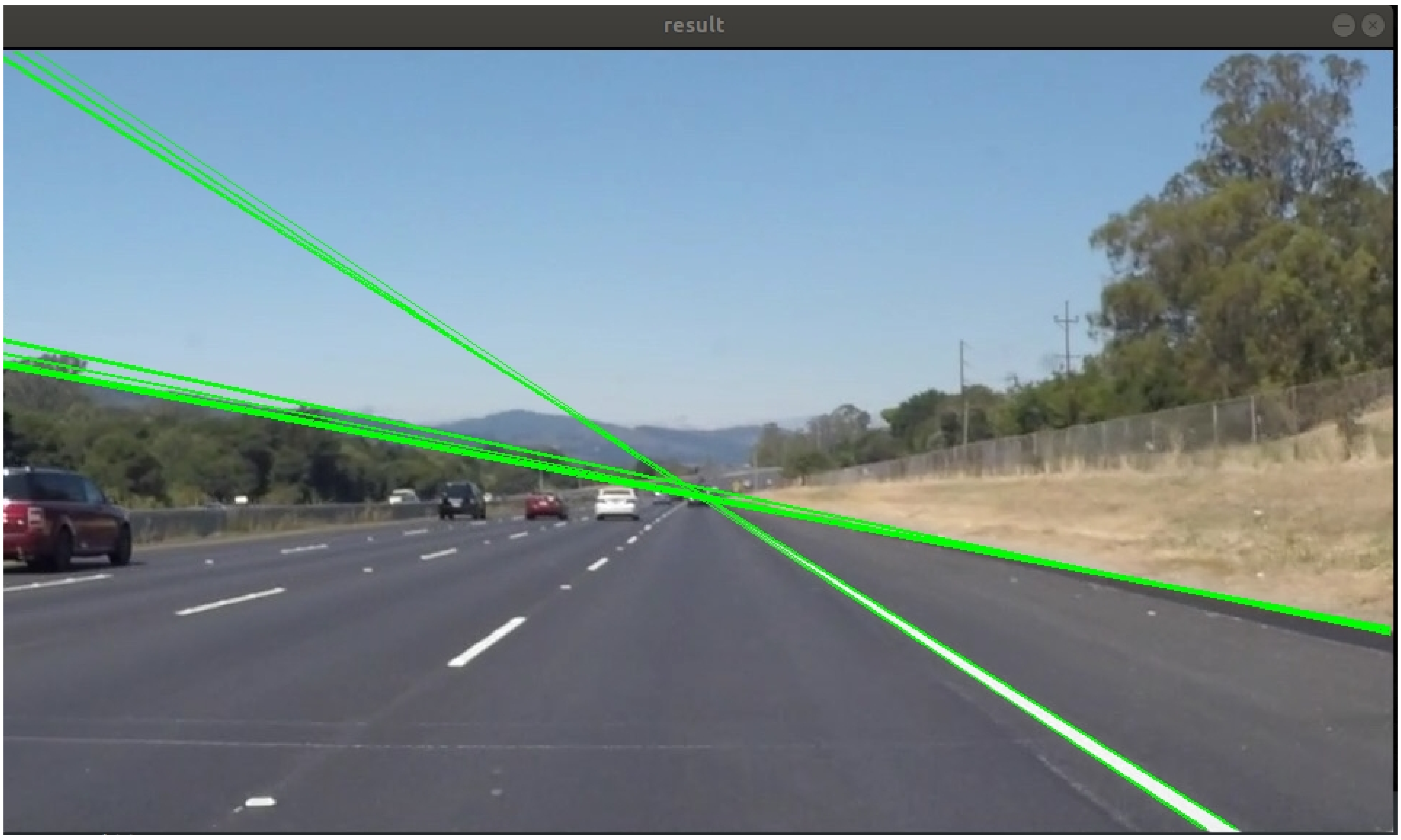

canny

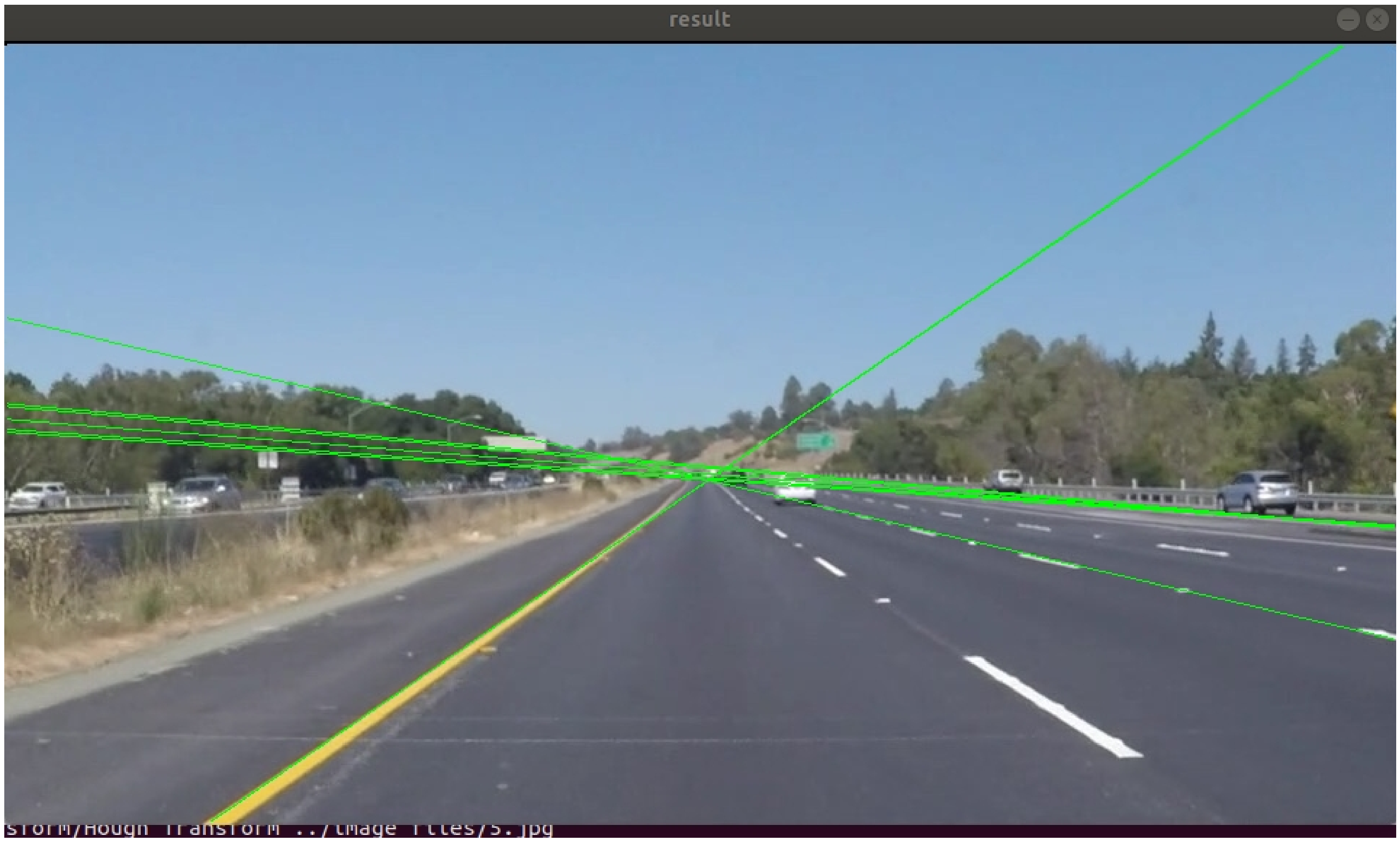

canny


canny

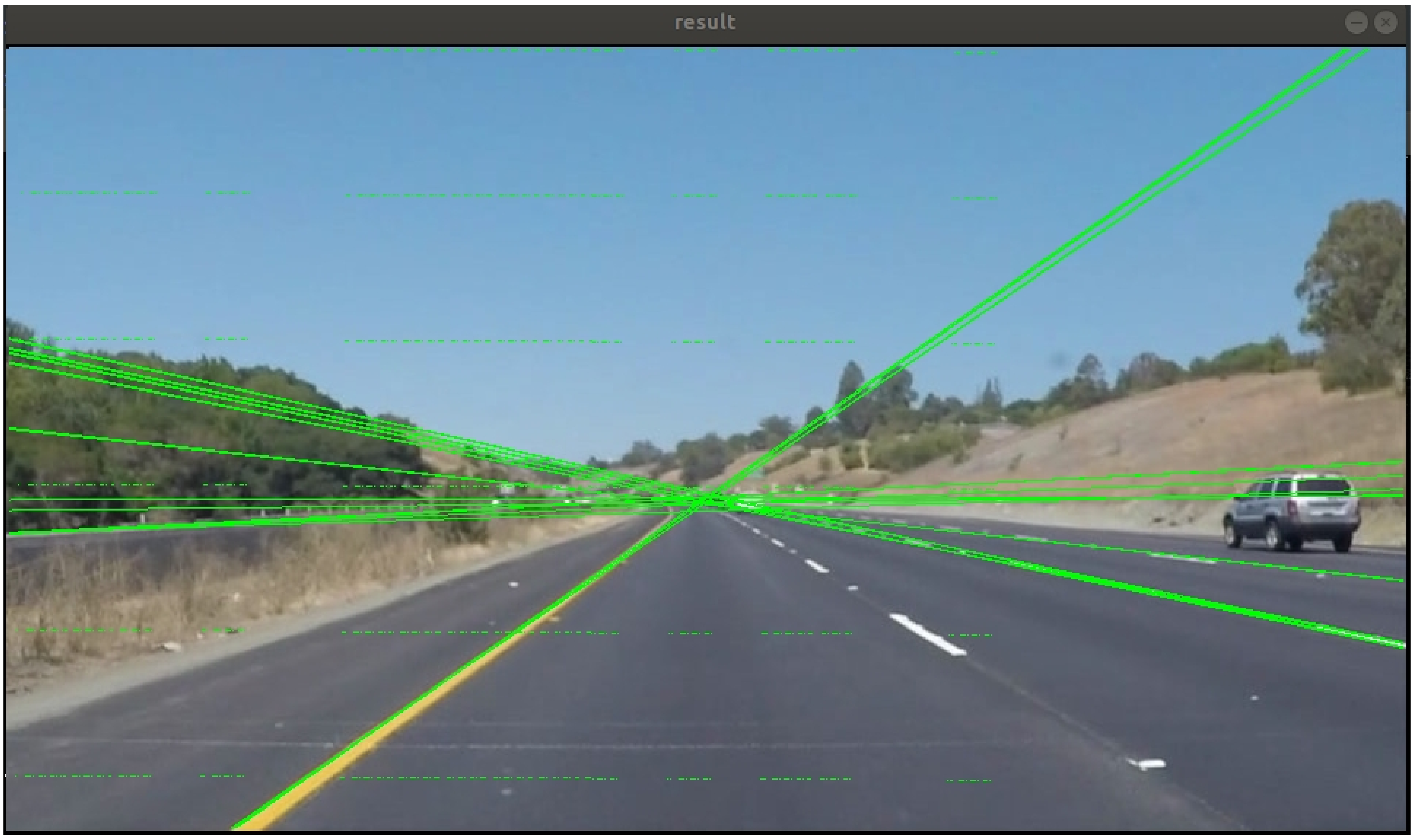
Hough Transform

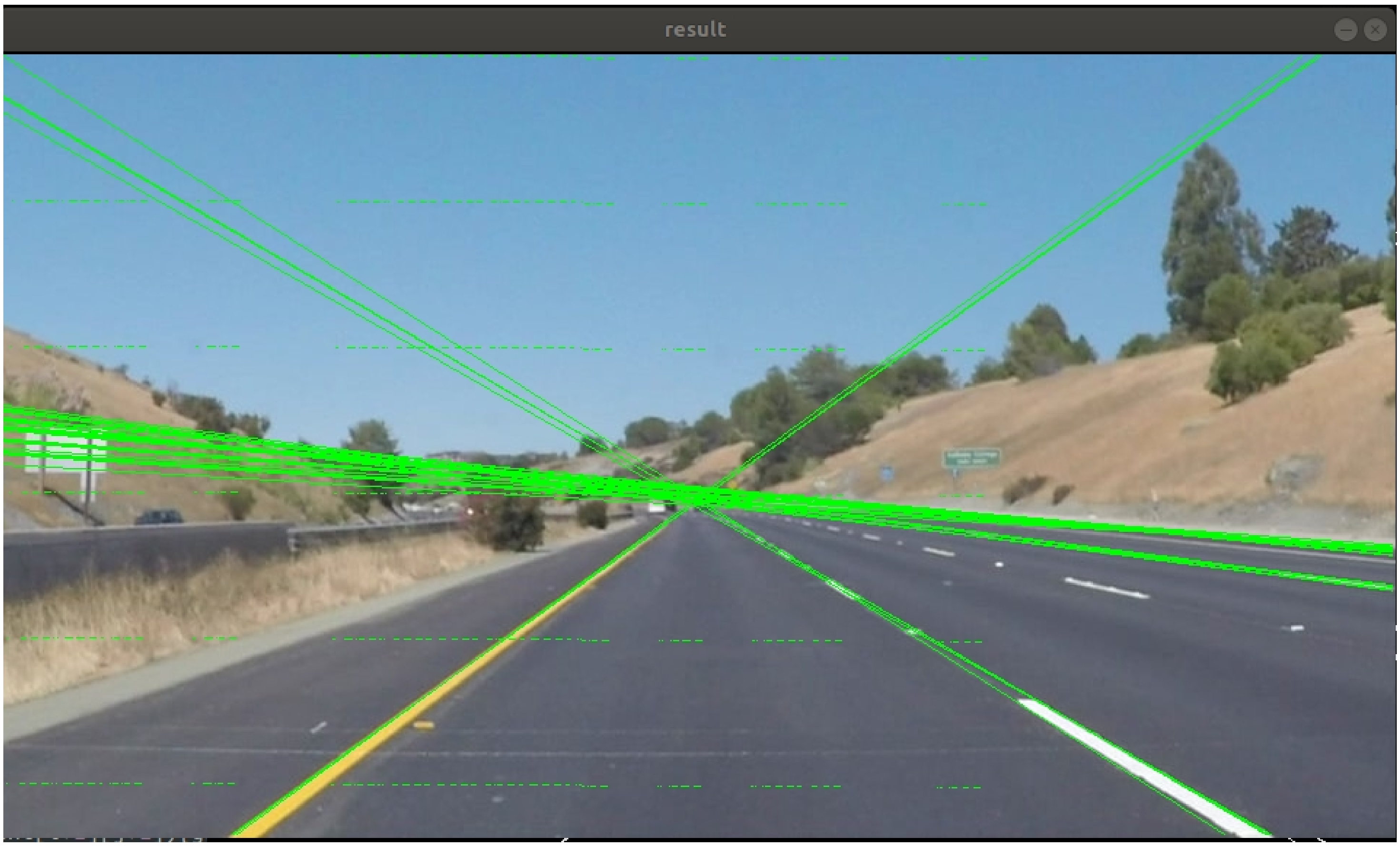


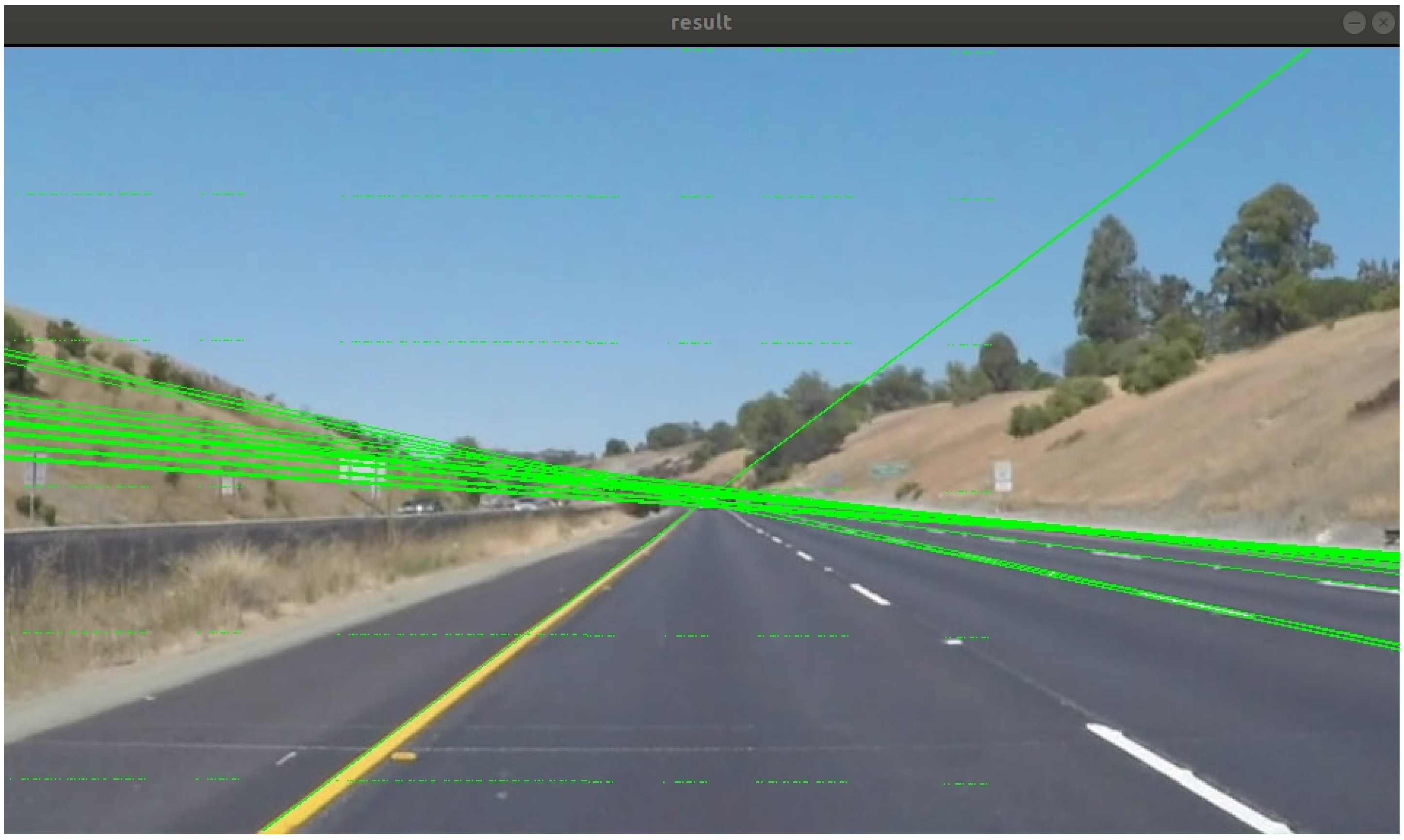




```
sform/Hough Transform ../image files/5.jpg
```




result


result