浙江大学

计算机视觉(本科)作业报告

作业名称：Harris Corner Detection

姓　　名：杨照东

学　　号：3170105289

电子邮箱：3170105289@zju.edu.cn

联系电话：18888923196

导　　师：潘纲

2019 年　　12 月　　10 日

作业名称

Harris Corner Detection

一、 作业已实现的功能简述及运行简要说明

作业完成了 Harris 角点检测算法，利用小窗口在图像上一移动时窗口内灰度总值变化的差值来反映出角点的情况。利用 opencv 库，自己编写了梯度计算，R 图计算的函数，并求出 λ 最大值与最小值图，利用非极大值抑制找出角点，最后在原图中利用红圈画出角点。

二、 作业的开发与运行环境

作业的在 Windows 平台上进行开发，开发使用 opencv3.7.4 库，在 VS2017 平台上使用 C++进行编程。

三、 系统或算法的基本思路、原理、及流程或步骤等

算法的基本思路是利用一个小窗口在图像上移动，小窗口移动时窗口内灰度的总值会产生变化，不同的变化情况代表了窗口覆盖的图像部分的角点情况，例如若任何方向移动灰度的变化不大，那么说明此区域为平滑区域，如果在某个方向移动时灰度变化不大，而其它方向有较大的变化，说明次区域为边缘，如果在任何方向移动灰度都有较大变化，此区域为角点区域。于是我们定义自相关函数来表示角点存在的可能：

$$E_{\mathrm{AC}}(\Delta \boldsymbol{u}) = \sum_i w(\boldsymbol{x}_i)[I_0(\boldsymbol{x}_i + \Delta \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2$$

$$\begin{aligned} E_{\mathrm{AC}}(\Delta \boldsymbol{u}) &= \sum_i w(\boldsymbol{x}_i)[I_0(\boldsymbol{x}_i + \Delta \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 \\ &\approx \sum_i w(\boldsymbol{x}_i)[I_0(\boldsymbol{x}_i) + \nabla I_0(\boldsymbol{x}_i) \cdot \Delta \boldsymbol{u} - I_0(\boldsymbol{x}_i)]^2 \\ &= \sum_i w(\boldsymbol{x}_i)[\nabla I_0(\boldsymbol{x}_i) \cdot \Delta \boldsymbol{u}]^2 \\ &= \Delta \boldsymbol{u}^T \boldsymbol{A} \Delta \boldsymbol{u}, \end{aligned}$$

$$\nabla I_0(\boldsymbol{x}_i) = (\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y})(\boldsymbol{x}_i)$$

$$\boldsymbol{A} = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix},$$

而我们的重点是要求出 A 来，利用 A 的特征值来反映出角点存在的可能性，也就是角点响应。由于 A 为 2 阶矩阵，A 有两个特征值，特征值的大小反

映了角点存在的情况：若两个特征值 λ1 与 λ2 都很小，则此处为平滑区域，若一个特征值很小，另一个很大，那么此处为边缘，如果两个特征值的大小差不多，而且两个特征值都较大，那么此处为角点。同时我们也可以直接通过矩阵 A 的行列式和迹来反映出角点的存在性，也就是：

$$\det(A) - \alpha \operatorname{trace}(A)^2 = \lambda_0\lambda_1 - \alpha(\lambda_0 + \lambda_1)^2$$

这里的 α 为系数，取值在 0.04 到 0.06 之间。那么本次算法用到的主要的方法就是求出每个点的 A 矩阵，再利用 A 矩阵求出每个点的 R 值，再画出 R 图，对 R 图进行非极大值抑制，得到最终的角点检测结果。

四、 具体如何实现，例如关键（伪）代码、主要用到函数与算法等

首先是求梯度，之后要利用梯度来求解每个点的 A 矩阵，首先在全局定义了一些双精度指针的指针，作为之后的二维数组的指针，分别储存梯度的平方与乘积，特征值较大值与特征值较小值，以及 R 值。

```
double** Ixx;
double** Iyy;
double** Ixy;
double** lambda_max;
double** lambda_min;
double** R;
```

求梯度的函数，首先将图像向各个方向扩大一个像素值，保证求出的梯度矩阵阶数与原图像相等，接着使用 sobel 算子，计算每个点的梯度，并将 x 方向与 y 方向的求导结果分别平方，乘积后储存起来。

```
void gradient(Mat former)
{

    Mat extend;

    copyMakeBorder(former, extend, 1, 1, 1, 1, BORDER_REPLICATE);
    int row = former.rows;
    int col = former.cols;
    double Ix, Iy;

    Ixx = new double*[row];
    Iyy = new double*[row];
    Ixy = new double*[row];
    for (int i = 0;i < row;i++) {
        Ixx[i] = new double[col];
        Iyy[i] = new double[col];
        Ixy[i] = new double[col];
    }

```

```
    double kernelx[3][3] = { {-1,0,1},{-2,0,2},{-1,0,1} };
    double kernely[3][3] = { {1,2,1},{0,0,0},{-1,-2,-1} };

    for (int i = 0;i < row;i++) {
        for (int j = 0;j < col;j++) {
            Ix = kernelx[0][0] * extend.at<uchar>(i, j) + kernelx[0][1] * extend.at<uchar>(i, j + 1) + kernelx[0][2] * extend.at<uchar>(i, j + 2) + kernelx[1][0] * extend.at<uchar>(i + 1, j) + kernelx[1][1] * extend.at<uchar>(i + 1, j + 1) + kernelx[1][2] * extend.at<uchar>(i + 1, j + 2) + kernelx[2][0] * extend.at<uchar>(i + 2, j) + kernelx[2][1] * extend.at<uchar>(i + 2, j + 1) + kernelx[2][2] * extend.at<uchar>(i + 2, j + 2);
            Iy = kernely[0][0] * extend.at<uchar>(i, j) + kernely[0][1] * extend.at<uchar>(i, j + 1) + kernely[0][2] * extend.at<uchar>(i, j + 2) + kernely[1][0] * extend.at<uchar>(i + 1, j) + kernely[1][1] * extend.at<uchar>(i + 1, j + 1) + kernely[1][2] * extend.at<uchar>(i + 1, j + 2) + kernely[2][0] * extend.at<uchar>(i + 2, j) + kernely[2][1] * extend.at<uchar>(i + 2, j + 1) + kernely[2][2] * extend.at<uchar>(i + 2, j + 2);
            Ixx[i][j] = Ix * Ix;
            Iyy[i][j] = Iy * Iy;
            Ixy[i][j] = Ix * Iy;
        }
    }
}
```

接下来在求梯度之后进行加窗求卷积，声明三个 Mat 矩阵用来储存 Ix^2，Iy^2，Ix * Iy，接着用另外三个矩阵求得高斯滤波后的结果，那么另外三个矩阵 A，B，C 就分别储存了 Σ Ix^2， Σ Ix * Iy， Σ Iy^2 的结果。

$$A = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix},$$

```
gradient(grey_img);

    Mat A1(row, col, CV_32FC1);
    Mat B1(row, col, CV_32FC1);
    Mat C1(row, col, CV_32FC1);

    for (int i = 0;i < row;i++) {
        for (int j = 0;j < col;j++) {
            A1.at<float>(i, j) = Ixx[i][j];
            B1.at<float>(i, j) = Iyy[i][j];
            C1.at<float>(i, j) = Ixy[i][j];
        }
    }
```

```
14.
15.     Mat A, B, C;
16.
17.     GaussianBlur(A1, A, Size(3, 3), 1);
18.     GaussianBlur(B1, B, Size(3, 3), 1);
19.     GaussianBlur(C1, C, Size(3, 3), 1);
```

接下来利用 A，B，C 求 R 值，并求出矩阵：

$$A = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix},$$

的特征值，将最大值和最小值分别储存在两个二维数组。利用公式：

$$\det(A) - \alpha \operatorname{trace}(A)^2 = \lambda_0 \lambda_1 - \alpha(\lambda_0 + \lambda_1)^2$$

求出 R 值，并画出 R 值图。

```
1.  void CornerResponse(Mat A, Mat B, Mat C,double alpha)
2.  {
3.      Mat M(2,2,CV_32FC1);
4.      Mat value(2, 1, CV_32FC1);
5.      lambda_max = new double*[row];
6.      lambda_min = new double*[col];
7.      for (int i = 0;i < row;i++) {
8.          lambda_max[i] = new double[col];
9.          lambda_min[i] = new double[col];
10.     }
11.     R = new double*[row];
12.     for (int i = 0;i < row;i++) {
13.         R[i] = new double[col];
14.     }
15.
16.     for (int i = 0;i < row;i++) {
17.         for (int j = 0;j < col;j++) {
18.             M.at<float>(0, 0) = A.at<float>(i, j);
19.             M.at<float>(0, 1) = C.at<float>(i, j);
20.             M.at<float>(1, 0) = M.at<float>(0, 1);
21.             M.at<float>(1, 1) = B.at<float>(i, j);
22.             R[i][j] = determinant(M) - alpha * (M.at<float>(0, 0) + M.at<flo
    at>(1, 1)) * (M.at<float>(0, 0) + M.at<float>(1, 1));
23.             eigen(M, value);
24.             if (value.at<float>(0, 0) > value.at<float>(1, 0)) {
25.                 lambda_max[i][j] = value.at<float>(0, 0);
26.                 lambda_min[i][j] = value.at<float>(1, 0);
27.             }
```

```
            else {
                lambda_max[i][j] = value.at<float>(1, 0);
                lambda_min[i][j] = value.at<float>(0, 0);
            }
        }
    }
}
```

算法最关键的部分为以上部分，但之后遇到了很多细节上处理的方面。首先是 R 值与特征值的大小可能达到一个极大的数值，在输入图像的格式为 8 位无符号数时，即灰度值在 0 到 255 的范围时，R 值有时可到达 10 的 10 次方左右的数量级，特征值有时可以达到 10 的 5 次方数量级。但与此同时，有的点的 R 值与特征值并不会很大。这就造成了，首先，由于处理过程中使用 32 位浮点类型矩阵储存数据，但此类型的灰度值范围在 0 到 1，如果直接显示图像，数据明显溢出，必须进行归一化操作。但是由于最大值太大，归一化后大多数的点的灰度值变地非常非常小，显示为图像时基本整张图除了几个白点，其它都是黑的。特征值图像同理。与一些同学交流后发现，他们的 R 图亮度很大，而且有很多白色噪点，可以从中间看出原图像具体的轮廓。但是他们基本都没有做归一化，因此很多噪点是由于数据溢出造成的。于是我在归一化之后，对 R 值图和特征值图进行了求平方根甚至求五次方根的处理，这样一来点与点之间的像素值大小关系还可以保持，但平均灰度值大大增加。这样一来，在展示 R 图和特征值图的时候，看起来更加明显，至于真正之后非极大值抑制求角点的时候，这些操作不会提供什么帮助也不会有什么影响。
这里为求每个矩阵的最大值来进行归一化的部分，还对灰度值求了平方根使其保持原相对大小关系扩大：

```
CornerResponse(A, B, C, 0.04);

    double max2 = 0;

    for (int i = 0;i < row;i++) {
        for (int j = 0;j < col;j++) {
            if (R[i][j] > max2) {
                max2 = R[i][j];
            }
        }
    }

    double max = 0;
    for (int i = 0;i < row;i++) {
        for (int j = 0;j < col;j++) {
            if (lambda_max[i][j] > max) {
                max = lambda_max[i][j];
            }
```

```
        }
    }

    double max1 = 0;
    for (int i = 0;i < row;i++) {
        for (int j = 0;j < col;j++) {
            if (lambda_min[i][j] > max1) {
                max1 = lambda_min[i][j];
            }
        }
    }

    Mat R_img(row, col, CV_32FC1);
    Mat lambda_max_img(row, col, CV_32FC1);
    Mat lambda_min_img(row, col, CV_32FC1);
    for (int i = 0;i < row;i++) {
        for (int j = 0;j < col;j++) {
            if (R[i][j]/max2 > 0.005) {
                R_img.at<float>(i, j) = R[i][j] / max2;
            }
            else {
                R_img.at<float>(i, j) = 0;
            }
            lambda_max_img.at<float>(i, j) = sqrt(abs(lambda_max[i][j]/max))
;
            lambda_min_img.at<float>(i, j) = sqrt(abs(lambda_min[i][j]/max1)
);
        }
    }
```

这里是扩大 R 值的部分，对 R 求五次方根：

```
Mat show_R(row, col, CV_32FC1);
    for (int i = 0;i < row;i++) {
        for (int j = 0;j < col;j++) {
                show_R.at<float>(i, j) = pow(fabs(R[i][j] / max2) , 0.2);

        }
    }
```

之后是将 R 图转换为伪彩色的部分，利用转换公式：

```
for (int i = 0;i < row;i++) {
        for (int j = 0;j < col;j++) {
            if (show_R.at<float>(i, j) < 0.25) {
```

```
                color_R.at<Vec3f>(i, j)[0] = 0;
                color_R.at<Vec3f>(i, j)[1] = 1 - 4 * show_R.at<float>(i, j);

                color_R.at<Vec3f>(i, j)[2] = 1;

            }
            else {
                if (show_R.at<float>(i, j) < 0.5) {
                    color_R.at<Vec3f>(i, j)[0] = 0;
                    color_R.at<Vec3f>(i, j)[1] = 4 * show_R.at<float>(i, j) - 1;
                    color_R.at<Vec3f>(i, j)[2] = 2 - 4 * show_R.at<float>(i, j);
                }
                else {
                    if (show_R.at<float>(i, j) < 0.75) {
                        color_R.at<Vec3f>(i, j)[0] = 4 * show_R.at<float>(i, j) - 2;
                        color_R.at<Vec3f>(i, j)[1] = 1;
                        color_R.at<Vec3f>(i, j)[2] = 0;
                    }
                    else {
                        color_R.at<Vec3f>(i, j)[0] = 1;
                        color_R.at<Vec3f>(i, j)[1] = 4 - 4 * show_R.at<float>(i, j);
                        color_R.at<Vec3f>(i, j)[2] = 0;
                    }
                }
            }
        }
    }
```

接下来是非极大值抑制的部分，由于直接判断某点是否比周围八个点都大的话，遇到紧挨在一起的两个相等的极大值，程序会将其判断为非极大值，但如果改为大于等于会更糟糕，因为所有平坦的部分都满足等于。因此我做了一些操作，首先以一个 5*5 的窗口在图像上移动，找出 25 个点中最大的一个（一定有最大，不会像第一种写法那样存在找不到的情况）。但此时平坦的区域也会有最大值，因此进行判断，此最大值是否比周围八个点中任何一个大，如果比任何一个大，说明此最大点就可以被当作极大点。这样一来，保证了挨着的两个极大点不会被计算漏。得到 NMS 图像，最后将 NMS 图像中的点画到原图像中。

```
Mat NMS_img(row, col, CV_32FC1);
    for (int i = 0;i < row;i++) {
        for (int j = 0;j < col;j++) {
```

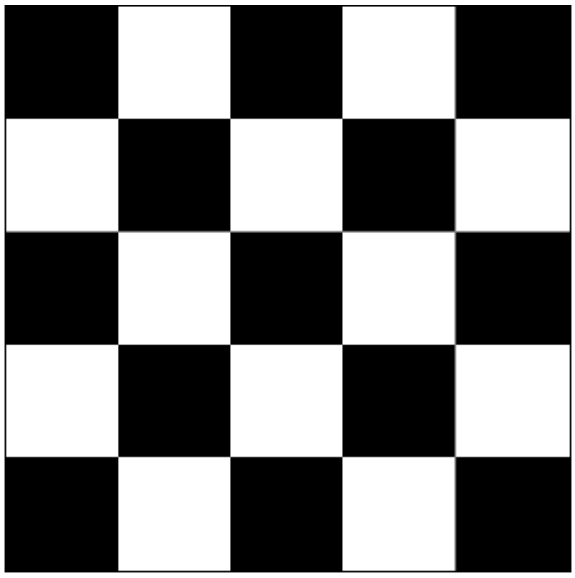
```
4.              NMS_img.at<float>(i, j) = 0;
5.          }
6.      }
7.
8.      Point maxpoint;
9.      maxpoint.x = 1;
10.     maxpoint.y = 1;
11.     double maxp_value = 0;
12.     for (int i = 3;i < row - 3;i++) {
13.         for (int j = 3;j < col - 3;j++) {
14.             for (int m = -2;m <= 2;m++) {
15.                 for (int n = -2;n <= 2;n++) {
16.                     if (R_img.at<float>(i + m, j + n) > maxp_value) {
17.                         maxp_value = R_img.at<float>(i + m, j + n);
18.                         maxpoint.x = j + n;
19.                         maxpoint.y = i + m;
20.                     }
21.                 }
22.             }
23.             maxp_value = 0;
24.             if (R_img.at<float>(maxpoint.y, maxpoint.x) > R_img.at<float>(ma
    xpoint.y - 1, maxpoint.x - 1) || R_img.at<float>(maxpoint.y, maxpoint.x) > R
    _img.at<float>(maxpoint.y - 1, maxpoint.x) || R_img.at<float>(maxpoint.y, ma
    xpoint.x) > R_img.at<float>(maxpoint.y - 1, maxpoint.x + 1) || R_img.at<floa
    t>(maxpoint.y, maxpoint.x) > R_img.at<float>(maxpoint.y, maxpoint.x - 1) ||

25.                 R_img.at<float>(maxpoint.y, maxpoint.x) > R_img.at<float>(ma
    xpoint.y, maxpoint.x + 1) || R_img.at<float>(maxpoint.y, maxpoint.x) > R_img
    .at<float>(maxpoint.y + 1, maxpoint.x - 1) ||
26.                 R_img.at<float>(maxpoint.y, maxpoint.x) > R_img.at<float>(ma
    xpoint.y + 1, maxpoint.x) || R_img.at<float>(maxpoint.y, maxpoint.x) > R_img
    .at<float>(maxpoint.y + 1, maxpoint.x + 1)) {
27.                 NMS_img.at<float>(maxpoint.y, maxpoint.x) = 1;
28.             }
29.         }
30.     }
31.
32.     Point center;
33.     for (int i = 0;i < row;i++) {
34.         for (int j = 0;j < col;j++) {
35.             if (NMS_img.at<float>(i, j) == 1) {
36.                 center.x = j;
37.                 center.y = i;
38.                 circle(primi_img, center, 2, Scalar(0, 0, 255), 1);
```

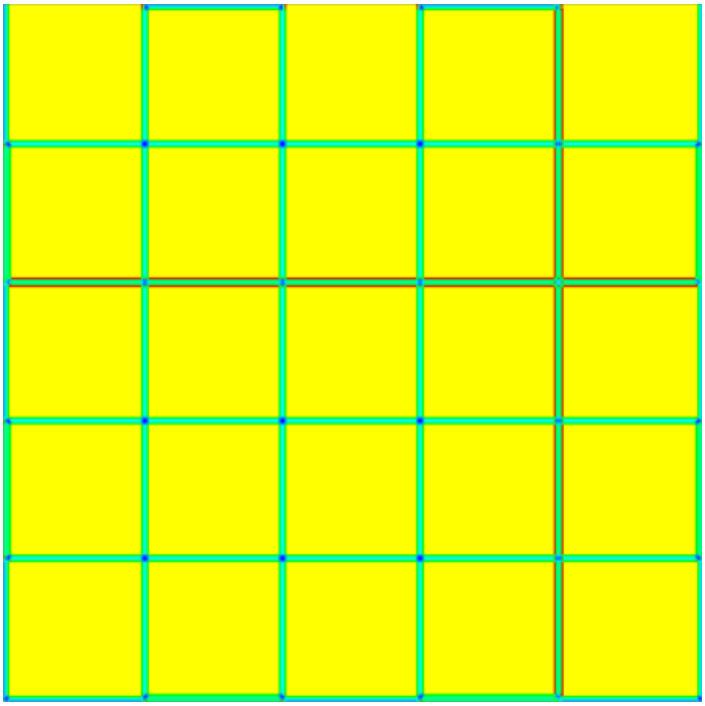
```
39.             }
40.         }
41.     }
```

五、 实验结果与分析

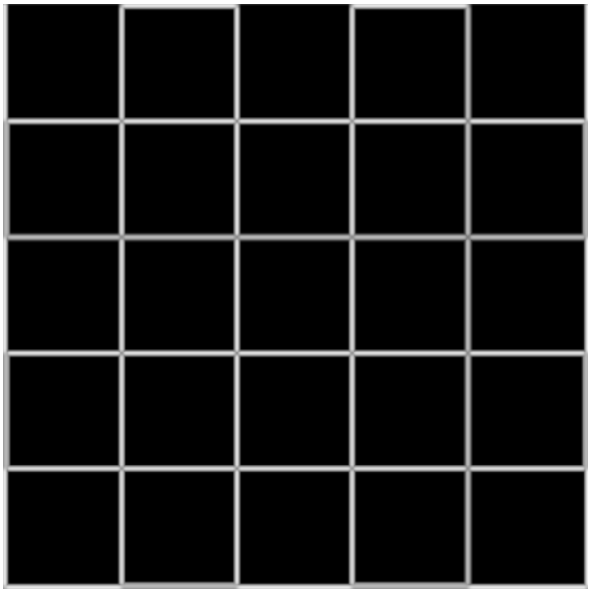
首先是最规整的格子图像：

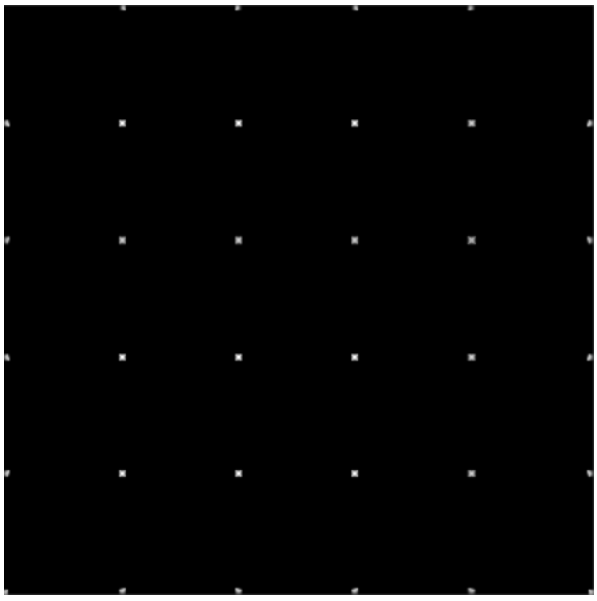


R 图彩色结果：

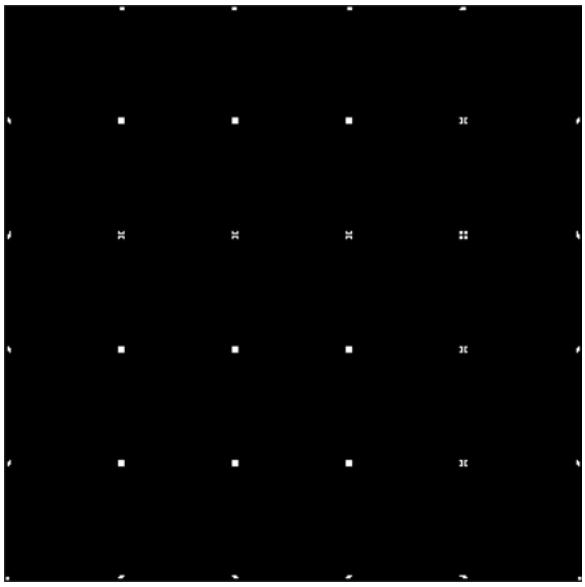


极大值图与极小值图：

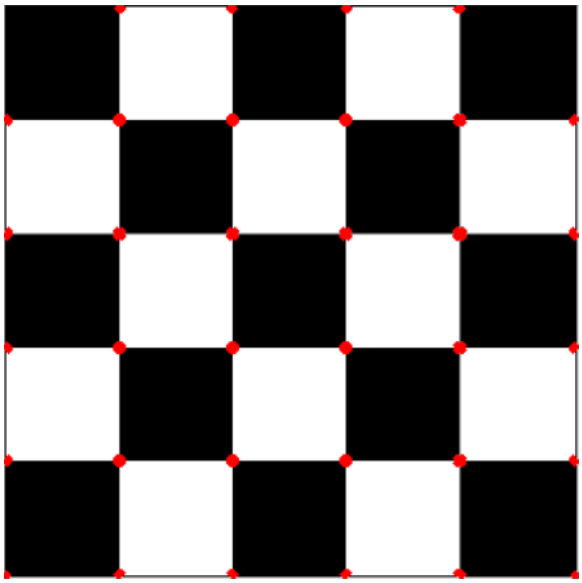




非极大值抑制图：

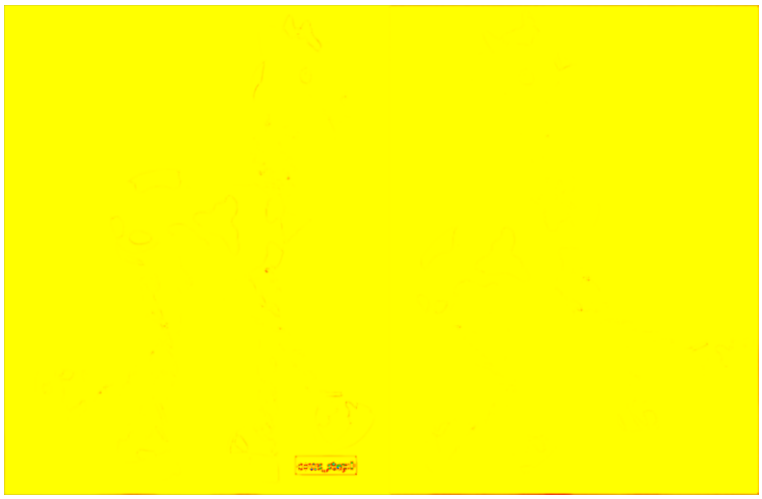


最终结果：

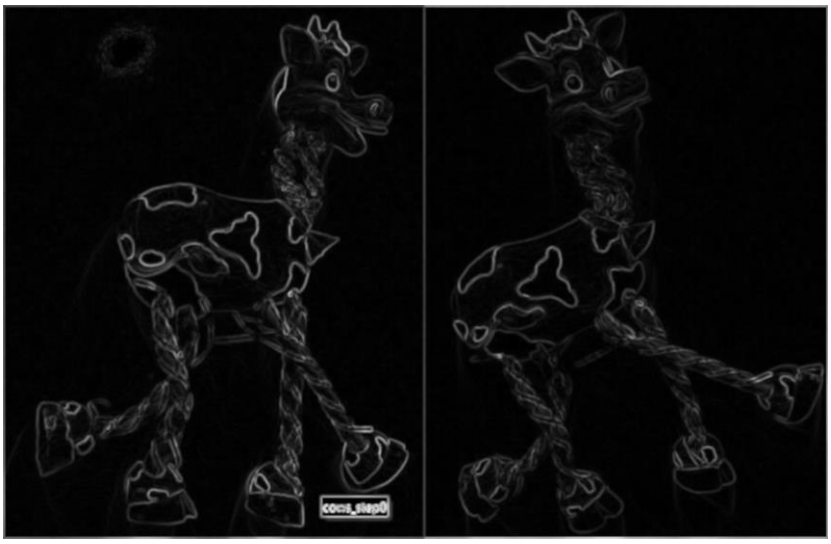


接下来是上课时用的长颈鹿图像，首先是 R 值（求五次方根后）：

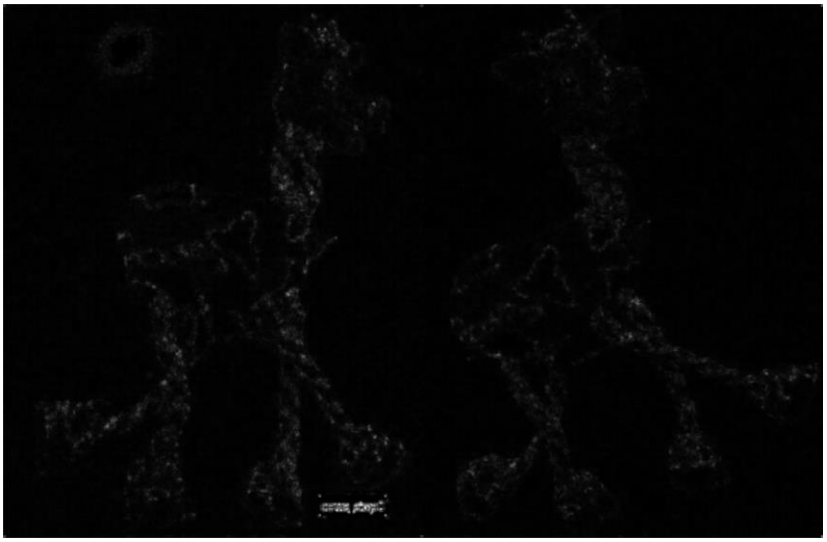
没有求五次方根的 R 图，可以看到只能看见极少的部分有明显轮廓，但其实并不影响接下来后续的角点检测：



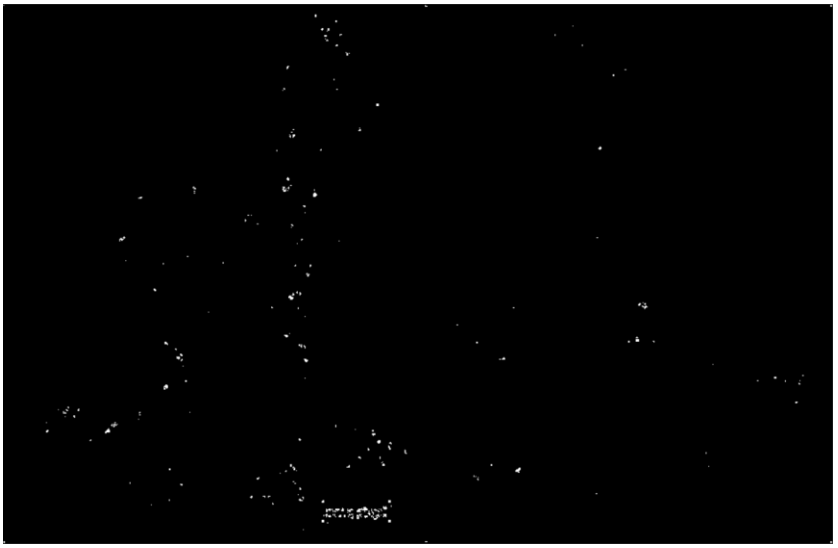
极大值图：



极小值图：



非极大值抑制图：

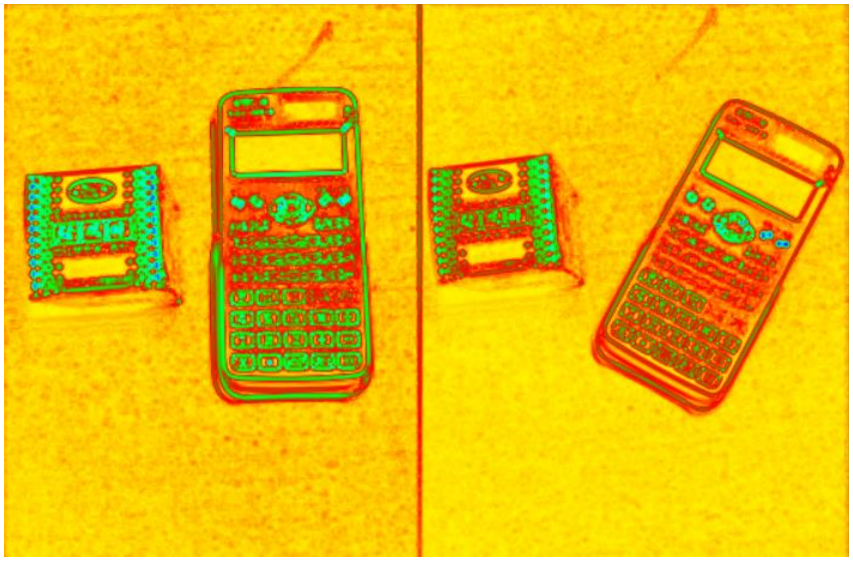


角点检测结果：

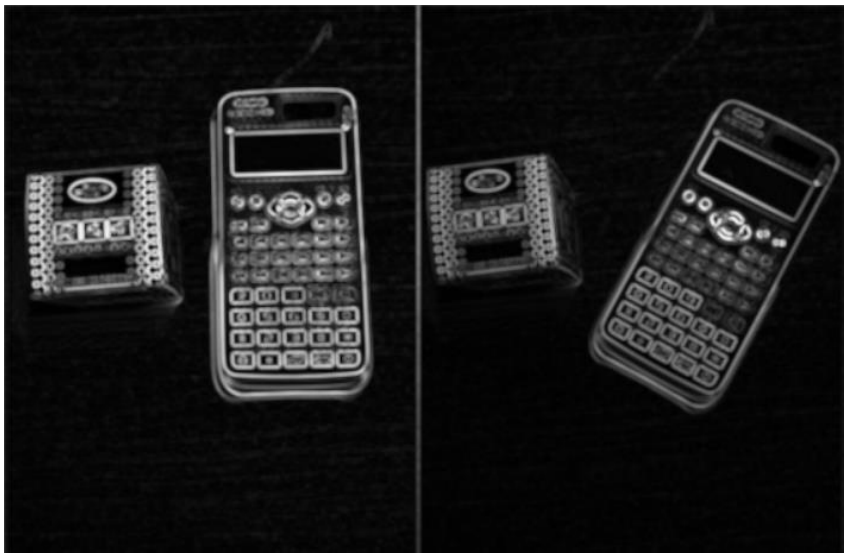
自己拍照的检测：

角度旋转，亮度改变

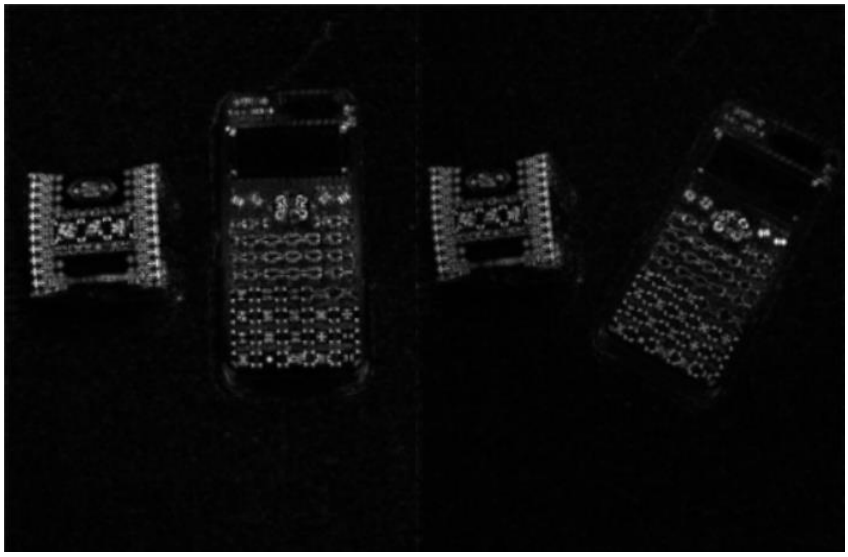
R 图：



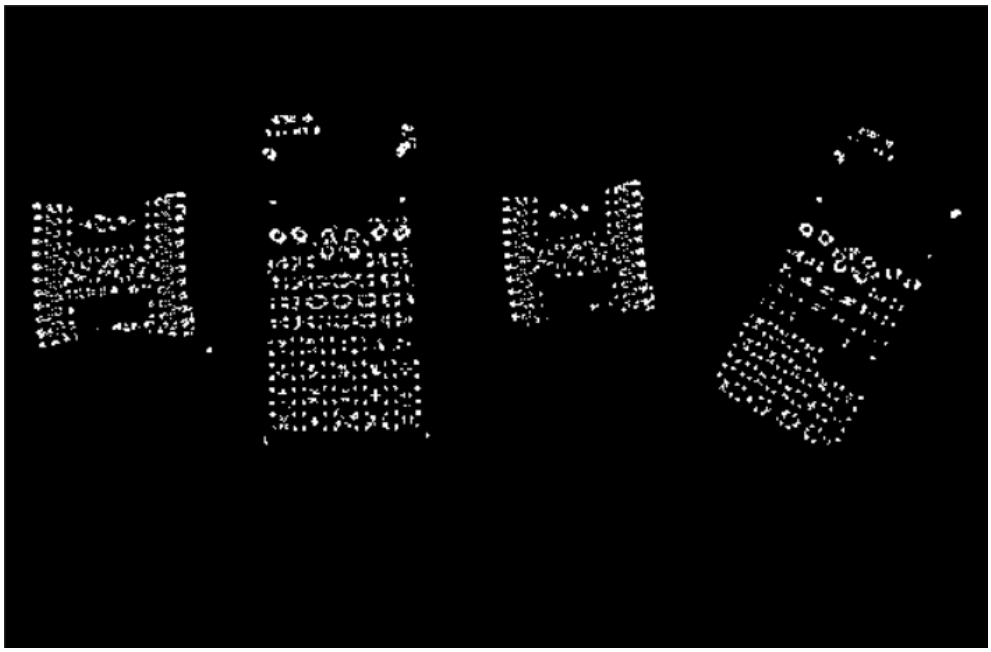
极大值：



极小值：



非极大值抑制：



由于计算器和墨水盒子以黑白为主，而且上面有大量符号，所以角点很多。

同时阈值的选取也与最终结果有影响：

同一幅图像，当求 R 值时选取 R>0.01 的点时：

当选择 R > 0.001:

\

可以说明当阈值越小，可以找到更多的细节，但是太小的阈值会很容易受到干扰，因此经过尝试，0.01 到 0.001 的阈值比较可靠。

六、 结论与心得体会

通过此次作业，对于我们生活中的图像检测有了一点初步的了解，因此之前看别人写的人脸检测，也是会有很多点打在脸部的各个位置，这也就是特征点吧，同时这样的特征点检测也不光能用在图像中，还可以在各种音频，电信号中使用，学习此特征检测算法，受益匪浅。