

浙江大学

计算机视觉作业报告

作业名称：实现 Harris Corner Detector

姓 名：陈润健

学 号：3160103989

电子邮箱：3160103989@zju.edu.cn

联系电话：18868104871

导 师：潘纲



2018 年 12 月 6 日

实现 Harris Corner Detector

一、作业已实现的功能简述及运行简要说明

1. 实现 Harris Corner Detector。
2. 使用不同的梯度计算算子（sobel 和 prewitt）进行对比实验。
3. 在不同的图片上进行对比实验。

二、作业的开发与运行环境

IDE: Xcode 10.1

操作系统: MacOS 10.14.1

SDK: macosx10.14

三、基本思路，用到的函数及流程

1. Corner Detector 类的概述:

- 1) 类的定义:

```
class ChenRJ_Detector
{
private:
    Mat prey;
    int patch_H,patch_W;
    double *patch_factor,*R_value,*X_gradient,*Y_gradient,R_Threshold;
    void Gradient_Calculator();
    void R_Calculator(double empirical_const);
    bool Local_Maximum(int x,int y);
    int Get_Index(int x,int y);

public:
    ChenRJ_Detector(Mat image,int H = 9 ,int W = 9):prey(image),patch_H(H),patch_W(W)
    {
        patch_factor = new double[patch_H*patch_W];
        for (int i = 0 ; i < H ; i++)
            for (int j = 0 ; j < W ; j++)
                if ((i!=H/2) || (j!=W/2))
                {
                    patch_factor[i*W+j] = 1.0 / ( (i-H/2)*(i-H/2) + (j-W/2)*(j-W/2) );
                }
                else
                    patch_factor[i*W+j] = 2.0;
        R_value = new double[image.cols*image.rows];
        X_gradient = new double[image.cols*image.rows];
        Y_gradient = new double[image.cols*image.rows];
        R_Threshold = 0;
    }
    ~ChenRJ_Detector();
    Mat Feature_Generator();
};


```

- 2) Gradient_Calculator 用于计算梯度值, 保存在 X_gradient 和 Y_gradient 中, 等待后续调用即可, 这里可以使用 sobel 或者 prewitt 算子:

```
void ChenRJ_Detector::Gradient_Calculator()
{
    int X_gra_factor[3][3]={-1,0,1,-2,0,2,-1,0,1},Y_gra_factor[3][3]={1,2,1,0,0,0,-1,-2,-1};

    for (int i = patch_H/2 ; i < prey.rows-patch_H/2 ; i++)
        for (int j = patch_W/2 ; j < prey.cols-patch_W/2 ; j++)
    {
        double sum_x = 0,sum_y = 0;
        for (int k = -1 ; k <= 1 ; k++)
            for (int m = -1 ; m <= 1 ; m++)
        {
            unsigned char * pos = prey.ptr<unsigned char>(i+k);

            sum_x += X_gra_factor[k+1][m+1] * pos[j+m];

            sum_y += Y_gra_factor[k+1][m+1] * pos[j+m];
        }
        X_gradient[Get_Index(i,j)] = sum_x;
        Y_gradient[Get_Index(i,j)] = sum_y;
    }
}
```

- 3) R_Calculator 用于计算 R 值:

```
void ChenRJ_Detector::R_Calculator(double empirical_const)
{
    Gradient_Calculator();
    double max = 0;

    for (int i = patch_H/2 ; i < prey.rows-patch_H/2 ; i++)
        for (int j = patch_W/2 ; j < prey.cols-patch_W/2 ; j++)
    {
        double sum1_1 = 0,sum1_2 = 0,sum2_1 = 0,sum2_2 = 0;
        for (int k = -patch_H/2 ; k <= patch_H/2 ; k++)
            for (int m = -patch_W/2 ; m <= patch_W/2 ; m++)
        {
            sum1_1 = sum1_1 + patch_factor[(k+patch_H/2)*patch_W+m+patch_W/2] *
                X_gradient[(i+k)*prey.cols+j+m]*X_gradient[(i+k)*prey.cols+j+m];

            sum1_2 = sum1_2 + patch_factor[(k+patch_H/2)*patch_W+m+patch_W/2] *
                X_gradient[(i+k)*prey.cols+j+m]*Y_gradient[(i+k)*prey.cols+j+m];

            sum2_1 = sum2_1 + patch_factor[(k+patch_H/2)*patch_W+m+patch_W/2] *
                X_gradient[(i+k)*prey.cols+j+m]*Y_gradient[(i+k)*prey.cols+j+m];

            sum2_2 = sum2_2 + patch_factor[(k+patch_H/2)*patch_W+m+patch_W/2] *
                Y_gradient[(i+k)*prey.cols+j+m]*Y_gradient[(i+k)*prey.cols+j+m];
        }
        R_value[i*prey.cols+j] = sum1_1*sum2_2 - sum2_1*sum1_2 - empirical_const * (sum1_1+sum2_2) * (sum1_1+sum2_2);
        if (max < R_value[i*prey.cols+j])
        {
            max = R_value[i*prey.cols+j];
            std::cout << max << " " << sum1_1 << " " << sum1_2 << " " << sum2_1 << " " << sum2_2 << std::endl;
        }
    }
    for (int i = patch_H/2 ; i < prey.rows-patch_H/2 ; i++)
        for (int j = patch_W/2 ; j < prey.cols-patch_W/2 ; j++)
    {
        R_value[i*prey.cols+j] /= max;
        R_value[i*prey.cols+j] *= 255;
    }
    R_Threshold = 0.8;
}
```

- 4) 调用 Local_Maximum 计算当前点是否是局部最大值:

```
bool ChenRJ_Detector::Local_Maximum(int x,int y)
{
    bool Local_Max = true;

    if (R_value[Get_Index(x,y)] < R_Threshold) Local_Max = false;

    for (int i = -patch_H*4 ; i <= patch_H*4 ; i++)
        for (int j = -patch_W*4 ; j <= patch_W*4 ; j++)
    {
        if ((Get_Index(x+i,y+j)>=0) && (Get_Index(x+i,y+j) < prey.cols*prey.rows) && (R_value[Get_Index(x,y)] <
            R_value[Get_Index(x+i,y+j)])) Local_Max = false;
    }
    return Local_Max;
}
```

2. 梯度计算方法的选择:

1) Sobel 梯度计算子:

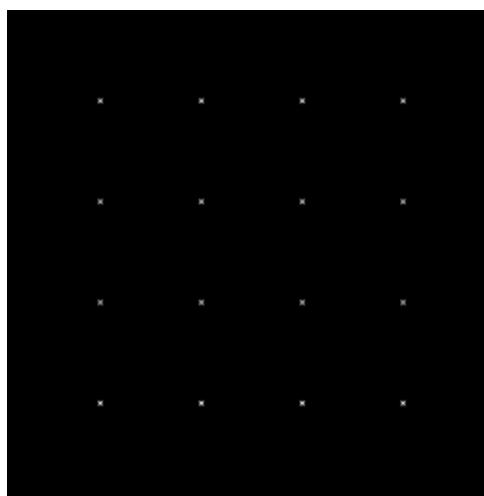
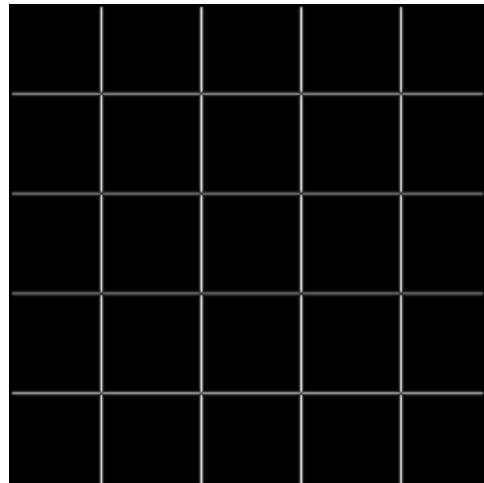
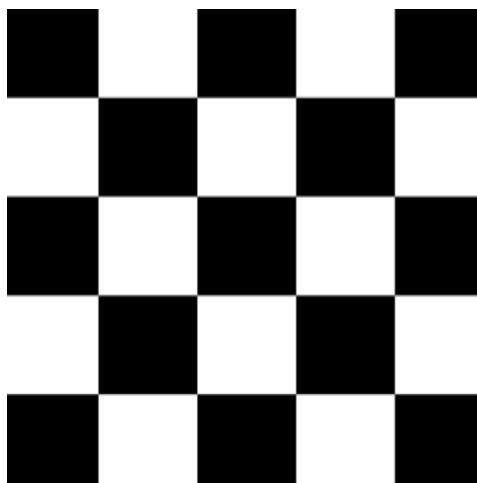
$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

2) Prewitt 梯度算子:

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

四、实验结果与分析：

1. 标定图: (左上角为原图, 右上角为最大特征值图, 左下角为最小特征值图)

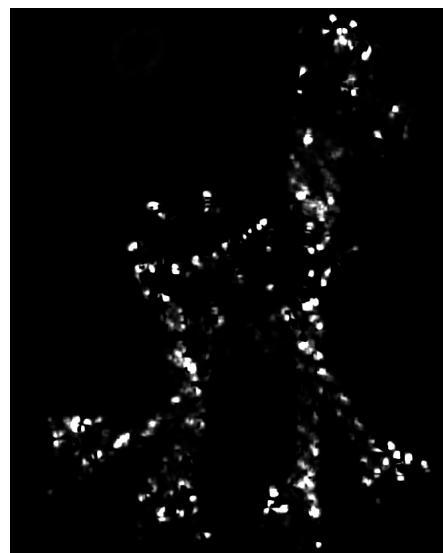


2. 对比 opencv 的哈里斯角点检测和自己写的角度检测：opencv 中的角度检测会把 R 值比阈值大的点筛选出来，然后根据 R 的大小给定灰度，而在本次作业中，我们最终需要的是找到 R 的局部极值，因此，在显示 R 值比阈值大的图片中，我没有将点用灰度值的不同显示出来，而是直接给出二值图，对比自己的二值图中的白色部分与 opencv 做出来的非黑色部分以展示效果。（图片如果看不清楚，在附件中有原图，麻烦老师检查！）

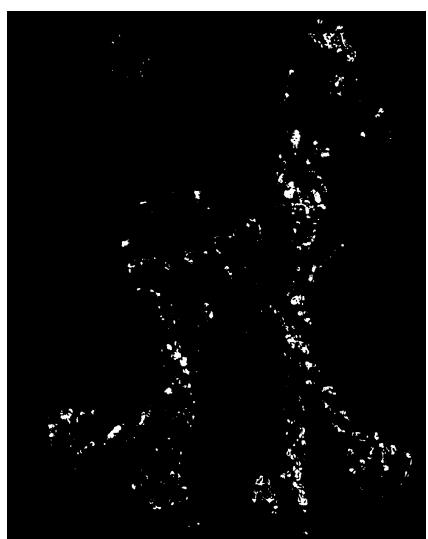
1) 第一张图片的实验结果：



原图



opencv 的效果



只采用阈值进行筛选



采用阈值+局部极值进行筛选



最大特征值 (scale 到区间 [0, 255])



最小特征值 (scale 到区间 [0, 255])

可以发现，只采用阈值筛选的结果与 opencv 的函数调用结果基本一致。(白色区域与 opencv 结果中的非黑色区域基本一致)

2) 第二张图片的实验结果：



原图



opencv 的效果



只采用阈值进行筛选



采用阈值+局部极值进行筛选



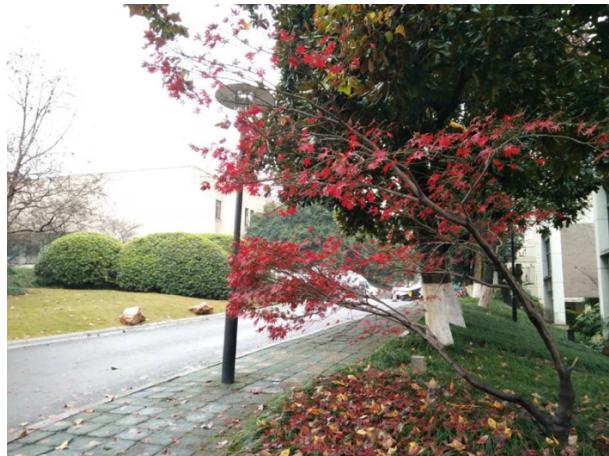
最大特征值 (scale 到区间 [0, 255])



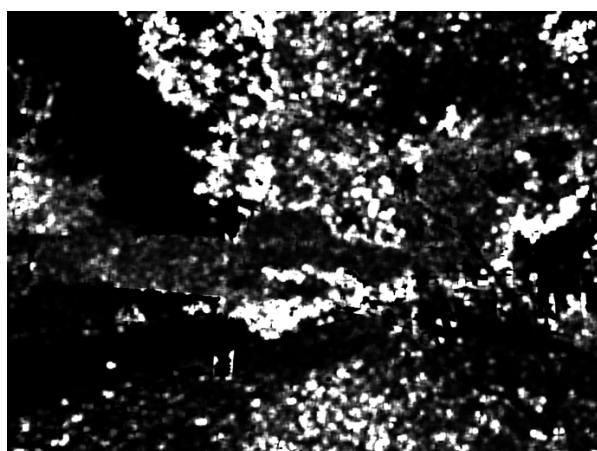
最小特征值 (scale 到区间 [0, 255])

可以发现，只采用阈值筛选的结果与 opencv 的函数调用结果基本一致。（白色区域与 opencv 结果中的非黑色区域基本一致）

3) 第三张图片的实验效果:



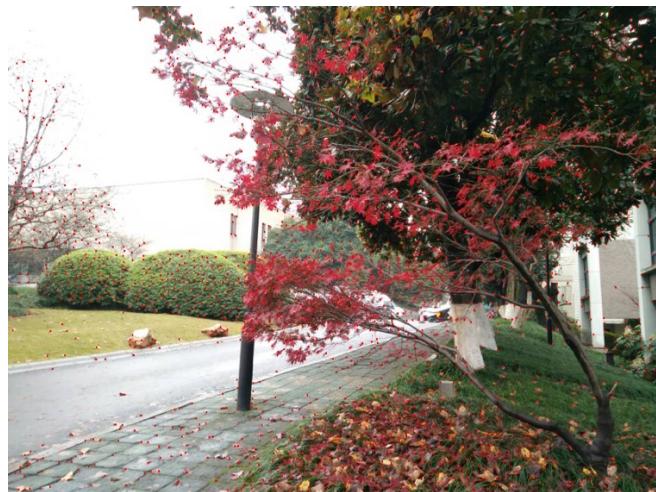
原图



opencv 的效果



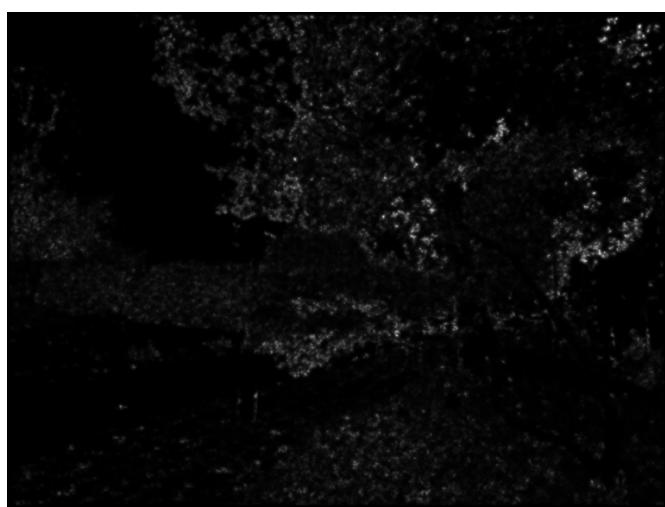
只采用阈值进行筛选



采用阈值+局部极值进行筛选



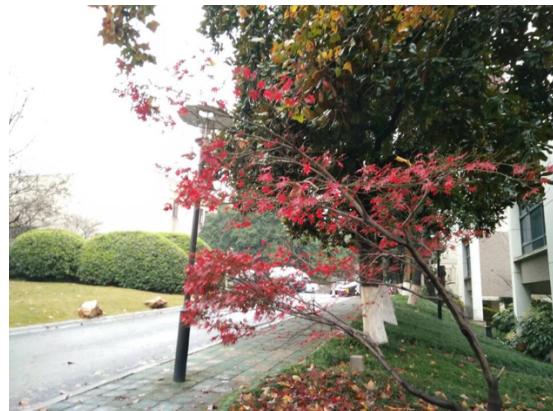
最大特征值 (scale 到区间 [0, 255])



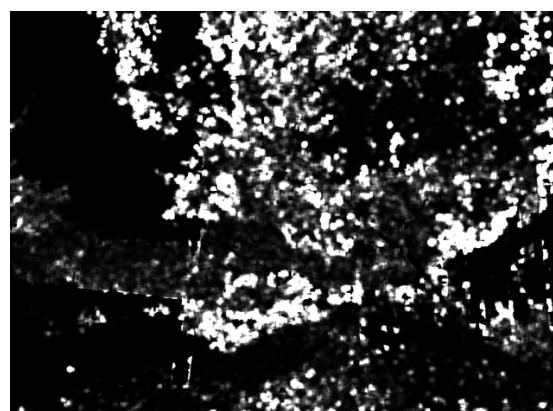
最小特征值 (scale 到区间 [0, 255])

可以发现，只采用阈值筛选的结果与 opencv 的函数调用结果基本一致。（白色区域与 opencv 结果中的非黑色区域基本一致）

4) 第四张图片的实验效果：



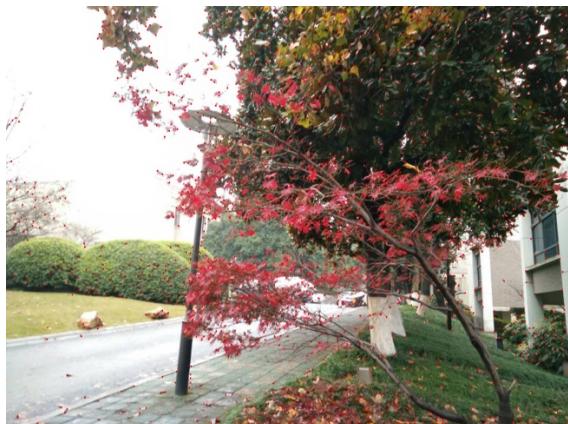
原图



opencv 的效果



只采用阈值进行筛选



采用阈值+局部极值进行筛选



最大特征值 (scale 到区间 [0, 255])

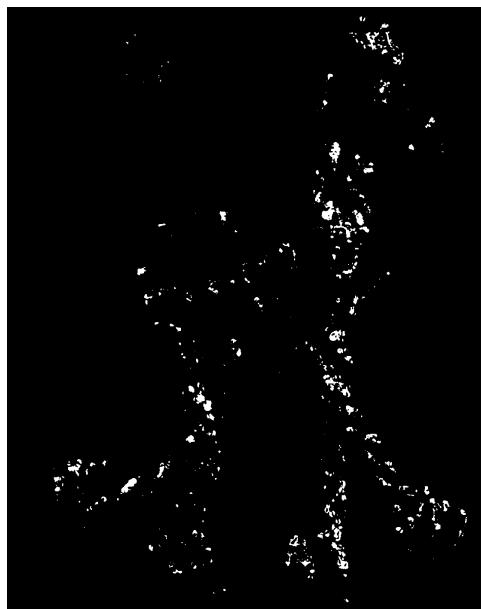


最小特征值 (scale 到区间 [0, 255])

可以发现，只采用阈值筛选的结果与 opencv 的函数调用结果基本一致。（白色区域与 opencv 结果中的非黑色区域基本一致）

结论：发现两组图片中，相似的特征基本上都可以被识别到。

3. 使用不同梯度算子进行计算：



Sobel 算子



Prewitt 算子

可以发现两种梯度算子的结果基本相同。

五、 结论与心得体会

在这一次的实验中，我实现了 Harris Corner Detector 并做了多组实验，实现效果比较好，希望在接下来的课程和作业中，能够接触到更多实用的算法，同时学习到更多关于计算机视觉的理论知识，尝试自己实现更多有趣的功能。