计算机视觉第三次作业

16340219 王亮岛

作业 1：

测试环境:

Windows，可执行文件为 a.exe。

算法描述：

1. 创建霍夫空间。使用极坐标创建霍夫空间，横坐标为角度 theta，纵坐标为图像长和宽的 1/2 的平方和再开根号。以图像中心为原点建立霍夫空间。根据直线的极坐标方程 p=xcos(theta)+ysin(theta)。同时对霍夫空间的每个点投票，如果位于直线上，则票数加一。

```
cimg_forXY(img, x, y) {
    int value = img(x, y), p = 0;
    if (value != 0) {
        int x0 = x - width / 2, y0 = height / 2 - y;
        for (int i = 0; i < thetaSize; i++) {
            //投票
            p = x0 * setCos[i] + y0 * setSin[i];
            if (p >= 0 && p < maxLength) {
                houghImage(p, i)++;
            }
        }
    }
}
```

2. 检测直线。使用局部最大值的方法检测。这里就需要设置局部范围，对于在范围内的除最大值外的点均设置为 0。然后把不为 0 的坐标和霍夫空间中的权重存入数组中。

```
int EdgeDetect::getMaxHough(CImg<float>& img, int& size, int& y, int& x) {
    int width = (x + size > img._width) ? img._width : x + size;
    int height = (y + size > img._height) ? img._height : y + size;
    int max = 0;
    for (int j = x; j < width; j++) {
        for (int i = y; i < height; i++) {
            max = (img(j, i) > max) ? img(j, i) : max;
        }
    }
    return max;
}
```

3. 在画边缘时，先对这些权重进行排序，由于 A4 纸只有 4 条边，所以取排序后的最大的四个值，筛选出四条边。同时创建一个检测交点的图，并对每次出现在 4 条边上的点加 255/2 作为权重。

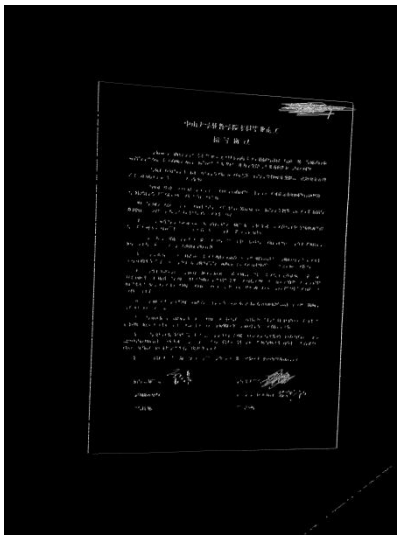
4. 画交点，在检测交点的图上判断每一个点，找出权重大于等于 255 的点，说明在该点处至少有两条直线通过，则标为红色，即找出四个交点。

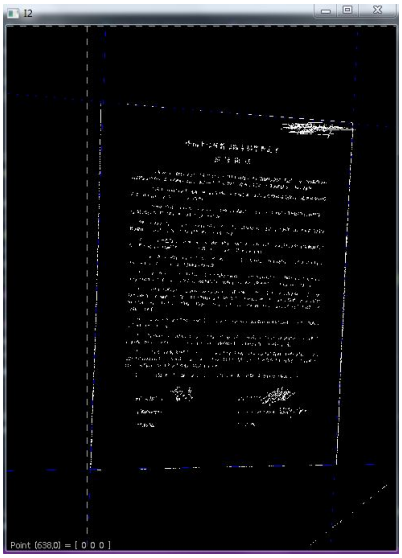
实验结果：

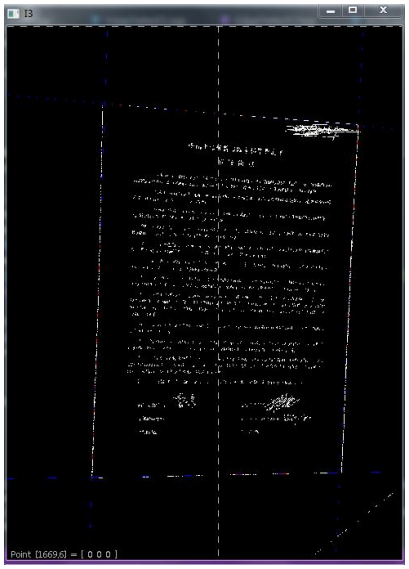
从左到右分别为 I1,I2,I3,I4.

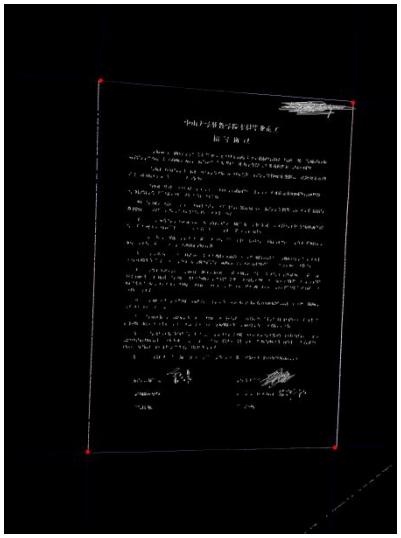
输入的参数包括灰度图，输出图像，以及 A4 纸边缘个数，一般为 4.

图 1：





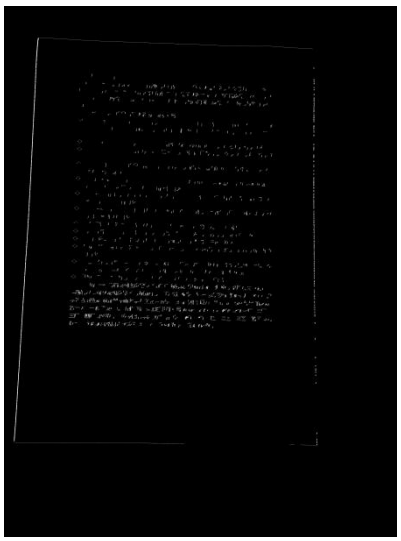


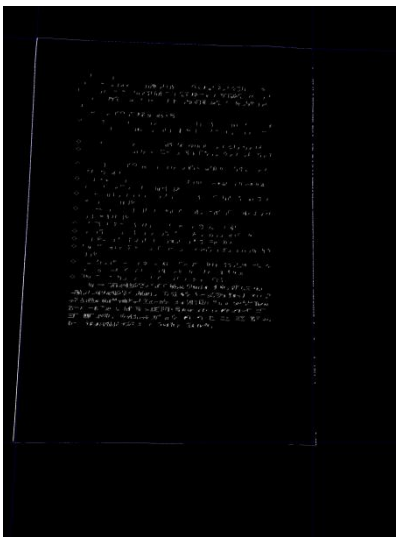


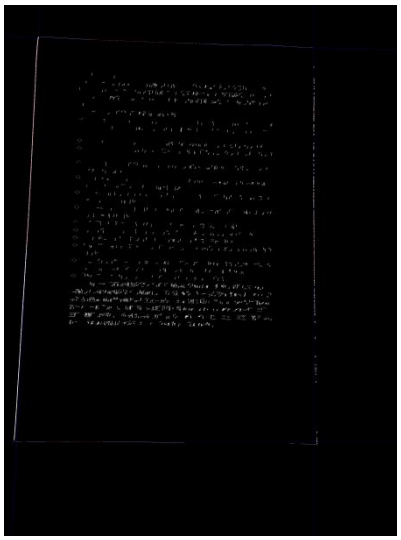
直线的极坐标方程如下：

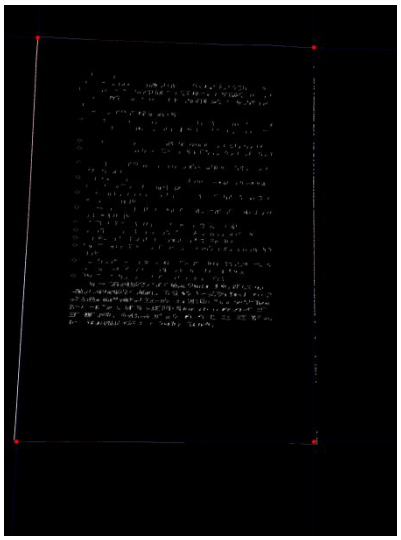
```
1138=x0*0.99863+y0*-0.052336
841=x0*-0.999391+y0*0.0348995
1419=x0*0.0871557+y0*0.996195
1434=x0*0.0174524+y0*-0.999848
```

图 2：





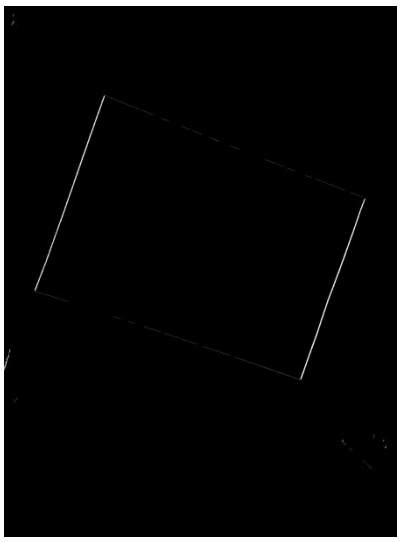


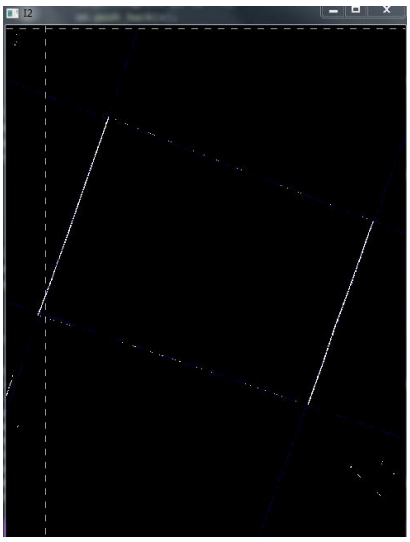


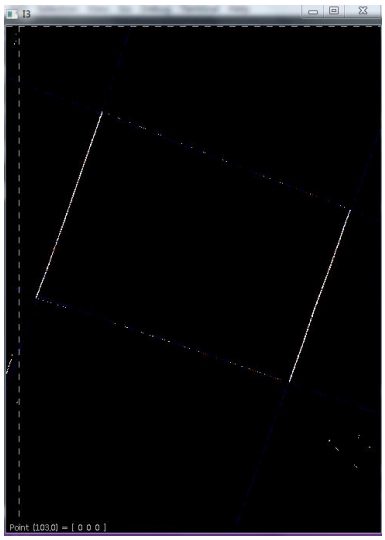
直线的极坐标方程为：

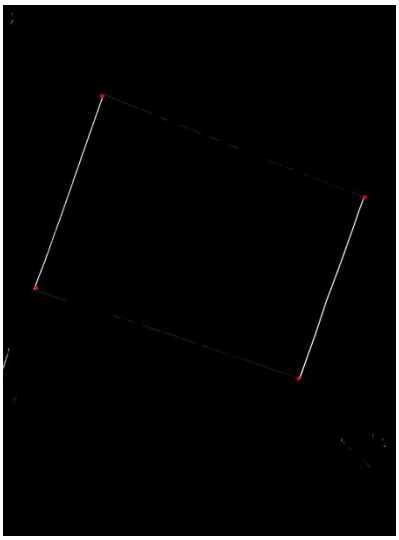
```
1383=x0*-0.99863+y0*0.052336
899=x0*1+y0*0
1794=x0*0.0348995+y0*0.999391
1354=x0*-8.8469e-010+y0*-1
```

图 3：





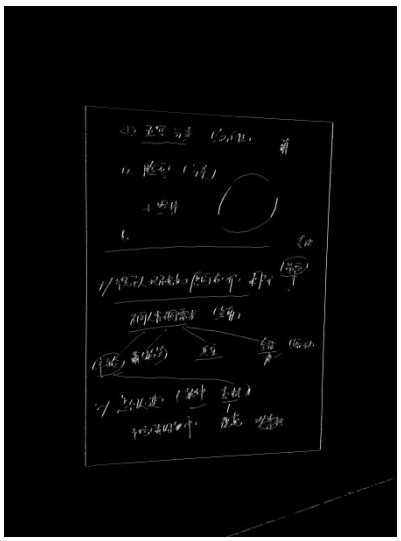


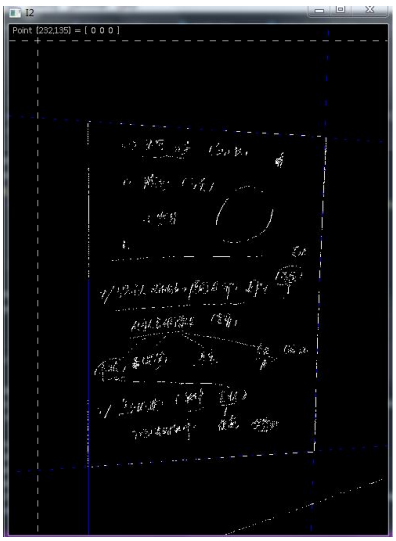


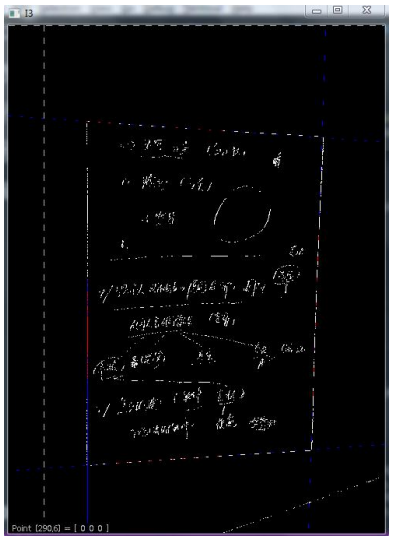
直线的极坐标方程为：

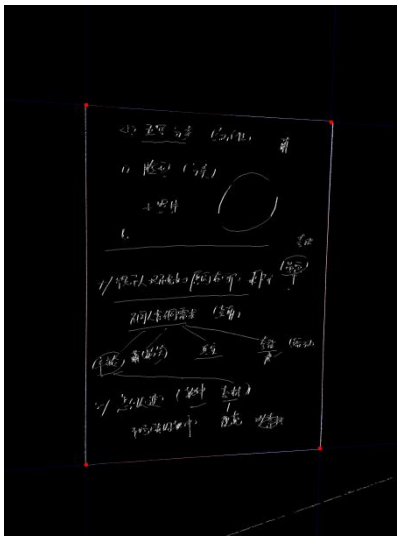
```
1178=x0*-0.945519+y0*0.325568
1031=x0*0.939693+y0*-0.34202
552=x0*-0.325568+y0*-0.945519
1010=x0*0.358368+y0*0.93358
```

图 4：





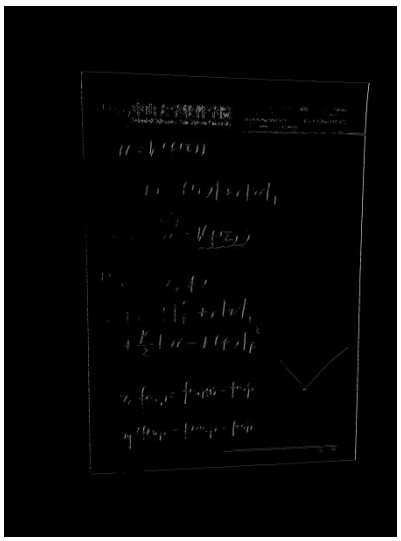


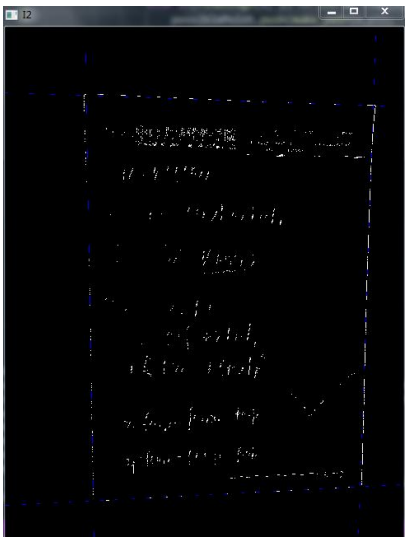


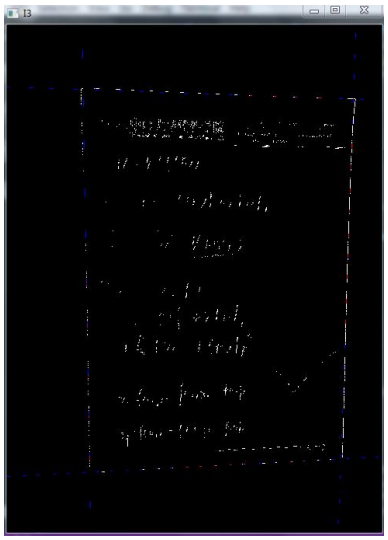
直线的极坐标方程：

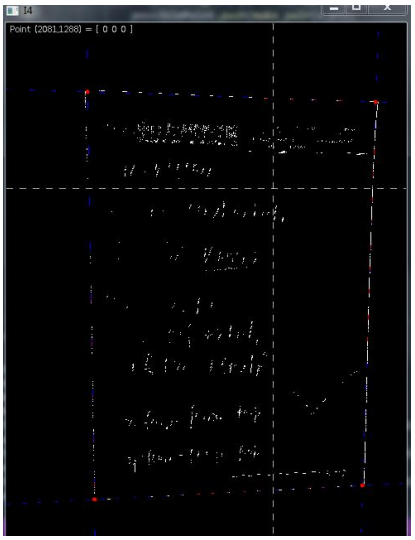
```
995=x0*0.999391+y0*-0.0348995
905=x0*-1+y0*5.89793e-010
1470=x0*0.0697565+y0*-0.997564
1242=x0*0.0697565+y0*0.997564
```

图 5：







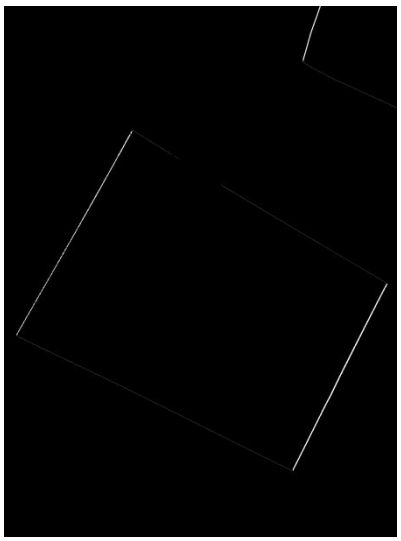


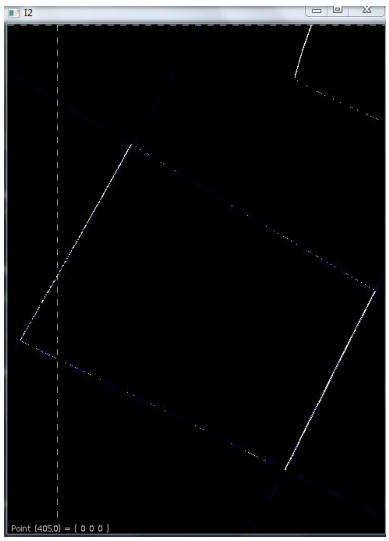
直线的极坐标方程：

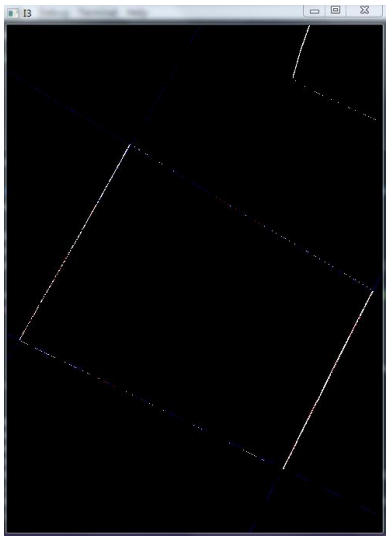
```
1264=x0*0.999391+y0*-0.0348995
1552=x0*0.052336+y0*-0.99863
899=x0*-0.999848+y0*-0.0174524
1536=x0*0.0348995+y0*0.999391
```

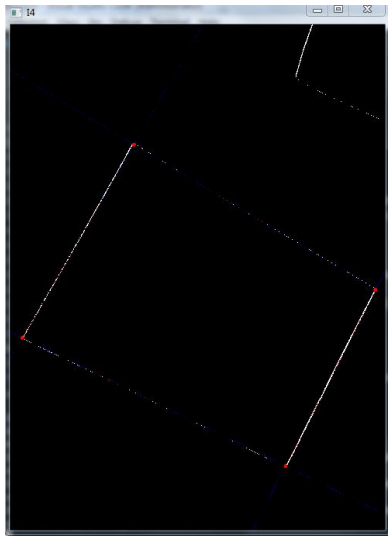
图 6：

由于需要把两条多余的边缘剔除在外，所以输入的 A4 纸边缘数量为 6.









直线的极坐标方程为：

```
1013=x0*-0.866025+y0*0.5
251=x0*0.951057+y0*-0.309017
1090=x0*-0.438371+y0*-0.898794
671=x0*0.515038+y0*0.857167
```

结果分析：

对于绝大多数边缘图都能找到边缘，检测出边缘主要取决于 A4 纸中边缘点的个数，如果边缘点的个数足够多，那么这个边缘就能很好的被检测出来，但如果干扰的边缘的边缘点的个数过多，就会干扰到需要检测出的边缘。解决的办法就是不断调整搜索局部最大值的区域范围，筛选掉干扰的边缘。

思考：

关于速度的提高问题，可以通过扩大寻找局部最大值的区域，减少搜索次数，但这样做的一个缺点是寻找的局部最大值过少，不足以检测出边缘。

作业 2：

测试环境：

Windows，可执行文件为 a2.exe。

算法描述：

1. 首先根据半径的上下阈值，通过建立霍夫空间为半径赋予相应的权重，图片的像素点在圆上的数量越多，则该半径的权重越大。然后对这些半径的权重进行排序，选择的半径数量为上下阈值的差除以 5，通过这样的半径选择找出圆的个数，即圆的个数取决于上下阈值的选择。

2. 通过对每一个半径建立霍夫图像，找到满足半径的圆心的像素点的个数给予权重。对这些圆心进行排序，确定圆的位置。

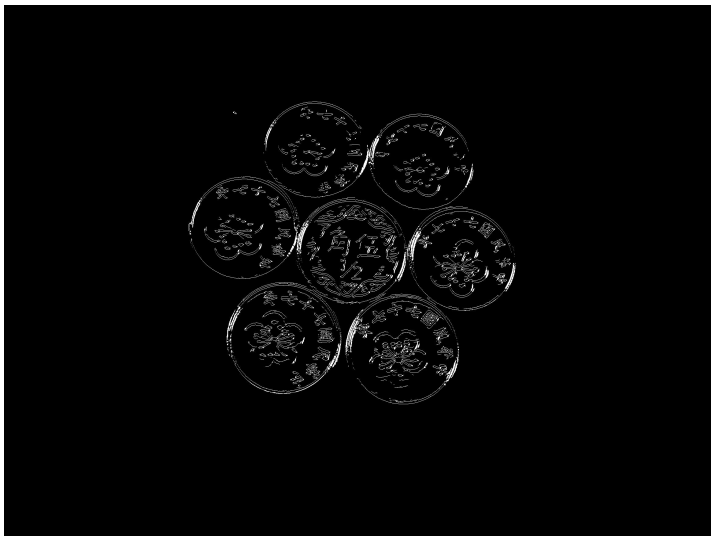
3. 在画圆的时候，判断检测出来的圆心坐标是否跟已检测的圆心坐标的距离，如果距离过小，默认是同个圆。这里的距离设定为最小的阈值。

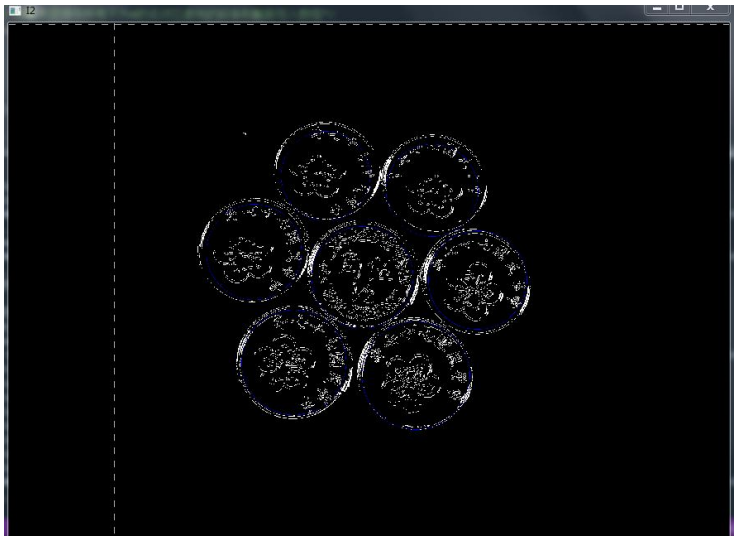
```
for (i = 0; i < center.size(); i++) {
    if (sqrt(pow((center[i].first - a), 2) + pow((center[i].second - b), 2)) < minRadius) {
        break;
    }
}
```

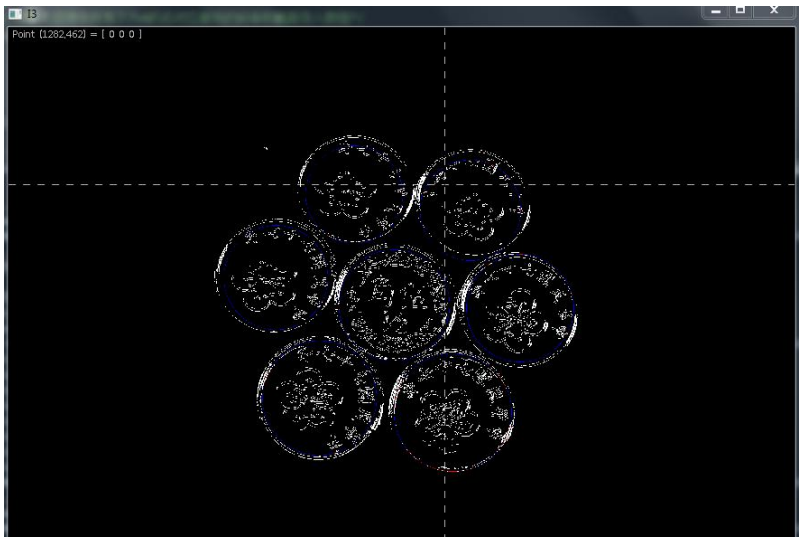
4. 画圆并标在边缘图上。

测试结果：

图 1：从上到下为 I1 到 I3。





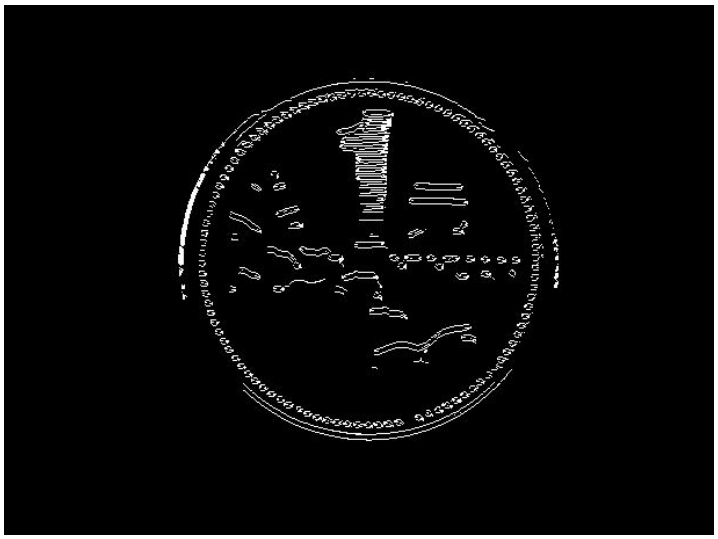


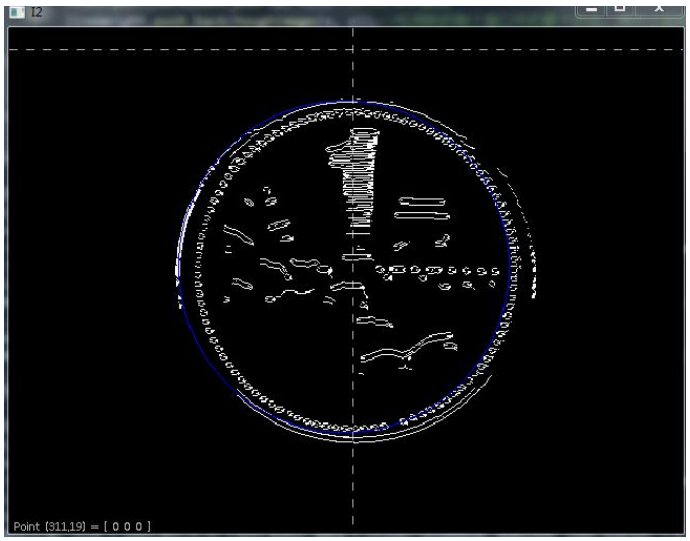
检测出的圆的数量为 7 个。半径为：

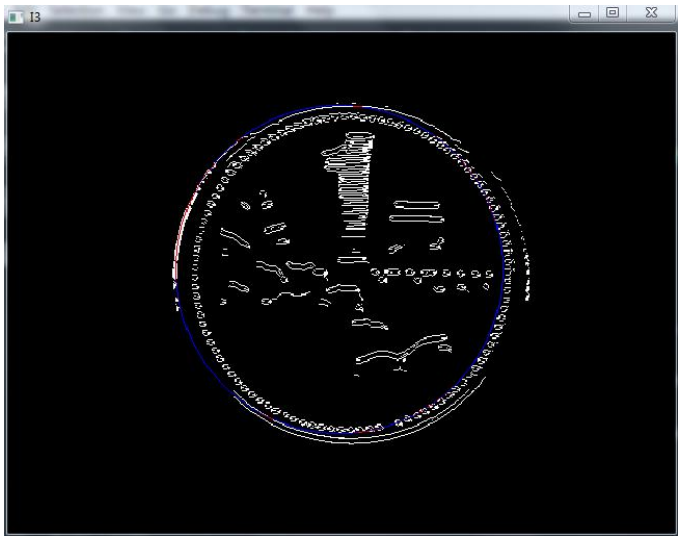
```
The radius is 173
The radius is 168
The radius is 163
The radius is 158
The radius is 143
The radius is 148
The radius is 153
```

输入的上下阈值为 143 和 178，用于检测半径在这个范围内的圆的数量。

图 2：





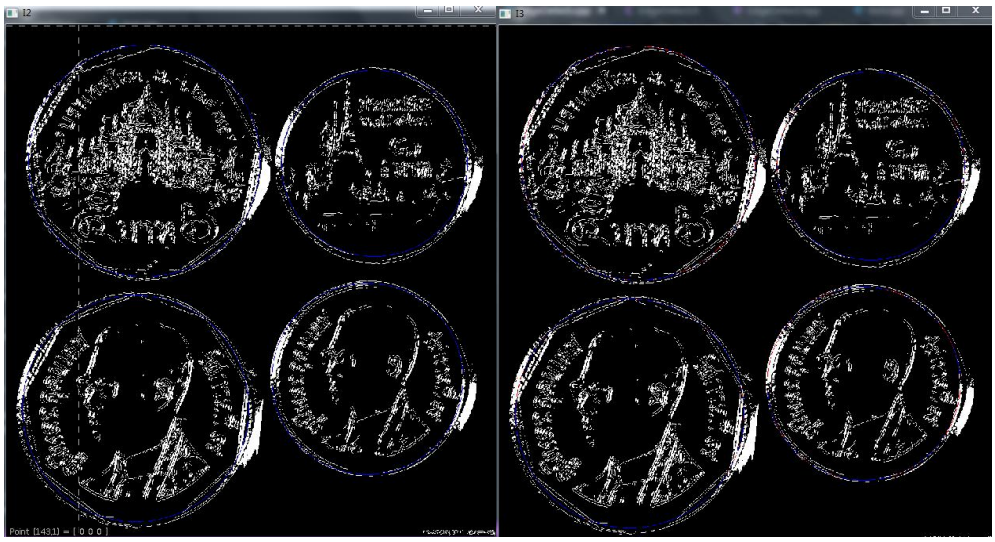


测出的圆的数量为 1，检测准确。半径为：

```
The radius is 150
```

输入上下阈值为 150 和 155，用于限制圆的半径。

图 3：

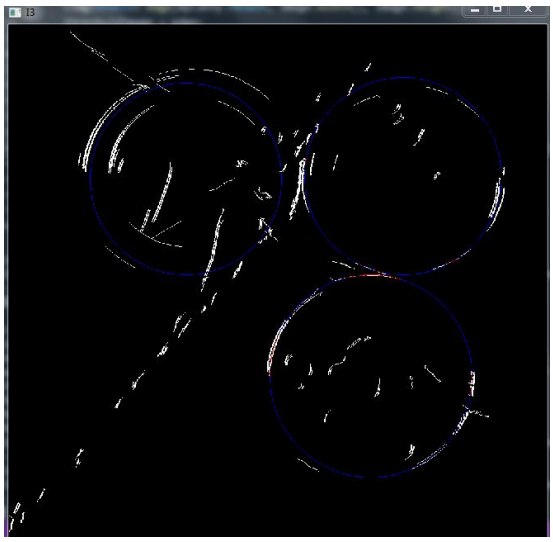


测出的圆的数量为 4，检测准确。半径为：

```
The radius is 190
The radius is 230
The radius is 185
The radius is 225
```

输入上下阈值为 180 和 240，用于限制圆的半径。

图 4：

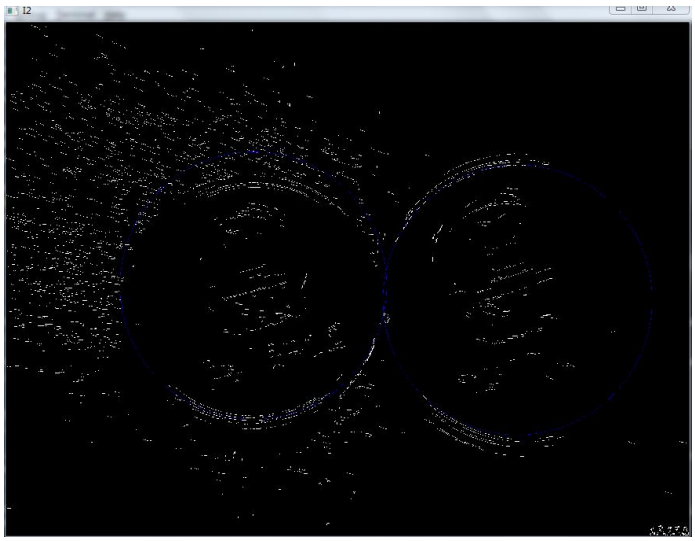


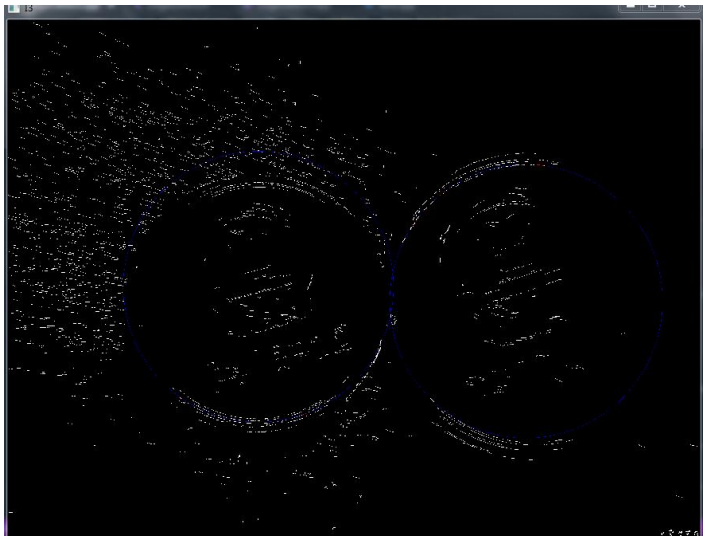
测出的圆的数量为 3，检测准确。半径为：

```
The radius is 187
The radius is 177
The radius is 182
```

输入上下阈值为 177 和 188，用于限制圆的半径。

图 5：





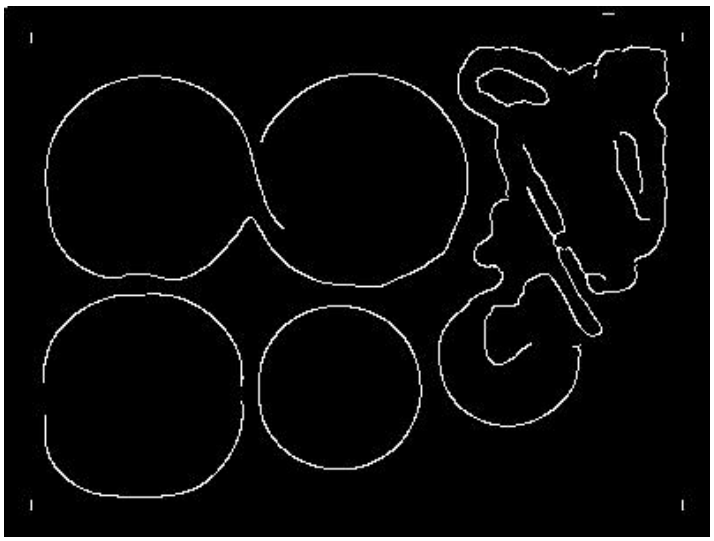
测出的圆的数量为 2，检测准确。半径为：

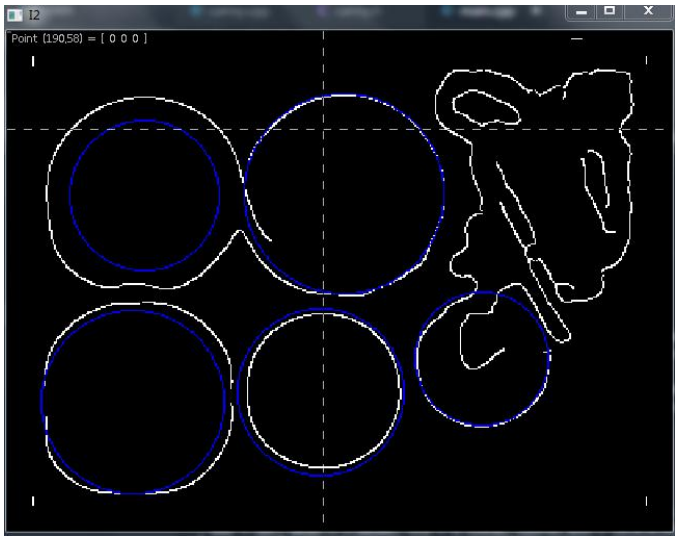
```
The radius is 510
The radius is 505
```

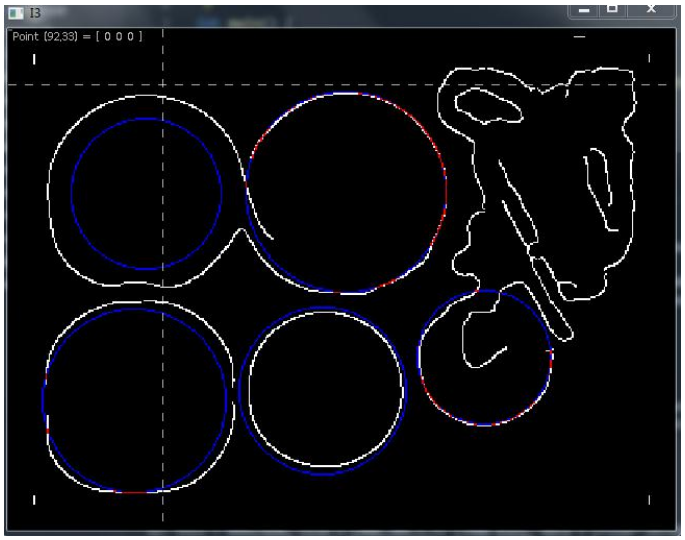
输入上下阈值为 505 和 515，用于限制圆的半径。

图 6：

由于在原来的 canny 算法无法得到可以用于检测圆的数量的边缘图，因此我使用了另一种 canny 算法。得到以下边缘图。







测出的圆的数量为 5，检测准确。半径为：

```
The radius is 60
The radius is 40
The radius is 45
The radius is 50
The radius is 55
```

输入上下阈值为 40 和 61，用于限制圆的半径。

结果分析：

我的方法可能有点投机取巧，主要思路是找到合适的半径，再为这个半径找到合适的能够匹配边缘图的圆的圆心，这样就能找到圆的位置。所以这个方法很大程度上取决于边缘图对边缘的描绘程度。对边缘描绘较好的边缘图能够很好的找到圆的位置，输出圆的个数。所以对 canny 算法进行调参和对局部最大值区域的选择决定了检测的好坏。

如果实验结果不够清晰，请移步检测圆和检测直线的文件夹中的测试结果，里面有清晰的结果图。