作业三实验报告及测试文档

数据科学与计算机学院 梁育诚 16340133

一、实验内容

1. 作业 3-1：输入图像：

普通 A4 打印纸，上面可能有手写笔记或者打印内容，但是拍照时可能角度不正。

输出：

1. 图像的边缘点（输出边缘图像I_{edge}）

2. 计算 A4 纸边缘的各直线方程，输出如下结果：（a）各个直线的参数方程；（b）在上面的边缘图I_{edge}绘制直线，用蓝色显示，得到图像I_2；（c）在I_2图上显示 A4 纸的相关边缘点，用红色点显示，得到图像I_3。

3. 输出 A4 纸的四个角点（在I_3上用半径为 5 的圆绘制角点，得到图像I_4）

思考：如何在保证精度的结果情况下加快运行速度。

2. 作业 3-2：输入图像：

图像上面有若干硬币。

输出：

1. 图像的边缘（输出边缘图像I_{edge}）

2. 把图像中边缘拟合成圆（在图像I_{edge}绘制出对应圆形，用蓝色显示，得到图像I_2），圆周相关的像素（在I_2上显示与圆形相关的像素，用红色显示，得到图像I_3）

3. 输出图像中硬币的数量。

二、实验原理

本次实验的目的是对图像进行边缘检测，主要是检测直线和圆形，用到的核心技术是霍夫变换。其基本思路可以简单描述为：使用实验二实现的 canny 算法得到图像的边缘图，对边缘图进行霍夫变换，然后统计在霍夫空间的局部极大值，再转换为直角坐标，就是我们需要检测出来的东西了。实验的关键有两个，一个是 canny 算法，由于霍夫变换是基于边缘图进行的，因此我们需要提取出一个比较好（噪声比较少）的边缘图。第二就是霍夫变换。无论是变换过程中的精度问题，还是找极大值的算法的好坏，都直接影响检测的结果。

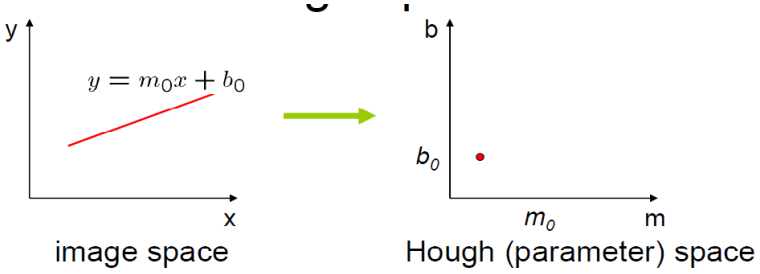
霍夫变换是一种特征检测，用于辨别图像中的物体的特征，如这次实验要求的直线和圆。通过将边缘点变换到参数空间中，进行投票来决定物体的形状，这里的决定的意思是找出参数空间中的局部极大值。

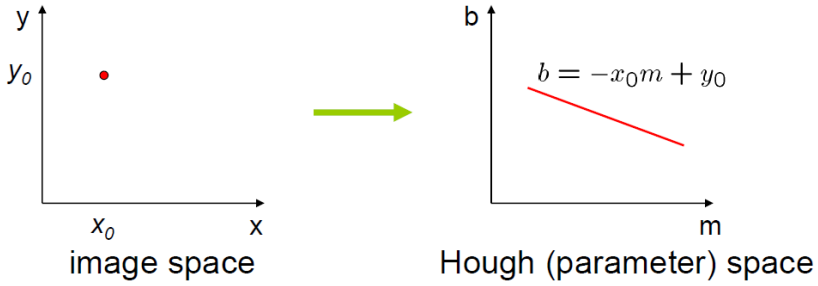
先说检测直线的霍夫变换。在平面直角坐标系中，一条直线的方程为$y = m_0x + b_0$，m_0是斜率，b_0是截距，则(m_0, b_0)是参数空间中的一点。因为考

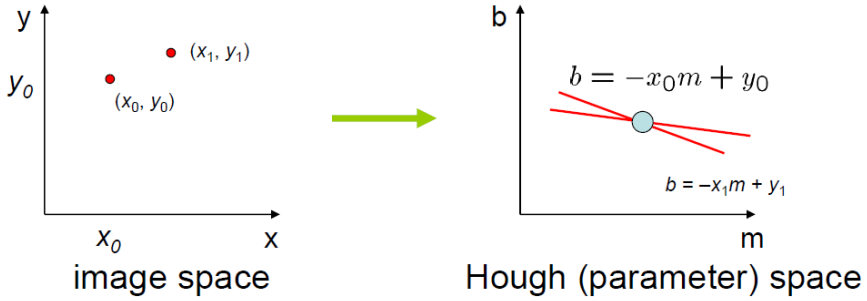
虑到直线斜率不存在的问题，所以使用极坐标可以避开这个问题。极坐标中直线表示为$r = xcos\theta + ysin\theta$。$r$是从原点到直线的距离，$\theta$是$\vec{r}$和$x$轴的夹角。直角坐标系的每一个点都对应参数空间中的一条曲线。完成转换后，参数空间中会有几个明显的亮点，这个过程称为投票，这些亮点在平面直角坐标系对应着一条直线。正是这种变换，能让我们提取到最长的直线，通过点投票的方式来找到它们，然后再输出。以上就是使用霍夫变换检测直线的基本原理了。

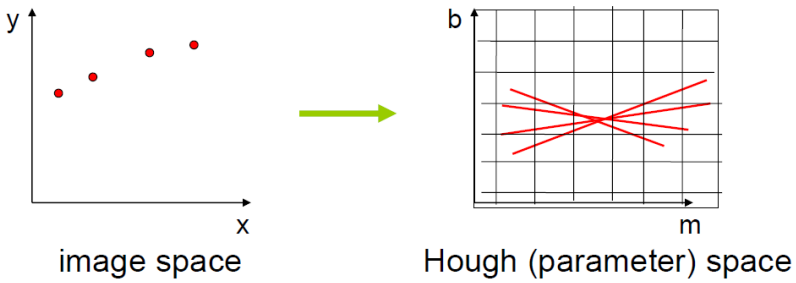
与直线类似，检测圆形的霍夫变换也是一个道理。不同之处在于此时的霍夫空间是一个三维的空间，因为在平面直角坐标系中的一个点对应着的圆有三个参数(a, b, r)。同样地使用投票的方式找出最亮的点，然后转换回在平面直角坐标系的圆即可。

直线的原理图如下（参考老师课件）：









三、实验过程

1. 检测直线

① 由平面直角坐标转换至极坐标系。通过极坐标直线方程转换，然后对在边缘图上是边缘点的点进行霍夫变换，映射到霍夫空间中的一条曲线。这里也就同时完成了转换和投票的工作了。每次转换都增加像素点的值，最后投票数多的点就是比较亮的点了。

```
// Line Hough Transformation
void canny::houghLine() {
  int maxLength;
  maxLength = sqrt(pow(rows / 2, 2) + pow(columns / 2, 2)); // transform to polar coordinate

  houghspace = CImg<float>(maxLength, 360);
  houghspace.fill(0);

  cimg_forXY(edge, x, y) {
    int value = (int)edge(x, y);
    int p;
    // Only care the edge points
    if (value == EDGE) {
      // transform to polar coordinate (p, i) => (length, angle)
      int x0 = x - rows / 2;
      int y0 = columns / 2 - y;
      // Calculate all lines gone through a point(all angles)
      for (int i = 0; i < 360; i++) {
        p = x0 * coss[i] + y0 * sins[i];
        // Voting
        if (p >= 0 && p < maxLength) {
          houghspace(p, i)++;
        }
      }
    }
  }
  houghspace.display("hough");
}
```

② 得到霍夫空间后，需要找出局部极大值。这里特别强调局部极大值，因为我一开始的时候找的是全局极大值，在某些图片中的效果还算可以，但是噪声大一点的图片就无法检测了，因此我们需要的局部极大值。这里我定义的局部是 50*50 的一个小区间。在里面寻找一个投票数最大的一个点，然后将除了这个点外的其他点设为 0，即去噪。

```
    for (int i = 0; i < wid; i += PART / 2) {
      for (int j = 0; j < hei; j += PART / 2) {
        maxVotes = getPartMax(houghspace, i, j);
        for (int x = i; (x < i + PART) && (x < wid); x++) {
          for (int y = j; (y < j + PART) && (y < hei); y++) {
            if ((int)houghspace(x, y) < maxVotes) {
              houghspace(x, y) = 0;
            }
          }
        }
      }
    }
```

```
// Find local maximum
int canny::getPartMax(CImg<float>& img, int &x, int &y) {
  int wid = (x + PART > img._width) ? img._width : x + PART;
  int hei = (y + PART > img._height) ? img._height : y + PART;
  int maxVotes = 0;
  for (int i = x; i < wid; i++) {
    for (int j = y; j < hei; j++) {
      maxVotes = ((int)img(i, j) > maxVotes) ? ((int)img(i, j)) : maxVotes;
    }
  }
  return maxVotes;
}
```

③ 在去噪后的霍夫空间找出所有的点，我们就认为这些点表示的是我们要检测的直线了。将这些点的坐标存入一个 vector 中，他们的坐标就是我们检测出来的直线的参数。同时将他们对应的投票数存入另一个 vector 中。用于后面的输出。

```
546     // save the bright points in houghspace
547     cimg_forXY(houghspace, x, y) {
548       if ((int)houghspace(x, y) != 0) {
549         lines.push_back(make_pair(x, y));
550         lineCount.push_back((int)houghspace(x, y));
551       }
552     }
```

④ 到这里我们已经完成了霍夫变换的内容了，然后就是在原图中画线了。我们根据亮度值（投票数）进行排序，对于一张 A4 纸而言，我们要的是最亮的四个点。然后根据前四个点我们利用参数将直线方程输出，并在原图中画出蓝线。特别地，对于是边缘点的，我们用红色标出。

```
568 // Draw Lines
569 void canny::drawLines() {
570   cout << "Begin drawLines" << endl;
571   int maxLength;
572   maxLength = sqrt(pow(rows / 2, 2) + pow(columns / 2, 2)); // width of houghspace
573
574   showEdge = CImg<float>(rows, columns, 1, 1, 0); // Initialize showEdge
575   afterHough = image; // Initialize output image with original image
576   sortedLineCount = lineCount; // Initialize the vector
577
578   // Sort lines by its votes, descending
579   sort(sortedLineCount.begin(), sortedLineCount.end(), greater<int>());
580
581   // Show the number of decteced size
582   cout << "Size: " << sortedLineCount.size() << endl;
583
584   // Record the parameters of the lines
585   vector<pair<int, int>> points;
586   for (int i = 0; i < 4; i++) {
587     int weight = sortedLineCount[i];
588     int indexInLines; // Index in lines vector
589     vector<int>::iterator it;
590     it = find(lineCount.begin(), lineCount.end(), weight);
591     indexInLines = it - lineCount.begin();
592     points.push_back(lines[indexInLines]);
593   }
```

```
595   // Draw lines in output image
596   cout << endl << endl;
597   for (int i = 0; i < points.size(); i++) {
598     int p = points[i].first;
599     int angle = points[i].second;
600     double k = -(coss[angle] * 1.0 / sins[angle]);
601     double b = p * 1.0 / sins[angle];
602     cout << "Line " << i << ": y = " << k << " * x + " << b << endl;
603     // dye the points that satisfy the equation with BLUE
604     cimg_forXY(showEdge, x, y) {
605       int x0 = x - rows / 2;
606       int y0 = columns / 2 - y;
607       int p_ = x0 * coss[angle] + y0 * sins[angle];
608
609       if (p == p_) {
610         showEdge(x, y) += 255.0 / 2;
611         // dye the edge points with RED
612         if ((int)edge(x, y) == EDGE) {
613           afterHough(x, y) = 255;
614         }
615         else {
616           // line is BLUE
617           afterHough(x, y, 0) = 0;
618           afterHough(x, y, 1) = 0;
619           afterHough(x, y, 2) = 255;
620         }
621       }
```

⑥ 然后对角点使用绿色的圆标出。这里上网参考了一种粗糙的算法，即检测某个像素点附近四个点，如果他们的像素值之和大于某一个值，则认为该像素点位于交点处，按照题目要求画一个半径为 5 的圆。

```
// Draw crosspoint with GREEN
void canny::drawPoints() {
  cout << "Begin drawPoints" << endl;
  unsigned char green[3] = {0, 255, 0};
  for (int x = 0; x < rows; x++) {
    for (int y = 0; y < columns; y++) {
      int area[4];
      area[0] = (int)showEdge(x, y);
      area[1] = (int)showEdge(x + 1, y);
      area[2] = (int)showEdge(x, y + 1);
      area[3] = (int)showEdge(x + 1, y + 1);
      // Enough point, indicates a crosspoint!
      if (area[0] + area[1] + area[2] + area[3] >= 255*3/2) {
        afterHough.draw_circle(x, y, 5, green);
      }
    }
  }
  cout << "End of drawPoints" << endl;
  afterHough.display("drawPoints");
}
```

2. 检测圆

① 由平面直角坐标系转换到霍夫空间。在极坐标系圆的方程表示为：$\begin{cases} x = a + r \times \cos\theta \\ y = b + r \times \sin\theta \end{cases}$，由直角坐标系中的一点$(x, y)$可以确定无数个圆，如果我们加上半径$r$，且这个$r$是控制在一个区间的，那么我们就可以生成一个三维的霍夫空间(a, b, r)。然后统计出全局最大的投票数，用于后面的过滤。

```
houghspace.assign(rows, columns, OFFSET_N);
houghspace.fill(0);
// change edge to houghspace
cimg_forXY(edge, x, y) {
  // Ignore non-edge point
  if ((int)edge(x, y) == NOEDGE) {
    continue;
  }
  for (int r = minRadius; r < maxRadius; r+=5) {
    // voting
    for (int j = 0; j < 360; j++) {
      float a = x - r * cos(j * M_PI / 180);
      float b = y - r * sin(j * M_PI / 180);
      if (a > 0 && a < rows && b > 0 && b < columns && r >= 0 && r < OFFSET_N) {
        houghspace((int)a, (int)b, r)++;
        maxVotes = max(maxVotes, (int)houghspace((int)a, (int)b, r));
      }

    }
  }
}
cout << "Voting Finish" << endl;
houghspace.display("hough");
```

② 然后我们要找到霍夫空间的局部最大值，我们需要过滤掉多余的点，这里我们的筛选规则是投票数低于最大值 70%的点不考虑。区域空间是 60*60*60，寻找局部最大值，并将该像素点坐标以及投票数存入自定义的 Point 结构中。

```
687   // Find local maximum in houghspace
688   cout << "Find Max!" << endl;
689   cimg_forXYZ(houghspace, x, y, z) {
690     float value = houghspace(x, y, z);
691     if (value < maxVotes * 0.7) {
692       continue;
693     }
694     bool isMax = true;
695     for (int ny = y - 30; ny <= y + 30; ny++) {
696       for (int nx = x - 30; nx <= x + 30; nx++) {
697           for (int nz = z - 30; nz <= z + 30; nz++) {
698             if (nx >= 0 && nx < rows && ny >= 0 && ny <= columns && nz >= 0 && nz < OFFSET_N) {
699               isMax = isMax && (houghspace(nx,ny, nz) <= value);
700             }
701           }
702       }
703     }
704     if (isMax) {
705       circles.push_back(Point(x, y, z, (int)houghspace(x, y, z)));
706     }
707   }
```

③ 根据投票数进行排序，数组的大小就是检测出来圆的数量，然后在图上用蓝色画出对应的圆。特别地，对于吻合的边缘点，使用红色标出。

```
708   cout << "Begin Sort" << endl;
709   // Sort circles by votes, descending
710   sort(circles.rbegin(), circles.rend());
711   cout << "Circle Number: " << circles.size() << endl;
712   cout << "Finish Sort" << endl;
713   circleNumber = circles.size();
714
715   // draw circles in BLUE
716   for (int i = 0; i < circleNumber; i++) {
717     float a = circles[i].x;
718     float b = circles[i].y;
719     float r = circles[i].z;
720     unsigned char blue[3] = {0, 0, 255};
721     afterHough.draw_circle(a, b, r, blue, 1, 1);
722     afterHough.draw_circle(a, b, r, blue, 0, 1);
723   }
724
725   // dye edges to RED
726   cimg_forXY(afterHough, x, y) {
727     if (((int)edge(x, y) == EDGE) && ((int)afterHough(x, y, 2) == 255)) {
728       afterHough(x, y, 0) = 255;
729       afterHough(x, y, 1) = 0;
730       afterHough(x, y, 2) = 0;
731     }
732   }
733   cout << "End of Draw!" << endl;
734   afterHough.display();
```

四、测试及分析

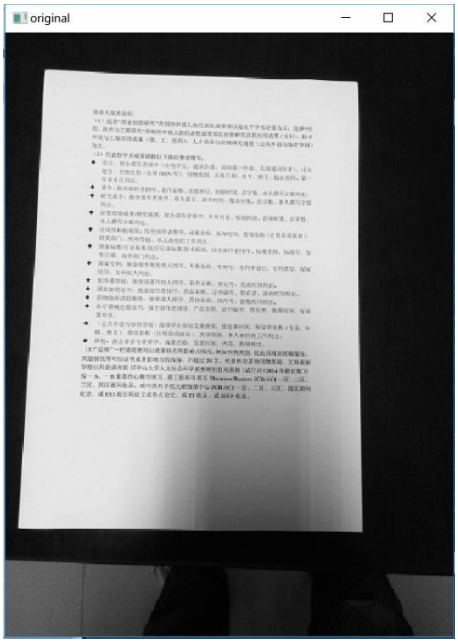
测试过程说明：对于不同的图片，给出的是我经过多次测试后效果比较好的参数。输出的图片包括：原图、边缘图、霍夫空间图、原图（含标注）共四张图。基本能够清晰地说明检测效果。参数输入为：sigma, tlow, thigh, minRadius, maxRadius（如果检测的是直线，则 minRadius 和 maxRadius 均为 0）。

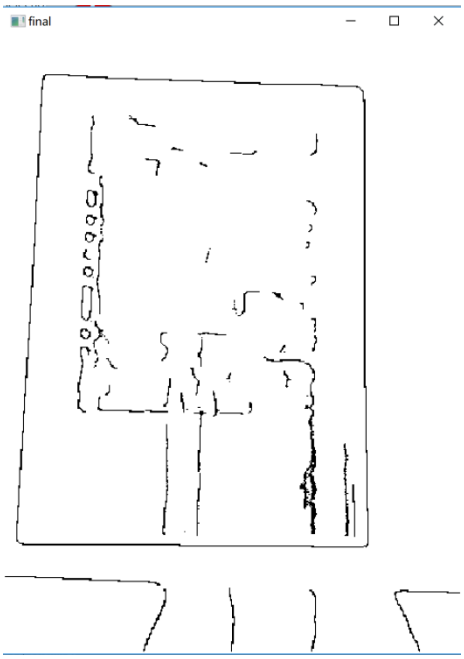
1. 检测直线

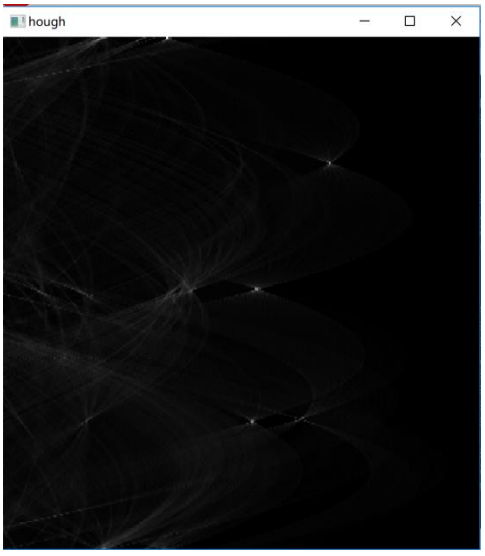
① 1.bmp

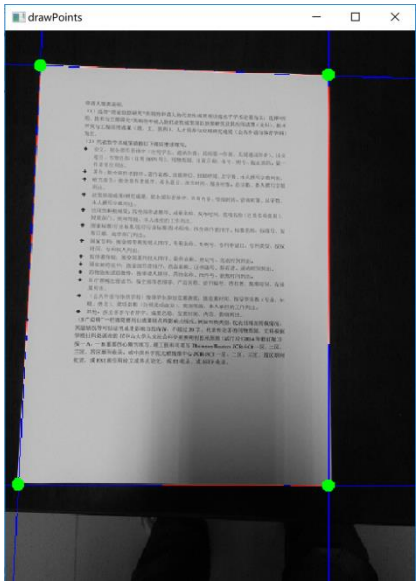
参数：4 0.4 0.9 0 0

测试效果：









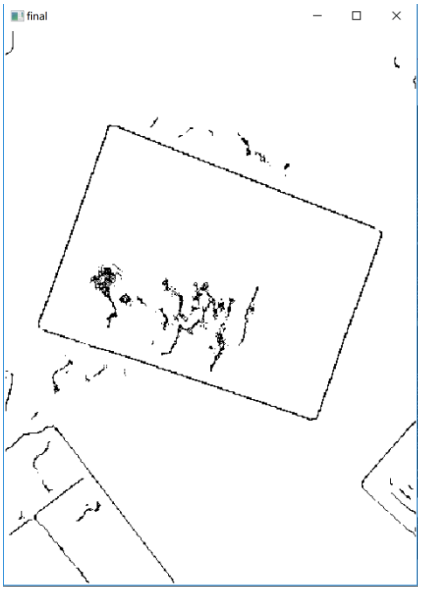

```
Line 0: x = 115
Line 1: y = -0.0349208 * x + 229.14
Line 2: y = 19.0811 * x + 3401.1
Line 3: y = -1.83691e-016 * x + -175
```

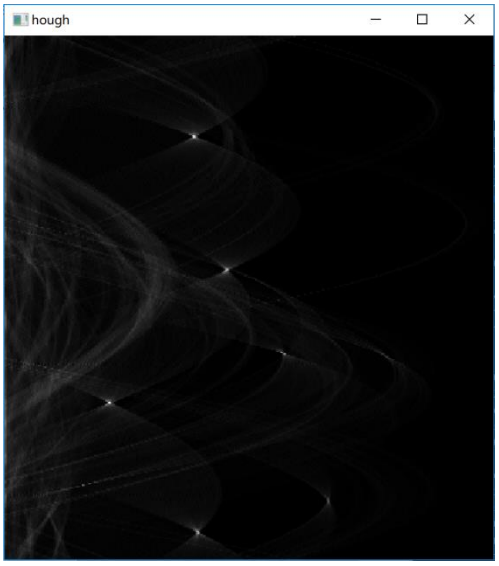
分析：这一张图的检测难点在于下方的桌面边缘的噪声。由于桌子边缘也是一条很明显的直线，因此会对检测造成干扰。因此需要在高斯模糊的时候调大 sigma，使得下方桌子边缘能够被模糊掉，从而凸显出 A4 纸的边缘。其他方面影响不大，比较好检测，霍夫空间中的亮点数也是很明显的有四个。

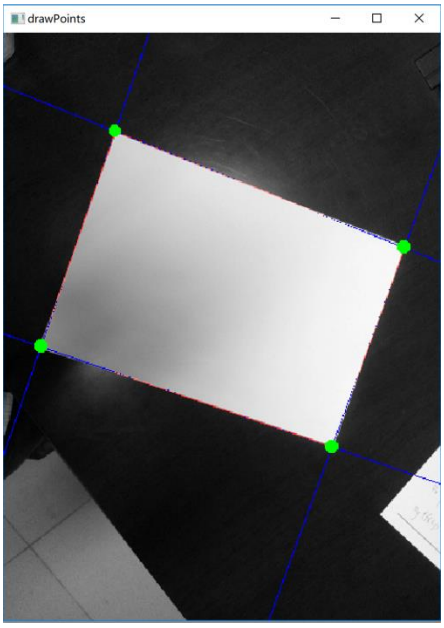
② 2.bmp

参数：4 0.4 0.9 0 0

测试效果：








```
Line 0: y = -0.344328 * x + -75.0911
Line 1: y = -0.404026 * x + 139.131
Line 2: y = 2.90421 * x + 463.805
Line 3: y = 2.74748 * x + -383.018
```

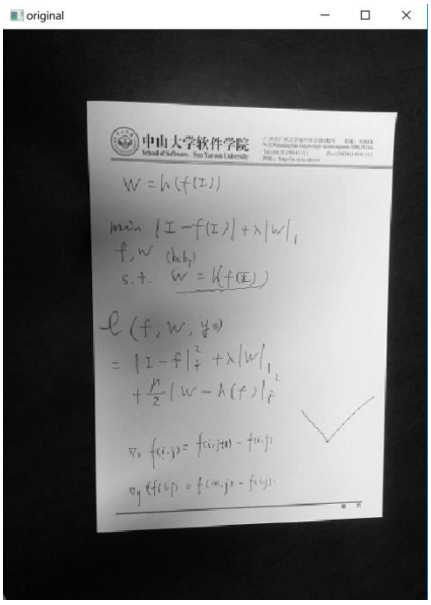
分析：这幅图的难点在于有其他直线的干扰。左下方和右下方都有清晰的直线，尤其是左下方的直线与 A4 纸的高的长度非常接近，这对检测来说干扰很大。采取的办法主要是使用 sigma 较高的高斯模糊，尽可能地将左下角比较暗的地方模糊掉，消除直线。然后在寻找亮点的时候寻找局部极大值，这幅图的霍夫空间亮点的数量明显要比上一幅图要多。因为 A4 纸的四条边映射到霍夫空间的四个点会比较分散，主要的干扰是在

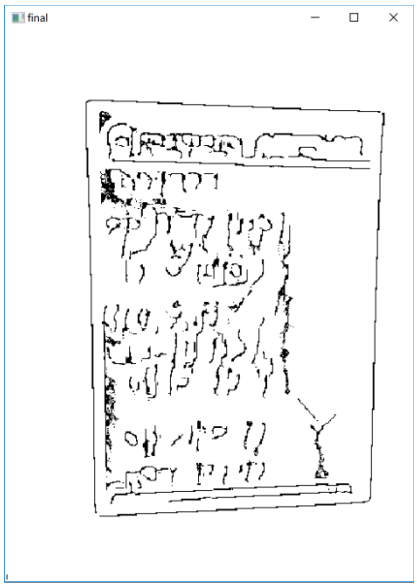
这四个亮点附近的其他点。使用全局极大值有可能会将干扰点误认为是 A4 纸的边缘。

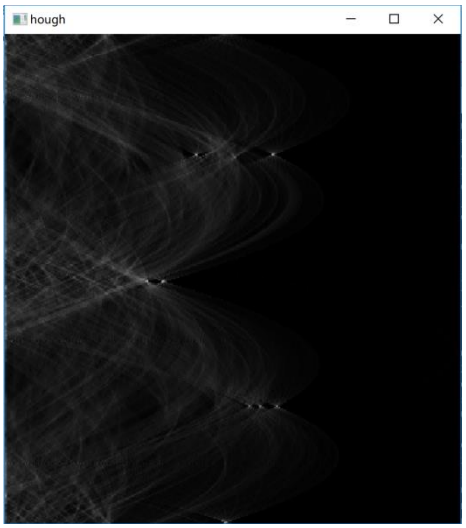
③ 3.bmp

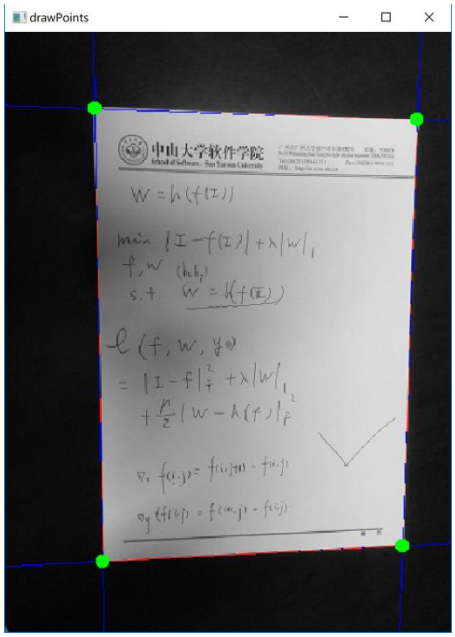
参数：4 0.3 0.8 0 0

测试效果：










```
Line 0: y = 0.0524078 * x + -199.273
Line 1: y = 28.6363 * x + -4641.9
Line 2: y = -57.29 * x + -6646.65
Line 3: y = -0.0349208 * x + 196.119
```

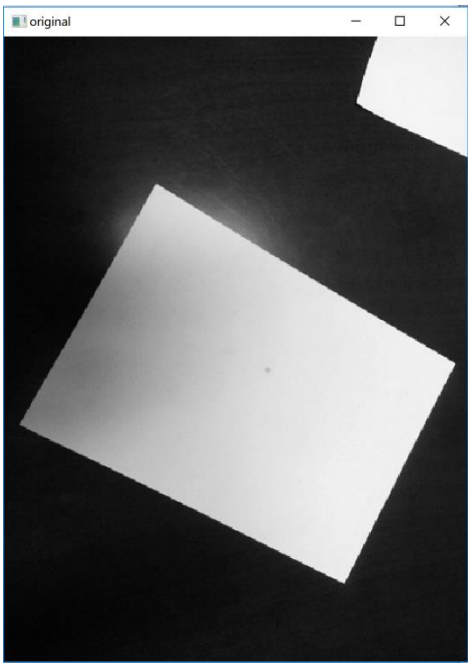
分析：这张图在 A4 纸外部没有什么噪声，光线上也没什影响，最大的干扰来自纸张上的干扰。纸的上下方各有一条与边几乎等宽的水平线，这种

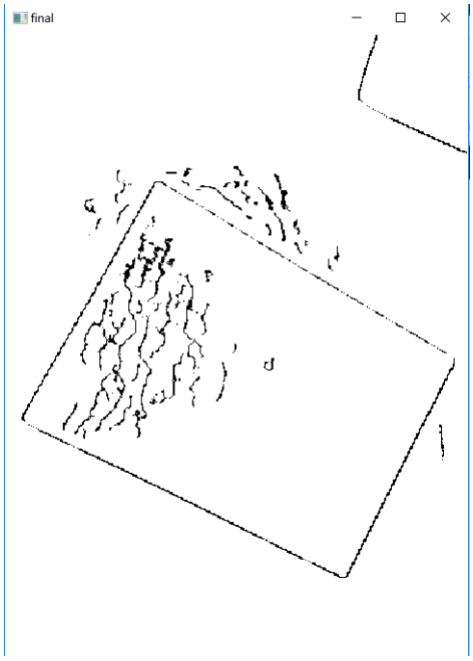
干扰是比较严重的，从霍夫空间中看到的亮点数也是较多，但只要我们寻找局部极大值的算法写得好，这种干扰将不会影响实验效果。

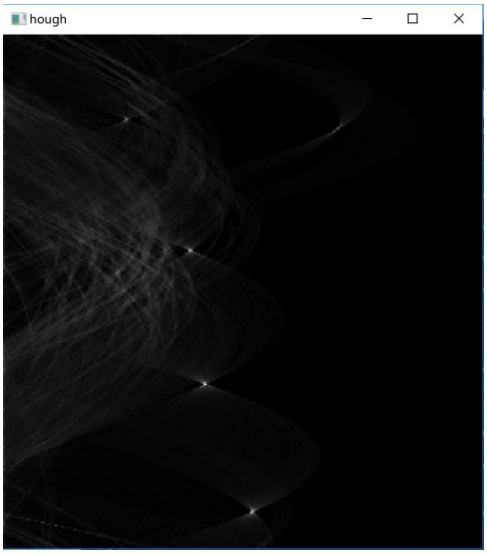
④ 4.bmp

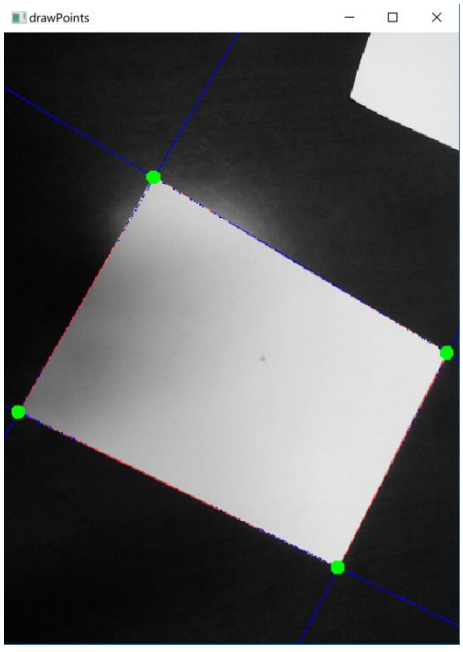
参数：3 0.4 0.9 0 0

测试效果：










```
Line 0: y = -0.487733 * x + -155.764
Line 1: y = 1.96261 * x + -383.268
Line 2: y = 1.73205 * x + 260
Line 3: y = -0.600861 * x + 100.33
```

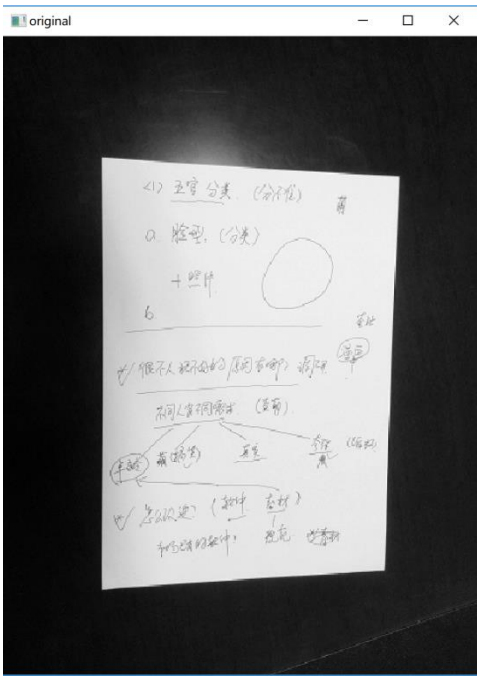
分析：这幅图我认为在这组测试数据中比较难的一幅图。因为 A4 纸的左

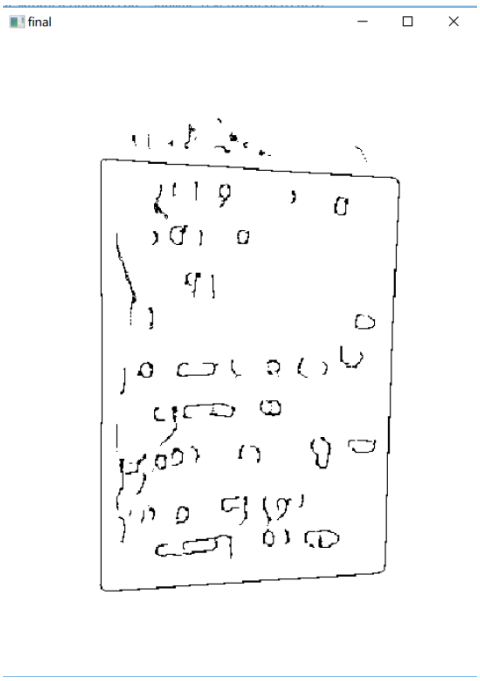
上角的部分光线特别亮，高斯模糊后的 nms 很容易将这些边消掉。而右上方的直线则非常清晰，因此测试的时候经常都是能检测出三条边，然后就被右上方的直线干扰了，由霍夫空间上看就是最上方的两个点的亮度非常接近，第二个亮点附近有干扰。解决方案是使用较低的 sigma 进行模糊，尽量保留 A4 纸的边缘，用高阈值来过滤，只要保证 A4 纸的边足够完成（有足够多的点），我们就能够检测出来。

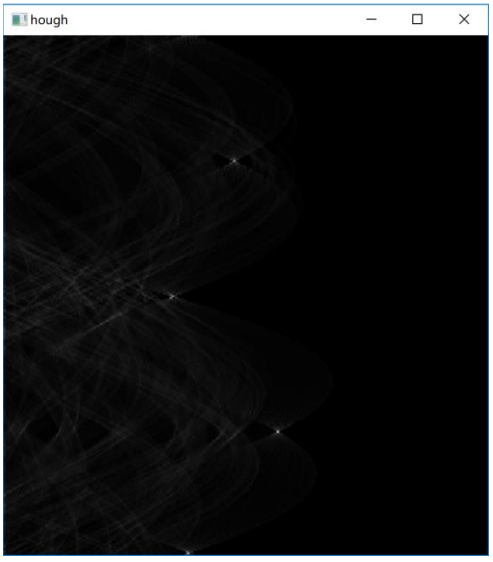
⑤ 5.bmp

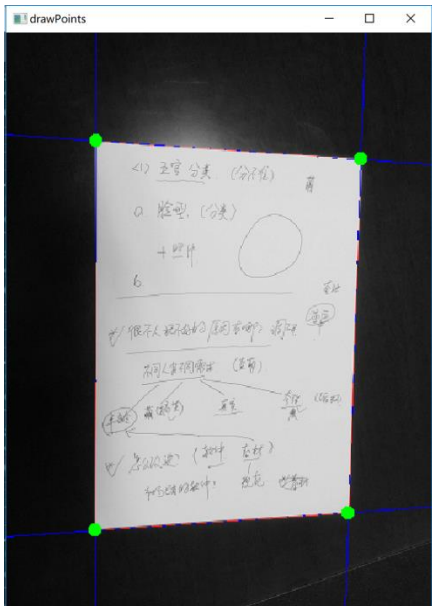
参数：4 0.4 0.9 0 0

测试效果：









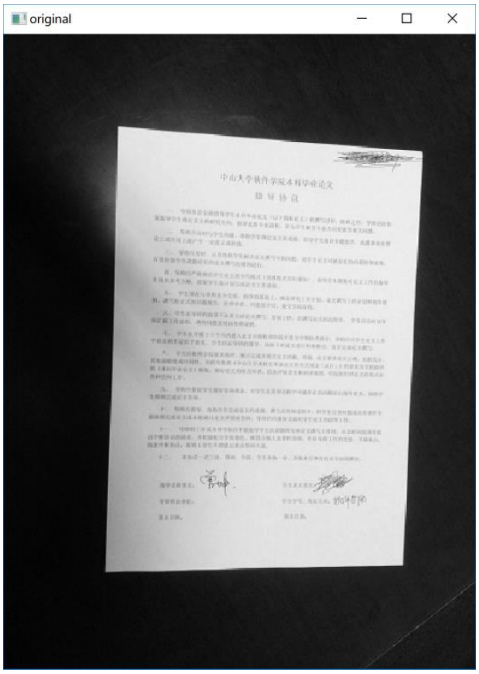
```
Line 0: y = 8.16589e+015 * x + 9.47243e+017
Line 1: y = 28.6363 * x + -3639.02
Line 2: y = 0.0699268 * x + -189.462
Line 3: y = -0.0699268 * x + 159.388
```

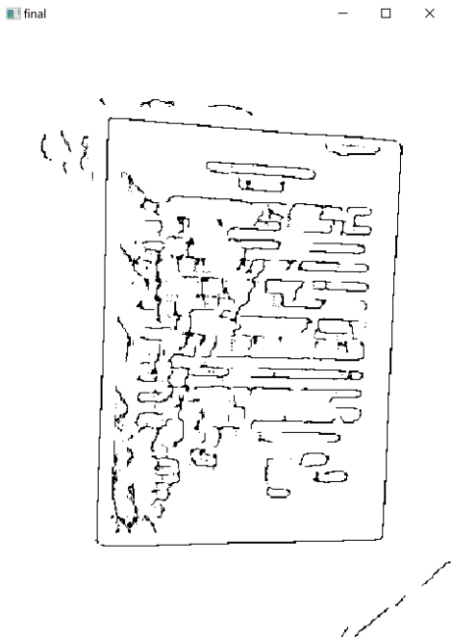
分析：这张图是比较好检测的，A4 纸边缘清晰，没有受光线的影响，同时干扰比较少。在霍夫空间中能看到四个明显的亮点，检测难度较低。

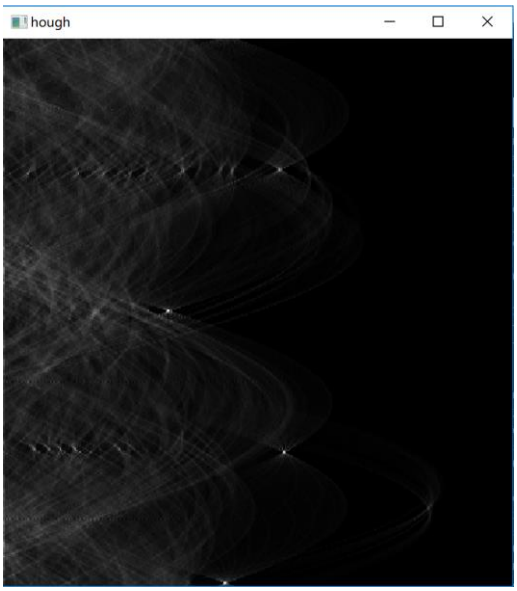
⑥ 6.bmp

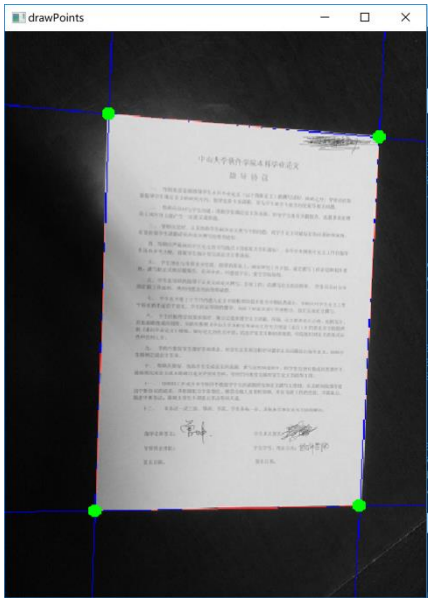
参数：5 0.3 0.8 0 0

测试效果：









```
Line 0: y = 28.6363 * x + 3094.6
Line 1: y = 19.0811 * x + -2770.56
Line 2: y = 0.0174551 * x + -184.028
Line 3: y = -0.0874887 * x + 181.691
```

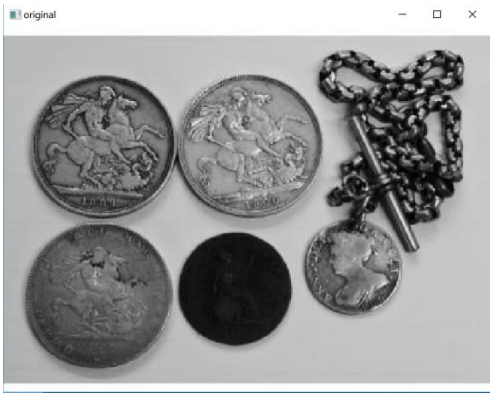
分析：这幅图检测的难点在于 A4 纸左上方有光线的干扰，以及内部文字的干扰。为了见到内部文字的干扰我选择了 sigma 较大的高斯模糊，然后调宽阈值的范围，接受尽可能多的点以保证 A4 纸边缘的完整性。由霍夫空间可以看出，尽管第一第二个两点附近的干扰很多，但四个亮点还算明显。

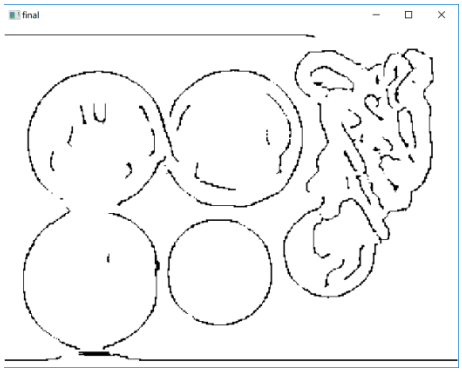
2. 检测圆

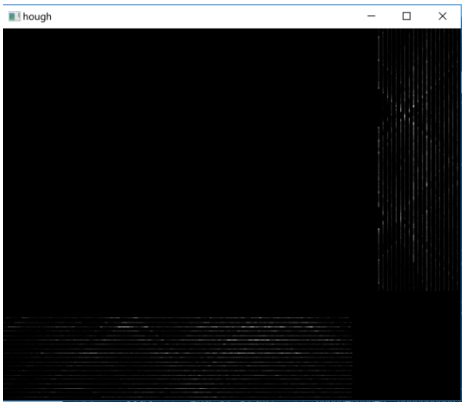
① 1.bmp

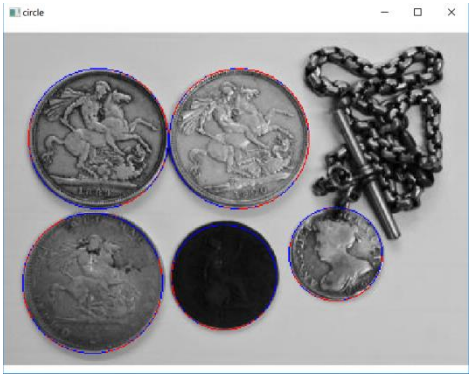
参数：5 0.3 0.7 30 200

测试效果：









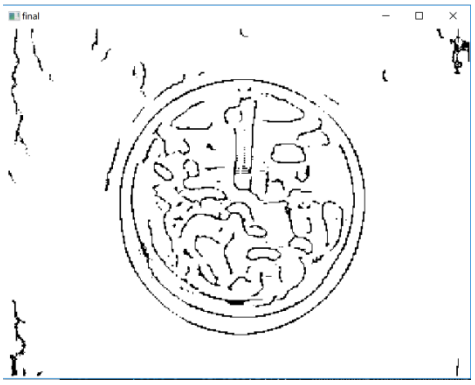

```
Circle Number: 5
```

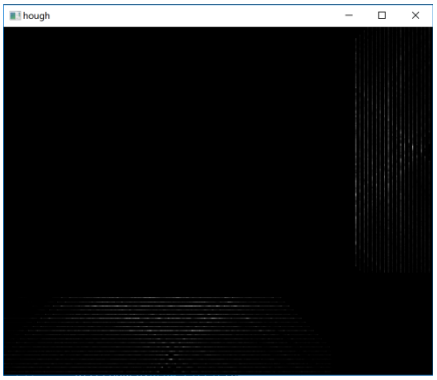
分析：这幅图检测的难度在于右上方有很大的干扰，因此需要较大的高斯模糊将链子部分模糊掉，然后调宽阈值区间，接受尽可能完整的硬币边缘。

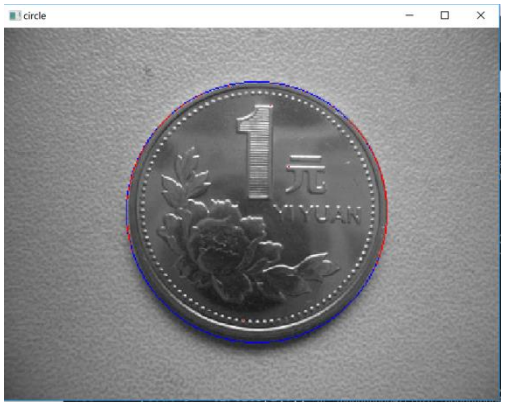
② 2.bmp

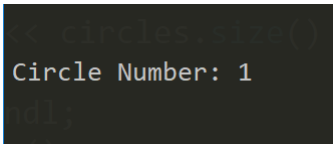
参数：4 0.3 0.7 30 200

测试效果：







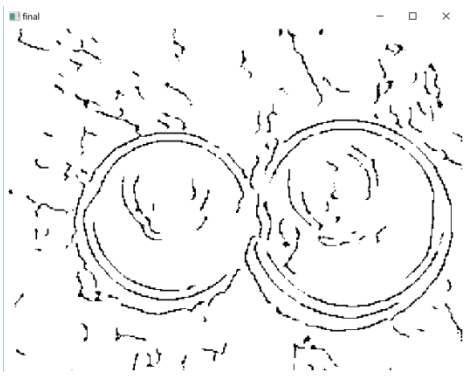


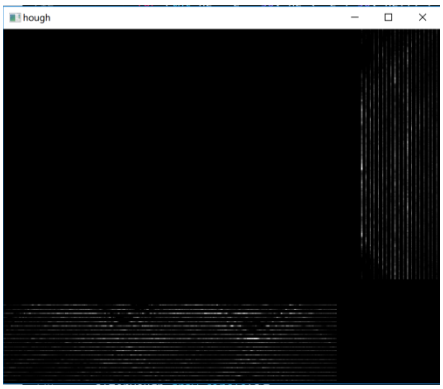
分析：这张图是最容易检测出来的，因为其干扰很少，边缘清晰。唯一的干扰是硬币边缘内白色线的干扰，通过高斯模糊就可以解决。

③ 3.bmp

参数：3 0.4 0.8 30 200

测试效果：





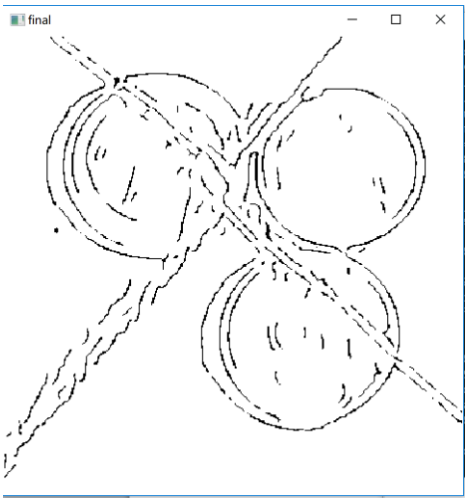

```
Circle Number: 2
```

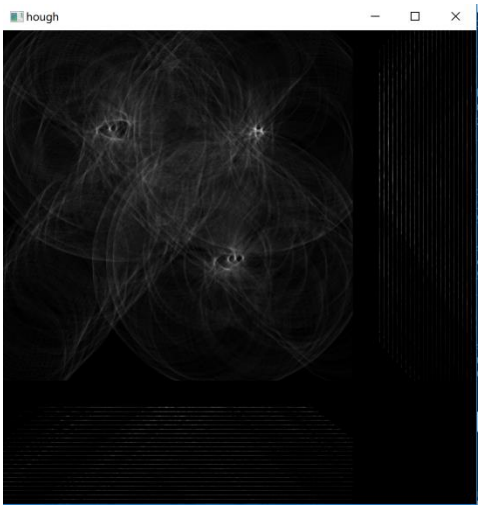
分析：这幅图的检测有一定难度，干扰有两个：一是左边的硬币与底色很接近，高斯模糊的 sigma 过大很容易丢失硬币的边缘；二是右边硬币里面国徽的边缘很容易被检测到。解决方法是使用较低的 sigma，然后调高 thigh 以消除国徽的边缘。

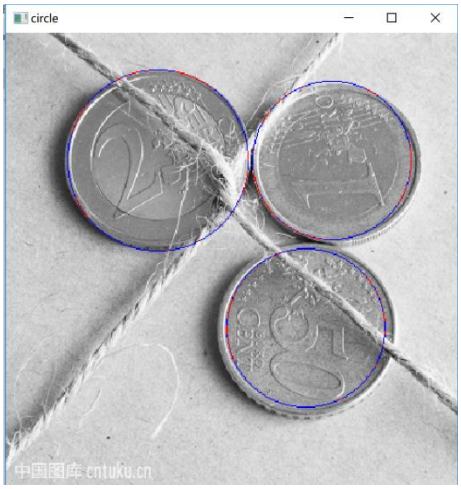
④ 4.bmp

参数：3 0.7 0.8 30 200

测试效果：







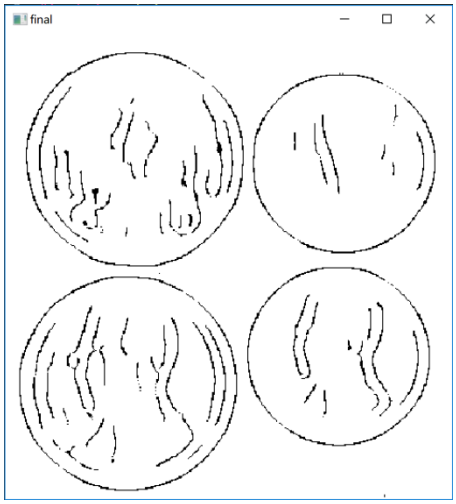

```
Circle Number: 3
```

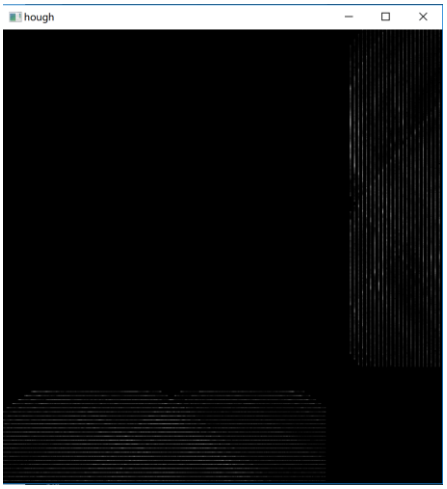
分析：这幅图看上去比较复杂，但是检测效果却很不错。因为硬币的边缘都比较地清晰，硬币内部也没什么干扰。但是为了排除绳子的干扰，这里用了很大的 tlow。

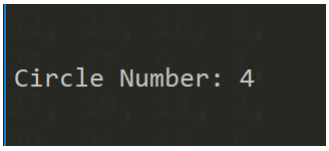
⑤ 5.bmp

参数：4.5 0.5 0.7 30 200

测试效果：





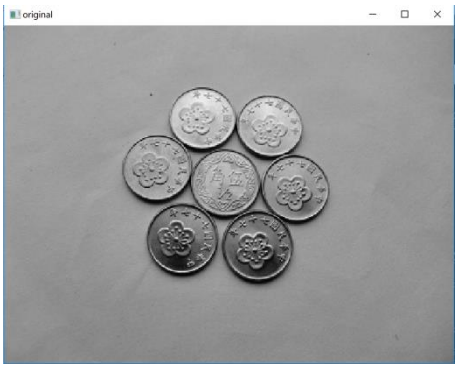


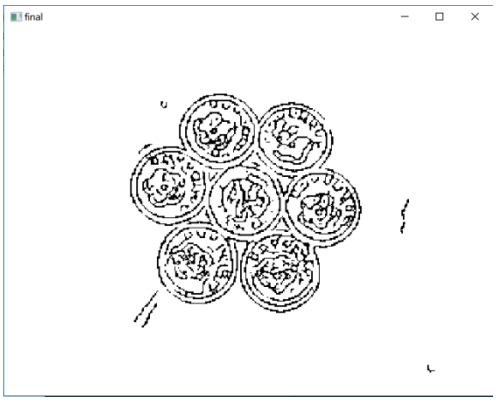
分析：这幅图的检测也是比较简单的。硬币的边缘很清晰，相互间基本没有干扰。硬币里面的人头可能会有一些影响，稍微将 tlow 调大就可以消除这种噪声。

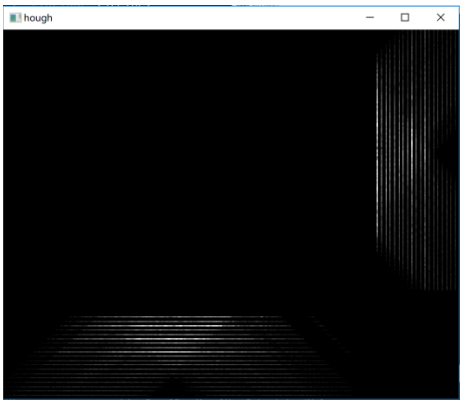
⑥ 6.bmp

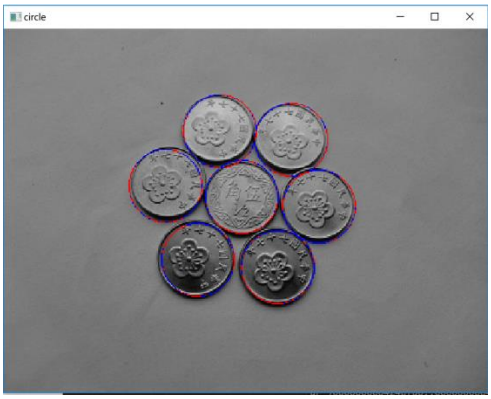
参数：2 0.2 0.8 30 200

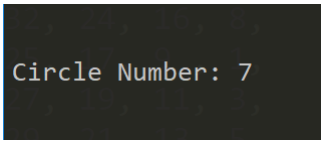
测试效果：











分析：这幅图检测的难度在于硬币的数量以及光线的影响。位于上方的两个硬币光线较亮，如果阈值 tlow 过高则很容易消掉边缘。又因为硬币之间靠得很近，因此如果使用 sigma 较高的高斯模糊，检测出来的硬币数很可能会少。所以我用的是一个较低的 sigma 和较低的 tlow。

五、难点、思考、总结与体会

难点：1. 解决直线斜率不存在的问题。一开始的想法是直接使用平面直角坐标系，但是很快就遇到了斜率不存在的问题。上网搜了一下解决方案，发现使用极坐标系可以避开这个问题，于是直线和圆都使用极坐标的霍夫空间了。

2. 找极值点。一开始找投票值最高的点是用全局搜索，对于某几张图，这种算法还可以，但是对其他的测试数据，则一直无法跑出正确的结果。于是我想这是不是找极值的问题。考虑到霍夫空间的分布情况，如果使用全局搜索，确实有可能接纳了噪声而抛弃了我们真正想要的点，因此我就改为使用局部搜索。那么局部搜索的范围又是多少？这个需要多次测试来得出。以 A4 纸的检测为例，霍夫空间的大小大概是 300*300，我们检测 A4 纸的直线是 4 条，平均每个极值点是 150*150 区域的一个局部最大值，因此区域的上限不应该超过这个值。至于下限，只要在计算机计算速度允许的范围内，都可以选择。这也回答了思考题，如果要提高计算速度，可以适当调高这个区域的值，但不能超过上述讨论的上限，否则有可能影响到精度。

3. 排序。无论是检测直线还是检测圆，我们都需要找到最亮的点，即投票数最高的点，这里需要用到排序，我都使用了 C 提供的 sort 函数。方法有两种，我分别在直线和圆都使用了。第一种方法是分开存储，使用一个数组存坐标，一个数组存投票数，利用下标来将他们关联在一起；第二种方法是自定义一个结构，将该点的坐标和投票数一起存放，然后重载比较运算符来进行排序，两种方法都可以。

总结：这次实验是基于上一次作业的 canny 算法实现的，因此上一次实验的结果也很重要。本次实验的重点在于霍夫变换以及找局部极大值。只要找到局部极大值，再排序，就能得到结果了。通过这次实验，我对 canny 算法的三个参数的理解更加深入，能够根据不同特征的图片来调整参数，使得检测的效果更佳。同时我也掌握了霍夫变换的思想，能够运用霍夫变换来检测具有解析方程的图像特征。整体来说比上次实验要顺利一些，对 CImg 的使用更加熟悉，但还是遇到了不少问题。虽然过程很辛苦，但做出结果后还是很开心，希望在日后的实验能够学到更多的东西！