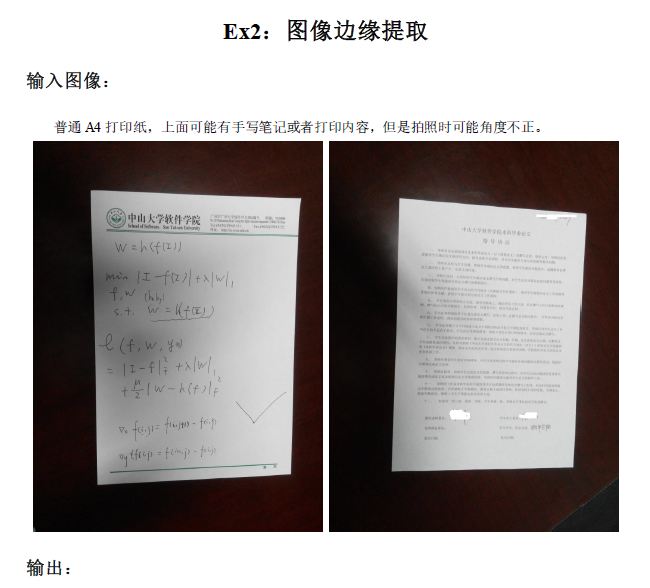
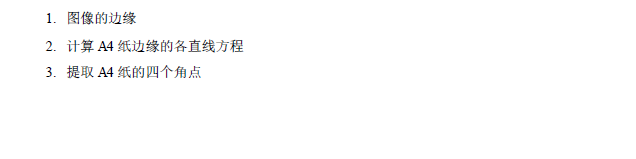
**测试报告之Ex2:图像边缘提取**



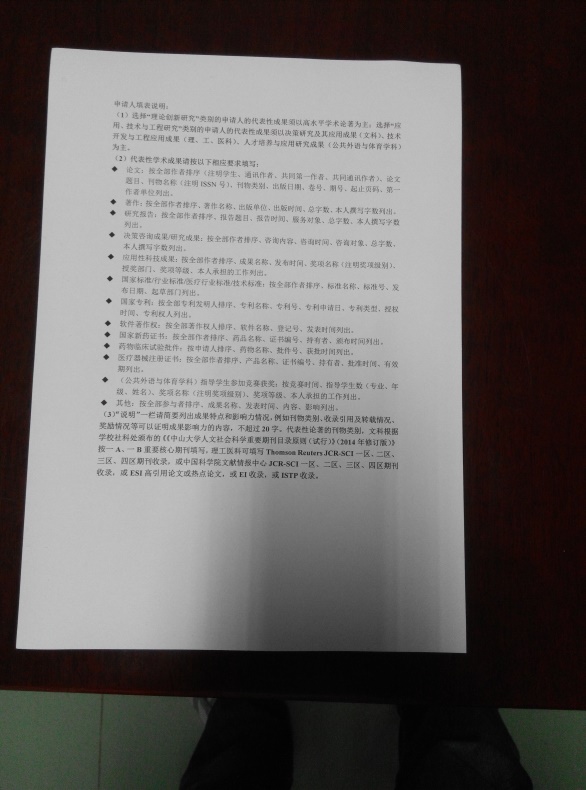
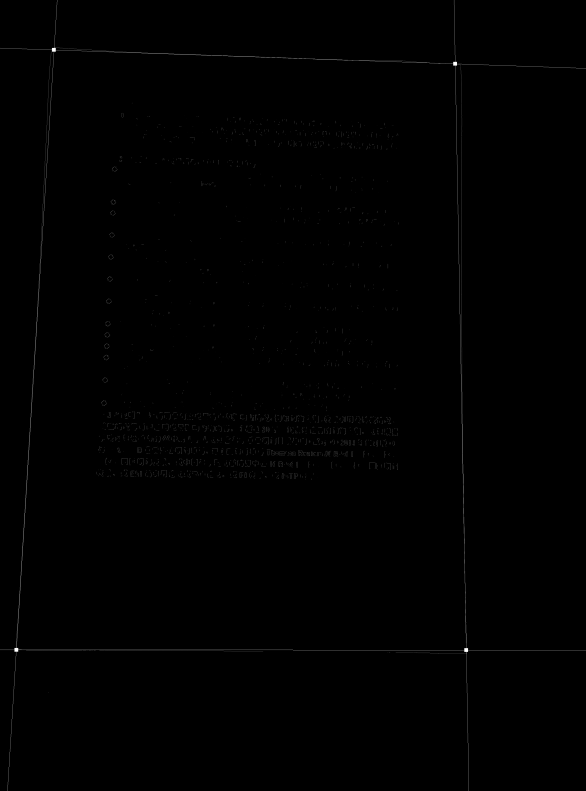


测试环境：visual Studio 12

测试数据：提供的DataSet

测试结果：

原图1： 结果图：

检测到的角点分别为(287,265)，(2423,339)，(87,3457)，(2482,3458)；

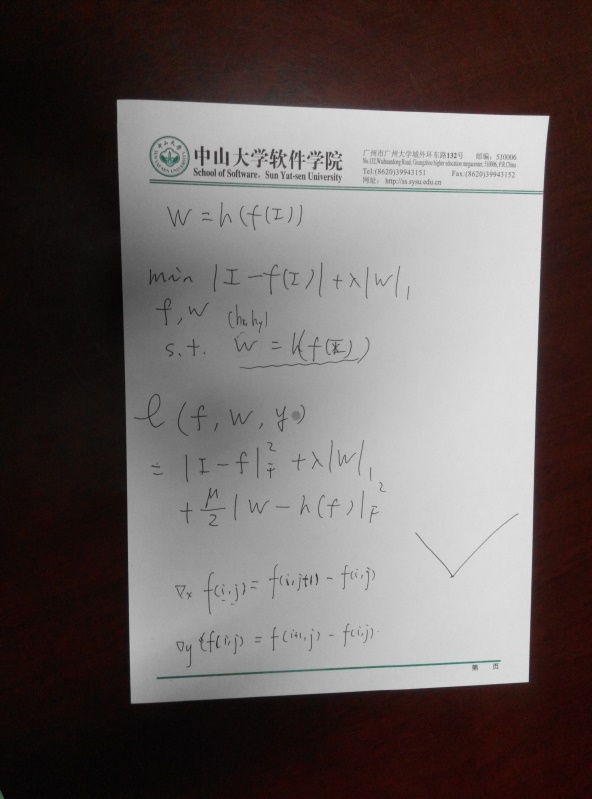
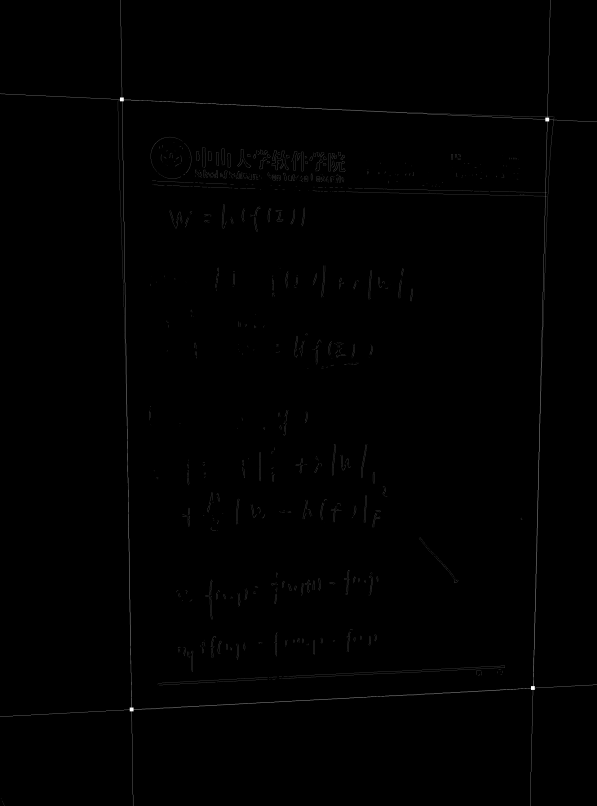
直线方程为：0.0629147x – 22073.4 = y;

-28.9257x – 1795.07 = y;

-0.0188518x – 47590.1 = y;

-3.73205e+007x + 1354 = y;

原图2： 结果图：

检测到的角点分别为(688,3704)，(2785,3592)，(637,519)，(2960,624)；

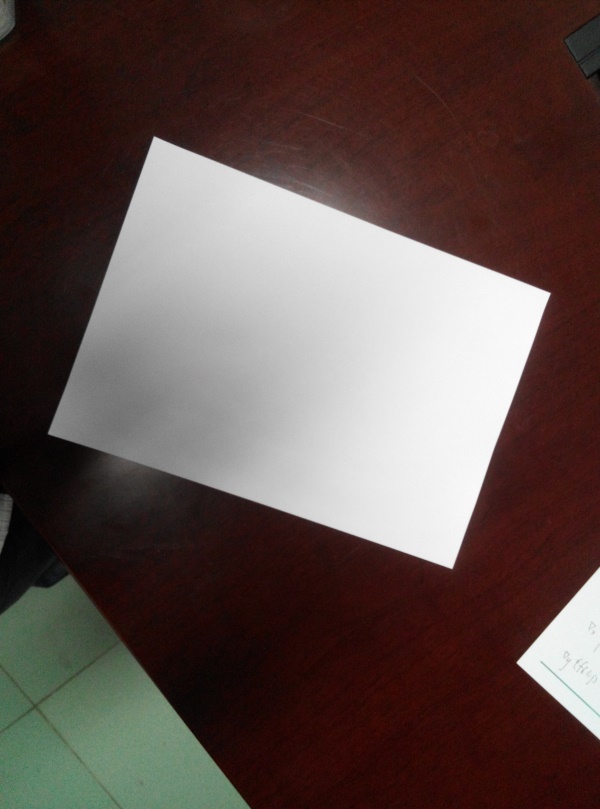
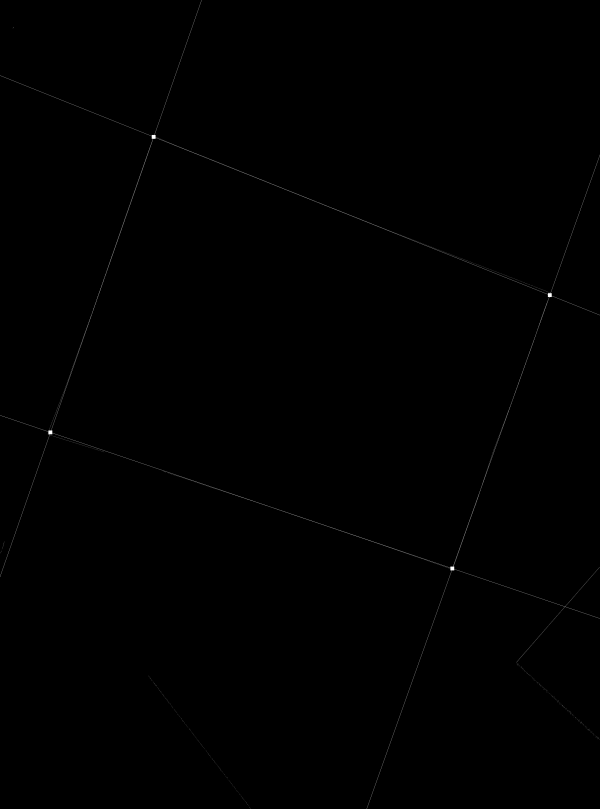
直线方程为：-0.0157093x + 57170.8 = y;

18.7063x + 1554.22 = y;

0.0251381x + 50258.4 = y;

-21.2049x - 1541.71 = y;

原图3： 结果图：

检测到的角点分别为(799,712)，(262,2249)，(2859,1535)，(2352,2957)；

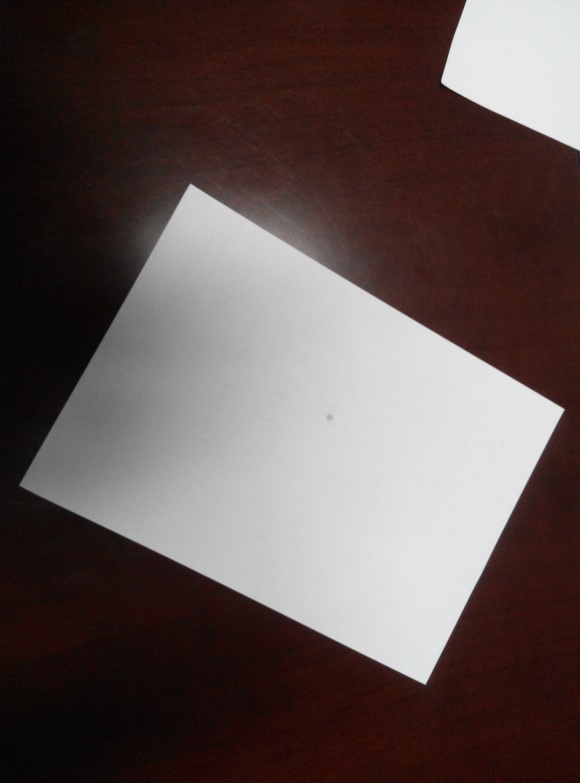
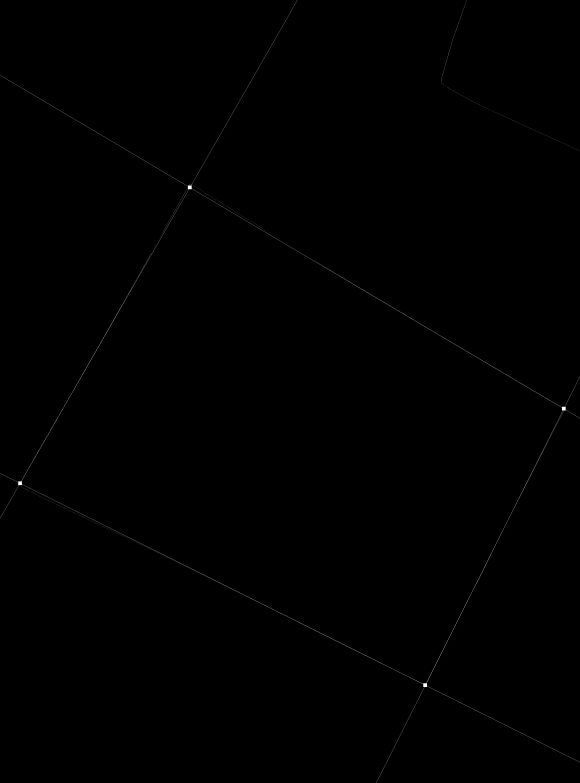
直线方程为：0.349412x - 3571.26 = y;

-2.50271x - 1088 = y;

-2.95098x + 584.945 = y;

0.356477x - 3076.42 = y;

原图4： 结果图:

检测到的角点分别为(108,2597)，(2287,3682)，(1021,1007)，(3033,2196)；

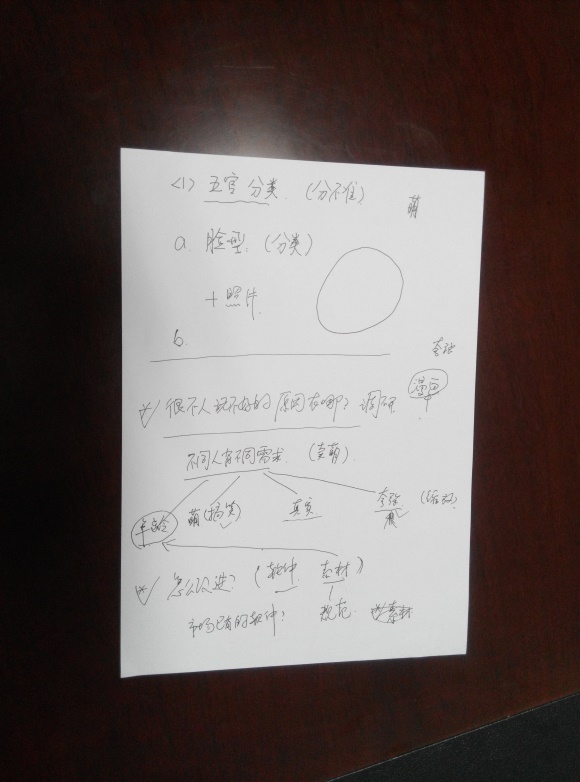
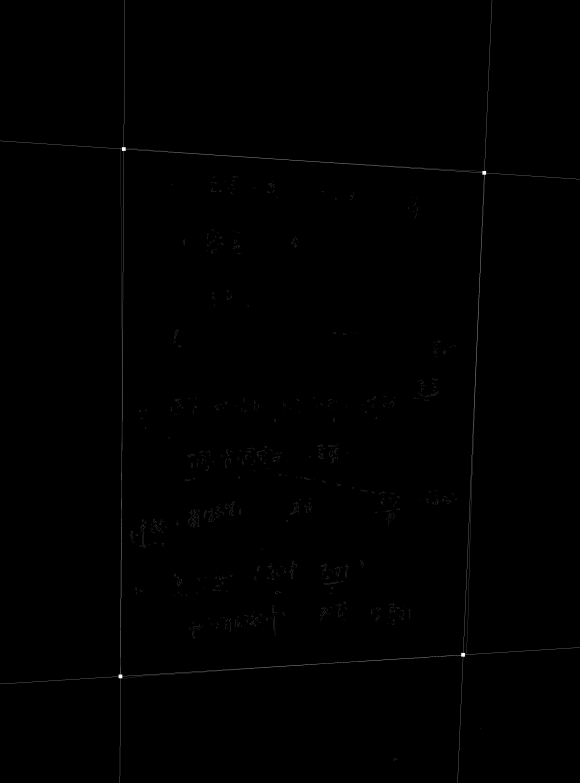
直线方程为：0.501636x + 3028.66 = y;

0.574561x – 2035.39 = y;

-2.0092x + 1216.43 = y;

-1.69091x – 778.398 = y;

原图5： 结果图：

检测到的角点分别为(648,3635)，(2491,3519)，(666,801)，(2605,929)；

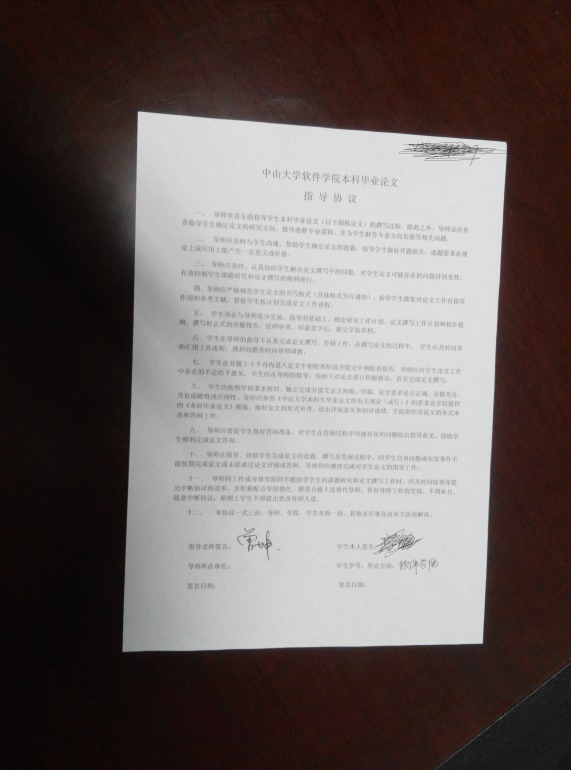
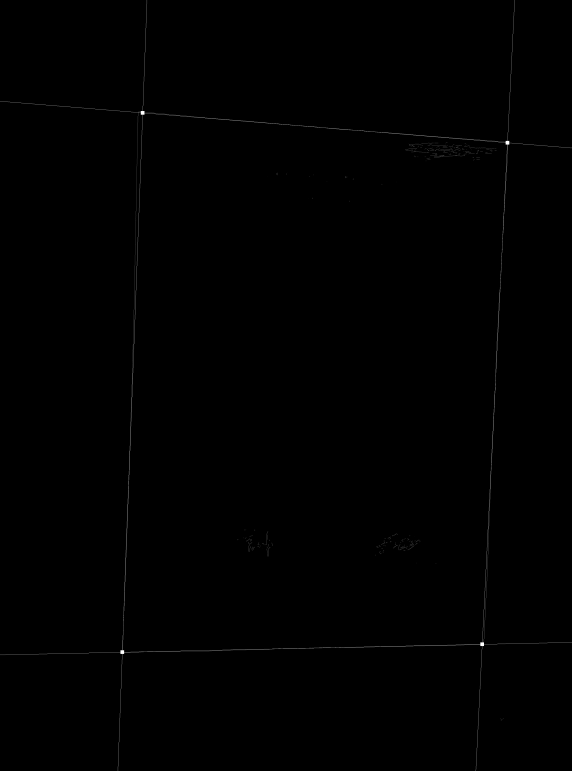
直线方程为：0.00628332x - 143717 = y;

15.8946x + 1473.91 = y;

0.0440107x + 22584.5 = y;

-15.1356x – 1244.71 = y;

原图6： 结果图：

检测到的角点分别为(778,616)，(2768,779)，(667,3559)，(2630,3516)；

直线方程为：0.037717x – 22233.9 = y;

-12.2154x – 1424.75 = y;

0.0503079x + 22689.1 = y;

45.4656x – 1436.35 = y;

测试分析：

由上面的结果图与原图比较，发现找到的A4纸边缘的直线和A4纸的边缘基本吻合，但在一些地方可能因为在hough space中找到r和theta有一点点偏差，所以导致有点偏差，但误差不大，整体的轮廓并没太大的偏差。

程序分析：

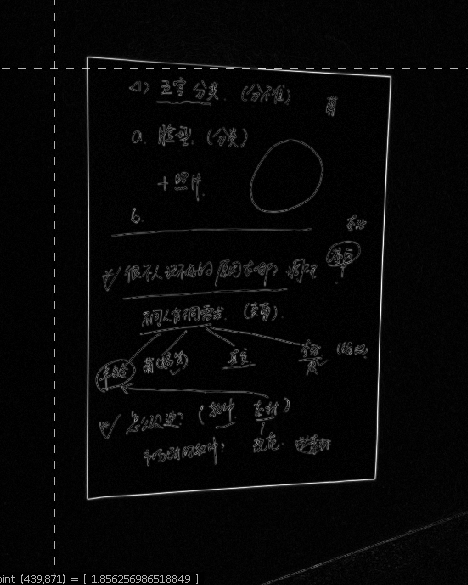
要找到A4纸的边缘直线方程，我首先要进行canny边缘检测，然后把原图的x,y笛卡尔坐标空间转到hough space 的r 和theta空间，找到较亮的点。然后再把亮点转换到原图的x,y笛卡尔空间，这样就可以找到直线方程了。

Canny检测：

我先用了cimg库中一个效果类似高斯模糊的滤波器blur函数对原图滤波，目的是去除噪声。

然后我把图片从RGB三维空间转到GREY灰度一维空间。这样是为了减少后面处理的计算量。然后我求图片的x,y方向的梯度，用了cimg库中的sobel算子，使用get\_gradient函数得到CimgList系列。

这是我得到的其中一张图的magnitude图像：



后面是我自己实现的函数部分：

因为仅求梯度得到的magnitude图像的边缘太粗，我还需要对magnitude进行非最大值抑制，找到较为准确的边缘。

在John Canny提出的Canny算子的论文中，非最大值抑制就只是在0、90、45、135四个梯度方向上进行的，每个像素点梯度方向按照相近程度用这四个方向来代替。这种情况下，非最大值抑制所比较的相邻两个像素就是：

          1） 0：左边 和 右边

          2）45：右上 和 左下

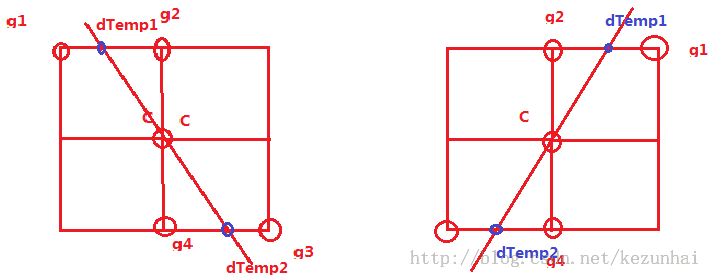
          3）90： 上边 和 下边

          4）135： 左上 和 右下

      这样做的好处是简单， 但是这种简化的方法无法达到最好的效果， 因为，自然图像中的边缘梯度方向不一定是沿着这四个方向的。因此，就有很大的必要进行插值，找出在一个像素点上最能吻合其所在梯度方向的两侧的像素值。

找出插值点的示意图如下：

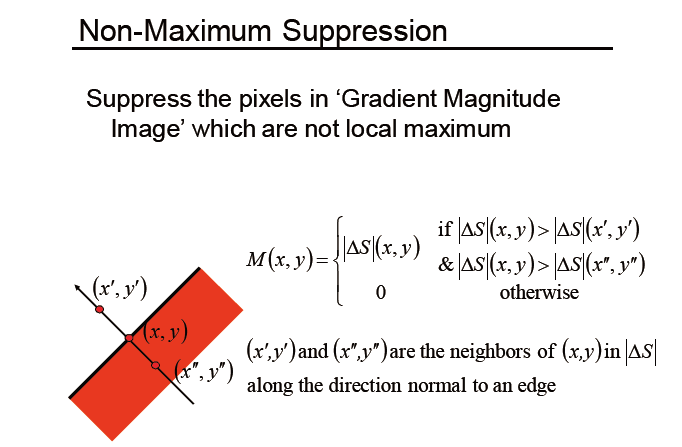
当梯度方向更靠近Y轴方向时，左图为gx,gy同向，右图为gx,gy反向



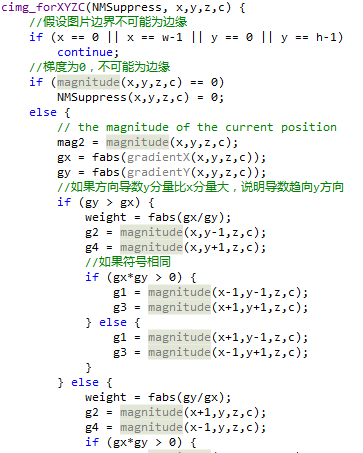
上图中，C表示中心位置点，斜的直线表示梯度方向（非极大值抑制是在梯度方向上的极大值），左边的一副表示gy与gx的方向相同，而右边的这幅这表示gy与gx的方向相反（注意原点在左上角），而权重则为weight = |gx|/|gy|，因此则根据此种情况，其插值表示则为：

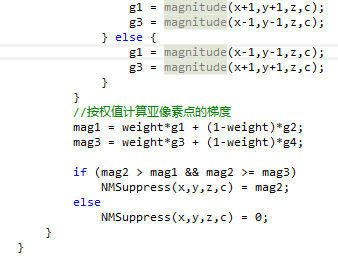
1. dTemp1 = weight\*g1 + (1-weight)\*g2;
2. dTemp2 = weight\*g3 + (1-weight)\*g4;

当梯度方向更靠近X轴，同样的原理。根据这张图:

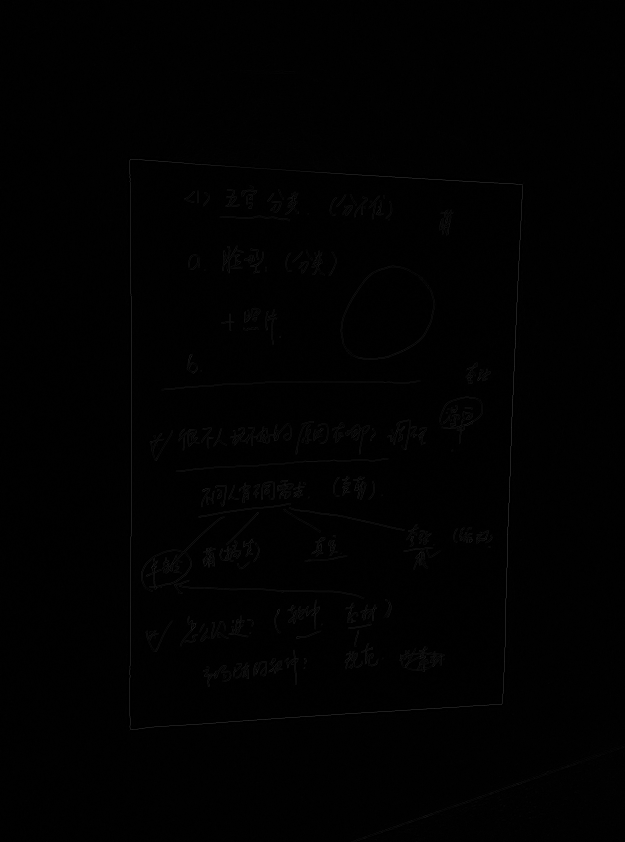


我的实现如下：



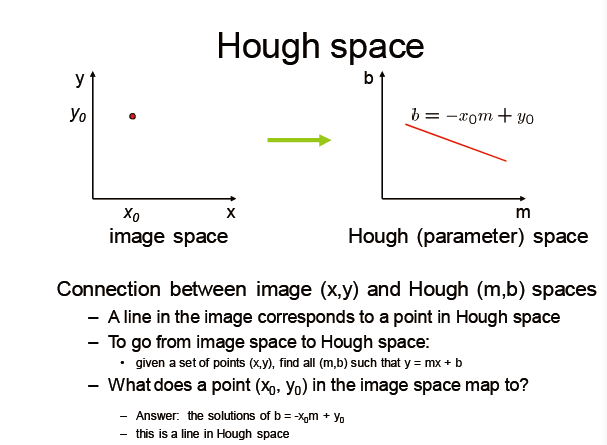


非极大值抑制的直接结果就是图像的边缘会细很多，这样的边缘肯定要比求梯度后的要好，基本满足好边缘的特性。结果图如下：

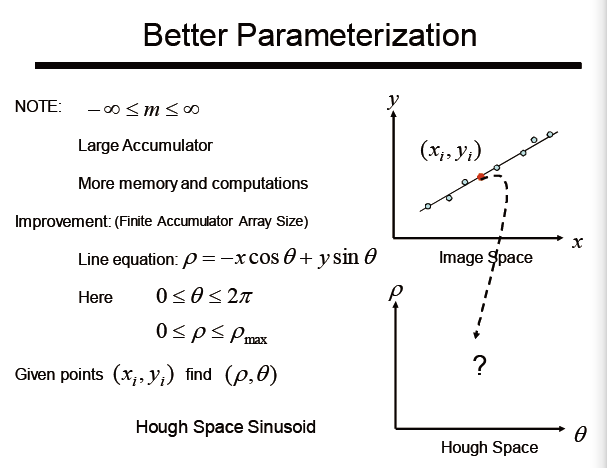


然后还要对非极大值抑制后的图像做阀值处理，目的是把边缘连起来。这一步在这个测试中贡献不大，因为非极大值抑制后的图像的边缘基本都是连起来的。我实现二阀值控制，当像素值的梯度大于我设置的high值，则说明这是我要找的边缘点，如果小于low值，说明这不是我要的边缘点。如果在low与high之间，那么看该点的8邻域有没有边缘点，如果有就把该点也设置为边缘点。这样来实现把边缘连接起来。

比较有难的是hough空间转换函数的实现。Hough space转换的原理大概就是在image space 中的一个(x,y)点映射到 hough space 中的一条直线。如果image space 中有一条直线 1 y = mx + b，那么映射到hough space 空间就是直线2 b = -xm + y。也就是image space 的斜率和截距在hough space中就成了自变量。感性的认识就是一个image space的点是可以被无数不同的m和b约束的直线穿过。参考图：



这样的模型很直观，但缺点是当image space的斜率趋近无穷时，在hough space中表示起来就很吃力了。所以有了极坐标模型：



我做了一个坐标平移，把原点平移到图像的中心，方便后面对theta角的观察。依照数学公式：

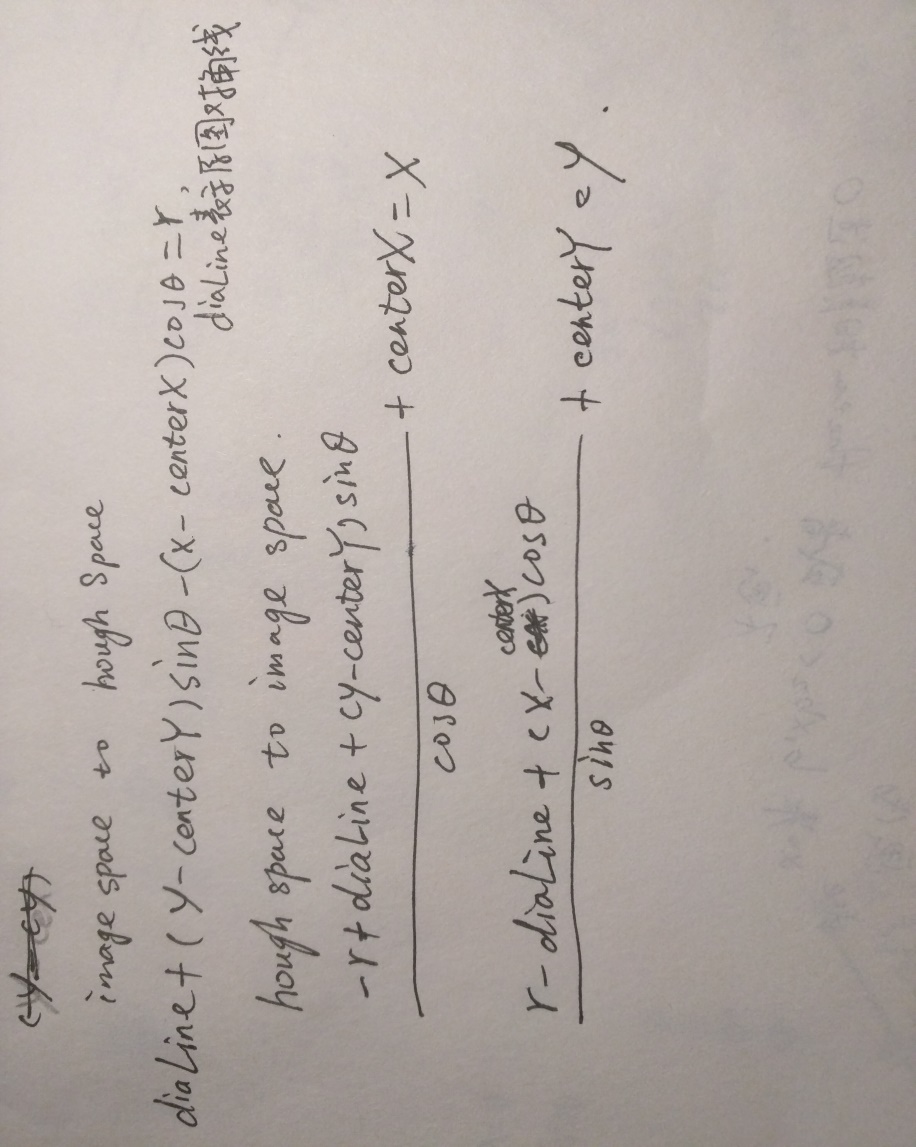
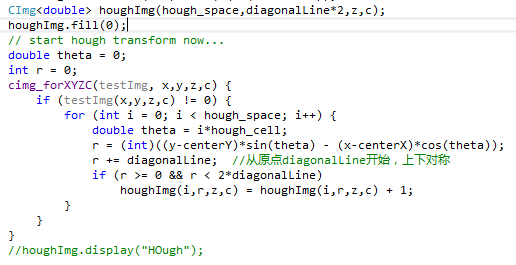


Image space 转到 hough space代码实现如下：



当从hough space 转回image space要注意的是角度问题。当theta角大于45度并小于135度时，说明这条直线是偏向x轴的，也就是在x方向的变化速度要比y方向快。所以在离散的坐标轴中，x每次加减1，y的变化要小于1。这时可以假设x是知道的，然后可以按公式求出y，这样就基本遍历了一遍原图，找到合适的点。这样假设比另一种假设已知，求x误差要小。因为当每次y变换1，x的变化都比1大，这样会导致x方向失真。同理当theta小于45度或者大于135度，则假设y已知，求x。

当然我们的目标不是将图像转到hough space 然后又转回来。我们的目标是找到4条A4纸的直线方程，4个角点。如果直线确定了，相交点就是角点了。如何确定四条直线，我用了比较笨的方法，先判断角度问题，看是偏x还是偏y。然后分类对x和y中的theta角度和截距符号进行过滤。

如果两条直线的theta角在Pi的两端，那么我要考虑两者的截距符号了，如果异号，则说明是一条直线是另一条直线旋转后的结果，两条直线会基本重合，这样的边我们取一条就够了。如果两条直线都是A4纸的长，那么两者的theta角不能差别太大，我设置10为边界，或者两条直线的theta角在Pi的两端，两者的截距符号相同。当然两条长的对角线之差也不能太小，我设置边界为10，还有对角线不能为0。为0肯定不是我要找的边缘，因为原点我是设置纸的中心的。

如果判断两条直线是否为A4纸的宽，条件也差不多。这是个比较笨的方法，因为条件都太具体了，推广性是不强的。

还有在找直线时，我是从hough value最大值开始找的，这样可以避免短线的干扰。所以我对hough图像进行了一次递减排序。这样带来的问题是，每次找一个合适的hough点，我都要从hough图像的起点开始找，这样的重复计算开销有点大。但因为我已经排序过，保证我要找的点排在前，所以如果判定条件设置得好的话，效率还是可以接受的。但是，最坏情况就有些糟糕了，要遍历完整张hough图。。。

一旦我找到两条宽，我就把finish\_x设置为真，不再找宽；一旦我找到两条长，我就把finish\_y设置成真，不再找长。如果长宽找齐了，count等于4，直接跳出查找。然后就是画线求角点。

具体代码如下：

