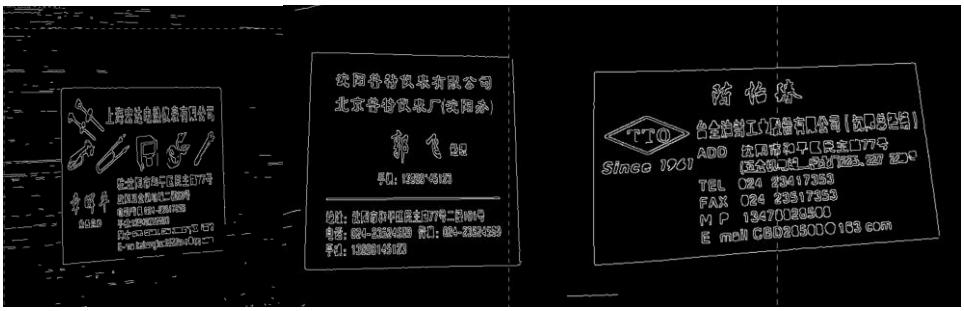
Final Project

1. 名片矫正

1) 首先要把名片的轮廓找出来，思路是用 canny 边缘提取和霍夫变换。由于名片颜色和反光的问题，有两张名片的轮廓不完整，导致无法识别。其他名片效果较好。用 canny 算法进行边缘提取的效果如下：



像左边这样的图片，由于桌面纹理的原因产生了很多噪声点，要对这些噪声点进行过滤，以防影响后面的结果。过滤的思路类似连通块的提取，以基准点开始遍历八邻域，如果线/块的大小小于固定阈值，就删去这些点，代码如下：

```
1.  CImg<unsigned char> delete_line(CImg<unsigned char> picture)
2.  {
3.      int rows = picture._width;
4.      int cols = picture._height;
5.      int dx[8] = { 1,1,0,-1,-1,-1,0,1 },
6.          dy[8] = { 0,1,1,1,0,-1,-1,-1 };
7.      int length = 0;
8.      int pos = 0;
9.      CImg<unsigned char> final_result = picture;
10.     queue<pair<int, int>> Queue;
11.     vector<vector<bool>> isVisited(rows, vector<bool>(cols, false));
12.     vector<stack<pair<int, int>>> Stack(10000, stack<pair<int, int>>());

13.     for (int r = 0; r < rows; r++) {
14.         for (int c = 0; c < cols; c++) {
```

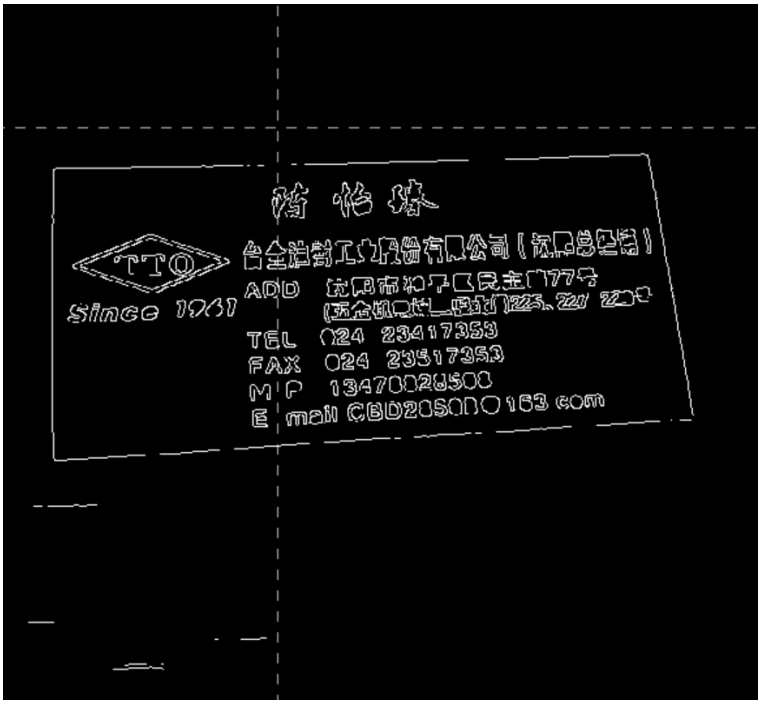
```
            if ((int)final_result(r, c) == 0) {
                isVisited[r][c] = true;
            }
        }
    }

    //广度优先搜索
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            length = 0;
            if (!isVisited[r][c]) {
                Queue.push(make_pair(r, c));
                Stack[pos].push(make_pair(r, c));
                while (!Queue.empty()) {
                    int Row = Queue.front().first;
                    int Col = Queue.front().second;
                    if (!isVisited[Row][Col]) {
                        isVisited[Row][Col] = true;
                        length++;
                        Stack[pos].push(make_pair(Row, Col));
                        for (int i = 0; i < 8; i++) {
                            if (Row + dx[i] >= 0 && Row + dx[i] < rows &
&
                                Col + dy[i] >= 0 && Col + dy[i] < cols &
&
                                !isVisited[Row + dx[i]][Col + dy[i]]) {

                                Queue.push(make_pair(Row + dx[i], Col +
dy[i]));
                            }
                        }
                    }
                    Queue.pop();
                }

                //删除长度小于 30 的连线
                if (length < 30) {
                    while (!Stack[pos].empty()) {
                        int Row = Stack[pos].top().first;
                        int Col = Stack[pos].top().second;
                        final_result(Row, Col, 0) = 0;
                        final_result(Row, Col, 1) = 0;
                        final_result(Row, Col, 2) = 0;
                        isVisited[Row][Col] = true;
```

```
                        Stack[pos].pop();
                    }
                }
                pos++;
            }
        }
    }
    return final_result;
}
```

删除的效果如下：

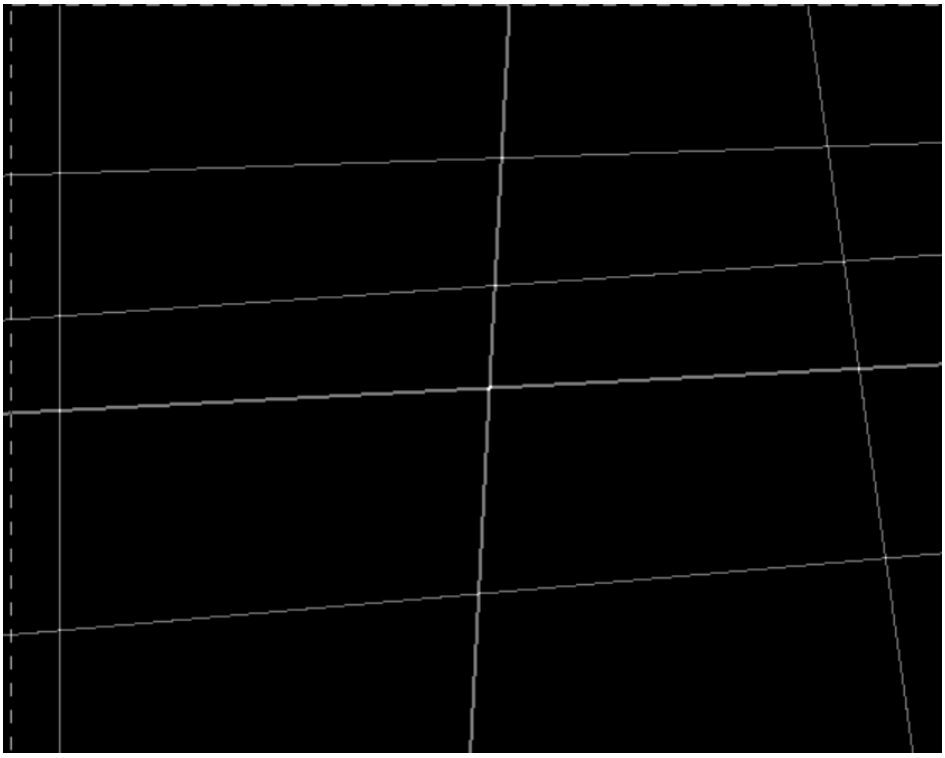
删除前的图片：

删除之后的图片：



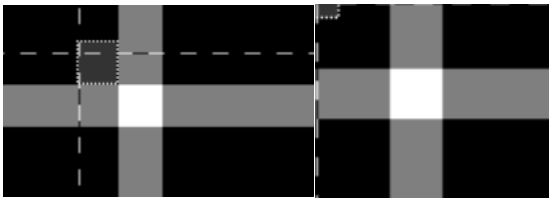
可以看出噪声点少了很多，效果较好。但是这种方法的缺点在于如果边缘提取时边缘点没有连在一起容易把边缘给删掉，导致无法提取轮廓。所以这里对阈值的调整比较重要。

2) 然后要找到名片的四条边和边的交点。我的做法是进行霍夫变换，但是由于要稳定提取出边缘的四条直线，霍夫变换的阈值要调的比较小，所以会识别出多条直线，这样就有超过四个交点。这些交点可能在名片的边缘上，也可能在边缘线的延长线上，把得到的直线画在新的空白图片上结果如下：



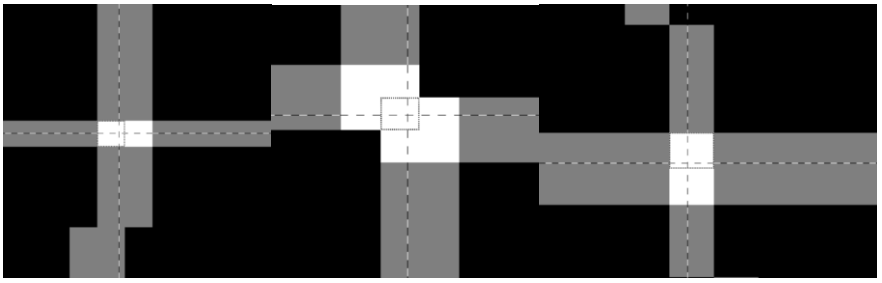
经过观察，边缘的四条直线的交点或者与边缘接近平行的直线的交点一定是 2 个像素，且周围八邻域的像素个数均少于 2 个：

边缘/平行边缘的线产生的交点：



而倾斜角较大的线与边缘的交点的八邻域一定有某个像素点的像素为 2：

斜线产生的交点：



根据实验结果，有很少的情况会有三条直线交于一点，这种情况都出现在该点为角点时，所以可以单独归为一类。

根据这些特点，可以把交点都限制在名片的边缘上：

```
int arr[9];
arr[0] = edge(x, y);
arr[1] = edge(x + 1, y);
arr[2] = edge(x, y + 1);
arr[3] = edge(x + 1, y + 1);
arr[4] = edge(x - 1, y);
arr[5] = edge(x, y - 1);
arr[6] = edge(x - 1, y - 1);
arr[7] = edge(x + 1, y - 1);
arr[8] = edge(x - 1, y + 1);
if ((arr[0] >= 2 && arr[1] < 2 && arr[2] < 2 && arr[3] < 2 && arr[4] < 2
    && arr[5] < 2 && arr[6] < 2 && arr[7] < 2 && arr[8] < 2)|| arr[0] == 3)
    {
    corss_point.push_back(make_pair(x, y));//得到所有的交点坐标
    //I4.draw_circle(x, y, 5, red);
}
```

下一步就是确定四个角点，把四个角点分别设为左上角、右上角、左下角、右下角，考虑到实验中名片的长宽比和倾斜角，可以得出左上角的横纵坐标之和最小，右下角的横纵坐标之和最大，把所有交点按横纵坐标和进行排序可以很容易找到这两个点。然后把图片沿着 Y 轴翻转，则翻转后的左上角其实是原图的左下角，同理，右下角是原图的右上角，再次进行筛选就可以把这两个点找出来，这样就可以把四个角点全都找到，代码如下：

```
bool cmp1(pair<int,int> a, pair<int, int> b)
{
    return a.first + a.second < b.first + b.second;
}
bool cmp2(pair<int, int> a, pair<int, int> b)
{
```

```
7.      return a.first - a.second < b.first - b.second;
8.  }
9.
10. void CutImage::Rotate()
11. {
12.     vector<pair<int, int>>sorted1 = corss_point;
13.     sort(sorted1.begin(), sorted1.end(), cmp1);
14.     pair<int, int>left_top = sorted1[0];
15.     pair<int, int>right_bottom = sorted1[sorted1.size() - 1];
16.     vector<pair<int, int>>sorted2 = corss_point;
17.     sort(sorted2.begin(), sorted2.end(), cmp2);
18.     pair<int, int>left_bottom = sorted2[0];
19.     pair<int, int>right_top = sorted2[sorted2.size() - 1];
20. }
```

3) 确定完角点，即可进行矫正。由于拍摄角度问题，得到的图片是投影变换之后的结果，所以名片可能不是矩形，而且有一定的形变，所以要进行逆的投影变换矫正。矫正的原理如下：

假设现在有一幅原图像$f(x_0, y_0)$产生了一定的变形(存在一定的几何关系)，变成的另一幅目标图像$g(x_1, y_1)$，这种变化仅仅是像素位置的变化，并不改变像素的值。 各像素点位置关系可以用函数的形式表示：

$$x_1 = s(x_0, y_0)$$
$$y_1 = t(x_0, y_0)$$

其中，$s(x_0, y_0)$和$t(x_0, y_0)$为由$f(x_0, y_0)$到$g(x_1, y_1)$的坐标变换函数。如果我们确定了两副图像各像素点之间位置的转换函数。那么对于几何失真的图像校正就是其变换的逆过程。 函数表达式为：

$$x_0 = s^{-1}(x_1, y_1)$$
$$y_0 = t^{-1}(x_1, y_1)$$

投影变换可以通过双线性方程进行建模。如下式：

$$x_1 = s(x_0, y_0) = c_1x_0 + c_2y_0 + c_3x_0y_0 + c_4$$
$$y_1 = t(x_0, y_0) = c_5x_0 + c_6y_0 + c_7x_0y_0 + c_8$$

在这两个式子中我们可以看到总共有 8 未知个参数，如果可以找到 4 组对应点，就可以建立 8 个方程。求解出这 8 未知个参数，求出这个双线性方程，即失真图片上的四个点，和希望变换到的图片上的四个点。分

别把八个点的坐标带入即可求解。这里我用的是 openCV 库中的 warpPerspective 函数：

```
1.  //透视变换前的四个点所在坐标，+5 是为了防止直线方程误差导致的角点偏移
2.  vector<Point2f> src_corner(4);
3.  src_corner[0].x = left_top.first+5;
4.  src_corner[0].y = left_top.second+5;
5.  src_corner[1].x = right_top.first-5;
6.  src_corner[1].y = right_top.second+5;
7.  src_corner[2].x = right_bottom.first-5;
8.  src_corner[2].y = right_bottom.second-5;
9.  src_corner[3].x = left_bottom.first+5;
10. src_corner[3].y = left_bottom.second-5;
11.
12.     //透视变换后的四个点所在坐标
13. Mat resultImg(800, 1600, CV_8UC3);
14. vector<Point2f> dst_corner(4);
15. dst_corner[0] = Point(0, 0);
16. dst_corner[1] = Point(resultImg.cols, 0);
17. dst_corner[2] = Point(resultImg.cols, resultImg.rows);
18. dst_corner[3] = Point(0, resultImg.rows);
19.
20. Mat M = getPerspectiveTransform(src_corner, dst_corner);
21. warpPerspective(src, resultImg, M, resultImg.size(), INTER_LINEAR);
```

这样可以把原图中的名片单独提取出来并统一化大小为 800*1600。得到的结果如下：

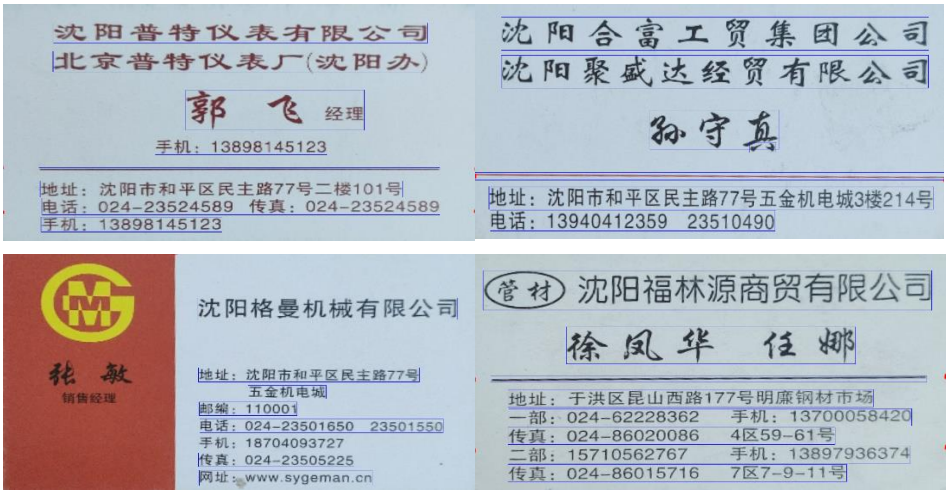
至此完成第一步的名片矫正。

2. 名片主要部分切割

这一部分主要是基于连通块做的，所以要对名片进行二值化、前后景分割、提取连通块等操作，这里用的还是前几次作业的方法就不在重复叙述。然后是如何判断是属于同一个模块的内容，一开始我的想法是按行查找，找到有黑色像素的开始划分，到没有黑色像素停止，但是由于给出的名片排版各种各样，导致这种方法效果很差。后面思考到相同模块内容基本都在一行，而且遍历连通块是从上到下进行的，所以可以考虑用简化的层次聚类算法：找到所有连通块的中心坐标，以第一个连通块为基准，找中心点纵坐标与基准纵坐标接近的连通块，把他们归为一类，并更新基准纵坐标，由于连通块的查找是按顺序进行的，所以一行的连通块都是相连的，所以只要从前往后遍历即可，遍历到与基准纵坐标相差较大的中心，就把当前类存储并开始下个聚类，算法伪代码如下：

```
Algorithm Cluster
---------------------------
Input:link areas
middles = get_middle(link areas)
base = middles[0].y
c.push(link areas[0])
vector<typeof<c>> clusters
For i From 1 To size(link areas)
    If (middles[i].y - base) < threshold then do
        c.push(link areas[i])
        update base
    Else
        clusters.push(c)
        c.clear
        c.push(link areas[i])
End
clusters.push(c)
return clusters
```

这样就可以按行把名片中的内容进行分割，最后画出来所有的板块结果如下：



但是这样的方法在上面这样板块明确，上下分布的名片效果很好，但是在一些图片很多，或者是左右排版的名片中效果就不是很好，如下：

至此完成第二部分的主要部分分割。