中山大學
SUN YAT-SEN UNIVERSITY

《计算机视觉与模式识别》
Experiment 4
Testing Report

学院名称	:	数据科学与计算机学院
学生姓名	:	吴若宇
学号	:	14301030
专业（班级）	:	14级计算机科学与技术
时间	:	2017 年 3 月 27 日

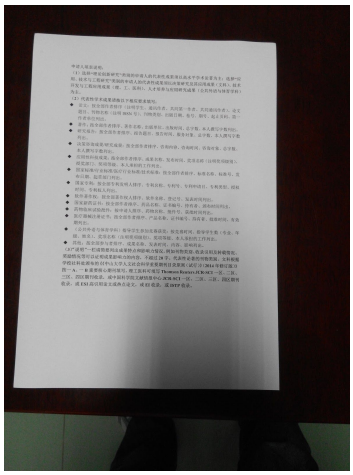
灰度图像和彩色图像的直方图均衡

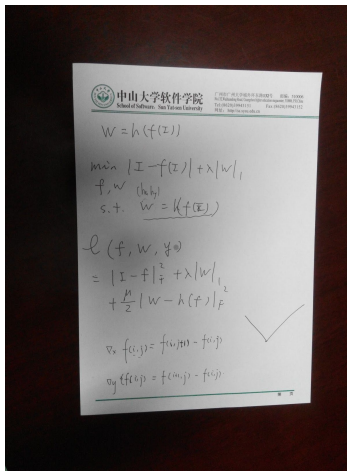
1、 测试环境

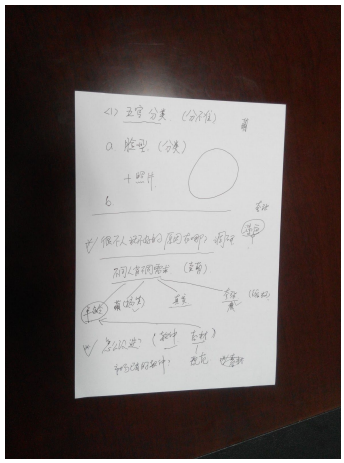
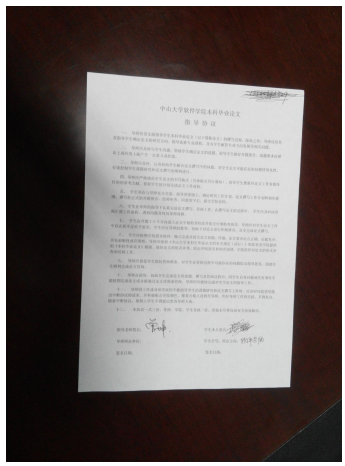
Ubuntu 15.10

2、 测试数据

不同角度拍摄的A4纸





3、 测试代码及过程

整个实验流程包括canny边缘检测、霍夫变换和从霍夫空间转换为灰度图。

1）canny边缘检测

canny边缘检测采用了Ex2的code0代码，这在Ex2的实验报告中有很多讨论，在这里不加赘述。

2）霍夫变换（voting算法）

首先从canny检测之后的灰度图变换到hough空间，在霍夫空间上进行投票，存储在名为houghSpace的空间中，houghSpace也是由CImg存储。

```
void gray2Hough() {
    cimg_forXY(edgeImg, x, y) {
        if (edgeImg(x, y) == 255) {
            cimg_forX(houghSpace, theta) {
                double rTheta = (double)theta * 3.14 / 180.0;
                int rho = (int)(x * cos(rTheta) + y * sin(rTheta));
                if (rho >= 0 && rho < houghSpace.height())
                    houghSpace(theta, rho) ++;
            };
        }
    };
}
```

之后，在houghSpace中把投票值前四的找到，这里用最小堆来实现。如果houghSpace上的一个点的值比现在最小堆里的堆顶的值还要小，并且它是局部最大值而且映射回去还在图像之内，那么就把最小堆的堆顶pop掉，把该点push进最小堆。

```
void findMax() {
    while (max.size() < 4) {
        max.push(Point(-1, -1, 0));
    }

    cimg_forXY(houghSpace, theta, rho) {
        if (houghSpace(theta, rho) > max.top().weight) {
            if (isLocalMax(theta, rho)) {
                if (inRegion(theta, rho)) {
                    max.pop();
                    max.push(Point(theta, rho, houghSpace(theta, rho)));
                }
            }
        }
    };
}
```

下面为判断是不是局部最大值和映射回去还在不在图像内的函数:

```
bool inRegion(int theta, int rho) {
    double a = cos(theta * 3.14 / 180.0);
    double b = sin(theta * 3.14 / 180.0);
    double c = -rho;

    const int minY = 0;
    const int maxY = img.height() - 1;
    const int minX = 0;
    const int maxX = img.width() - 1;

    if (b == 0){
        if (-c/a > minX && -c/a < maxX)
            return true;
        return false;
    }
    else {
        double k = -a/b;
        c = -c/b;
        if (c > minY && c < maxY)
            return true;
        if (k * maxX + c > minY && k * maxX + c < minY)
            return true;
        if (-c/k > minX && -c/k < maxX)
            return true;
        if (maxY/k - c/k > minX && maxY/k - c/k < maxX)
            return true;
        return false;
    }
}
```

```
bool isLocalMax(int theta, int rho) {
    for (int thetaShift = -thetaFilterWidth; thetaShift <= thetaFilterWidth; thetaShift ++) {
        for (int rhoShift = -rhoFilterWidth; rhoShift <= rhoFilterWidth; rhoShift ++) {
            if (thetaShift == 0 && rhoShift == 0)
                continue;
            if (theta + thetaShift < 0 || theta + thetaShift >= houghSpace.width())
                continue;
            if (rho + rhoShift < 0 || rho + rhoShift >= houghSpace.height())
                continue;
            if (houghSpace(theta + thetaShift, rho + rhoShift) > houghSpace(theta, rho))
                return false;
        }
    }
    return true;
}
```

thetaFilterWidth和rhoFilterWidth是给定的参数,

是判断是不是局部最大值的窗口的半径

这样可以防止在一个局部找到多个最大值

3）变换回灰度图

```
void hough2gray() {
    std::vector<Line> line;
    CImg<double> result(this->img);

    while (!max.empty()) {
        line.push_back(Line(cos(max.top().theta * 3.14 / 180.0), sin(max.top().theta * 3.14 / 180.0), -max.top().rho));
        max.pop();
    }

    // print out the line's coefficients
    for (int i = 0; i < line.size(); i ++) {
        cout << line[i].a << "x + " << line[i].b << "y + " << line[i].c << " = 0" << endl;
    }

    drawLine(result, line);

    drawIntersection(result, line);

    result.display();
}
```

把存储在最小堆里里面的四对参数拿出来，转换到图像上的四条直线。把它们画出来和求它们的交点。

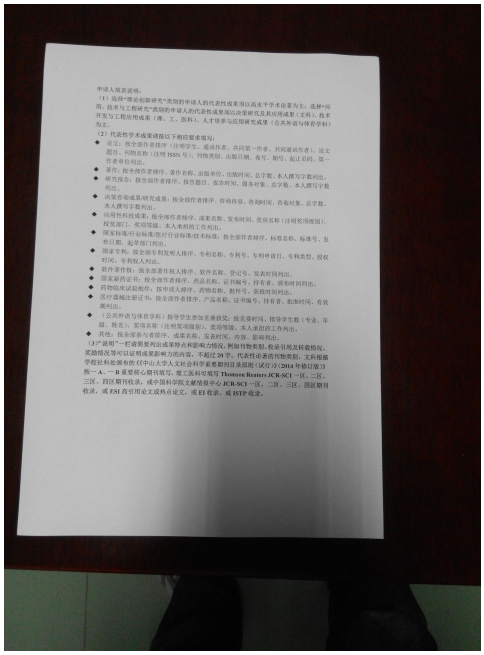
画线和求交点的代码如下：

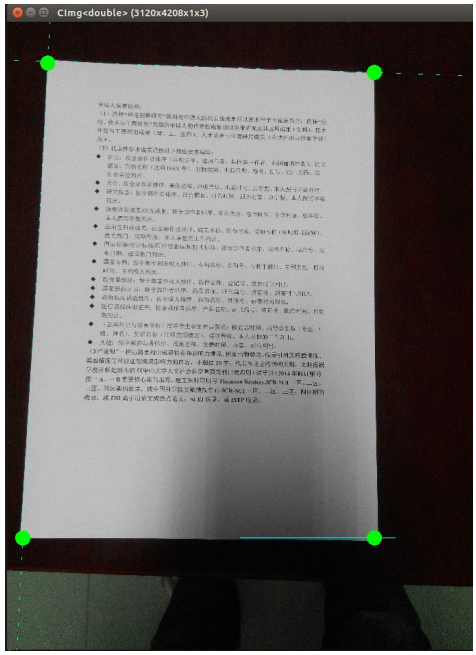
```
void drawLine(CImg<double>& result, std::vector<Line>& line) {
    for (int i = 0; i < line.size(); i ++) {
        const int minY = 0;
        const int maxY = result.height() - 1;
        const int minX = 0;
        const int maxX = result.width() - 1;

        const int color[] = {0, 255, 255};

        std::vector<std::pair<int, int>> crossPoint;

        if (line[i].a == 0) {
            crossPoint.push_back(std::pair<int, int>(minX, -line[i].c / line[i].b));
            crossPoint.push_back(std::pair<int, int>(maxX, -line[i].c / line[i].b));
        }
        else if (line[i].b == 0) {
            crossPoint.push_back(std::pair<int, int>(-line[i].c / line[i].a, minY));
            crossPoint.push_back(std::pair<int, int>(-line[i].c / line[i].a, maxY));
        }
        else {
            if (- line[i].c / line[i].b > minY && - line[i].c / line[i].b < maxY)
                crossPoint.push_back(std::pair<int, int>(0, - line[i].c / line[i].b));
            if (- line[i].a / line[i].b * maxX - line[i].c / line[i].b > minY && - line[i].a / line[i].b * maxX - line[i].c / line[i].b < maxY)
                crossPoint.push_back(std::pair<int, int>(maxX, - line[i].a / line[i].b * maxX - line[i].c / line[i].b));
            if (- line[i].c / line[i].a > minX && - line[i].c / line[i].a < maxX)
                crossPoint.push_back(std::pair<int, int>(- line[i].c / line[i].a, 0));
            if (- line[i].b / line[i].a * maxY - line[i].c / line[i].a > minX && - line[i].b / line[i].a * maxY - line[i].c / line[i].a < maxX)
                crossPoint.push_back(std::pair<int, int>(- line[i].b / line[i].a * maxY - line[i].c / line[i].a, maxY));
        }
        result.draw_line(crossPoint[0].first, crossPoint[0].second, crossPoint[1].first, crossPoint[1].second, color);
    }
}
```

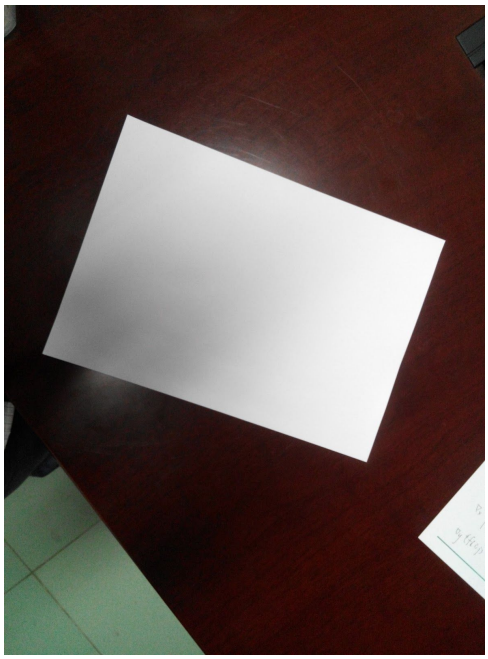
```
void drawIntersection(CImg<double>& result, std::vector<Line> line) {
    const int color[] = {0, 255, 0};

    const int minY = 0;
    const int maxY = result.height() - 1;
    const int minX = 0;
    const int maxX = result.width() - 1;

    for (int i = 0; i < line.size(); i ++) {
        for (int j = i + 1; j < line.size(); j ++) {
            if (ifVertical(line[i], line[j])) {
                int x = (-line[j].b * line[i].c + line[i].b * line[j].c) / (-line[j].a * line[i].b + line[i].a * line[j].b);
                int y = 0;
                if (line[i].b != 0)
                    y = - line[i].a / line[i].b * x - line[i].c / line[i].b;
                else
                    y = -line[j].a / line[j].b * x - line[j].c / line[j].b;
                if (x > minX && x < maxX && y > minY && y < maxY) {
                    result.draw_circle(x, y, 50, color);
                }
            }
        }
    }
}
```

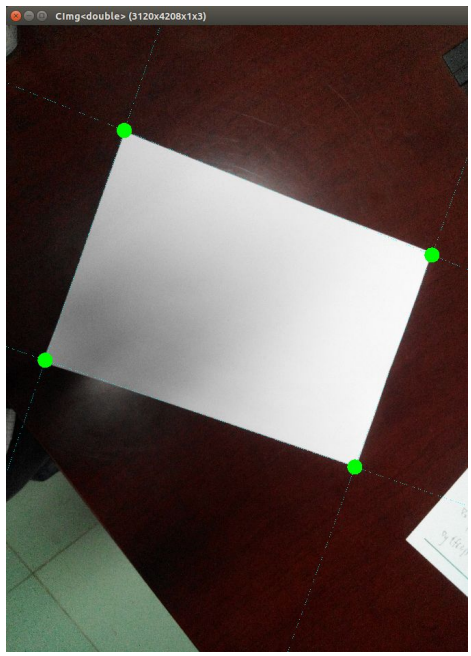
4、 实验结果

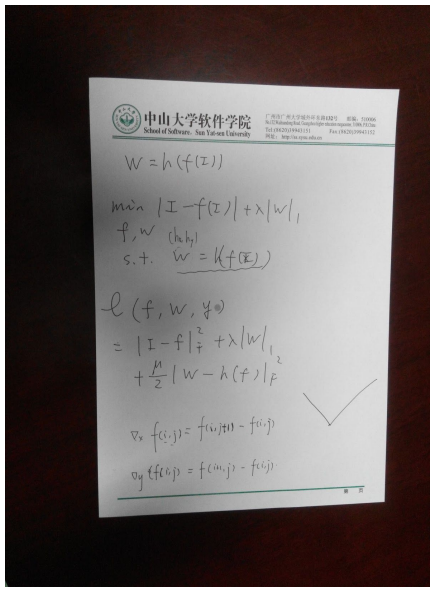
下列图片分别为原图和最终效果。

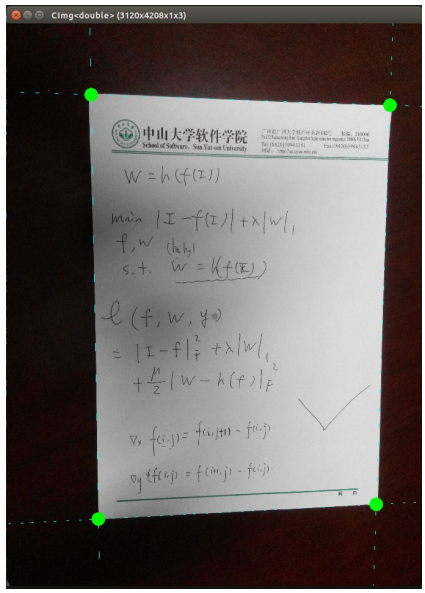


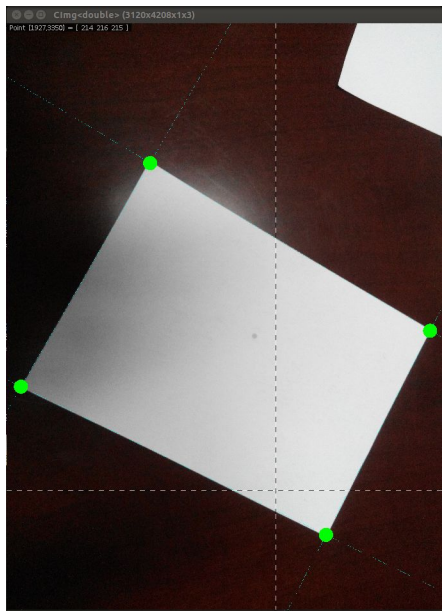


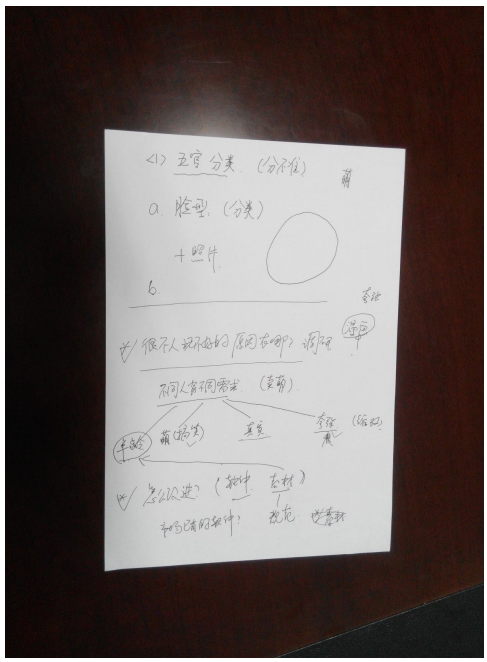


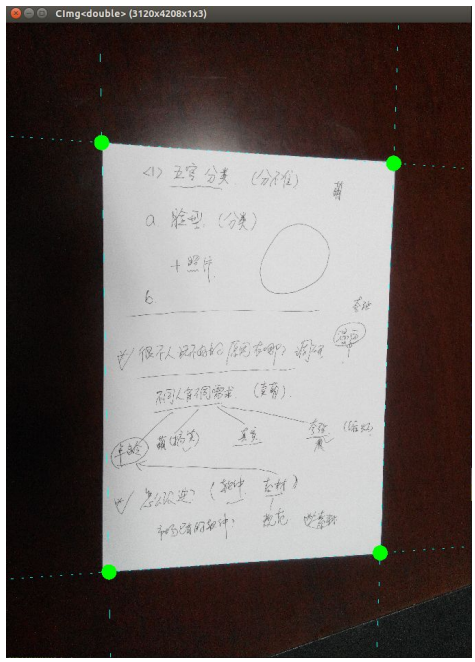


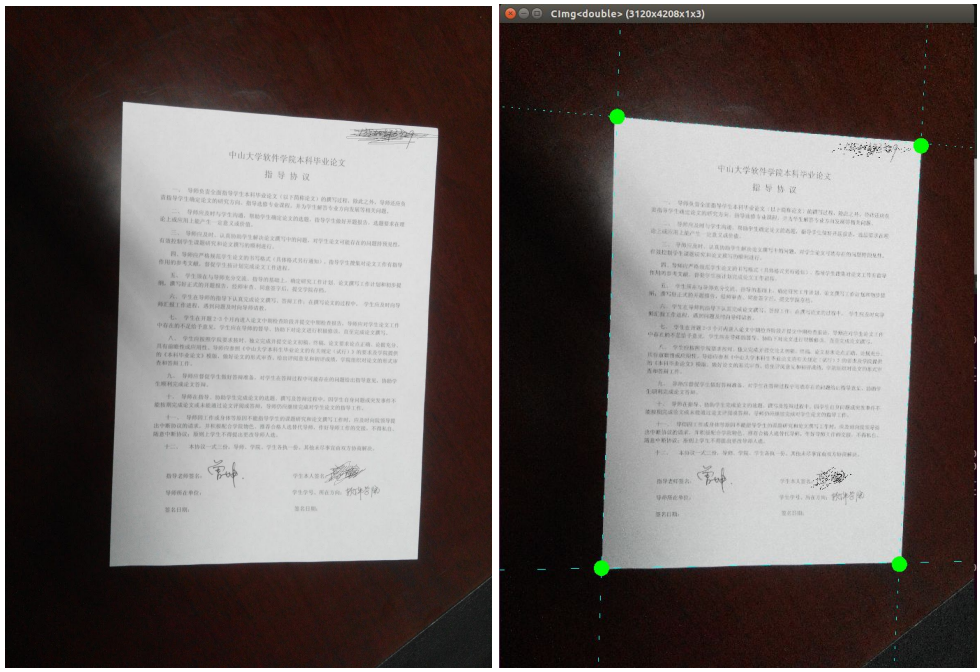



CImg<double> (3120x4208x1x3)


CImg<double> (3120x4208x1x3)
Point [1927,3850] = [214 216 215]




CImg<double> (3120x4208x1x3)



5、 总结及提高

可以看出，对于各张图，都能得到相当不错的结果，没有受到阴影、旁边的纸张和纸张上的字的影响。那么可能可以优化的地方就在于性能上。

我首先想到的就是可以采用并行化的方法提高霍夫变换检测直线的性能。因为在投票的时候各个点之间是没有依赖关系的，所以在PRAM的模型上可以分给多个进程/核进行处理。

然后其实我们没有必要对完整的图像进行处理，因为我们只需要边缘信息，所以如果能把图像通过某种方式缩小再进行霍夫检测再放大，即使这样图片失去了一些信息，但是不会影响到我们对边缘信息的提取。缩小的方法我们可以用最新一周讲到的双线性插值进行处理。