

实验环境：Ubuntu 16.04 LTS 虚拟机

可执行文件：a.out

代码文件：canny.h canny.cpp HoughTransform.h HoughTransform.cpp

sharp.h sharp.cpp main.cpp

一、 实验部分

1. 首先需要确定四个角点的顺序，用于裁切的时候使用。在实验中使用 4 个点中最上方的两个作为上方的边界，最下方的两个点作为下方的边界。
2. 使用 inverse warping 的方式，将图片进行旋转。在映射的时候，使用 bilinear interpolation 的方式获取在原图像中位于非整数点处的像素点的数值。

Inverse Warping:

```
void sharp::rotate() {
    // theta = 1;
    double angle = atan(this->theta);
    for(int q = 0; q < 4; q++) {
        double x1 = corners[q].x;
        double y1 = corners[q].y;
        rotatedCorners.push_back(point(x1*cos(-angle)-y1*sin(-angle), x1*sin(-angle)+y1*cos(-angle)));
    }

    cout << "angle: " << angle << endl;

    CImg<unsigned char> tempImg(inputImg._width*2, inputImg._height*2, 1, 3, 0);
    cimg_forXY(tempImg, x, y) {
        double x1, y1;
        x1 = x*cos(angle)-y*sin(angle);
        y1 = x*sin(angle)+y*cos(angle);
        if(x1 < 0 || x1 > inputImg._width || y1 < 0 || y1 > inputImg._height) continue;
        else {
            tempImg(x,y,0) = bilinearInt(x1, y1, 0);
            tempImg(x,y,1) = bilinearInt(x1, y1, 1);
            tempImg(x,y,2) = bilinearInt(x1, y1, 2);
        }
    }
    // tempImg.display("rotate");
    crop(tempImg);
}
```

Bilinear Interpolation:

```
unsigned char sharp::bilinearInt(double x, double y, int channel) {
    int x0 = x / 1;
    int x1 = (x+1) / 1;
    int y0 = y / 1;
    int y1 = (y+1) / 1;
    int a = x - x0;
    int b = y - y0;
    double result = a*b*inputImg(x0, y0, channel) + a*(1-b)*inputImg(x1, y0, channel)
        + (1-a)*b*inputImg(x0, y1, channel) + (1-a)*(1-b)*inputImg(x1, y1, channel);
    return (unsigned char) result;
}
```

3. 按照角点的位置进行裁切，得到最后的 A4 纸的图像。

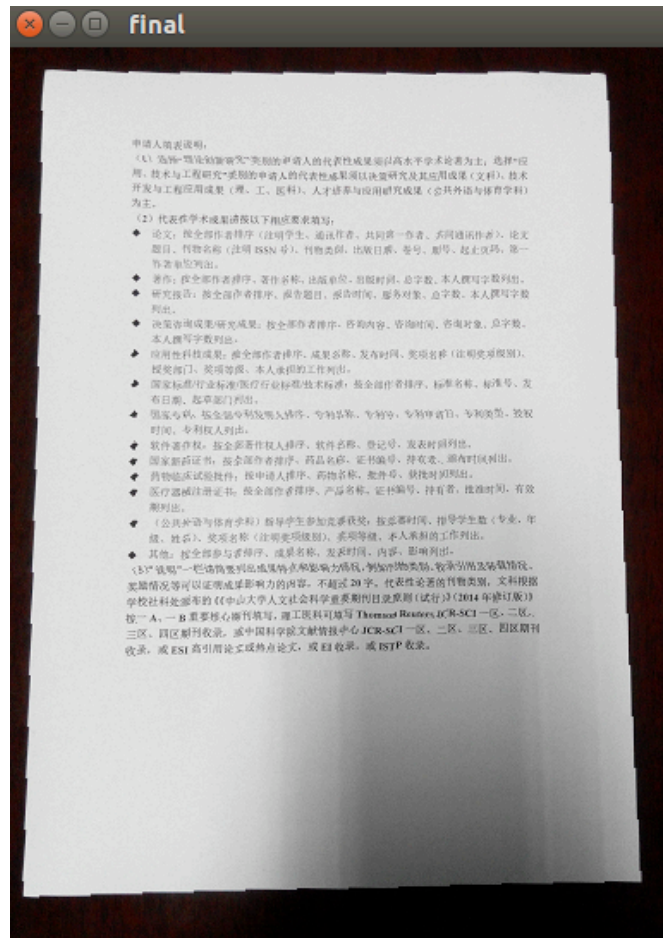
```

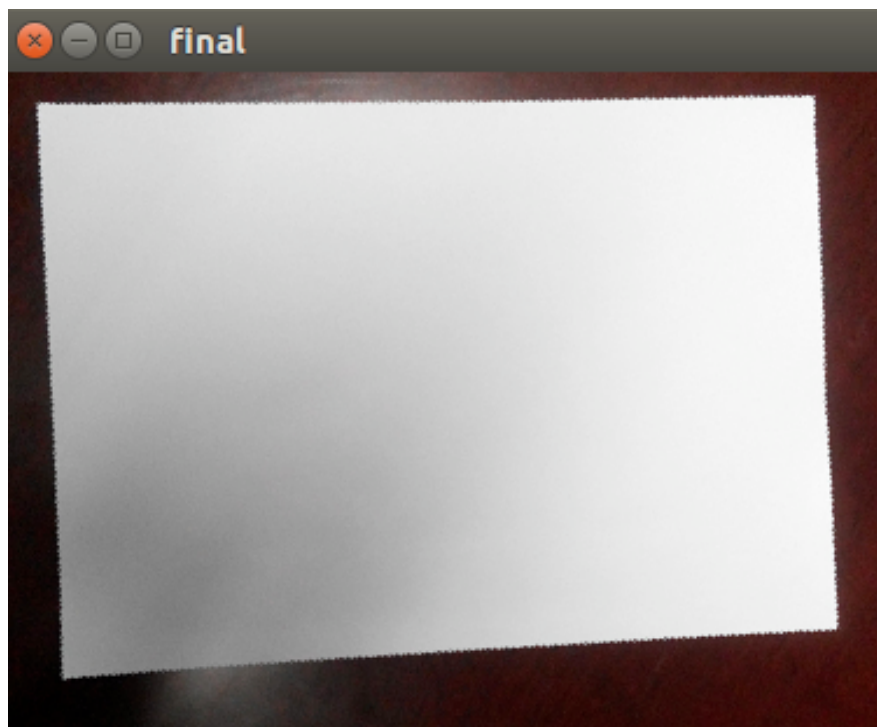
void sharp::crop(CImg<unsigned char> tempImg) {
    int minX=100000, minY = 1000000, maxX = -10, maxY = -10;
    for(int i = 0; i < 4; i++) {
        if(rotatedCorners[i].x <minX) minX = rotatedCorners[i].x;
        if(rotatedCorners[i].x > maxX) maxX = rotatedCorners[i].x;
        if(rotatedCorners[i].y < minY) minY = rotatedCorners[i].y;
        if(rotatedCorners[i].y > maxY) maxY = rotatedCorners[i].y;
    }
    // int minX1 = x
    minX -= 10;
    minY -= 10;
    maxX += 10;
    maxY += 10;
    cout << "aaa" << endl;
    CImg<unsigned char> sharpenedImage(maxX-minX+20, maxY-minY+20, 1, 3, 0);
    cimg_forXY(sharpenedImage, x, y) {
        sharpenedImage(x, y, 0) = tempImg(x+minX, y+minY, 0);
        sharpenedImage(x, y, 1) = tempImg(x+minX, y+minY, 1);
        sharpenedImage(x, y, 2) = tempImg(x+minX, y+minY, 2);
    }
    sharpenedImage.display("final");
}


```

二、 实验结果

实验的结果已经存在了 dataset 文件夹中，文件的命名以 answer_起头。






中山大学软件学院
 School of Software, Sun Yat-sen University

广州市广州大学城外环东路112号 邮编: 510006
 No.112 Guangda Road/Guangzhou Higher Education Region, 510006, P.R.China
 Tel: (8620) 39943151 Fax: (8620) 39943152
 网址: <http://sw.sjys.zhu.cn>

$$W = h(f(I))$$

$$\min_{f, w} |I - f(I)| + \lambda |w|_1$$

$f, w \in \mathbb{R}^{n \times n}$
 s.t. $w = h(f(I))$

$$\mathcal{L}(f, w, y)$$

$$= |I - f|_F^2 + \lambda |w|_1 + \frac{\mu}{2} |w - h(f)|_F^2$$

$$\nabla_x f(i, j) = f(i, j+1) - f(i, j)$$

$$\nabla_y f(i, j) = f(i+1, j) - f(i, j)$$

要提高速度的话，可以通过提升 HoughTransform 以及在 canny 中算法的复杂度来提高速度。同时依旧可以使用缩小图片的大小的方式来提升速度。