车牌识别定位与字符分割

背景

汽车的出现改变了以往出行徒步和以马代步的时代，极大地改变了人们的生活方式，扩大了人们的活动范围，加强了人与人之间的交流。全世界的汽车拥有量呈爆炸性增长，汽车虽方便了我们的出行，但同时也造成了城市交通压力，应用现代科技解决汽车不断增长而出现的交通问题已经成为一项重要的研究课题，智能交通系统应孕而出。

智能交通系统(Intelligent Transportation System，简称 ITS)是一种充分利用各种先进的高新技术来实现实时、准确、高效的交通管理系统，使交通更畅通更安全，它也是一种交通信息服务系统，使人们出行更方便更快捷。随着智能交通系统的快速发展，智能交通系统已经融入人们的日常生活，使人们的生活越来越方便。车辆是智能交通系统中的重点研究对象，每辆车都有自身唯一的车牌号码，车牌号码反映了车辆信息以及关联着车主信息，通过车牌号码可以记录对应车辆的交通行为，因此，车牌识别技术是智能交通系统中最核心最基础的技术之一，决定着智能交通系统的发展速度和技术水平。

它能够实时地对城市的车辆进行检测、监控和管理，实现智能交通的实时性和高效性；它不仅可以有效地减少人工操作的参与，节约成本；还可以在一定程度上杜绝一些交通工作人员的违规、舞弊操作，解决收费流失等问题；它还可以对城市的过往车流量进行检测、指导相关工作，减少交通拥堵现象。

在这个大力倡导智慧型城市概念的社会，随着互联网技术的提升，网络的发展，智能的车牌识别系统早已经深入人们的生活中，监测车流量等。

电子警察系统：一种抓拍车辆违章违规行为的智能系统，大大降低了交通管理压力。

卡口系统：对监控路段的机动车辆进行全天候的图像抓拍，自动识别车牌号码，通过公安专网与卡口系统控制中心的黑名单数据库进行比对，当发现结果相符合时，系统自动向相关人员发出警报信号。

高速公路收费系统：自动化管理，当车辆在高速公路收费入口站时，系统进行车牌识别，保存车牌信息，当车辆在高速公路收费出口站时，系统再次进行车牌识别，与进入车辆的车牌信息进行比对，只有进站和出站的车牌一致方可让车辆通行。

停车场收费系统：随处可见，收费系统抓拍车辆图片进行车牌识别，保存车辆信息和进入时间,并语音播报空闲车位，当车辆离开停车场时，收费系统自动识别出该车的车牌号码和保存车辆离开的时间，并在数据库中查找该车的进入时间，计算出该车的停车费周，车主交完费用后，收费系统自动放行。

智能公交报站：当公交车进入和离开公交站台时，报站系统对其进行车牌识别，然后与数据库中的车牌进行比对，语音报读车牌结果和公交线路。

车牌识别技术应用广泛，当然，上面所指的应用只是其中的一小部分。随着智能交通的迅猛发展，社会对车牌自动识别的需求量会越来越高多，技术上也会越来越高。

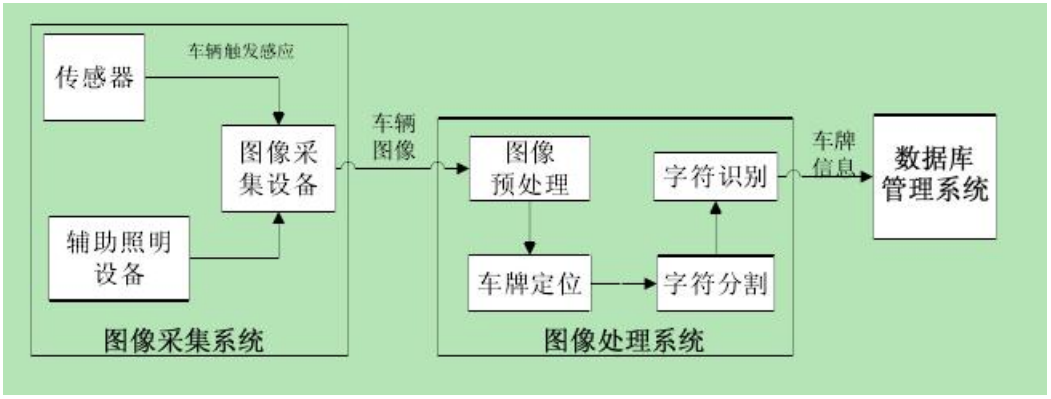
车牌自动识别系统也叫做LPR（License Plate Recognition）系统，目前国内做的比较成熟的产品有北京汉王科技有限公司开发的“汉王眼”车牌识别系统，厦门宸天电子科技有限公司研发的 Supplate系列，深圳吉通电子有限公司研发的“车牌通”车牌识别产品、亚洲视觉科技有限公司研发的 VECON-VIS 自动识别系统等。也有很多高校在研究这个课题。国外相对的在这个方面开始的比较早，同时他们的车牌种类单一，字符简单，容易定位识别有关，取得不错的成就。

关于车牌识别的研究，虽然国内外学者已经作了大量的工作，但仍然存在一些问题。在车辆还比较新的时候，车牌上的字迹清晰，较容易识别，随着车龄越来越大，车子经过风吹雨淋，车牌难免受到一定程度的磨损，这样就会造成识别的难度。比如车牌图像的倾斜、车牌自身的磨损、光线的干扰都会影响到定位的精度。

车牌字符识别是在车牌准确定位的基础上，对车牌上的汉字、字母、数字进行有效确认的过程。目前已有的方法很多，但其效果与实际的要求相差很远，难以适应现代化交通系统高速度、快节奏的要求。因而对字符识别的进一步研究也同样具有紧迫性和必要性。

车牌及牌照图像：

流程图：



车牌规格

我国目前使用的汽车牌号标准是 2007 年开始实施的《中华人民共和国机动车号牌》GA36-2007（2010 年修订）。根据 GA36-2007 对机动车牌号编排规则规定，我国汽车的车牌构造特点如下：

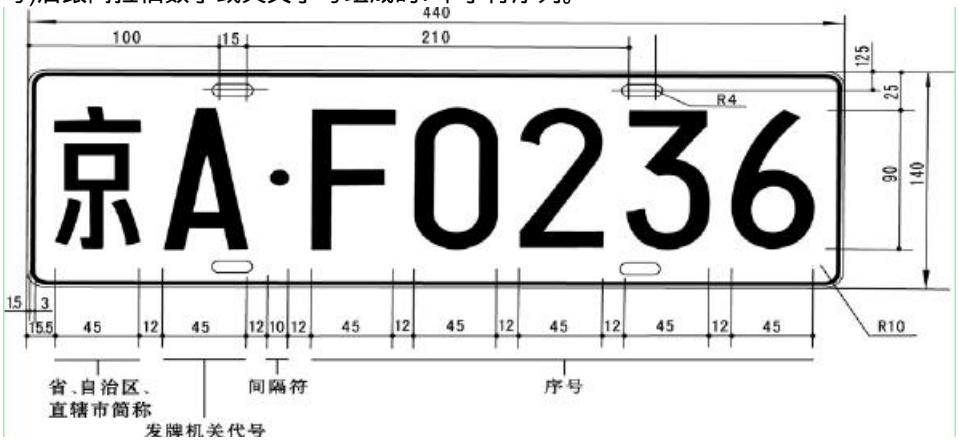
汽车车牌号的编排规则：我国的标准车辆车牌是由一个省份汉字（军警车牌为其他汉字）后跟字母或阿拉伯数字组成的 7 个字序列。标准车牌的的具体排列格式是：X1X2·X3X4X5X6X7，X1是各省、直辖市的简称或军警，X2是英文字母，代表该汽车所在地的地市代码，比如 A 代表省会，B 代表该省的第二大城市，C 代表该省的第三大城市，X3X4X5X6X7为英文字母或阿拉伯数字，2010年以前车牌号码的分布规律是，前面是字母，后面是数字。但是，随着车辆保有量的增加，每个字母所属号段越来越不够用。按照新《中华人民共和国机动车号牌》（2010 年修订）标准，将允许字母在后五位编码中任意一位出现，但不能超过两个。除了第一个汉字外，字母和数字的笔画在竖直方向都是联通的。

牌照类型如下图：

表 2.1 号牌的分类、规格、颜色及使用范围

序号	分类	外廓尺寸 mm×mm	颜色	数量	适用范围
1	小型汽车号牌	440×140	蓝底白字白线框	2	中型以下的载客、载货汽车和专项作业车
2	使馆汽车号牌		黑底白字，红“使”、“领”、字白框线		驻华使馆的汽车
3	领馆汽车号牌				驻华领事馆的汽车
4	港澳入出境车号牌		黑底白字，白“港”、“澳”字白框线		港澳地区入出内地的汽车
5	教练汽车号牌		黄底黑字，黑“学”字黑框线		教练用汽车
6	警用汽车号牌		白底黑字，红“警”字黑框线		汽车类警车

绝大部分的汽车牌照的宽度为1100px，高度为350px，牌照上一共有7个或8个字符，其中，每个字符的宽度为45mm，高度为90mm，间隔符的宽度为10mm，除了第二个和第三个字符之间的间距为 34mm 外，字符之间的间隔宽度12mm。民用的汽车牌照上有所属省、自治区或直辖市的简称(军用、警察牌为其他汉字)，监督机关及发证照的代号(大写的英文字母)后跟阿拉伯数字或英文字母组成的7个字符序列。



可以简单的归纳为以下特征：

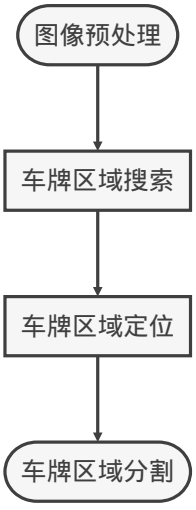
- 1、有两种或这三种颜色组成
- 2、车牌具有统一的标准尺寸 宽：高=3.14，这便于字符的分割和车牌的定位。字符的面积大约占整个车牌面积的20%
- 3、边缘特征：汽车的车牌边框是有规则的边缘，由于汽车车牌的字符排列规则，汽车车牌的垂直边缘比水平边缘更为丰富，而汽车的车身却有丰富的水平边缘，垂直边缘不明显。
- 4、黑白跳变特征：车牌区域二值化后，字符和背景为一黑一白，存在明显的黑白跳变，且跳变的次数在一定范围内。
- 5、投影特征：汽车车牌图像进行垂直投影后的图像是由波峰、波谷交替组成的连续分布图，垂直投影后的图像会有约七个波峰或波谷区；汽车车牌图像进行水平投影后的图像中灰度跳变的像素点数累加值很大。

    了解我国车牌的特征，有利于后续对车牌进行的各种操作，在项目步骤中，这属于对需求目标的全面了解。对车牌的了解不能跳过，因清楚的知道我们所要处理的的目标的各个特性，这样才有利于我们利用这些特性来操作车票图像。

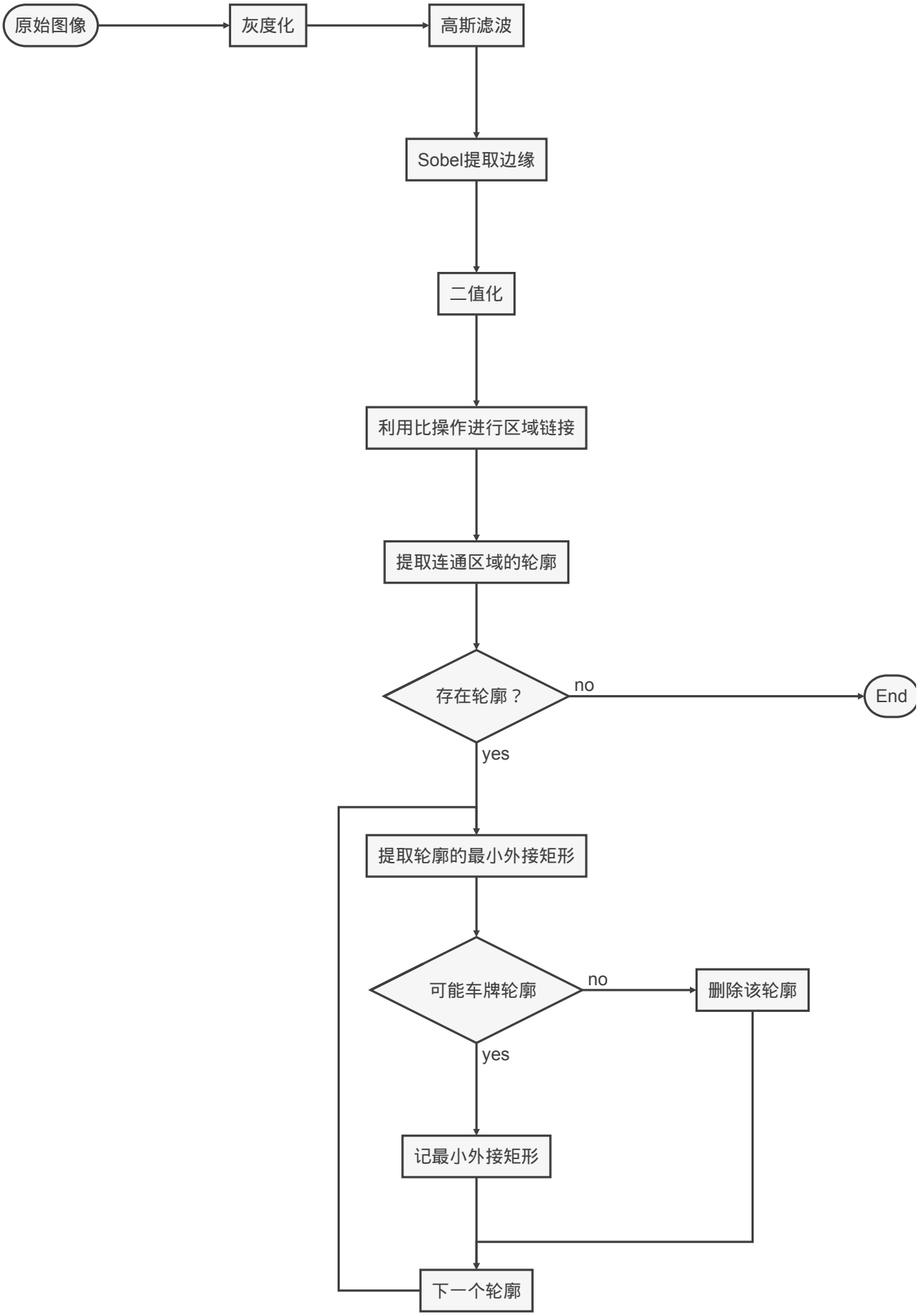
车牌定位：

车牌定位就是采用一系列图像处理或者数学的方法从一幅图像中将车牌准确地定位出来。车牌定位提取出的车牌是整个车牌识别系统的数据来源，它的效果的好坏直接影响到整个系统的表现，只有准确地定位出车牌，才会有后续的车牌分割与字符识别。目前车牌定位有两大类、基于灰度、基于彩色。

车牌定位流程图：



详细步骤：



灰度化：

```
//单通道灰度值做差，binary or disbinary
```

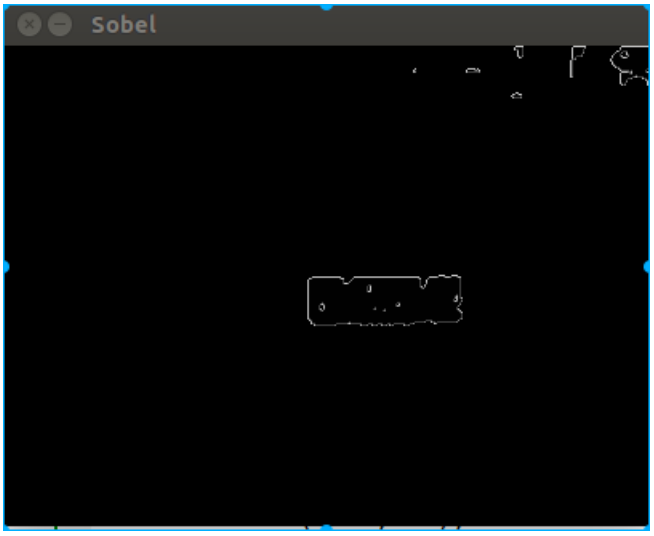
```
void subPixel(Mat &first,Mat &secend,int flag=DBINARY,int threshold=30){
    Point_<uchar> *p1;
    Point_<uchar> *p2;
    for(int row=0;row<first.size().height;row++){
        for(int col=0;col<secend.size().width;col++){
            p1=first.ptr<Point_<uchar> >(row,col);
            p2=secend.ptr<Point_<uchar> >(row,col);
            int fx,sx;
            fx=p1->x;
            sx=p2->x;
//            cout << fx << " " << sx  << " ";
            fx=fx-sx>0?fx-sx:0;
            if(flag==BINARY){
                fx=fx>threshold?255:0;
            }
            p1->x=fx;
        }
    }
//    p1=first.ptr<Point_<uchar> >(first.size().height-1,first.size().width-2);
//    p2=secend.ptr<Point_<uchar> >(secend.size().height-1,secend.size().width-2);
//    cout << (int)p1->x << (int)p2->x << endl;
}
```

高斯滤波

```
//Gauss平滑滤波
void Gauss(Mat origin,Mat &dst){
    int gauss[3][3];
    int sum_gauss=0;
    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            gauss[i][j]=pow(2,i+j);
            sum_gauss+=gauss[i][j];
        }
    }


    Point_<uchar> *p1;
    Point_<uchar> *p2;
    for(int row=0;row<origin.size().height;row++){
        for(int col=0;col<origin.size().width;col++){
            p2=dst.ptr<Point_<uchar> > (row,col);
            int sum=0;
            for(int i=-3/2;i<3/2;i++){
                for(int j=-3/2;j<3/2;j++){
                    p1=origin.ptr<Point_<uchar> > (row+i,col+j);
                    if(row+i>=0&&row+i<origin.size().height&&col+j>=0&&col+j<origin.size().width){
                        sum+=p1->x*gauss[i+1][j+1]/sum_gauss;
                    }
                }
            }
           p2->x=sum;
        }
    }
}
```

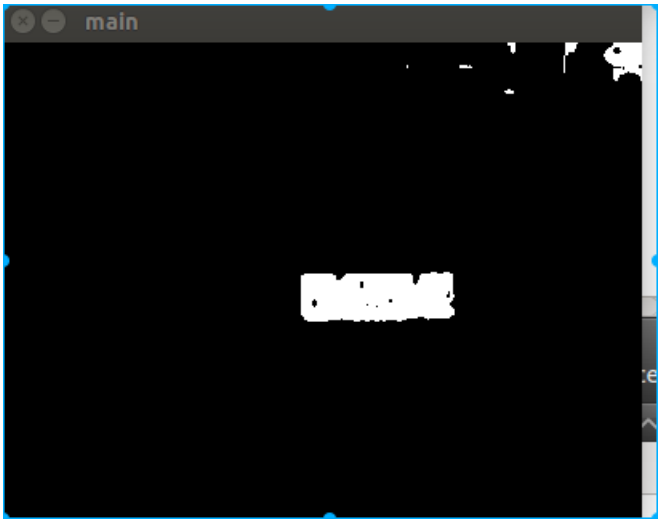
Sobel

```
//Sobel算子,一阶差分算子
void Sobel(Mat origin,Mat &dst,Grad grad=DIAGONAL,int threshold=30){
    int sobel[3][3];
    memset(sobel,0,sizeof(sobel));
    if(grad==HORIZONTAL){
        sobel[0][0]=-1;sobel[0][2]=1;
        sobel[1][0]=-2;sobel[1][2]=2;
        sobel[2][0]=-1;sobel[2][2]=1;
    }else if(grad==VERTICAL){
        sobel[0][0]=-1;sobel[0][1]=-2;sobel[0][2]=-1;
        sobel[2][0]=1;sobel[2][1]=2;sobel[2][2]=1;
    }else{
        sobel[0][0]=-2;sobel[0][1]=-1;
        sobel[1][0]=-1;
        sobel[1][2]=1;
        sobel[2][1]=1;sobel[2][2]=2;
    }
...
...
...
    if(_max==_min){
        _max=255;
        _min=0;
    }
    for(int row=0;row<origin.size().height;row++){
        for(int col=0;col<origin.size().width;col++){
            p2=dst.ptr<Point_<uchar> > (row,col);
            int sum=0;
            for(int i=-3/2;i<3/2;i++){
                for(int j=-3/2;j<3/2;j++){
                    p1=origin.ptr<Point_<uchar> > (row+i,col+j);
                    if(row+i>=0&&row+i<origin.size().height&&col+j>=0&&col+j<origin.size().width){
                        sum+=p1->x*sobel[i+1][j+1];
                    }
                }
            }
           p2->x=sum*255/(_max-_min);
        }
    }
}
```



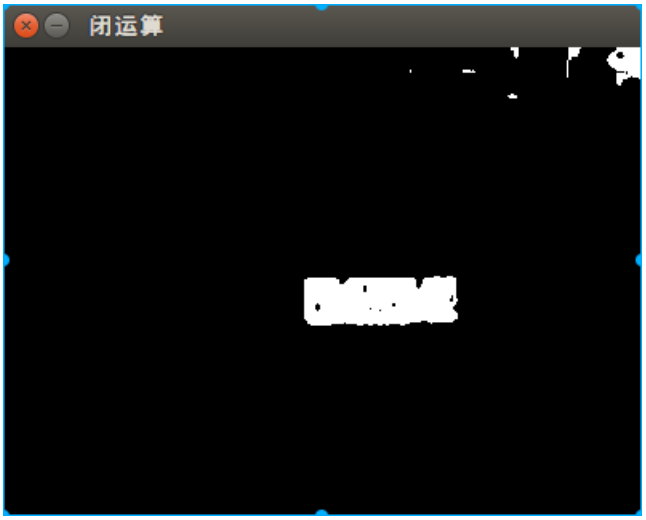
二值化

```
//二值化
void Threshold(Mat origin,Mat &dst,int threshold=30){
    Point_<uchar> *p1;
    Point_<uchar> *p2;

    for(int row=0;row<origin.size().height;row++){
        for(int col=0;col<origin.size().width;col++){
            p1=origin.ptr<Point_<uchar> > (row,col);
            p2=dst.ptr<Point_<uchar> > (row,col);
            if(p1->x>=threshold){
                p2->x=255;
            }else{
                p2->x=0;
            }
        }
    }
}
```



闭运算

```
//闭运算
void Closed(Mat origin,Mat &dst,int kernal1=KERNAL3,int kernal2=KERNAL3){
    Mat eImage=origin.clone(),dImage=origin.clone();
    dst=origin.clone();
    Erode(origin,eImage,kernal1);
    Dilate(eImage,dst,kernal2);
}
```

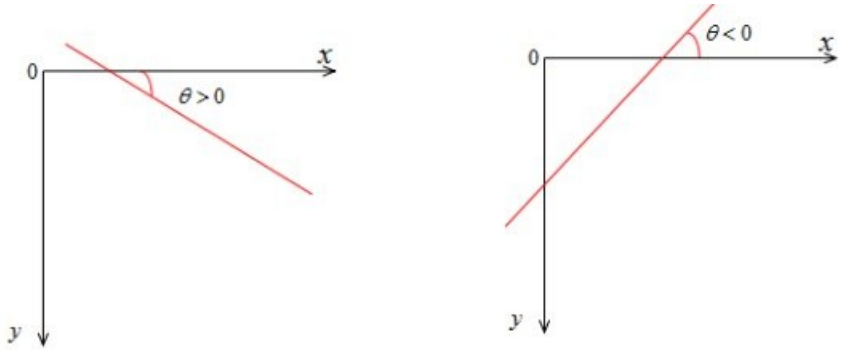


提取连通区域

```
//标记联通区,广度优先搜索,传入二值图，搜索过的标记0
void num_area(Mat origin,int &n){
    num_before=0;
    cout << "start" << endl;
    Mat temp=origin.clone();
    n=0;

    Point_<uchar> *p1;
    Point_<uchar> *p2;

    for(int i=0;i<origin.size().height;i++){
        for(int j=0;j<origin.size().width;j++){
            p1=temp.ptr<Point_<uchar> > (i,j);
            if(p1->x!=0){
                bfs(origin,temp,i,j);
                n++;
            }
        }
    }
    cout << n << endl;
    cout << "end" << endl;
}
```

剩下处理代码均在压缩文件当中(轮廓判断，矩形判断，车牌区域提取)

旋转至水平方向

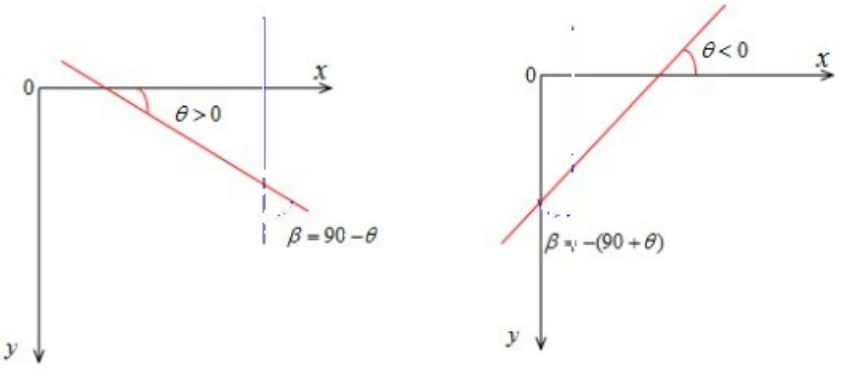


由上图知，$\theta>0$需要沿角度减小方向旋转，$\theta<0$需要沿角度增加方向旋转，因此，旋转矩阵为：

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \Rightarrow A(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

旋转至垂直方向

根据主轴方向求与 y 轴方向夹角，其中角度 β 为与 y 正方向最小夹角，如果位于 x 轴正方向，则 $\beta>0$，如果位于 x 轴正方向，则 $\beta<0$：



由上图知，$\beta>0$ 需要沿角度增大方向旋转，$\beta<0$ 需要沿角度减小方向旋转，因此，旋转矩阵为：

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \Rightarrow B(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

由于 $B(\theta)=A(-\theta)$，所以也可以采用 $A(\theta)$，只需要 $\beta=-\beta$ 即可：

$$\begin{cases} \beta = -(90-\theta), \theta > 0 \\ \beta = (90+\theta), \theta < 0 \end{cases}$$

求连通域最小外接矩形长宽比

1）求主轴方向 θ；

2）根据旋转至水平或垂直方向，选择合适的旋转矩阵：

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

3）求取连通域边界坐标或全部坐标： vector<CPoint>&Boundry

4）求最小外接矩形长宽比

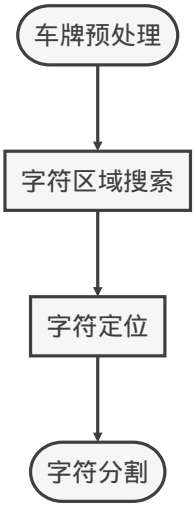
```
//bfs广度优先搜索
void bfs(Mat apple,Mat &origin,int x,int y){
    while(!q.empty()){
                .......
                ......
                ......
    }

    if(num<2500){
        num_before=num;
        return;
    }


    cout << "x: " << x_std << " y: " << y_std << " num: " << num <<endl;
    x_std=x_std/num*times;//col
    y_std=y_std/num*times;//row

    cout << "x_std:" << x_std << " y_std:" << y_std << endl;

    //计算m00_std,m11_std,m20_std,m02_std;
    double m11_std=0,m00_std=0,m20_std=0,m02_std=0;//中心矩

    m00_std=coordinate.size();
    cout << "m00_std^2: " << m00_std*m00_std << ' ' << num*num << endl;

    for(int i=0;i<coordinate.size();i++){
```

```
        m11_std+=(coordinate[i].first-y_std)*(coordinate[i].second-x_std);
        m20_std+=(coordinate[i].second-x_std)*(coordinate[i].second-x_std);
        m02_std+=(coordinate[i].first-y_std)*(coordinate[i].first-y_std);
    }
                    .......
                    .......
                    .......
                    .......
    //逆变换
    double temp_x,temp_y;
//    cout << "逆变化：" << endl;
    temp_x=a.x;temp_y=a.y;
    a.x=costheta*temp_x-sintheta*temp_y;
    a.y=sintheta*temp_x+costheta*temp_y;

    temp_x=b.x;temp_y=b.y;
    b.x=costheta*temp_x-sintheta*temp_y;
    b.y=sintheta*temp_x+costheta*temp_y;

    temp_x=c.x;temp_y=c.y;
    c.x=costheta*temp_x-sintheta*temp_y;
    c.y=sintheta*temp_x+costheta*temp_y;
    ......
    ......
    ......

    if(distance(a.x,a.y,b.x,b.y)>distance(b.x,b.y,c.x,c.y)){
        if(distance(a.x,a.y,b.x,b.y)/distance(b.x,b.y,c.x,c.y)
<2.5&&distance(a.x,a.y,b.x,b.y)/distance(b.x,b.y,c.x,c.y)>3){
            return;
        }
    }else{
        if(distance(b.x,b.y,c.x,c.y)/distance(a.x,a.y,b.x,b.y)
<2.5&&distance(c.x,c.y,b.x,b.y)/distance(b.x,b.y,a.x,a.y)>3){
            return;
        }
    }
    if(num/distance(b.x,b.y,c.x,c.y)*distance(a.x,a.y,b.x,b.y)<0.9){
        return;
    }
```

结果：

字符分割

字符分割算法流程同车牌分割不再详述



定位

分割