北 京 林 业 大 学

2021 学年一 2022 学年第 一 学期 数字视频技术及应用

课设报告书

专　　业：数字媒体技术　　班　　级：数媒 19-1

姓　　名：陈秋莹　　学　　号：191002426

实验地点：计算中心　　任课教师：刘文萍

实验题目：视频车牌识别

实验环境：VS+OpenCV

实验内容:

编程实现视频中车牌号的识别。输入为含有车辆车牌的视频，经过图像预处理、车牌定位、字符分割和字符识别一系列处理，输出为车牌号。

实验内容指导

需要实现的具体任务包括：

1. 在视频序列中提取出含有车辆的图像。
2. 图像预处理。采集的视频是彩色视频，环境因素、硬件设备等原因会导致视频抖动或图像模糊，其背景和噪声会严重影响字符的准确分割和识别，经过图像预处理后，可提高图像质量。
3. 车牌区域提取 。对经过预处理后的车牌的二值图像运用形态学进行滤波，使得车牌区域能够形成一个连通区域，然后以车牌的先验知识（如大部分车牌为蓝色等）为依据筛选所得到的连通区域，进而获得车牌区域的准确位置，最后完成从图片中提取车牌的任务。 另外，可选择采用支持向量机（SVM）的二分类算法对车牌和非车牌区域进行识别。
4. 车牌字符分割。车牌分割的过程首先对车牌图片进行水平方向的投影，去除水平边框，然后再进行垂直方向的投影。通过分析车牌投影可以得知，投影中最大峰值所对应的是车牌中的第二个字符和第三个字符之间的间隔，第二大峰值中心距离对应的是车牌字符的宽度，以此类推就可以对车牌字符进行分割。另外，可选择采用形态学轮廓检测方法进行字符分割。
5. 车牌字符识别。字符识别的方法有很多种，模板匹配方法是应用最广泛的，在进行识别的过程中，要先建立标准字库，然后将分割所得到的字符进行分类，将分类后的字符与标准字库中的字符进行比较，最后以误差最小的字符作为结果显示出来。 另外，可选择采用支持向量机（SVM）、K-近邻算法方法进行字符识别。

实现方法:

1. 在视频序列中提取出含有车辆的图像&图像预处理。

根据先验知识，中国车牌为长 45cm,宽 15cm 的矩形车牌，长宽比 3:1。车牌颜色通常为蓝底白字、黄底黑字等，因此我受到一篇论文的启发[1]，首先采用基于 HSV 颜色模型的方法对车牌定位，从而进一步判断，提取含有车辆的图像。

表 3-1 HSV 的取值范围，“\”表示不予考虑的项目。

	蓝色	黄色	白色	黑色
色调	200~255 度	25~55 度	\	\
饱和度	0.4~1	0.4~1	0~0.1	\
亮度	0.3~1	0.3~1	0.9~1	0~0.35

这里有一个 RGB 转 HSV 的问题。图像以 8 位 3 通道的形式保存，每个颜色分量占用 8 位，值的范围为 0 到 255，而 HSV 模型各分量的取值范围为：H 为 0 到 360°，S 为 0 到 100%，V 为 0 到 255，因此需要先更改图像的数据格式为 32 位 float 3 通道。转换代码如下：

```
//调整格式32fc3
img.convertTo(BGRimg, CV_32FC3, 1.0 / 255, 0);
//转换hsv图
cvtColor(BGRimg, HSVimg, COLOR_BGR2HSV);
```

转换成 HSV 后，根据蓝色车牌 H 在 200~255，S 在 0.4~1.0，V 在 0.3~1.0 的范围内过滤，黄色车牌同理。筛选代码如下：

```
for (int i = 0; i < imageHeight; ++i) {
    uchar* ptr = regionImg.ptr<uchar>(i);
    for (int j = 0; j < imageWidth; ++j)
    {
        float pixelH = HSVimg.at<Vec3f>(i, j)[0];  //读取通道值
        float pixelS = HSVimg.at<Vec3f>(i, j)[1];
        float pixelV = HSVimg.at<Vec3f>(i, j)[2];

        //颜色定位图像--蓝色
        if (pixelH > 200.0 && pixelH < 255.0 )
        {
            if (pixelS > 0.4 && pixelS < 1.0 && pixelV > 0.3 && pixelV < 1.0)
                    ptr[j] = 255;
        }
        //黄色车牌
        else if (pixelH > 25.0 && pixelH < 55.0)
        {
            if (pixelS > 0.4 && pixelS < 1.0 && pixelV > 0.3 && pixelV < 1.0)
                ptr[j] = 255;
        }
        else
        {
            ptr[j] = 0;
        }
    }
}
```

但是后来还了解到另外一种筛选方式：inRange，输入要处理的图像、HSV 最小值和最大值，输出是一个筛选后的二值图像。效果和上方代码相同。

```
//inRange(HSVimg, Scalar(200, 0.4, 0.3), Scalar(255, 1, 1), regionImg);
```

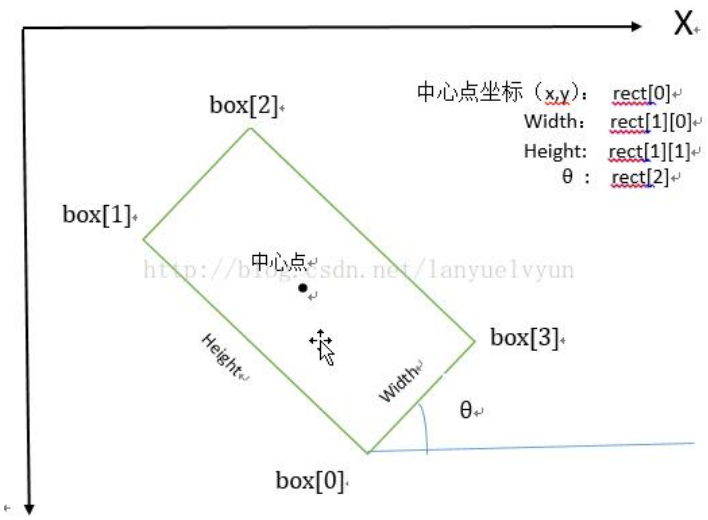
基于颜色的初步过滤的效果如下：

然后采用基于形状的方法。利用 OpenCV 的 findContours 函数检测物体的轮廓，通过判断物体的面积大小、长宽比来提取含车辆的帧图像。findContours 函数的输入为二值图像，黑色为背景，白色为目标，该函数会修改原图像。

先进行形态学处理，再提取轮廓，对提取的每个轮廓绘制包围矩形。有 2 种矩形：boundingRect 最小正矩形和 minAreaRect 最小斜矩形，如下图所示，绿色是 boundingRect，红色是 minAreaRect。

在这里我使用的是最小斜矩形 RotatedRect minAreaRect(InputArray points)，旋转角度θ是水平轴（x 轴）逆时针旋转，与碰到的矩形的第一条边的夹角。并且这个边的边长是 width，另一条边边长是 height。也就是说，在这里，width 与

height 不是按照长短来定义的。



可以计算这个矩形的长、宽，再计算面积、长宽比，分析区域的形状特征能不能达到车牌的比例。限制范围如下：

```
#define MIN_CONTOUR_RATIO  2.0
#define MAX_CONTOUR_RATIO  5.0
#define MIN_CONTOUR_AREA  1000.0
#define MAX_CONTOUR_AREA  50000.0
```

3. 车牌区域提取。

如果某个矩形满足以下判定条件，就视作车牌，对车牌进行位置矫正。

```
rectArea = axisLong * axisShort;
rectDegree = area / rectArea;
//体态比or长宽比 最下外接矩形的长轴和短轴的比值
long2Short = axisLong / axisShort;
if (long2Short > MIN_CONTOUR_RATIO && long2Short < MAX_CONTOUR_RATIO && rectDegree > 0.63)
{
```

矫正图片也有两种方法。

第 1 种是获取矩形的四个角点，使用函数直接从图像中扭曲矩形，代码如下：

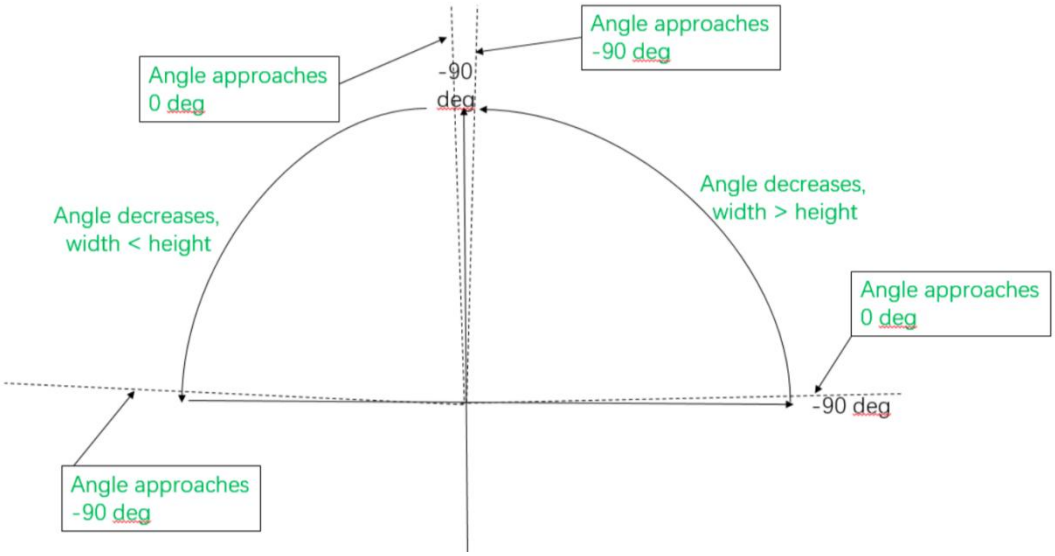
```
//自己设置一个ROI
Mat cutImg = (Mat_<float>(4, 2) << 0, height - 1,
    0, 0,
    width - 1, 0,
    width - 1, height - 1);
cout << "裁剪后的车牌"<<cutImg << endl;
Mat MP = getPerspectiveTransform(boxPtsMat, cutImg);
warpPerspective(srcFrame, rotImg, MP, Size(width, height));
```

需要注意的是：如果矩形垂直放置（宽度<高度），则检测到的角度为 -90。如果矩形水平放置，则检测到的角度也是 -90 度。

如果矩形的顶部位于第一象限中，则当矩形从水平位置旋转到垂直位置时，检测到的角度减小，直到检测到的角度变为 -90 度。在第一象限中，检测到的矩形的宽度大于其高度。

如果检测到的矩形的顶部位于第二象限中，则当矩形从垂直位置旋转到水平位置时，角度会减小。但是第二象限和第一象限之间是有区别的。如果矩形接近垂直位置，但尚未处于垂直位置，则其角度接近 0。如果矩形接近水平位置但尚未处于水平位置，则其角度接近 -90 度。

两个象限的矩形旋转方向不同，需要转换顶点顺序，否则矫正图像就是颠倒的。



第 2 种是根据矩形偏转的角度，进行逆旋转。

```
//方法2
//旋转纠正
//Point center = Point(imageWidth / 2, imageWidth / 2);
////获得变换矩阵
//Mat rot(2, 3, CV_32FC1);
//rot = getRotationMatrix2D(center, degree, 0.8);//getRotation
//warpAffine(srcFrame, srcFrame, rot, srcFrame.size());
```

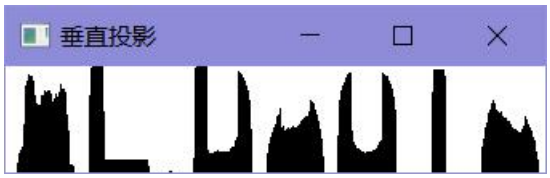
效果如下：

4. 车牌字符分割。

首先对车牌进行灰度化、二值化处理，如果是黄底黑字的车牌二值化后还需要反色，保证二值化后是黑底白字。结果如下所示：

再通过行扫描去掉车边框，结果如下图所示。

然后用垂直投影分割字符，垂直投影是二维图像在 x 轴上的投影，整体上看黑色集中的区域就是一个字符，然后 2 个白色区域之间黑色区域就是一个字符，利用这个特点，分割出每个字符。



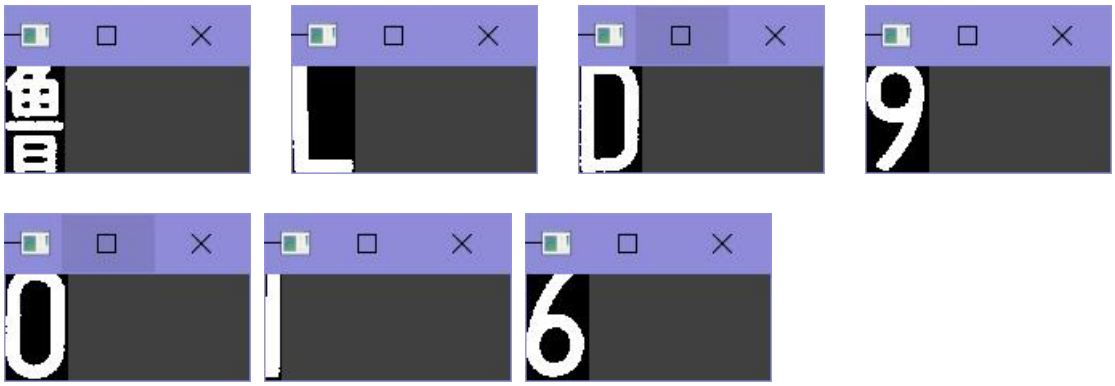
生成垂直投影的代码为：

```
for (int col = 0; col < width; col++)
{
    for (int row = 0; row < height; row++)
    {
        perPixelValue = srcImg.at<uchar>(row, col);
        if (perPixelValue == 0)
        {
            projectValArry[col]++;
        }
    }
}
```

```
Mat verticalProjectionMat(height, width, CV_8UC1);//垂直投影的画布
for (int i = 0; i < height; i++)
{
    for (int j = 0; j < width; j++)
    {
        perPixelValue = 255;  //背景设置为白色
        verticalProjectionMat.at<uchar>(i, j) = perPixelValue;
    }
}
for (int i = 0; i < width; i++)//垂直投影直方图
{
    for (int j = 0; j < projectValArry[i]; j++)
    {
        perPixelValue = 0;  //直方图设置为黑色
        verticalProjectionMat.at<uchar>(height - 1 - j, i) = perPixelValue;
    }
}
imshow("垂直投影", verticalProjectionMat);
waitKey(0);
vector<Mat> roiList;//用于储存分割出来的每个字符
int startIndex = 0;//记录进入字符区的索引
int endIndex = 0;//记录进入空白区域的索引
bool inBlock = false;//是否遍历到了字符区内
for (int i = 0; i < srcImg.cols; i++)//cols=width
{
    if (!inBlock && projectValArry[i] > 0)//进入字符区
    {
        inBlock = true;
        startIndex = i;
    }
    else if (projectValArry[i] == 0 && inBlock)//进入空白区
```

分割的结果如下：

```
//先进行垂直投影
vector<Mat> characters = verticalProjectionMat(noboundImg);
cout << "字符" << characters.size() << endl;
for (int i = 0; i < characters.size(); i++)
{
    imshow("垂直投影", characters[i]);
    cv::waitKey(6000);
}
```



5. 车牌字符识别。

车牌上面有 8 个字符，含汉字、大写字母 A~Z、圆点、数字 0~9。
识别方法一：模板匹配

模板匹配是通过滑框在采集到的原图像上进行滑动寻找与模板图像相似的目标。模板匹配是基于图像的灰度值匹配。将滑框在图像上以一定的像素步长滑动将窗口内的图像灰度矩阵与参考模板按照设定的相似度度量算法进行搜索比较。为了利用模板匹配从源图像中得到匹配区域，从源图像选取该区域作为进行匹配的模板。模板从源图像左上角开始每次以一个像素点为单位进行移动，每到达一个位置，就会计算模板矩阵和源图像当前位置矩阵匹配的“好”“坏”程度即两个矩阵的相似程度。

OpenCV 中提供了 matchTemplate()并配合 minMaxLoc()函数实现图像的模板匹配过程。通过 matchTemplate()函数根据输入模板搜寻输入图像中与模板相似的地方，获得匹配结果图像。通过 minMaxLoc()函数来找到最大值和最小值。

a. **平方差匹配 method=CV_TM_SQDIFF**

这类方法利用平方差来进行匹配，最好匹配为0.匹配越差，匹配值越大.

$$R(x,y)=\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2$$

b. **标准平方差匹配 method=CV_TM_SQDIFF_NORMED**

$$R(x,y)=\frac{\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}}$$

c. **相关匹配 method=CV_TM_CCORR**

这类方法采用模板和图像间的乘法操作，所以较大的数表示匹配程度较高，0标识最坏的匹配效果.

$$R(x,y)=\sum_{x',y'}(T(x',y')\cdot I(x+x',y+y'))$$

d. **标准相关匹配 method=CV_TM_CCORR_NORMED**

$$R(x,y)=\frac{\sum_{x',y'}(T(x',y')\cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}}$$

e. **相关匹配 method=CV_TM_CCOEFF**

这类方法将模版对其均值的相对值与图像对其均值的相关值进行匹配，1表示完美匹配，-1表示糟糕的匹配，0表示没有任何相关性（随机序列）.

$$R(x,y)=\sum_{x',y'}(T'(x',y')\cdot I(x+x',y+y'))$$

在这里

$$T'(x',y')=T(x',y')-1/(w\cdot h)\cdot\sum_{x'',y''}T(x'',y'')$$
$$I'(x+x',y+y')=I(x+x',y+y')-1/(w\cdot h)\cdot\sum_{x'',y''}I(x+x'',y+y'')$$

f. **标准相关匹配 method=CV_TM_CCOEFF_NORMED**

$$R(x,y)=\frac{\sum_{x',y'}(T'(x',y')\cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'}T'(x',y')^2\cdot\sum_{x',y'}I'(x+x',y+y')^2}}$$

代码应用如下，使用了相关匹配算法：

```
if (temp.size() == 0) return;
int maxscore = 0, maxindex = 0;
Mat resultMat;
remove_border(image, image);
resize(image, image, Size(20, 40));
for (int i = 0; i < temp.size(); i++)
{
    matchTemplate(image, temp[i], resultMat, CV_TM_CCOEFF);
    //进行匹配和标准化
    double minValue;
    double maxValue;
    //location point
    Point minLocP;
    Point maxLocP;
    //通过函数 minMaxLoc 定位最匹配的位置，越大越好
    minMaxLoc(resultMat, &minValue, &maxValue, &minLocP, &maxLocP, Mat());
    if (maxValue > maxscore)
    {
        maxscore = maxValue;
        maxindex = i;
    }
}
result = templabel[maxindex];
myvalue = to_string(maxscore);
```

识别方法二：SVM

支持向量机（support vector machines，SVM）是一种用来分类的机器学习算法，对字符分类要用多类分类的 SVM 方法。

基本流程就是输入一组带有标签（label）的特征进行模型训练（train），保存这个训练模型（save），然后再读这个模型（read），输入要预测的特征，就能得到预测(predict)的结果。

OpenCV 中有相应的 API（官方文档跳转）。Ptr<SVM> model 参数设置为 CvSVM::C_SVC 类型，该类型可以用于 n-类分类问题。调用函数 CvSVM::train 来建立 SVM 模型。根据资料，把图像的 HOG 特征作为输入。

HOG 的计算：

```
HOGDescriptor *hog;
hog->compute(inMat, descriptors, Size(1, 1), Size(0, 0));
```

```
{
    featruemodel = feature;
    model = cv::ml::SVM::create();
    model->setType(SVM::C_SVC);
    model->setKernel(SVM::LINEAR);
    model->setDegree(0);
    model->setGamma(1);
    model->setCoef0(0);
    model->setC(1);
    model->setNu(0);
    model->setP(0);
    model->setTermCriteria(CvTermCriteria(CV_TERMCRIT_ITER, 1000, 0.01));
    hog = new HOGDescriptor(Size(20, 40), Size(15, 15), Size(5, 5), Size(5, 5), 9);
```

对于切割出来的字符，汉字和数字、字母相比形态更加复杂，为了避免识别错误，汉字一个模型，字母数字一个模型分别训练。

```
NumModelHOG.init(HOG);
NumModelHOG.train(NumImages, NumLabels);
NumModelHOG.savemodel("nummodel-hog.xml");
ProvinceModelHOG.init(HOG);
ProvinceModelHOG.train(ProvinceImages, ProvinceLabels);
ProvinceModelHOG.savemodel("provincemodel-hog.xml");
```

预测函数：Model.predict(letterMat, predictStr)；预测结果就是 predictStr，然后把预测结果的 string 转换成 Cstring，显示到用户界面上。

6. 把程序转到 MFC 中

OpenCV 的图像 Mat 在控制台用 imshow 就可以显示，但是在 MFC 中要用 StretchDIBits 函数获取 Mat 图像数据转换成 BITMAPINFO 图像数据，再显示到 MFC 的 picture 控件中：

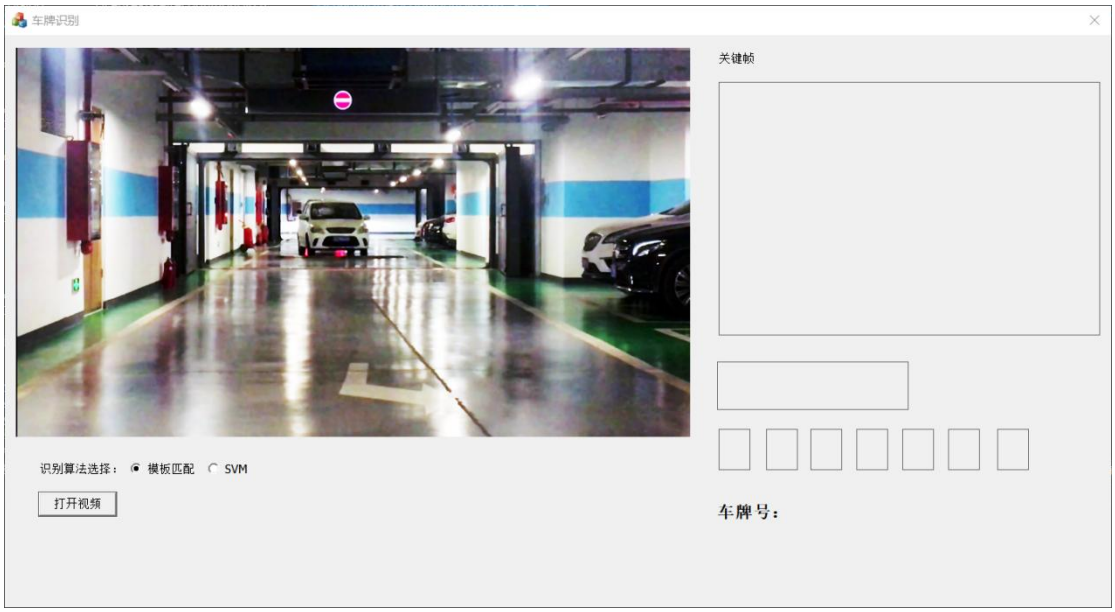
```
StretchDIBits(pDC->GetSafeHdc(), 0, 0, rect.Width(), rect.Height(), 0, 0, rect.Width(), rect.Height(), img.data, (BITMAPINFO*)m_bmi, DIB_RGB_COLORS, SRCCOPY);
```

```cpp
//在窗口显示视频
void CLicensePlateRecognitionDlg::DrawcvMat(cv::Mat m_cvImg, UINT ID)
{
    cv::Mat img;
    CRect rect;

    GetDlgItem(ID)->GetClientRect(&rect);
    if (rect.Width() % 4 != 0){ ... }

    cv::Rect dst(rect.left, rect.top, rect.right, rect.bottom);
    cv::resize(m_cvImg, img, cv::Size(rect.Width(), rect.Height()));  //使图像适应控件大小

    unsigned int m_buffer[sizeof(BITMAPINFOHEADER) + sizeof(RGBQUAD) * 256];
    BITMAPINFO* m_bmi = (BITMAPINFO*)m_buffer;
    BITMAPINFOHEADER* m_bmih = &(m_bmi->bmiHeader);
    memset(m_bmih, 0, sizeof(*m_bmih));
    m_bmih->biSize = sizeof(BITMAPINFOHEADER);
    m_bmih->biWidth = img.cols;   //必须为4的倍数
    m_bmih->biHeight = -img.rows; //在自下而上的位图中 高度为负
    m_bmih->biPlanes = 1;
    m_bmih->biCompression = BI_RGB;
    m_bmih->biBitCount = 8 * img.channels();

    if (img.channels() == 1)  //当图像为灰度图像时需要设置调色板颜色
    {
        for (int i = 0; i < 256; i++)
        {
            m_bmi->bmiColors[i].rgbBlue = i;
            m_bmi->bmiColors[i].rgbGreen = i;
            m_bmi->bmiColors[i].rgbRed = i;
            m_bmi->bmiColors[i].rgbReserved = 0;
        }
    }

    CDC* pDC = GetDlgItem(ID)->GetDC();
    ::StretchDIBits(pDC->GetSafeHdc(), 0, 0, rect.Width(), rect.Height(), 0, 0, rect.Width()
    ReleaseDC(pDC);
}
```

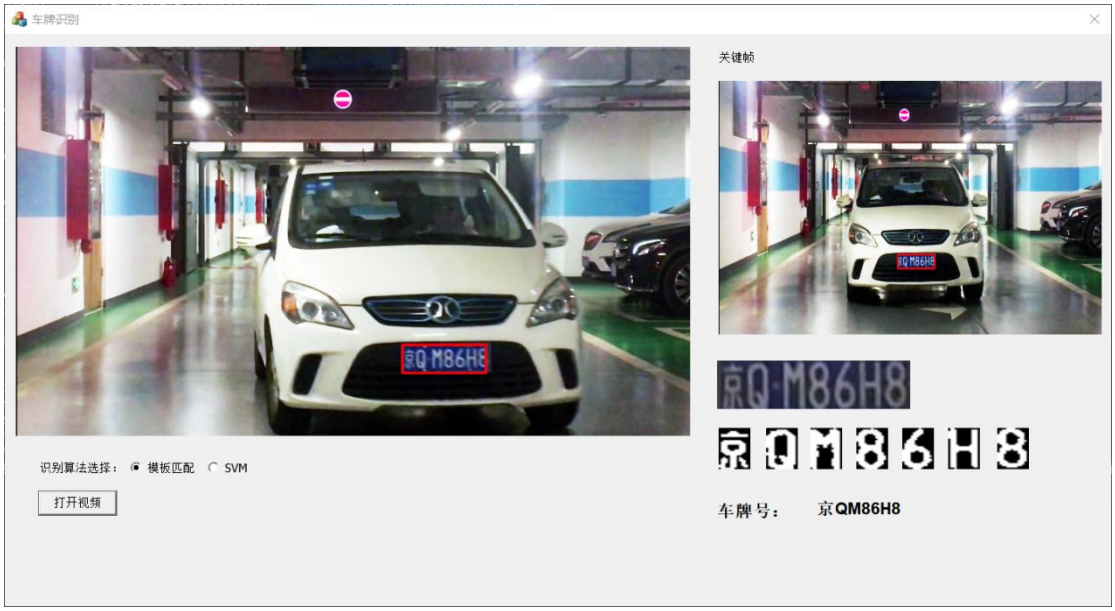
实验结果：

一、模板匹配模式

点击打开视频。当没有符合要求的结果时，显示如下：



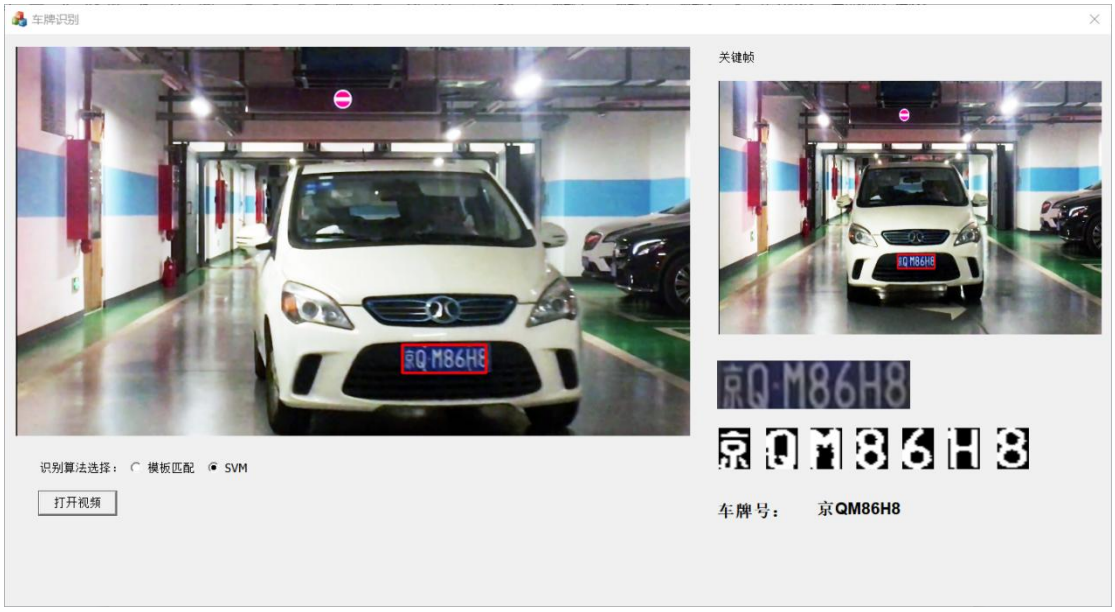
当检测到车牌并成功切割出七个字符时，右侧就会显示关键帧、车牌截取、字符分割的结果，最后成功显示识别的车牌号京 Q M86H3。



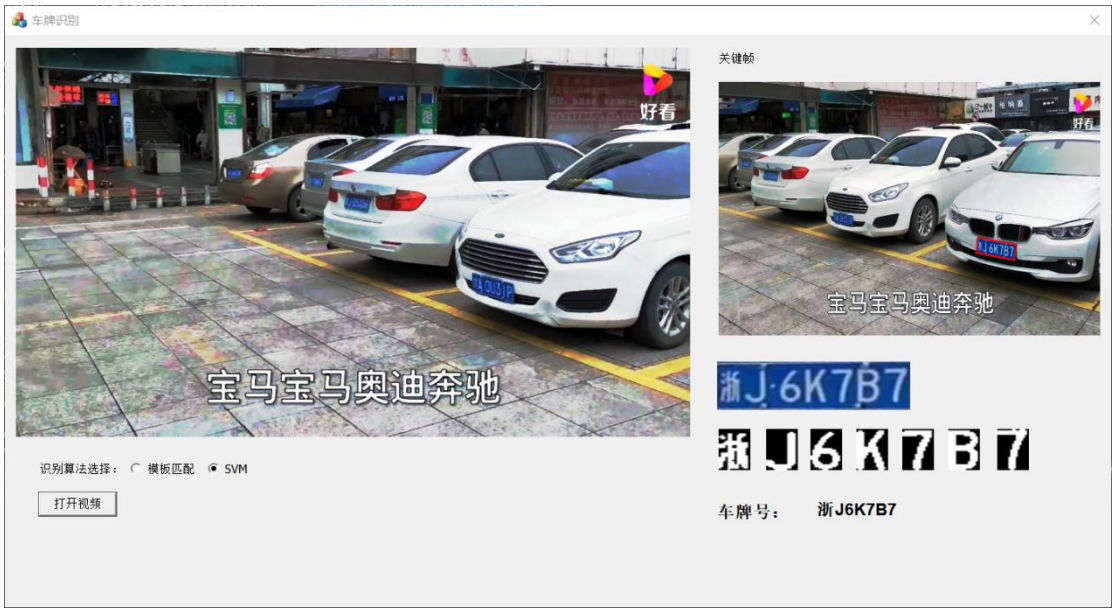
但是在测试其他视频的时候汉字识别错误，把浙识别成湘，因为这两个字偏旁相似，有概率出现错误。

二、SVM 模式

京 Q M86H3 识别成功。



测试其他视频，浙 J 6K7B7 识别正确，同时可以看到车牌的矫正效果比较好：



总结

识别车牌是一个处理图像、分析图像最后理解图像的过程，其中运用了大量的图像处理的知识，比如灰度化、阈值分割、形态学处理、连通性分析等，这也是我第一次尝试分割字符，垂直投影是一种巧妙的分割方法，让我感触很深。

效果方面，我认为最重要的在车牌定位、提取车牌这一步，在查找资料的时候看到了有很多识别车牌的论文，方法多种多样。如果能提取到符合规范的车牌，切割字符以及识别都不会有大问题。在测试中，可以看到模板匹配的准确率没有 SVM 高，比如在实验结果中模板匹配把“浙”识别成“湘”，但是 SVM 是准确的，可以体现出 SVM 的优势。

如果没有 OpenCV，我没有办法轻松地使用模板匹配、SVM 算法，OpenCV 功能比我想象的更强大，在课设中不断学习新的知识，我感觉到我还有很多东西

需要学习。

参考资料：

使用OpenCV从图像中裁剪旋转的矩形 - jdhao的博客
利用OPenCV在MFC中显示图像
HOG
[1] 陈永超. 基于数字图像处理的车牌识别研究[D]. 武汉理工大学, 2006.
[2] 张引, 潘云鹤. 彩色汽车图象牌照定位新方法[J]. 中国图象图形学报：A 辑, 2001, 6(4):4.