北京交通大學

《数字图像处理》实验 4

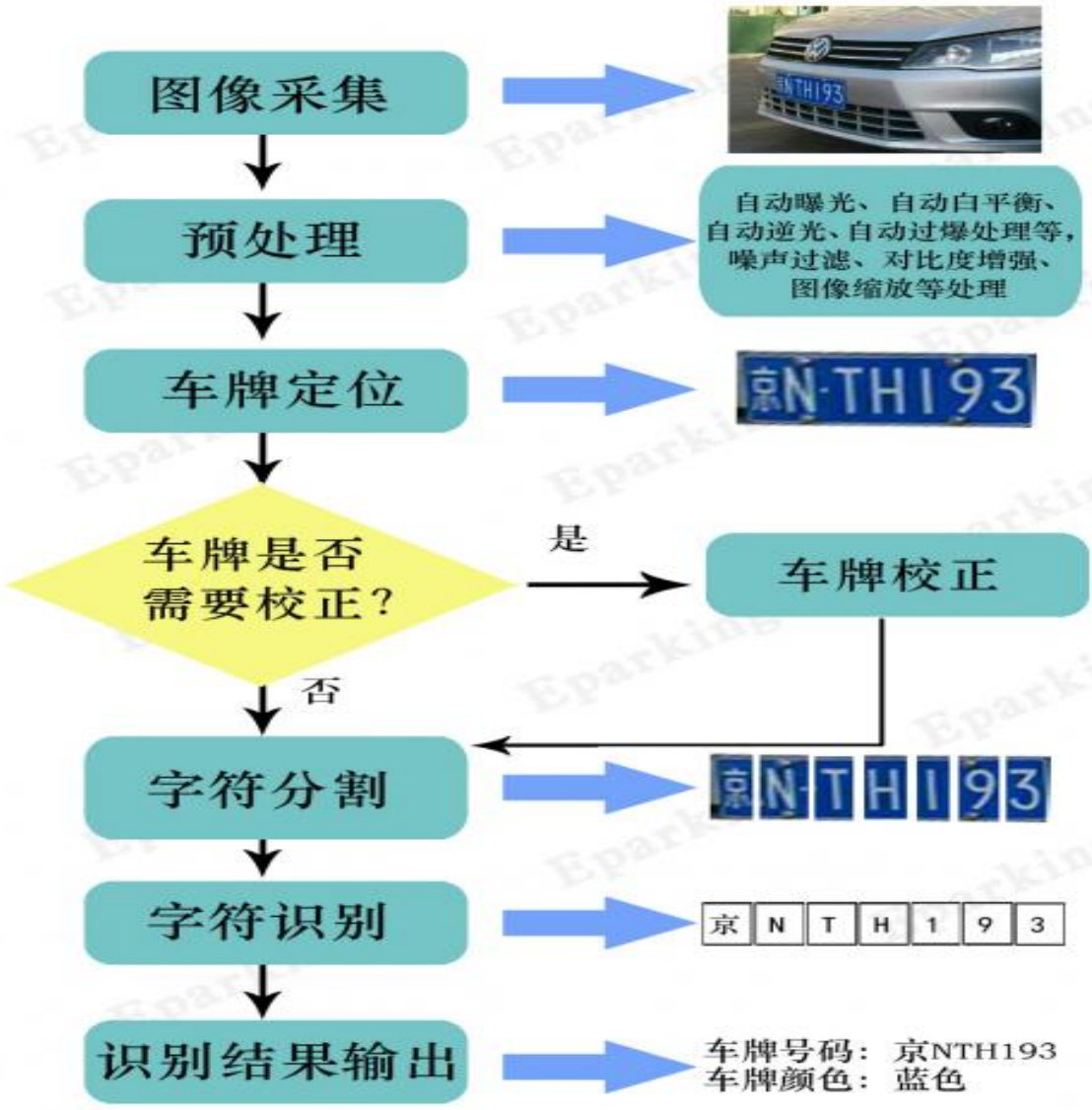
学　　院：	计算机与信息技术学院
专　　业：	计算机科学与技术
学生姓名：	麻锦涛　　谭天云
学　　号：	16281262　　16281048

目录

摘 要

这里我们小组自主选题选择做一个小型的车牌识别系统。整个车牌识别系统的功能分为车牌提取、倾斜校正、字符分割、字符识别四大部分。现有研究算法的理论基础上，在Visual C ++平台下，采集到的图像经过一系列预处理；在车牌定位模块中，采用了二值化方法的基础上，利用灰度跳变的特点，检测到车牌的上下边界和左右边界。至此，车牌完全定位出来；在车牌图像的倾斜矫正模块中，通过对图像左右半边目标像素平均高度的统计来进行倾斜的调整；经过倾斜校正和去噪处理后，字符分割部分采用一种基于相邻字符最大间隔宽度的方法来对车牌中的字符进行分割。识别部分主要提取了样本特征，然后进行模板匹配。

关键词：车牌定位，倾斜校正，字符分割、字符识别



1 车牌识别的研究背景：意义、应用

1.1 意义

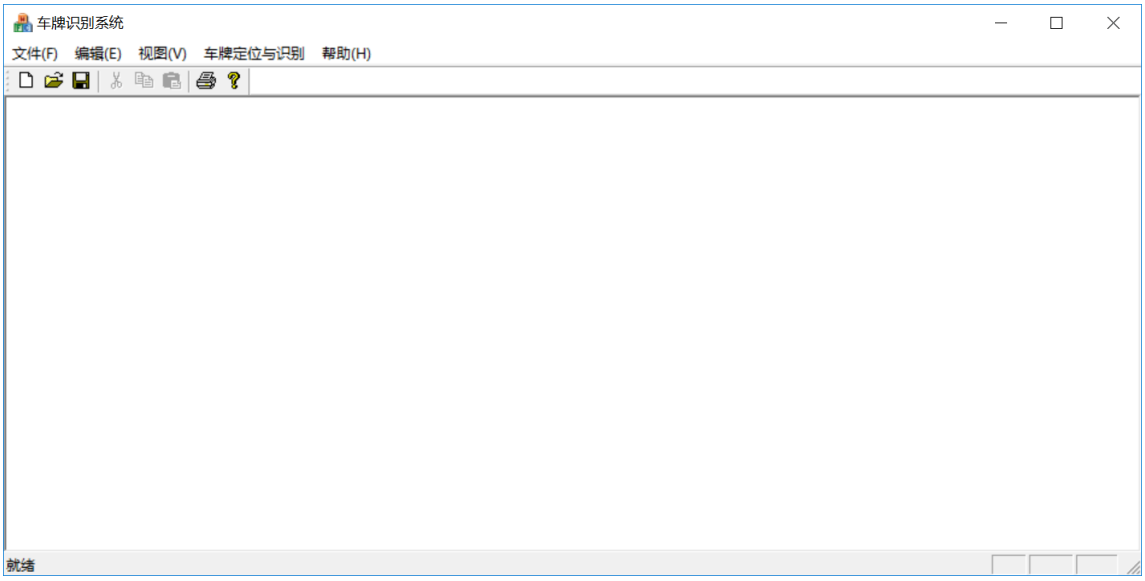
车牌识别技术要求能够将运动中的汽车牌照从复杂背景中提取并识别出来，通过车牌提取、图像预处理、特征提取、车牌字符识别等技术，识别车辆牌号，目前的技术水平为字母和数字的识别率可达到 96%，汉字的识别率可达到 95%

1.2 应用

车牌识别在高速公路车辆管理中得到广泛应用，电子收费（ETC）系统中，也是结合 DSRC 技术识别车辆身份的主要手段。

在停车位管理中，车牌识别技术也是识别车辆身份的主要手段。在深圳市公安局建设的《停车库（场）车辆图像和号牌信息采集与传输系统技术要求》中，车牌识别技术成为车辆身份识别的主要手段。

车牌识别技术结合电子不停车收费系统（ETC）识别车辆，过往车辆通过道口时无须停车，即能够实现车辆身份自动识别、自动收费。在车场管理中，为提高出入口车辆通行效率，车牌识别针对无需收停车费的车辆（如月卡车、内部免费通行车辆），建设无人值守的快速通道，免取卡、不停车的出入体验，正改变出入停车场的管理模式。 如图是我们开发的一款车牌识别系统。



2 程序的实现方法：

2.1 车牌提取：

在预处理基础上，检测出已经转化为白色的部分，然后划定白色区域的范围，最终检测出车牌的上下左右边界位置，将其提取出来。

【程序如下】

```
void CPlateIdentifyDoc::OnPlateextract()
{
    // TODO: 在此添加命令处理程序代码
    //读入原始图像
    IplImage *pSrc_Image=NULL;
    pSrc_Image=cvCreateImage(cvGetSize(src_image),src_image->depth,src_image->nChannels);
    pSrc_Image=cvCloneImage(src_image);

    //将图像转化为灰度图像
    gray_image=cvCreateImage(cvGetSize(pSrc_Image),8,1);
    cvCvtColor(pSrc_Image,gray_image,CV_BGR2GRAY);


    //灰度拉伸
    //定义拉伸后的图像
    IplImage * pStretch_Image=NULL;
    pStretch_Image=cvCreateImage(cvGetSize(pSrc_Image),8,1);
    m_Func.Stretch(gray_image,pStretch_Image,0,255);


    //图像腐蚀
    //定义腐蚀后的图像
    IplImage *pErode_Image=NULL;
    pErode_Image=cvCreateImage(cvGetSize(pSrc_Image),8,1);
    cvErode(pStretch_Image,pErode_Image,0,1);



    //拉伸后的图像减去腐蚀后的图像，得到车牌的轮廓
    //定义做差后的图像
    IplImage *sub_Image=NULL;
    sub_Image=cvCreateImage(cvGetSize(pSrc_Image),8,1);
    cvAbsDiff(pStretch_Image,pErode_Image,sub_Image);


    //对图像进行二值化操作
    //定义二值化图像
    IplImage *pBinary_Image=NULL;
    pBinary_Image=cvCreateImage(cvGetSize(pSrc_Image),8,1);
    cvThreshold(sub_Image,pBinary_Image,0,255,CV_THRESH_OTSU);
```

```
//对图像进行中值滤波，核采用矩阵(1,1,1,1,1)的转置
//定义中值滤波图像
IplImage *pFilter_Image=NULL;
pFilter_Image=cvCreateImage(cvGetSize(pSrc_Image),8,1);
//先复制一下要进行中值滤波的图像，因为前两行和最后两行没有进行处理
pFilter_Image=cvCloneImage(pBinary_Image);
m_Func.Middle_Smooth(pBinary_Image,pFilter_Image);



//对二值图像做腐蚀处理，去除噪声，采用矩阵（1,1,1,1,1）的转置作为核
//定义腐蚀后的图像
IplImage* pKernel_Erode_Image=NULL;
pKernel_Erode_Image=cvCreateImage(cvGetSize(pSrc_Image),8,1);
IplConvKernel * pKernel_Erode=NULL;
pKernel_Erode=cvCreateStructuringElementEx(1,3,0,1,CV_SHAPE_RECT,NULL);
cvErode(pFilter_Image, pKernel_Erode_Image,pKernel_Erode,2);

//对二值图像在做膨胀处理，恢复车牌区域，采用矩阵（1,1,1）的转置作为核
IplConvKernel * pKernel_Dilate=NULL;
pKernel_Dilate=cvCreateStructuringElementEx(1,3,0,1,CV_SHAPE_RECT,NULL);
//定义膨胀后的图像
IplImage* pKernel_Dilate_Image=NULL;
pKernel_Dilate_Image=cvCreateImage(cvGetSize(pSrc_Image),8,1);
cvDilate(pKernel_Erode_Image, pKernel_Dilate_Image,pKernel_Dilate,1);



//对车辆图像做水平投影
//定义像素累加和数组，图像高度不应大于2048
int level_shadow[2048];
int height=pSrc_Image->height;
int width=pSrc_Image->width ;
CvScalar s_shadow;

//清空并初始化数组
memset(level_shadow,0,sizeof(level_shadow));

//对图像做水平投影
for(int y=height-1;y>=0;y--)
    {
        for(int x=0;x<width;x++)
            {
                s_shadow=cvGet2D(pKernel_Dilate_Image,y,x);
                if(s_shadow.val[0]!=0)
                level_shadow[y]++; //统计每行不为零的像素的个数
            }
    }

//对图像水平投影的值做聚类，如果小行数值除以大行数值的商大于小于0.6，
//则认为这两行不是一类，将小行数值置为0
for(int y=height-1;y>=1;y--)
```

```
    {
        if(level_shadow[y-1]!=0)
        {
            if((float(level_shadow[y]))/(float(level_shadow[y-1]))<0.6)
                level_shadow[y]=0 ;
        }
    }

    //统计水平投影中不为零的区间
    for(int y=height-1;y>=0;y--)
    {
        if(level_shadow[y]!=0)
            level_shadow[y]=level_shadow[y+1]+1;
    }

    //求出水平投影数组中连续不为零的最大区间，此即认为是车牌大致高度
    int Y_min=0;//车牌高度小坐标
    int Y_max=0;//车牌高度大坐标
    int M_max_value=0;
    M_max_value=level_shadow[0]; //把level_shadow的第一个值赋给M_max_value
    for(int y=0;y<height;y++)
    {
        if(level_shadow[y]>M_max_value)
        {
            M_max_value=level_shadow[y];
            Y_min=y;
            Y_max=Y_min+M_max_value;
        }
    }

    if(M_max_value<10)
        AfxMessageBox("提取车牌高度失败!");

    //定义ROI区域，切割出车牌的高度
    CvRect ROI_Plate_Height;
    ROI_Plate_Height.x=0;
    ROI_Plate_Height.y=Y_min;
    ROI_Plate_Height.width=pSrc_Image->width;
    ROI_Plate_Height.height=M_max_value;
    cvSetImageROI(pKernel_Dilate_Image,ROI_Plate_Height);

    //将此区域复制一份，以便后续处理
    IplImage * pROI_Height_Image=NULL;
    pROI_Height_Image=cvCreateImage(cvSize(ROI_Plate_Height.width,ROI_Plate_Height.height
),8,1);
    cvCopyImage(pKernel_Dilate_Image,pROI_Height_Image);


    //对车牌高度区域做闭运算，得出车牌的矩形区域，以切割出车牌
    //运算核大小采用（车牌的高度*0.6)）
    int Copy_M_max_value=M_max_value; //复制此值，
    int Close_width=0;
    int Close_height=0;
```

```
    //核大小规定为奇数
    while((Copy_M_max_value%3)!=0)
    {
        Copy_M_max_value--;
    }
    Close_width=int(Copy_M_max_value*0.6);
    Close_height=Copy_M_max_value;
    IplConvKernel * pKernel_Close=NULL;
    pKernel_Close=cvCreateStructuringElementEx(Close_width,Close_height,Close_width/2,Close_height/2,CV_SHAPE_RECT,NULL);
    cvMorphologyEx(pROI_Height_Image,pROI_Height_Image,NULL,pKernel_Close,CV_MOP_CLOSE,1);


    //求联通区域的最大宽度，定位车牌的横坐标
    int X_min=0;//车牌宽度小坐标
    int X_max=0;//车牌宽度大坐标
    int M_row_max_value=0;
    int count_row[2048];//图像宽度不应大于2048
    memset(count_row,0,sizeof(count_row));

    //取车牌中间的一条直线进行检测,求此直线上连续不为0的像素的最大宽度，此即为车牌宽度
    int mid_height=M_max_value/2;
    uchar *ptr_mid=(uchar*)(pROI_Height_Image->imageData+mid_height*pROI_Height_Image->widthStep);
    for(int x=width-1;x>=0;x--)
    {
        if(ptr_mid[x]!=0)
            count_row[x]=count_row[x+1]+1;
    }

    //求出count_row数组中的最大值
     int Max_value_count_row=0;
     Max_value_count_row=count_row[0];
     for(int x=0;x<width;x++)
     {
         if(count_row[x]>Max_value_count_row)
        {
            Max_value_count_row=count_row[x];
            X_min=x;
            X_max=X_min+Max_value_count_row;
        }
     }

     //车牌的宽度应大于高度的三倍，对切割出的车牌进行验证

if(float(Max_value_count_row)/float(M_max_value)<3||float(Max_value_count_row)/float(M_max_value)>6)
        AfxMessageBox("提取车牌失败!");

    //切割出车牌,对边界区域适当调整
    CvRect ROI_Plate;
```

```
    ROI_Plate.x=X_min+2;
    ROI_Plate.y=Y_min+5;
    ROI_Plate.width=Max_value_count_row-6;
    ROI_Plate.height=M_max_value+3;

    //判断车牌定位区域是否合法
    if(ROI_Plate.x<0||ROI_Plate.x>width)
        AfxMessageBox("提取车牌失败!");
    if(ROI_Plate.y<0|| ROI_Plate.y>height)
        AfxMessageBox("提取车牌失败!");
    if((ROI_Plate.x+ROI_Plate.width)>width)
        AfxMessageBox("提取车牌失败!");
    if((ROI_Plate.y+ROI_Plate.height)>height)
        AfxMessageBox("提取车牌失败!");

    cvSetImageROI(gray_image,ROI_Plate);
    plate_image=cvCreateImage(cvSize(ROI_Plate.width,ROI_Plate.height),8,1);
    cvCopyImage(gray_image,plate_image);


    //释放灰度图像的ROI区域
    cvResetImageROI(gray_image);

    //对车牌图像进行灰度拉伸
    m_Func.Stretch(plate_image,plate_image,0,255);



    //对车牌图像进行二值化
    cvThreshold(plate_image,plate_image,0,255,CV_THRESH_OTSU);



    //将车牌图像颜色反转
    cvNot(plate_image,plate_image);

   //图像显示
    m_Cimage.mSetImg(plate_image);
    UpdateAllViews(NULL);


}
```

【实验效果】


S320
黑A 00001
车牌识别系统
文件(F) 编辑(E) 视图(V) 车牌定位与识别 帮助(H)
黑A 00001


京N·8P8F8
车牌识别系统
文件(F) 编辑(E) 视图(V) 车牌定位与识别 帮助(H)
京N·8P8F8


冀R·7819Q
东沽港
车牌识别系统
文件(F) 编辑(E) 视图(V) 车牌定位与识别 帮助(H)
冀R·7819Q


B 7D138
车牌识别系统
文件(F) 编辑(E) 视图(V) 车牌定位与识别 帮助(H)

2.2 倾斜校正：

2.2.1 应用图像自身的倾斜度构成一定斜率的直线，将图像等分为左右两边，风别求得两边的平均高度，最后求得直线斜率。

2.2.2 求得车牌的倾斜的等效直线斜率之后，可通过旋转矫正和几何下拉的多种校正方法，小组使用了几何校正，将图像上的像素点减去直线上同一条纵行的点的纵坐标得到校正后的图像。

【程序如下】

```
void CPlateIdentifyDoc::OnSplit()
{
    // TODO: 在此添加命令处理程序代码
    //字符分割
    //清空用来保存每个字符区域的链表
    CRectLink charRect1,charRect2;
    charRect1.clear();
    charRect2.clear();
    int num2=0;   //统计第一次切割出的字符的个数
    int num_char=0;//统计最终字符的个数
    CvScalar s;
    int iHeight=plate_image->height;
   int iWidth=plate_image->width;
    int top;//字符的大致顶部
    int bottom;//字符的大致底部

   //遇到第一个黑点确定顶部坐标位置
   for(int ht=0;ht<iHeight;ht++)
   {
       for(int wt=0;wt<iWidth;wt++)
       {
           s=cvGet2D(plate_image,ht,wt);
           if(0==s.val[0])
           {
               //黑点
```

```
            top=ht;
            ht=iHeight;//强制跳出循环
            break;
        }
    }
}

//遇到第一个黑点确定底部坐标位置
for(int ht=iHeight-1;ht>=0;ht--)
{
    for(int wt=0;wt<iWidth;wt++)
    {
        s=cvGet2D(plate_image,ht,wt);
        if(0==s.val[0])
        {
            bottom=ht;//强制跳出循环
            ht=-1;
            break;
        }
    }
}

bool lab=FALSE;  //是否进入一个字符分割状态
bool black=FALSE; //扫描中是否发现黑点
CRect rect; // 存放位置信息的结构体

for(int wt=0;wt<iWidth;wt++)
{
     //开始扫描一列
     black=false;
    for(int ht=0;ht<iHeight;ht++)
    {
         // 获得当前像素值
        s=cvGet2D(plate_image,ht,wt);
        if(0==s.val[0])//判断是否是黑点
        {
            //如果发现黑点，设置标志位
            black=true;
            //如果还没有进入一个字符分割
            if(lab==false)
            {
                //设置左侧边界
                rect.left=wt;
                //字符分割开始
                lab=true;
            }
            //如果字符分割已经开始了
            else
                //跳出循环
                break;
         }
     }
```

```
        //如果已经扫描到了最右边那列，说明整幅图像扫描完毕。退出
        if(wt==(iWidth-1))
            break;

        //如果到此时balck仍为false，说明扫描了一列都没有发现黑点。表面当前字符分割结束
        if((lab==true)&&(black==false))
        {
            //将位置信息存入结构体中
            //设置右边界
            rect.right=wt;

            //设置上边界
            rect.top=top;

            //设置下边界
            rect.bottom=bottom;

            //将框外扩一个像素，以免压倒字符
            rect.InflateRect(1,1);

            //将这个结构体插入存放位置信息的链表1的后面
            charRect1.push_back(rect);

            //设置标志位，开始下一次的字符分割
            lab=false;

            //字符个数加1
            num2++;
        }
            //进入下一列的扫描
}

//再将矩形轮廓的top和bottom精确化

//将链表1赋值给链表2
charRect2=charRect1;

//将链表2的内容清空
charRect2.clear();

//建立一个新的存放位置信息的结构体
CRect rectnew;

//对于链表1从头到尾逐个进行扫描
while(!charRect1.empty())
{
    //从链表1头上得到一个矩形
    rect=charRect1.front();

    //从链表1头上面删掉一个
    charRect1.pop_front();

    //判断字符的宽度，以去除中间的小圆点
```

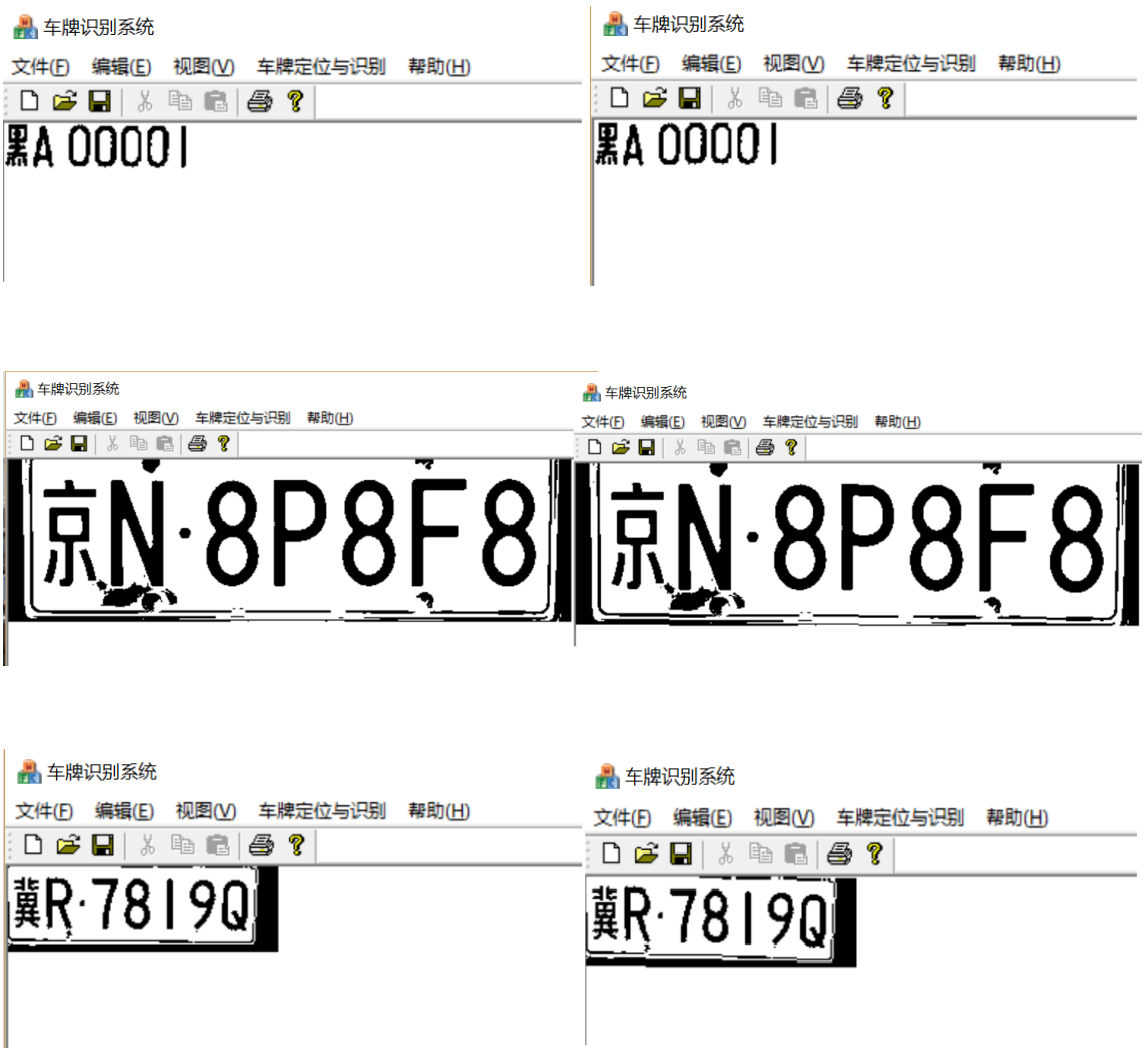
```
//字符个数加一
num_char++;

//计算更加精确的矩形区域
//获得精确的左边界
rectnew.left=rect.left-1;

//获得精确的右边界
rectnew.right=rect.right-1;

//通过获得的精确左右边界对上下边界重新进行精确定位
//由上而下扫描，计算上边界
//行
for(int ht=rect.top;ht<rect.bottom;ht++)
{
    //列
    for(int wt=rect.left;wt<rect.right;wt++)
    {
        // 获得当前像素值
        s=cvGet2D(plate_image,ht,wt);
        if(0==s.val[0])
        {
            //设置上边界
            rectnew.top=ht-1;

            //对ht进行强制定义以跳出循环
            ht=rect.bottom;

            //跳出循环
            break;
        }

    }
}

//由下而上扫描，计算下边界
//行
for(int ht=rect.bottom-1;ht>=rect.top;ht--)
{
    //列
    for(int wt=rect.left;wt<rect.right;wt++)
    {
        // 获得当前像素值
        s=cvGet2D(plate_image,ht,wt);
        if(0==s.val[0])
        {
            //设置上边界
            rectnew.bottom=ht+1;

            //对ht进行强制定义以跳出循环
            ht=-1;
```

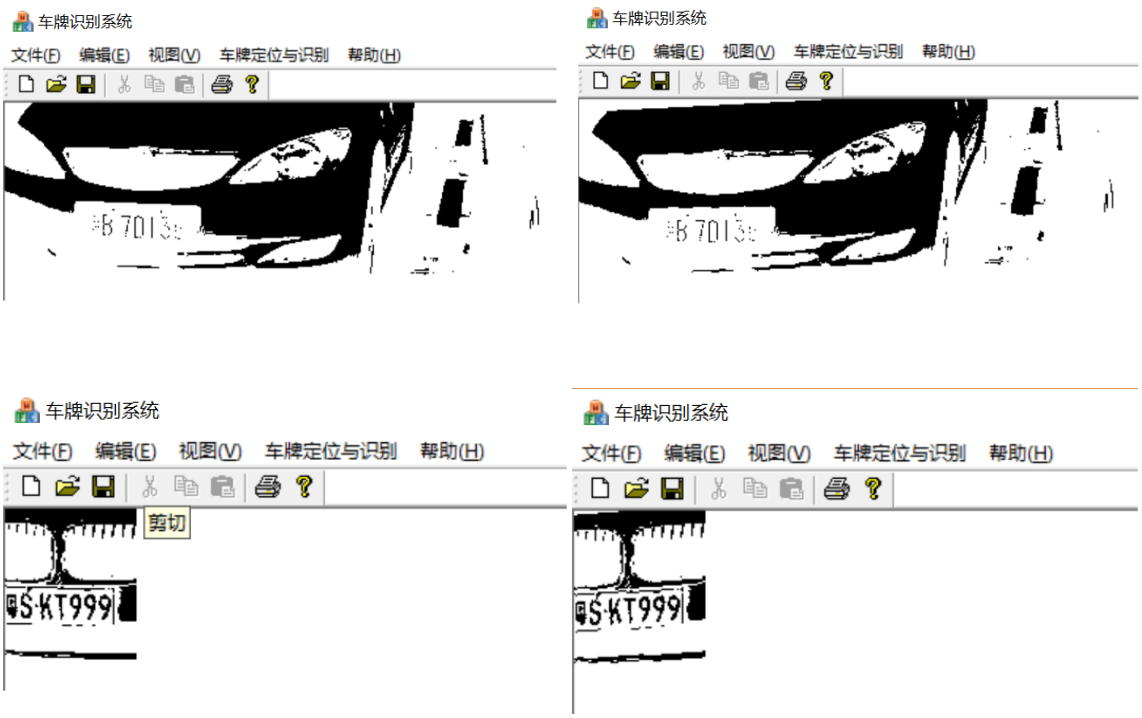
```
                    //跳出循环
                    break;
                }
            }
        }
    //将得到的新的准确位置信息从后面插入到链表2的尾上
    charRect2.push_back(rectnew);
    }

//将链表2传递给链表1
charRect1=charRect2;

//切割图像
CvRect dst_rect[100];
CRect rect2;
for(int i=0;i<num_char;i++)
{
    if(!charRect1.empty())
    {
        rect2=charRect1.front();
        charRect1.pop_front();
        dst_rect[i].x=rect2.left;
        dst_rect[i].y=rect2.top;
        dst_rect[i].width=rect2.right-rect2.left+1;
        dst_rect[i].height=rect2.bottom-rect2.top+1;
    }
}

//字符归一化和字符细化

for(int i=0;i<num_char;i++)
{
    dst_image[i]=cvCreateImage(cvSize(g_width,g_height),8,1);
    cvSetImageROI(plate_image,dst_rect[i]);
    cvNot(plate_image,plate_image);//字符反色处理
    cvResize(plate_image,dst_image[i],CV_INTER_NN);//图像缩放
    m_thin.Thin(dst_image[i],dst_image[i]);//图像细化
    cvResetImageROI(plate_image);
}

//显示分割后的字符
IplImage * plate_char;
//将分割出来的字符整齐排列到图像plate_char上，
//为便于显示分割效果，将字符间的间距设为12个像素
plate_char=cvCreateImage(cvSize(g_width*num_char+(num_char-1)*12,g_height),8,1);
cvZero(plate_char);

for(int i=0;i<num_char;i++)
    {
        int x=0+i*g_width+i*12;
        int y=0;
        int width=g_width;
        int height=g_height;
```

```
            cvSetImageROI(plate_char,cvRect(x,y,width,height));
            cvCopyImage(dst_image[i],plate_char);
            cvResetImageROI(plate_char);
        }
    //图像显示
    cvShowImage("分割出的字符",plate_char);
    UpdateAllViews(NULL);
}
```

【实验效果】

左边为提取的图像，右边为校正后的图像





2.3 字符分割：

在字符分割之前，首先进行预处理，选用另外的阈值来重新处理图像，获得新的二值化图像。

然后通过投影法，统计出垂直方向上字符像素的个数来确定车牌的每个字符的位置坐标，并在二值化的图像中标记出来。

【程序如下】

```
void CPlateIdentifyDoc::OnSplit()
{
    // TODO: 在此添加命令处理程序代码
    //字符分割
    //清空用来保存每个字符区域的链表
    CRectLink charRect1,charRect2;
    charRect1.clear();
    charRect2.clear();
    int num2=0;   //统计第一次切割出的字符的个数
    int num_char=0;//统计最终字符的个数
    CvScalar s;
    int iHeight=plate_image->height;
    int iWidth=plate_image->width;
    int top;//字符的大致顶部
    int bottom;//字符的大致底部

    //遇到第一个黑点确定顶部坐标位置
    for(int ht=0;ht<iHeight;ht++)
    {
        for(int wt=0;wt<iWidth;wt++)
        {
            s=cvGet2D(plate_image,ht,wt);
            if(0==s.val[0])
            {
                //黑点
                top=ht;
                ht=iHeight;//强制跳出循环
                break;
            }
        }
    }

    //遇到第一个黑点确定底部坐标位置
    for(int ht=iHeight-1;ht>=0;ht--)
    {
        for(int wt=0;wt<iWidth;wt++)
        {
            s=cvGet2D(plate_image,ht,wt);
            if(0==s.val[0])
            {
                bottom=ht;//强制跳出循环
```

```
            ht=-1;
            break;
        }
    }
}

bool lab=FALSE;  //是否进入一个字符分割状态
bool black=FALSE; //扫描中是否发现黑点
CRect rect; // 存放位置信息的结构体

for(int wt=0;wt<iWidth;wt++)
{
    //开始扫描一列
    black=false;
    for(int ht=0;ht<iHeight;ht++)
    {
        // 获得当前像素值
        s=cvGet2D(plate_image,ht,wt);
        if(0==s.val[0])//判断是否是黑点
        {
            //如果发现黑点，设置标志位
            black=true;
            //如果还没有进入一个字符分割
            if(lab==false)
            {
                //设置左侧边界
                rect.left=wt;
                //字符分割开始
                lab=true;
            }
            //如果字符分割已经开始了
            else
                //跳出循环
                break;
        }
    }

    //如果已经扫描到了最右边那列，说明整幅图像扫描完毕。退出
    if(wt==(iWidth-1))
        break;

    //如果到此时balck仍为false，说明扫描了一列都没有发现黑点。表面当前字符分割结束
    if((lab==true)&&(black==false))
    {
        //将位置信息存入结构体中
        //设置右边界
        rect.right=wt;

        //设置上边界
        rect.top=top;

        //设置下边界
        rect.bottom=bottom;
```

```
            //将框外扩一个像素，以免压倒字符
            rect.InflateRect(1,1);

            //将这个结构体插入存放位置信息的链表1的后面
            charRect1.push_back(rect);

            //设置标志位，开始下一次的字符分割
            lab=false;

            //字符个数加1
            num2++;
        }
            //进入下一列的扫描
}

//再将矩形轮廓的top和bottom精确化

//将链表1赋值给链表2
charRect2=charRect1;

//将链表2的内容清空
charRect2.clear();

//建立一个新的存放位置信息的结构体
CRect rectnew;

//对于链表1从头到尾逐个进行扫描
while(!charRect1.empty())
{
    //从链表1头上得到一个矩形
    rect=charRect1.front();

    //从链表1头上面删掉一个
    charRect1.pop_front();

    //判断字符的宽度，以去除中间的小圆点

        //字符个数加一
        num_char++;

        //计算更加精确的矩形区域
        //获得精确的左边界
        rectnew.left=rect.left-1;

        //获得精确的右边界
        rectnew.right=rect.right-1;

        //通过获得的精确左右边界对上下边界重新进行精确定位
        //由上而下扫描，计算上边界
        //行
        for(int ht=rect.top;ht<rect.bottom;ht++)
        {
```

```
                //列
                for(int wt=rect.left;wt<rect.right;wt++)
                {
                    // 获得当前像素值
                    s=cvGet2D(plate_image,ht,wt);
                    if(0==s.val[0])
                    {
                        //设置上边界
                        rectnew.top=ht-1;

                        //对ht进行强制定义以跳出循环
                        ht=rect.bottom;

                        //跳出循环
                        break;
                    }

                }
            }

            //由下而上扫描，计算下边界
            //行
            for(int ht=rect.bottom-1;ht>=rect.top;ht--)
            {
                //列
                for(int wt=rect.left;wt<rect.right;wt++)
                {
                    // 获得当前像素值
                    s=cvGet2D(plate_image,ht,wt);
                    if(0==s.val[0])
                    {
                        //设置上边界
                        rectnew.bottom=ht+1;

                        //对ht进行强制定义以跳出循环
                        ht=-1;

                        //跳出循环
                        break;
                    }
                }
            }
        //将得到的新的准确位置信息从后面插入到链表2的尾上
        charRect2.push_back(rectnew);
        }

    //将链表2传递给链表1
    charRect1=charRect2;

    //切割图像
    CvRect dst_rect[100];
    CRect rect2;
    for(int i=0;i<num_char;i++)
```

```
    {
        if(!charRect1.empty())
        {
            rect2=charRect1.front();
            charRect1.pop_front();
            dst_rect[i].x=rect2.left;
            dst_rect[i].y=rect2.top;
            dst_rect[i].width=rect2.right-rect2.left+1;
            dst_rect[i].height=rect2.bottom-rect2.top+1;
        }
    }

    //字符归一化和字符细化

    for(int i=0;i<num_char;i++)
    {
        dst_image[i]=cvCreateImage(cvSize(g_width,g_height),8,1);
        cvSetImageROI(plate_image,dst_rect[i]);
        cvNot(plate_image,plate_image);//字符反色处理
        cvResize(plate_image,dst_image[i],CV_INTER_NN);//图像缩放
        m_thin.Thin(dst_image[i],dst_image[i]);//图像细化
        cvResetImageROI(plate_image);
    }

    //显示分割后的字符
    IplImage * plate_char;
    //将分割出来的字符整齐排列到图像plate_char上，
    //为便于显示分割效果，将字符间的间距设为12个像素
    plate_char=cvCreateImage(cvSize(g_width*num_char+(num_char-1)*12,g_height),8,1);
    cvZero(plate_char);

    for(int i=0;i<num_char;i++)
        {
            int x=0+i*g_width+i*12;
            int y=0;
            int width=g_width;
            int height=g_height;
            cvSetImageROI(plate_char,cvRect(x,y,width,height));
            cvCopyImage(dst_image[i],plate_char);
            cvResetImageROI(plate_char);
        }
    //图像显示
    cvShowImage("分割出的字符",plate_char);
    UpdateAllViews(NULL);
}
```

尚未成功识别的一张车牌照片：

图 2.4-16

【原因探索】

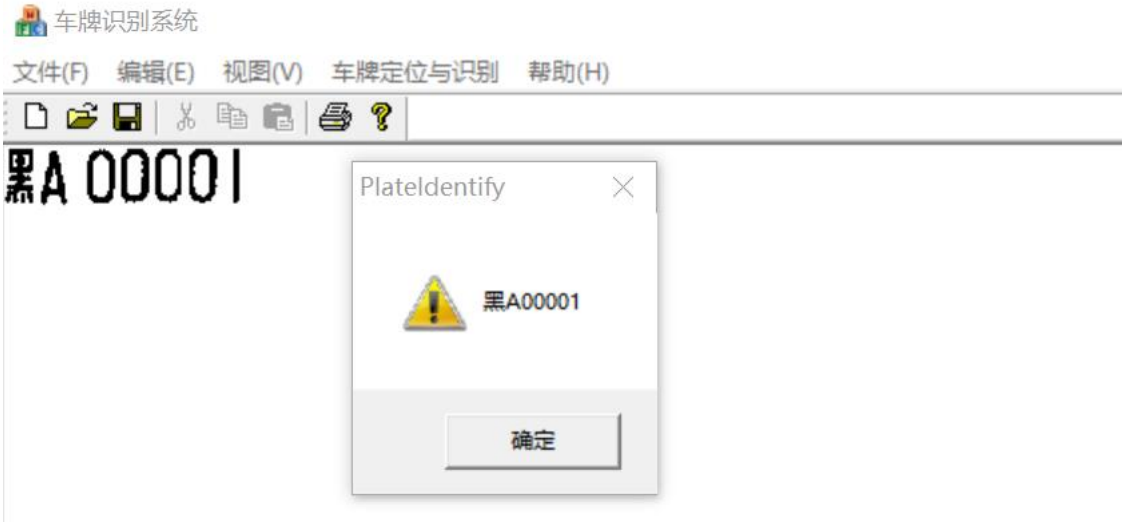
仔细观察车牌可知，在车牌拍摄的时候。光照很强，与其余图片相比，车牌的 HSV 各分量均有较大的差别，因此我们小组在二值化处理时为了适应大多数照片而没有的得到最佳阈值，二值化后，字符边缘模糊不清，对字符分割有很大的影响。

【解决方案】

应该在进行二值化处理之前，首先判断一下照片的光照强度，根据判断结果来设定不同的阈值范围，使得所有的图片都得到最佳的二值化处理。

2.4 字符识别：

实验运行截图：



【程序如下】：

```
void CPlateIdentifyDoc::OnCharidentify()
{
    // TODO: 在此添加命令处理程序代码

    IplImage * char_sample[34];//字符样本图像数组
    IplImage * hanzi_sample[9];//汉字样本图像数组
    pattern char_pattern[34];//定义字符样品库结构数组
    pattern hanzi_pattern[9];//定义汉字样品库结构数组
    pattern TestSample[7];//定义待识别字符结构数组

    //载入字符模板
```

```
char_sample[0]=cvLoadImage("template\\0.bmp",0);
char_sample[1]=cvLoadImage("template\\1.bmp",0);
char_sample[2]=cvLoadImage("template\\2.bmp",0);
char_sample[3]=cvLoadImage("template\\3.bmp",0);
char_sample[4]=cvLoadImage("template\\4.bmp",0);
char_sample[5]=cvLoadImage("template\\5.bmp",0);
char_sample[6]=cvLoadImage("template\\6.bmp",0);
char_sample[7]=cvLoadImage("template\\7.bmp",0);
char_sample[8]=cvLoadImage("template\\8.bmp",0);
char_sample[9]=cvLoadImage("template\\9.bmp",0);
char_sample[10]=cvLoadImage("template\\A.bmp",0);
char_sample[11]=cvLoadImage("template\\B.bmp",0);
char_sample[12]=cvLoadImage("template\\C.bmp",0);
char_sample[13]=cvLoadImage("template\\D.bmp",0);
char_sample[14]=cvLoadImage("template\\E.bmp",0);
char_sample[15]=cvLoadImage("template\\F.bmp",0);
char_sample[16]=cvLoadImage("template\\G.bmp",0);
char_sample[17]=cvLoadImage("template\\H.bmp",0);
char_sample[18]=cvLoadImage("template\\J.bmp",0);
char_sample[19]=cvLoadImage("template\\K.bmp",0);
char_sample[20]=cvLoadImage("template\\L.bmp",0);
char_sample[21]=cvLoadImage("template\\M.bmp",0);
char_sample[22]=cvLoadImage("template\\N.bmp",0);
char_sample[23]=cvLoadImage("template\\P.bmp",0);
char_sample[24]=cvLoadImage("template\\Q.bmp",0);
char_sample[25]=cvLoadImage("template\\R.bmp",0);
char_sample[26]=cvLoadImage("template\\S.bmp",0);
char_sample[27]=cvLoadImage("template\\T.bmp",0);
char_sample[28]=cvLoadImage("template\\U.bmp",0);
char_sample[29]=cvLoadImage("template\\V.bmp",0);
char_sample[30]=cvLoadImage("template\\W.bmp",0);
char_sample[31]=cvLoadImage("template\\X.bmp",0);
char_sample[32]=cvLoadImage("template\\Y.bmp",0);
char_sample[33]=cvLoadImage("template\\Z.bmp",0);


//载入汉字模板
hanzi_sample[0]=cvLoadImage("template\\川.bmp",0);
hanzi_sample[1]=cvLoadImage("template\\鄂.bmp",0);
hanzi_sample[2]=cvLoadImage("template\\黑.bmp",0);
hanzi_sample[3]=cvLoadImage("template\\京.bmp",0);
hanzi_sample[4]=cvLoadImage("template\\辽.bmp",0);
hanzi_sample[5]=cvLoadImage("template\\琼.bmp",0);
hanzi_sample[6]=cvLoadImage("template\\湘.bmp",0);
hanzi_sample[7]=cvLoadImage("template\\粤.bmp",0);
hanzi_sample[8]=cvLoadImage("template\\浙.bmp",0);


//提取字符样本特征
for(int i=0;i<34;i++)
{
    GetFeature(char_sample[i],char_pattern[i]);
}
```

```
//提取汉字字符特征
for(int i=0;i<9;i++)
{
    GetFeature(hanzi_sample[i],hanzi_pattern[i]);
}
//提取待识别字符特征
for(int i=0;i<7;i++)
{
    GetFeature(dst_image[i],TestSample[i]);
}

//进行模板匹配
double min=100000.0;
for(int num=0;num<1;num++)
{
    for(int i=0;i<9;i++)
    {
        double diff=0.0;
        for(int j=0;j<25;j++)
        {
            diff+=fabs(TestSample[num].feature[j]-hanzi_pattern[i].feature[j]);
        }
        for(int j=25;j<33;j++)
        {
            diff+=fabs(TestSample[num].feature[j]-hanzi_pattern[i].feature[j])*9;
        }
        if(diff<min)
        {
            min=diff;
            TestSample[num].number=i;
        }
    }
}

for(int num=1;num<7;num++)
{
    double min_min=1000000.0;
    for(int i=0;i<34;i++)
    {
        double diff_diff=0.0;
        for(int j=0;j<25;j++)
        {
            diff_diff+=fabs(TestSample[num].feature[j]-char_pattern[i].feature[j]);
        }
        for(int j=25;j<33;j++)
        {
            diff_diff+=fabs(TestSample[num].feature[j]-char_pattern[i].feature[j]);
        }
        if(diff_diff<min_min)
        {
            min_min=diff_diff;
            TestSample[num].number=i;
        }
```

```
    }
}

CString result="";//存放识别出的字符

for(int i=0;i<1;i++)
{
    switch (TestSample[i].number)
    {
    case 0:
        result+="川";
        break;
    case 1:
        result+="鄂";
        break;
    case 2:
        result+="黑";
        break;
    case 3:
        result+="京";
        break;
    case 4:
        result+="辽";
        break;
    case 5:
        result+="琼";
        break;
    case 6:
        result+="湘";
        break;
    case 7:
        result+="粤";
        break;
    case 8:
        result+="浙";
        break;
    default:
        AfxMessageBox("识别失败");
        break;
    }
}

for(int i=1;i<7;i++)
{
    switch (TestSample[i].number)
    {
    case 0:
        result+="0";
        break;
    case 1:
        result+="1";
        break;
    case 2:
```

```
        result+="2";
        break;
    case 3:
        result+="3";
        break;
    case 4:
        result+="4";
        break;
    case 5:
        result+="5";
        break;
    case 6:
        result+="6";
        break;
    case 7:
        result+="7";
        break;
    case 8:
        result+="8";
        break;
    case 9:
        result+="9";
        break;
    case 10:
        result+="A";
        break;
    case 11:
        result+="B";
        break;
    case 12:
        result+="C";
        break;
    case 13:
        result+="D";
        break;
    case 14:
        result+="E";;
        break;
    case 15:
        result+="F";
    case 16:
        result+="G";
        break;
    case 17:
        result+="H";
        break;
    case 18:
        result+="J";
        break;
    case 19:
        result+="K";
        break;
    case 20:
```

```
            result+="L";
            break;
        case 21:
            result+="M";
            break;
        case 22:
            result+="N";
            break;
        case 23:
            result+="P";
            break;
        case 24:
            result+="Q";
            break;
        case 25:
            result+="R";
            break;
        case 26:
            result+="S";
            break;
        case 27:
            result+="T";
            break;
        case 28:
            result+="U";
            break;
        case 29:
            result+="U";
            break;
        case 30:
            result+="W";
            break;
        case 31:
            result+="X";
            break;
        case 32:
            result+="Y";
            break;
        case 33:
            result+="Z";
            break;
        default:
            AfxMessageBox("识别失败");
            break;
        }
    }
    AfxMessageBox(result);//显示结果

}
```

3 实验结果及分析：

3.1 车牌检测准确率:20/20=100%

3.2 字符分割准确率: 19/20=95%

3.3 字符识别准确率：8/20=40%

4 本课程收获及建议：

4.1 收获：

数字图像处理课程是一名实践性很强的一门专业课程，在进行较少的课上学习之后，老师便穿插进行了很多图像处理的编程实践，先是让我们仿照老师的模板，对照着老师的 ppt 将正确的程序敲击到电脑里最终实现一定的效果，诸如直方图均衡化处理等等。在大作业的过程中，老师仅仅提供了一部分的模板，这对一向对老师的现有程序有很大依赖性的我有很大挑战，在编程过程中， 有很多时候程序的运行结果就是事与愿违，我在和队友的密切配合中慢慢地达到了预期效果，一步一个脚印，卓有成效，培养了缜密的思维和非凡的耐力，养成了团队配合的良好习惯。

目前的市场上面识别车牌的技术水平为字母和数字的识别率可达到 96%，汉字的识别率可达到 95%，我们达不到它的一半，可能和算法有些关系。本科阶段没有接触过机器学习的理论是非常遗憾的，（运气不好没选到课）希望接触机器学习之后识别率能够大大提高。

4.2 建议：

这门课建议开 16 周的，8 周上课太紧了，好多知识我都没掌握，只能说是囫囵吞枣，不求甚解。同时这是一门实用性非常强的课程，希望以后可以多开设一些开放性的实验。

另外先修课信号与系统和数字信号处理这两门课我们计算机专业本科阶段是接触不到的，希望可以再开两门这个课，这样我们也就不会因为数学基础差看不懂傅里叶变换而感到头疼。

5 组员贡献率

组长：麻锦涛 60%
组员：谭天云 40%

6 参考文献

[1] 冈萨雷斯 R C，温茨 P 著. 数字图像处理. 阮秋琦等译. 北京: 电子工业出版社, 2011.
[2] 阮秋琦著. 数字图像处理基础. 北京: 清华大学出版社, 2009.
[3] 翟伟芳. 具有倾斜矫正功能的车牌定位和字符分割算法研究. [硕士论文], 河北工业大学, 2007.
[4] 吴昊. 汽车牌照自动识别系统. [硕士论文], 电子科技大学, 2008.