天津理工大学

计算机科学与工程学院

实验报告

2017 至 2018 学年 第 二 学期

实验九 图像处理综合实验

课程名称	数字图像处理				
学号	20152180	学生姓名	王帆	年级	2015
专业	计算机科学与技术	教学班号	2	实验地点	7-212
实验时间	2018 年 5 月 21 日 第 5 节 至 第 6 节				
主讲教师	杨淑莹				

实验成绩

软件运行	效果	算法分析	流程设计	报告成绩	总成绩

实验（九）	实验名称	**图像处理综合实验**
软件环境	Windows Visual Studio 2017	
硬件环境	PC	

实验目的说

提高图像处理综合能力，提高实践应用能力

实验内容（应包括实验题目、实验要求、实验任务等）

针对某一个你感兴趣的图像处理项目，如人脸识别、身份证号码识别、汽车牌照识别等，分析数据，写出设计流程，写出关键技术实现功能，实现每一个关键技术的步骤，编程实现图像综合处理。

实验过程与实验结果

项目名称：

车辆牌照识别系统的研究与实现

项目介绍：

车牌识别技术的任务是处理、分析摄取的车辆图像，实现车牌号码的自动识别。典型的车辆牌照识别系统是由图像采集系统、中央处理器、识别系统组成，一般还要连接相应的数据库以完成特定的功能。当系统发现(通过埋地线圈或者光束检测)有车通过时，则发出信号给图像采集系统，然后采集系统将得到的图像输入识别系统进行识别，其识别结果应该是文本格式的车牌号码。由于车辆牌照是机动车唯一的管理标识符号，在交通管理中具有不可替代的作用，因此车辆牌照识别系统应具有很高的识别正确率，对环境光照条件、拍摄位置和车辆行驶速度等因素的影响应有较大的容阈，并且要求满足实时性要求。

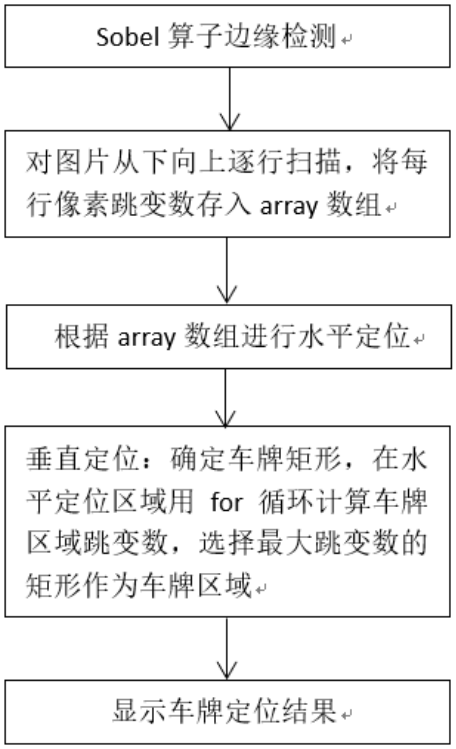
本实验是车牌识别系统的图像处理部分，基于自建样例车牌图像库，通过对其进行基本图像处理（灰度化、二值化、均值滤波、Sobel 边缘检测、轮廓提取、灰度跳变检测等方法）得到车牌单元，并使用模板匹配法对获取到的车牌单元进行识别，最终获取到车牌号数据，实现车牌识别的效果。

实验流程：

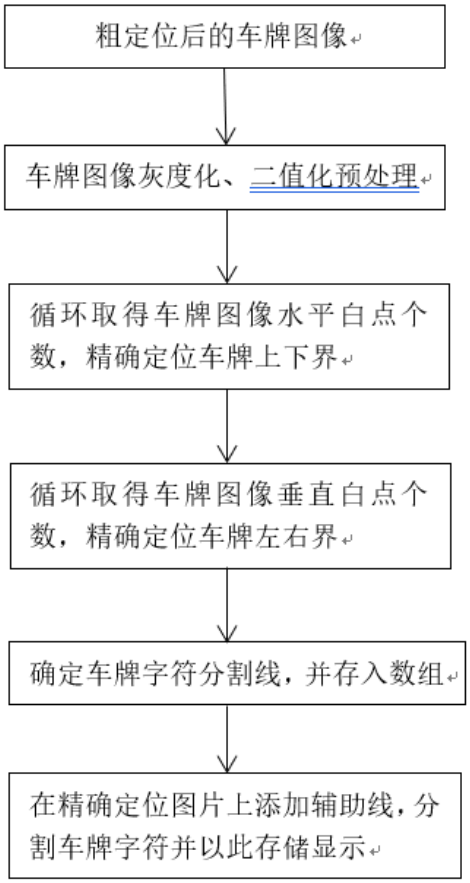
1. 总体实验流程

2. 分步实验流程

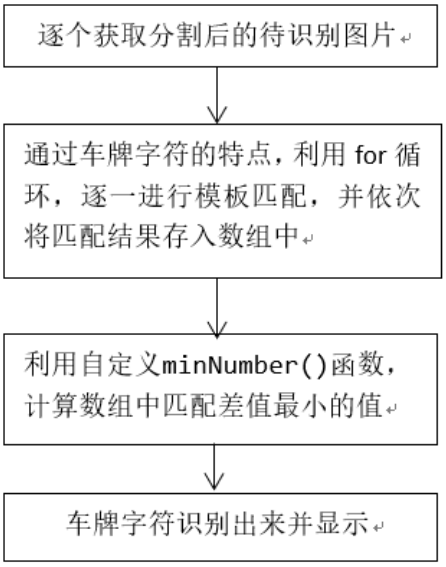
车牌定位：



车牌分割：



车牌识别：



关键技术实现：

1. 图片预处理

在车牌识别系统中我们通过采集得到的图片一般是彩色图片，在加上实际环境以及硬件设施的影响，图片质量不高，图像的背景噪声等会影响字符的分割与识别，因此我们在车牌分割及识别之前一般会进行图像的预处理。

本实验中，车牌图像的预处理包括图像灰度化，图像均衡化以及图像均值滤波。

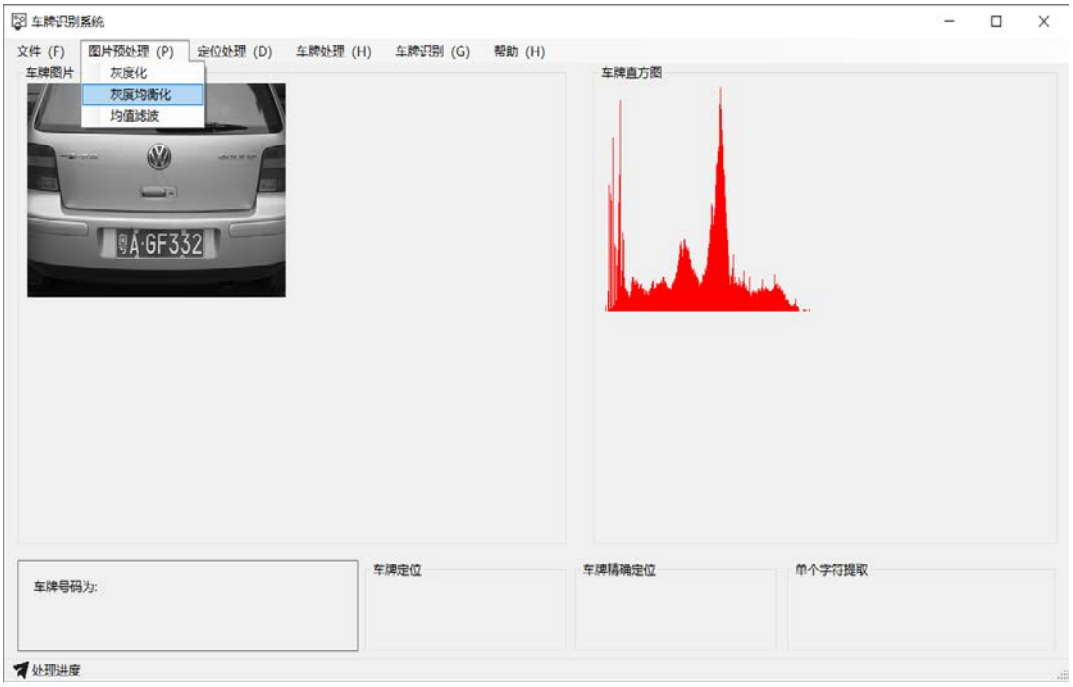


图 1 图片预处理

1.1 灰度化

将彩色图像转化成为灰度图像的过程成为图像的灰度化处理。灰度图像的描述与彩色图像一样仍然反映了整幅图像的整体和局部的色度和亮度等级的

分布和特征。

本实验中，我们根据 YUV 颜色空间 Y 的分量的物理意义是点的亮度，由该值反映亮度等级，根据 RGB 和 YUV 颜色空间的变换关系可建立亮度 Y 与 R、G、B 三个颜色分量的对应，并以亮度值表达图像的灰度值。

$$Y = 0.299R + 0.587G + 0.114B$$

代码：

```
private void 灰度化ToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        if (m_Bitmap != null)
        {
            Bitmap bitmap = new Bitmap(pictureBox1.Image);
            Color curColor;
            int ret;
            for (int i = 0; i < m_Bitmap.Width; i++)
            {
                for (int j = 0; j < m_Bitmap.Height; j++)
                {
                    curColor = m_Bitmap.GetPixel(i, j);
                    ret = (int)(curColor.R * 0.299 + curColor.G * 0.587 +
curColor.B * 0.114);
                    bitmap.SetPixel(i, j, Color.FromArgb(ret, ret, ret));
                }
            }
            pictureBox1.Image = bitmap;
            Invalidate();
        }
        flag = 1;
        graydo();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
    }
}
```

1.2 均衡化

均衡化原理是将原图像通过某种变换，得到一幅灰度直方图为均匀分布的新图像的方法。设图像均衡化处理后，图像的直方图是平直的，即各灰度级具有相同的出现频数(大体相同)，那么由于灰度级具有均匀的概率分布，图像看起来就更清晰了。

均衡化数学原理：

已知累积分布函数(CDF)：$S = T(r) = \int_0^r P_r(w)dw$，其中，T(r)在区间$0 \le r \le 1$中为单值且单调递增，当$0 \le r \le 1$时，$0 \le T(r) \le 1$。上式表明，当变换函数为r的累积直方图函数时，能达到直方图均衡化的目的。

均衡化步骤：

（1）计算各灰度级出现的概率；

（2）根据变换函数求新的灰度；

（3）与灰度级拟合；

（4） 求新的灰度级出现的概率。

代码：

```
private void 灰度均衡化ToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        Bitmap curBitmap = (Bitmap)pictureBox1.Image.Clone();
        if (curBitmap != null)
        {
            Bitmap bitmap = new Bitmap(pictureBox1.Image);
            int[] hist = getHist(curBitmap, curBitmap.Width,
curBitmap.Height);
            Color color = new Color();
            double p = (double)255 / (curBitmap.Width * curBitmap.Height);
            double[] sum = new double[256];
            int[] outg = new int[256];
            sum[0] = hist[0];
            for (int i = 1; i < 256; i++)
                sum[i] = sum[i - 1] + hist[i];
            for (int i = 0; i < 256; i++)
                outg[i] = (int)(p * sum[i]);
            for (int j = 0; j < curBitmap.Height; j++)
            {
                for (int i = 0; i < curBitmap.Width; i++)
                {
                    int g = (curBitmap.GetPixel(i, j).R);
                    color = Color.FromArgb(outg[g], outg[g], outg[g]);
                    bitmap.SetPixel(i, j, color);
                }
            }
            pictureBox1.Image = bitmap;
        }
        flag = 1;
        graydo();
    }
    catch (Exception ex)
```

```
    {
        MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
    }
}
```

图 2 均衡化

1.3 中值滤波

噪声对图像处理的影响很大，它影响图像处理的输入、采集和处理等各个环节以及输出结果。因此，在进行其它的图像处理前，需要对图像进行去噪处理。中值滤波方法是，对待处理的当前像素，选择一个模板，该模板为其邻近的若干个像素组成，对模板的像素由小到大进行排序，再用模板的中值来替代原像素的值的方法。

$$g = \mathrm{mid}[(x-1,y-1) + f(x,y-1) + f(x+1,y-1) + f(x-1,y) + f(x,y) + f(x+1,y) + f(x-1,y+1) + f(x,y+1) + f(x+1,y+1)]$$

权系数矩阵模板：

$$mid\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

代码：

```
private void 中值滤波ToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        Bitmap curBitmap = (Bitmap)pictureBox1.Image.Clone();
        Bitmap bitmap = new Bitmap(pictureBox1.Image);
        int height = curBitmap.Height;
        int width = curBitmap.Width;
        Color[] pixel = new Color[9];//暂时建立一个3*3模版
```

```
int[] red = new int[9];
int[] green = new int[9];
int[] blue = new int[9];
int temp1 = 0, temp2 = 0, temp3 = 0;
for (int i = 1; i < width - 1; i++)
{
    for (int j = 1; j < height - 1; j++)
    {
        pixel[0] = curBitmap.GetPixel(i - 1, j - 1);
        pixel[1] = curBitmap.GetPixel(i - 1, j);
        pixel[2] = curBitmap.GetPixel(i - 1, j + 1);
        pixel[3] = curBitmap.GetPixel(i, j - 1);
        pixel[4] = curBitmap.GetPixel(i, j);
        pixel[5] = curBitmap.GetPixel(i, j + 1);
        pixel[6] = curBitmap.GetPixel(i + 1, j - 1);
        pixel[7] = curBitmap.GetPixel(i + 1, j);
        pixel[8] = curBitmap.GetPixel(i + 1, j + 1);
        //取中值

        for (int s = 0; s < 9; s++)
        {
            red[s] = pixel[s].R;
            green[s] = pixel[s].R;
            blue[s] = pixel[s].R;
        }
        //起泡排序
        for (int x = 0; x < 8; x++)
        {
            for (int y = 0; y < 8 - x; y++)
            {
                if (red[y] < red[y + 1])
                {
                    temp1 = red[y];
                    red[y] = red[y + 1];
                    red[y + 1] = temp1;
                }
                if (green[y] < green[y + 1])
                {
                    temp2 = green[y];
                    green[y] = green[y + 1];
                    green[y + 1] = temp2;
                }
                if (blue[y] < blue[y + 1])
                {
```

```
                        temp3 = blue[y];
                        blue[y] = blue[y + 1];
                        blue[y + 1] = temp3;
                    }
                }
            }
            Color cc = Color.FromArgb(red[4], green[4], blue[4]);
            bitmap.SetPixel(i, j, cc);
        }
    }
    pictureBox1.Image = bitmap;

    flag = 1;
    graydo();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
}
}
```

图 3 中值滤波

2. 图像定位

车牌的定位主要是在经过图像预处理过程后的图像中确定车牌的具体位置。自然环境下，汽车图像背景复杂、光照不均匀，如何在自然背景中准确地确定牌照区域是整个识别过程的关键。首先对采集到的视频图像进行大范围相关搜索，找到符合汽车牌照特征的区域，然后对该候选区域做进一步分析、评判，最后选定最佳的区域作为牌照区域，并将其从图像中分割出来。

2.1 Sobel 算子边缘检测

图像定位之前首先采用 Sobel 算子对图像进行边缘检测。

该算子包含两组 3x3 的矩阵，分别为横向边缘检测，Sobel 算子是滤波算子的形式，用于提取边缘，可以利用快速卷积函数，简单有效。

代码：

```
private void sobel边缘检测ToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        if (m_Bitmap != null)
        {
            Bitmap bitmap = new Bitmap(pictureBox1.Image);
            Color color = new Color();
            int r;
            int w = m_Bitmap.Width;
            int h = m_Bitmap.Height;
            int[,] inred = new int[w, h];
            int[,] ingreen = new int[w, h];
            int[,] inblue = new int[w, h];
            int[,] ingray = new int[w, h];
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    color = m_Bitmap.GetPixel(i, j);
                    inred[i, j] = color.R;
                    ingreen[i, j] = color.G;
                    inblue[i, j] = color.B;
                    ingray[i, j] = (int)((color.R + color.G + color.B) / 3.0);
                }
            }
            int[,] sobel1 = { { -1, 0, 1 }, { -2, 0, 2 }, { -1, 0, 1 } };
            int[,] sobel2 = { { 1, 2, 1 }, { 0, 0, 0 }, { -1, -2, -1 } };
            int[,] edge1 = edgeDetect(ingray, sobel1, w, h);
            int[,] edge2 = edgeDetect(ingray, sobel2, w, h);
            for (int j = 0; j < h; j++)
            {
                for (int i = 0; i < w; i++)
                {
                    if (Math.Max(edge1[i, j], edge2[i, j]) > 200)
                        r = 255;
                    else
                        r = 0;
                    color = Color.FromArgb(r, r, r);
                    bitmap.SetPixel(i, j, color);
```

```
                }
            }
            pictureBox1.Image = bitmap;
        }
        flag = 1;
        graydo();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
    }
}
```

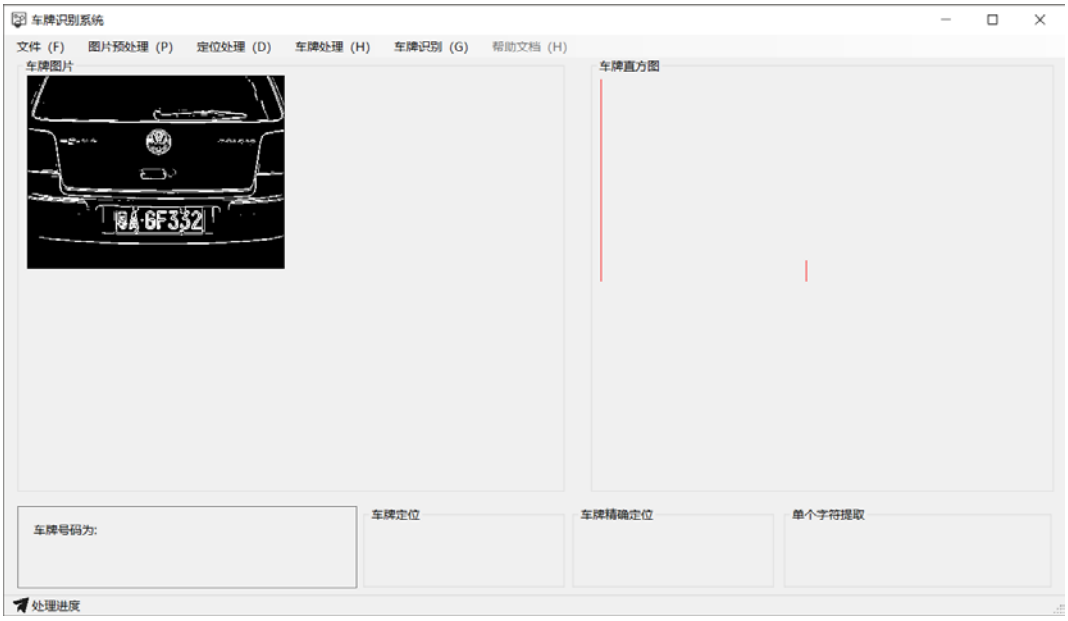


图 4 Sobel 边缘检测

2.2 行扫描算法车牌定位（灰度跳变法）

目前，车牌定位主要有以下几种方法：基于颜色的分割算法；基于遗传算法的分割算法；基于边缘检测的分割算法，基于数学形态学的分割算法等。

本实验中，我们采用了行扫描算法进行车牌定位。行扫描算法是利用了车牌的连续特性，经 Sobel 算子边缘检测后的图像具有黑白两种像素，车牌区域有连续 7 个字符，而且字符与字符之间的距离在一定范围内，因此车牌区域像素的 0、1 次数明显大于非车牌背景中的 0、1 跳变，因此定义从 0 到 1 或者重 1 到 0 为一个跳变，根据牌照区域相对于其它非车牌区域跳变多，而且间距在定范围内和跳变次数大于一定的阈值，就可以确定车牌的水平区域。因为车牌的位置一般在下方，因此我们从下到上的顺序扫描，对图像的每一行进行从左向右的扫描，遇到跳变点即记录当前位置；在本实验中，我们以 16 为阈值，如果某行连续 16 个跳变点以上，就记录下起始点和终止点位置，如果连续有 15 行上述跳变点，我们就认为该区域就是车牌预选区域。

接下来进行车牌的垂直定位，在车牌的水平区域中，最高行与最低行的差值即为车牌在图像中的高度，我国的车牌区域矩形的长高的比约为 3-4 倍，但

是由于我们在信息采集过程中的各种情况以及水平定位时得到的车牌的高，这个倍数可能出现误差，对垂直定位产生一定的影响，我们在本实验中暂时取 3.8 倍的高低行间差值作为车牌的长，然后在我们取得的水平区域中，以车牌长、高的窗口从左到右移动，统计窗口中相邻像素 0、1 的跳变次数；当窗口移动到车牌位置时，跳变次数应该最大，这样即可找到车牌的垂直区域。

代码：

```
private void 车牌定位ToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        Bitmap bitmap = (Bitmap)pictureBox1.Image.Clone();
        int height = bitmap.Height;
        int width = bitmap.Width;
        //定义上下左右边界
        int up = 0, down = 0, right = 0, left = 0;
        //定义车牌的高和宽
        int h, w;
        int[] array = new int[height];
        Color color1 = new Color();
        Color color2 = new Color();
        int number = 0, m = 0;
        if (bitmap != null)
        {
            //逐行自下而上扫描像素0、1跳变数
            for (int i = height; i > 0; i--)
            {
                for (int j = 0; j < width - 1; j++)
                {
                    color1 = bitmap.GetPixel(j, i - 1);
                    color2 = bitmap.GetPixel(j + 1, i - 1);
                    if (Math.Abs(color1.R - color2.R) > 200)
                        array[i - 1]++;
                }
            }
            //水平定位
            for (int i = height - 1; i > 0; i--)
            {
                if (array[i] > 16)
                {
                    if (m == 1)
                        number++;
                    if (m == 0)
                    {
                        m = 1;
```

```
                number++;
            }
        }
        if (array[i] <= 16)
        {
            if (m == 1)
            {
                m = 0;
                if (number > 15)
                {
                    up = i;
                    down = i + number + 3;
                }
            }
        }
        if (up != 0)
            break;
    }

    //垂直定位
    h = down - up;
    w = (int)(3.8 * h);
    int[] arraylist = new int[width - w];
    for (int i = 0; i < width - w; i++)
    {
        for (int j = 0; j < h; j++)
        {
            for (int k = 0; k < w - 1; k++)
            {
                color1 = bitmap.GetPixel(k + i, j + up);
                color2 = bitmap.GetPixel(k + i + 1, j + up);
                if (Math.Abs(color1.R - color2.R) > 200)
                    arraylist[i]++;
            }
        }
    }
    int max = this.maxNumber(arraylist);
    left = max;
    right = max + w;
    Rectangle sourceRectangle = new Rectangle(left, up, w, h);
    c_Bitmap = m_Bitmap.Clone(sourceRectangle,
        PixelFormat.DontCare);
    pictureBox3.Image = c_Bitmap;
    Graphics g = pictureBox1.CreateGraphics();
```

```
            Pen pen = new Pen(Color.Red);
            g.DrawImage(bitmap, 0, 0, m_Bitmap.Width, m_Bitmap.Height);
            g.DrawLine(pen, left, up, right, up);
            g.DrawLine(pen, left, down, right, down);
            g.DrawLine(pen, left, up, left, down);
            g.DrawLine(pen, right, up, right, down);
            flag = 2;
            graydo();
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
    }
}
```

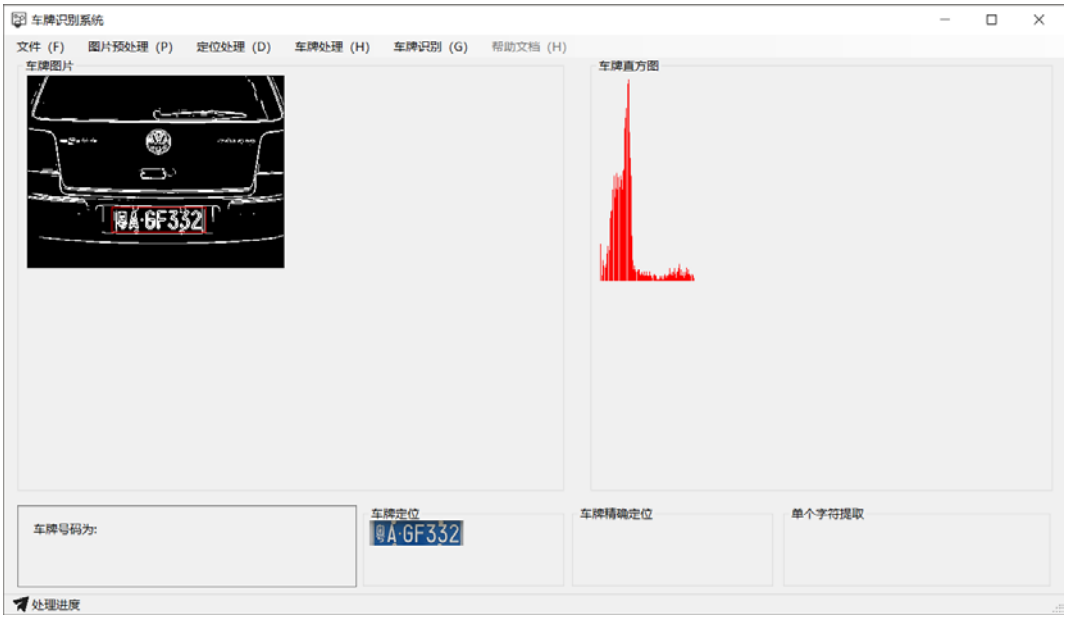


图 5 车牌定位

3. 车牌分割

要进行车牌分割，我们首先要对定位好的车牌进行一些预处理。在本实验中我们采用灰度化、二值化预处理车牌，然后采用一定的算法对车牌进行精确定位提取，得到精确定位的图像之后采用垂直投影法对车牌进行字符分割。

垂直投影法原理如下：对车牌进行垂直投影，得到车牌的垂直投影图（在本实验中我们采用计算白色像素点个数），在垂直投影图上从左到右检测每一个坐标的投影数值。在本实验中，我们检测到第一个大于 3 的投影数值出现时，即是第一个投影区域的左边界；继续向右检测，检测到下一个大于 3 的位置即是第一个投影区域的右边界，据此依次得出其余 6 个投影区域的边界。得到投影区域边界后我们就可以进行字符分割了。

需要说明的是，车牌分割之前得到的精确定位的图像的准确性，以及在算法中的阈值的选择都对车牌分割至关重要，因此我们获取的精确定位图像及在

本实验的算法中我们选取的阈值可能并不能普适地使每一个车牌都能十分准确地进行分割。最终，我们通过对数据进行筛选分析，选择局部最佳的阈值，使算法能够适应大多数的图片。

代码：

```
private void 字符分割ToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        Bitmap curBitmap = (Bitmap)pictureBox4.Image.Clone();
        if (curBitmap != null)
        {
            int cwidth = curBitmap.Width;
            int cheight = curBitmap.Height;
            Color color = new Color();
            int[] county = new int[cwidth];
            int[] array = new int[50];
            int flag2 = 0;
            int n = 0;
            for (int i = 0; i < cwidth; i++)
            {
                for (int j = 0; j < cheight - 1; j++)
                {
                    color = curBitmap.GetPixel(i, j);
                    if (color.R == 255)
                        county[i]++;
                }
            }
            for (int i = 1; i < cwidth; i++)
            {
                if (county[i] > 2)
                {
                    if (flag2 == 0)
                    {
                        array[n] = i;
                        n++;
                        flag2 = 1;
                    }
                }
                else
                {
                    if (flag2 == 1)
                    {
                        array[n] = i;
                        n++;
```

```
                    flag2 = 0;
                }
            }
        }
        Graphics g = pictureBox4.CreateGraphics();
        Pen pen = new Pen(Color.Red);
        g.DrawImage(curBitmap, 0, 0, curBitmap.Width, curBitmap.Height);
        g.DrawLine(pen, array[0], 0, array[0], cheight);
        g.DrawLine(pen, array[1], 0, array[1], cheight);
        g.DrawLine(pen, array[2], 0, array[2], cheight);
        g.DrawLine(pen, array[3], 0, array[3], cheight);
        g.DrawLine(pen, array[4], 0, array[4], cheight);
        g.DrawLine(pen, array[5], 0, array[5], cheight);
        g.DrawLine(pen, array[6], 0, array[6], cheight);
        g.DrawLine(pen, array[7], 0, array[7], cheight);
        g.DrawLine(pen, array[8], 0, array[8], cheight);
        g.DrawLine(pen, array[9], 0, array[9], cheight);
        g.DrawLine(pen, array[10], 0, array[10], cheight);
        g.DrawLine(pen, array[11], 0, array[11], cheight);
        g.DrawLine(pen, array[12], 0, array[12], cheight);
        g.DrawLine(pen, array[13], 0, array[13], cheight);

        Rectangle sourceRectangle0 = new Rectangle(array[0], 0, array[1] -
array[0], cheight);
        array_Bitmap[0] = curBitmap.Clone(sourceRectangle0,
PixelFormat.DontCare);
        pictureBox5.Image = array_Bitmap[0];
        objNewPic = new System.Drawing.Bitmap(array_Bitmap[0], 9, 16);
        array_Bitmap[0] = objNewPic;
        objNewPic.Save("E:\\0.bmp");
        objNewPic = null;

        Rectangle sourceRectangle1 = new Rectangle(array[2], 0, array[3] -
array[2], cheight);
        array_Bitmap[1] = curBitmap.Clone(sourceRectangle1,
PixelFormat.DontCare);
        pictureBox6.Image = array_Bitmap[1];
        objNewPic = new System.Drawing.Bitmap(array_Bitmap[1], 9, 16);
        array_Bitmap[1] = objNewPic;
        objNewPic.Save("E:\\1.bmp");
        objNewPic = null;

        Rectangle sourceRectangle2 = new Rectangle(array[4], 0, array[5] -
array[4], cheight);
```

```
            array_Bitmap[2] = curBitmap.Clone(sourceRectangle2,
PixelFormat.DontCare);
            pictureBox7.Image = array_Bitmap[2];
            objNewPic = new System.Drawing.Bitmap(array_Bitmap[2], 9, 16);
            array_Bitmap[2] = objNewPic;
            objNewPic.Save("E:\\2.bmp");
            objNewPic = null;

            Rectangle sourceRectangle3 = new Rectangle(array[6], 0, array[7] -
array[6], cheight);
            array_Bitmap[3] = curBitmap.Clone(sourceRectangle3,
PixelFormat.DontCare);
            pictureBox8.Image = array_Bitmap[3];
            objNewPic = new System.Drawing.Bitmap(array_Bitmap[3], 9, 16);
            array_Bitmap[3] = objNewPic;
            objNewPic.Save("E:\\3.bmp");
            objNewPic = null;

            Rectangle sourceRectangle4 = new Rectangle(array[8], 0, array[9] -
array[8], cheight);
            array_Bitmap[4] = curBitmap.Clone(sourceRectangle4,
PixelFormat.DontCare);
            pictureBox9.Image = array_Bitmap[4];
            objNewPic = new System.Drawing.Bitmap(array_Bitmap[4], 9, 16);
            array_Bitmap[4] = objNewPic;
            objNewPic.Save("E:\\4.bmp");
            objNewPic = null;

            Rectangle sourceRectangle5 = new Rectangle(array[10], 0, array[11]
- array[10], cheight);
            array_Bitmap[5] = curBitmap.Clone(sourceRectangle5,
PixelFormat.DontCare);
            pictureBox10.Image = array_Bitmap[5];
            objNewPic = new System.Drawing.Bitmap(array_Bitmap[5], 9, 16);
            array_Bitmap[5] = objNewPic;
            objNewPic.Save("E:\\5.bmp");
            objNewPic = null;

            Rectangle sourceRectangle6 = new Rectangle(array[12], 0, array[13]
- array[12], cheight);
            array_Bitmap[6] = curBitmap.Clone(sourceRectangle6,
PixelFormat.DontCare);
            pictureBox11.Image = array_Bitmap[6];
            objNewPic = new System.Drawing.Bitmap(array_Bitmap[6], 9, 16);
```

```
            array_Bitmap[6] = objNewPic;
            objNewPic.Save("E:\\6.bmp");
            objNewPic = null;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
    }
}
```

图 6 车牌精确定位与分割

4. 字符识别

本实验主要采用模板匹配的方法进行车牌的字符识别。

在车牌分割阶段，我们得到了车牌的七个分割部分的二值化图像，利用大量的车牌字符模板，然后通过代码进行模板图片的读入，依次与所有的模板进行匹配，通过迭代实现计算分割后的图片与模板不同点的个数，当不同点个数最小时，即认为匹配到了相应的字符。最后将匹配结果输出，就得到了车牌识别的结果。

代码：

```
private void 车牌识别ToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        int charBmpCount = this.TransformFiles(charSourceBath);//字母数字资源库
中bitmap文件个数
        int provinceBmpCount = this.TransformFiles(provinceSourceBath);//省份
资源库中bitmap文件个数
        int[] charMatch = new int[charBmpCount];//存储当前图片和资源库中图片比对
```

```
后所得的像素不同的个数
        int[] provinceMatch = new int[provinceBmpCount];

        charFont = new Bitmap[charBmpCount];//存储字母数字bitmap文件
        provinceFont = new Bitmap[provinceBmpCount];//存储省份bitmap文件
        for (int i = 0; i < charBmpCount; i++)
        {
            charMatch[i] = 0;
        }
        for (int i = 0; i < provinceBmpCount; i++)
        {
            provinceMatch[i] = 0;
        }
        for (int i = 0; i < charBmpCount; i++)
        {
            charFont[i] = (Bitmap)Bitmap.FromFile(charString[i],
false);//charString存储的是路径
        }
        for (int i = 0; i < provinceBmpCount; i++)
        {
            provinceFont[i] = (Bitmap)Bitmap.FromFile(provinceString[i],
false);
        }

        int matchIndex = 0;//最终匹配索引
        string[] digitalFont = new string[7];

        if (array_Bitmap[0] != null)
        {
            int nWidth = array_Bitmap[0].Width;
            int nHeight = array_Bitmap[0].Height;
            for (int i = 0; i < provinceBmpCount; i++)
            {
                for (int y = 0; y < nHeight; ++y)
                {
                    for (int x = 0; x < nWidth; ++x)
                    {
                        if ((array_Bitmap[0].GetPixel(x, y).R -
provinceFont[i].GetPixel(x, y).R) != 0)
                            provinceMatch[i]++;
                    }
                }
            }
            matchIndex = this.minNumber(provinceMatch);
```

```
            digitalFont[0] = provinceDigitalString[matchIndex].Substring(0,
1);
        }

        if (array_Bitmap[1] != null && array_Bitmap[2] != null &&
array_Bitmap[3] != null && array_Bitmap[4] != null && array_Bitmap[5] != null
&& array_Bitmap[6] != null)
        {
            for (int j = 1; j < 7; j++)
            {
                int nWidth = array_Bitmap[j].Width;
                int nHeight = array_Bitmap[j].Height;
                for (int i = 0; i < charBmpCount; i++)
                {
                    charMatch[i] = 0;
                }
                for (int i = 0; i < charBmpCount; i++)
                {
                    for (int y = 0; y < nHeight; ++y)
                    {
                        for (int x = 0; x < nWidth; ++x)
                        {
                            if ((array_Bitmap[j].GetPixel(x, y).R -
charFont[i].GetPixel(x, y).R) != 0)
                                charMatch[i]++;
                        }
                    }
                }
                matchIndex = this.minNumber(charMatch);
                digitalFont[j] = charDigitalString[matchIndex].Substring(0,
1);
            }
        }
        this.ResultLabel.Text = "" + digitalFont[0] + digitalFont[1] +
digitalFont[2] + digitalFont[3] + digitalFont[4] + digitalFont[5] +
digitalFont[6];
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
    }
    }
```

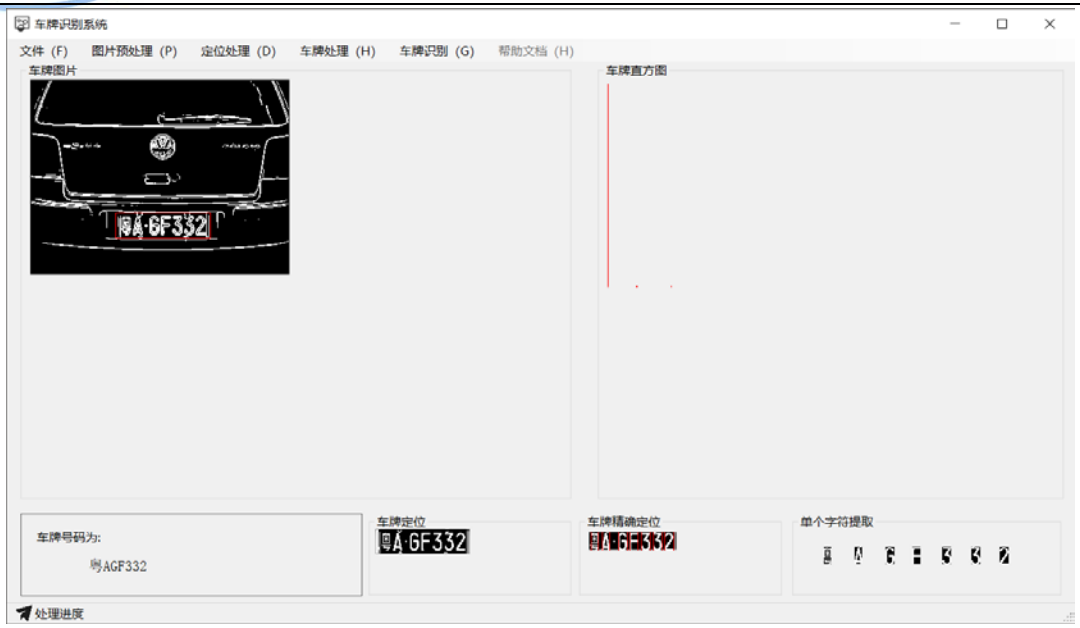


图 5 车牌处理与识别

心得体会

通过本次“车牌识别系统”项目的开发，我加深了对数字图像处理技术的理解，综合实现了对数字图像处理知识（图像灰度化、二值化、滤波、边缘检测、垂直投影定位、图像逻辑运算等）的融合运用，同时使用 C#进行 C/S 图像处理系统的开发，也提高了编程能力，从而将理论与实践充分结合。

回顾前八次实验，从对数字图像处理的陌生到熟悉，这个过程是十分具有意义的，也对我日后其他专业课的学习与计算机编程打好了基础。

附录

参考文献：

1. 郑兴. 基于 C#的车牌识别系统设计与实现. 大连理工大学专业学位硕士学位论文， 2014
2. 张云刚， 张长水等. 利用 Hough 变换和先验知识的车牌字符分割算法的车牌字符分割方法[J]. 计算机学报， 2004, 27: 131-133.
3. 李驰、苗顺占，《基于行扫描的车牌定位算法》，《科技信息》，2008
4. 王晓健，《车牌定位与字符分割算法研究及实现》，北京邮电大学，2009
5. 刘广起，郑晓势，张晓波，《基于图像纹理特征提取的车牌定位算法》,2005
6. 车牌号校验正则表达式 - Jack Tang - 博客园
https://www.cnblogs.com/jacktang/p/5609931.html