第 5 章 停车场车位检测

5.1 概述

随着国人生活水平的提高，汽车已经进入千家万户，停车难的问题也越来越突出。在寻找停车位时，能够实时提供车位信息的停车场大受欢迎，本章将给出一个根据图像自动检测车位的案例。

5.1.1 案例描述

本案例采用一张停车场的俯视图，如图 5-1 所示。这是一个有着 150 个车位的停车场，车位共有 6 排，每排 25 个，车位用白线标出。

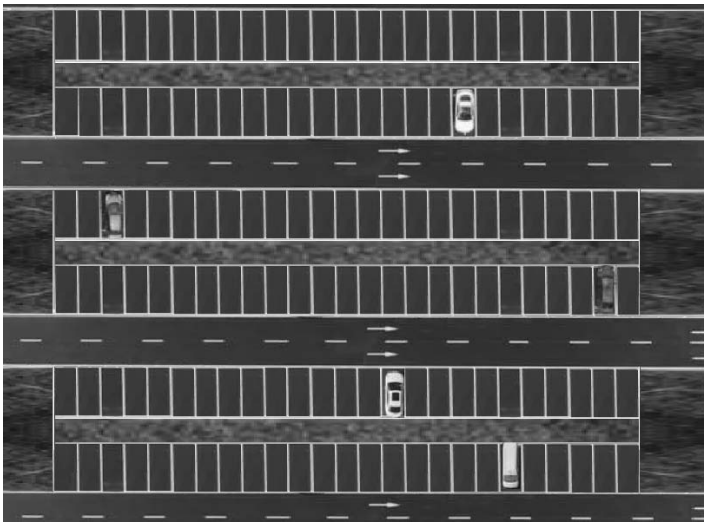

图 5-1 停车场俯视图

5.1.2 案例分析

一般来讲，停车场的图像是从固定的监控摄像头采集的，因此各车位的位置都是已知

的。在本案例中车位上的白线比较整齐，车位大小也基本一致，因此即使不知道车位坐标也可以直接从图像中提取车位信息。

提取这些白线的方法类似于第 4 章中提取围棋棋盘格子线的方法，不过现实中的白线不可能像计算机生成的图像那样整齐划一，因此在进行模板匹配前需要先进行边缘检测，对边缘图像的质量要求也较高，Canny 边缘检测将是较为理想的选择。通过这一步希望提取出如图 5-2 所示的线条，根据这些线条就可以计算出每个停车位的位置。

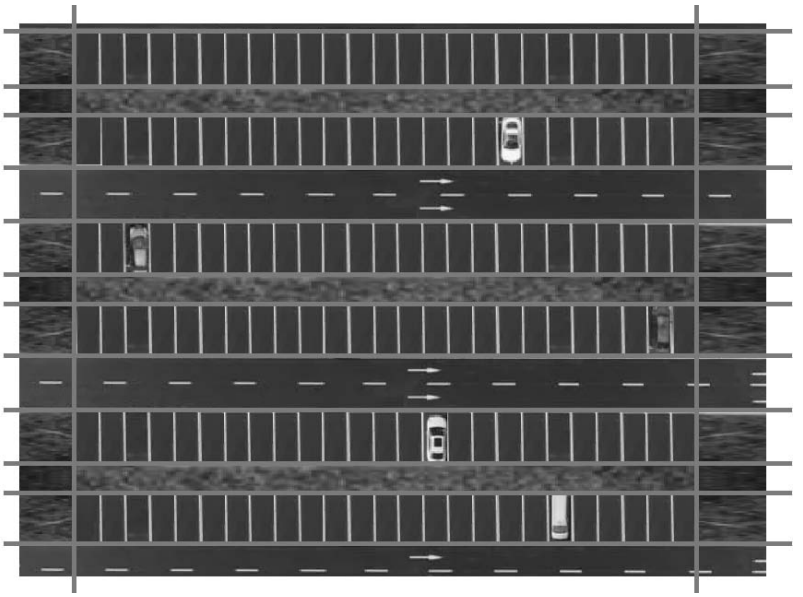

图 5-2　希望提取的线条

车位坐标确定后可以根据颜色信息来判断车位上是否有车，此时需要区分有车和无车时的颜色特征。

无车时车位的特征主要是以下两条：

（1）颜色均一。

（2）底色为较深的灰色。

有车时的特征较为复杂，在大多数情况下颜色并不均匀，例如车窗处的颜色会与车身明显不同，但是少数情况下车体颜色均匀也是可能的，此时就需要通过色彩值是否接近深灰色进行判断。技术上实现上述区分并不难，可以借鉴第 2 章中魔方色块的识别方法。

5.2　总体设计

5.2.1　系统需求

本案例只需 OpenCV，不需要任何第三方库。

5.2.2 总体思路及流程

根据上述分析，本案例的总体流程如下：

（1）Canny 边缘检测。

（2）模板匹配提取水平和垂直的线条。

（3）过滤重复线条并验证。

（4）颜色判断。

（5）判断有车无车并输出结果。

5.3 停车位车位检测的实现

5.3.1 Canny 边缘检测

在本案例中需要通过 Canny 边缘检测提取较为清晰的边缘图像，相关代码在 runCanny() 函数中，其中的关键代码如下：

```
Imgproc.cvtColor(src, gray, Imgproc.COLOR_BGR2GRAY);
Imgproc.Canny(gray, canny, 50, 200, 3, false);
```

经过 Canny 边缘检测后的结果如图 5-3 所示。从图中可以看出，有车的车位有车体轮廓的线条，而无车处则比较干净，根据此特征其实也可以判断是否有车，不过既然通过颜色特征已可区别也就无此必要了。

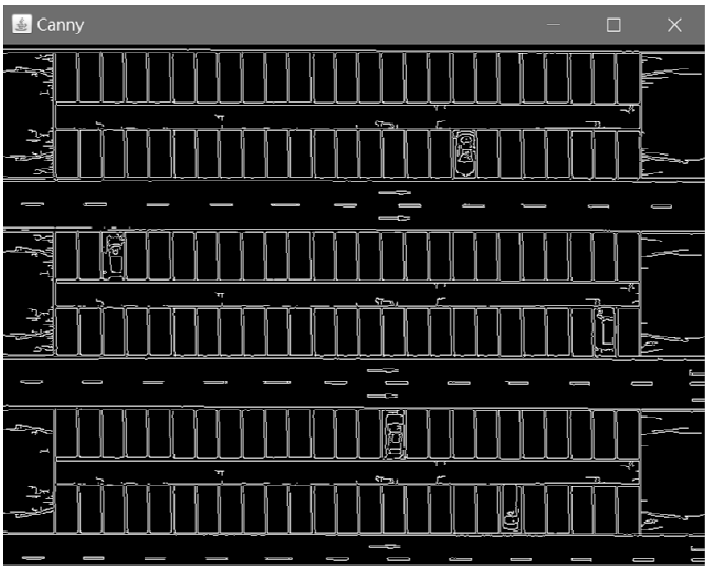

图 5-3 Canny 边缘检测结果

5.3.2　模板匹配

接下来通过模板匹配找出水平和垂直的直线，同样分为 matchLineH()函数和 matchLineV()函数。该函数的基本思想和围棋盘面识别类似，但是为了能匹配到所有有用的线条，模板图像的大小设为 100×3 像素。模板中的直线如果太短，则会导致匹配结果中出现有箭头等标志的线条，如果太长，则可能错过重要的线条。由于匹配出的线条中 y 坐标相同的情况较多，代码中用 pt 数组保存结果，检测到线条的坐标用 1 标记，未检测到的则标为 0。

5.3.3　过滤及验证

模板匹配时通过数组标记的方法避免坐标相同的线条重复出现，但即便如此也无法避免坐标相近的线条重复出现，例如 y 坐标为 7 和 8 处可能都检测出线条，此时需要将重复的线条过滤掉，程序中用 combineLine()函数来过滤。该函数的原理非常简单，如果相邻的两个坐标值非常接近，则直接跳过后面的坐标，这样重复值就被过滤掉了。过滤后还需要对有关线条的数量和距离进行简单验证，相关代码见 checkLines()函数。通过验证后可以将所有车位绘制出来，具体代码见 drawBlocks()函数，绘制结果如图 5-4 所示。

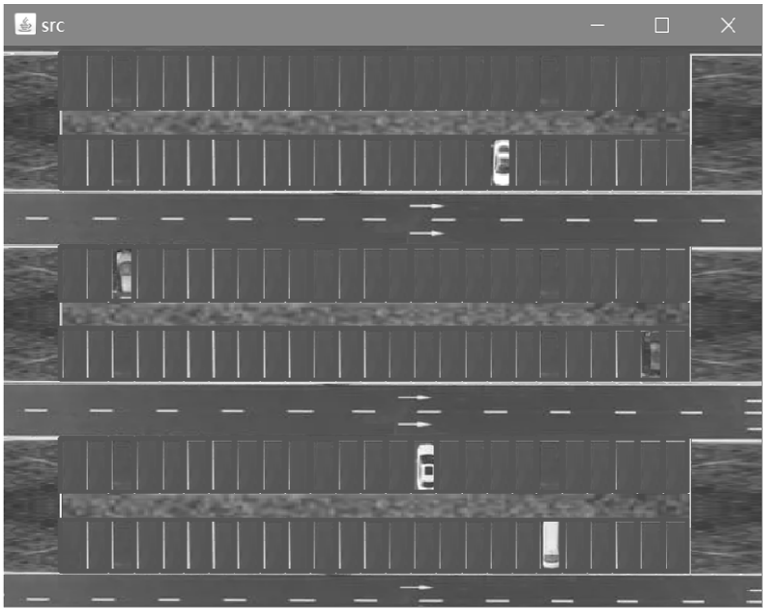

图 5-4　验证后车位的绘制结果

5.3.4　颜色识别

下一步是本案例的关键：通过颜色特征判断车位上是否有车。检测原理在 5.1.2 节已经

介绍过，具体代码见 carColor()函数。该函数中的代码与魔方色块的颜色识别类似，函数中的 rec 参数代表车位的矩形区域。函数通过 RGB 模式识别颜色，在此模式下灰色的 R、G、B 值非常接近。如果 R、G、B 值中任一个值与均值相差较大，则认为该颜色不是灰色。为了识别深灰色，可以将均值限制在较低范围。颜色是否均一可以通过标准差来判断，与魔方案例中类似。

5.3.5 车位检测

上述 carColor()函数可判断某区域内是否停有车辆，对所有车位进行检测后即可获得结果，程序中通过 testAll()函数完成此任务，识别的结果如图 5-5 所示。

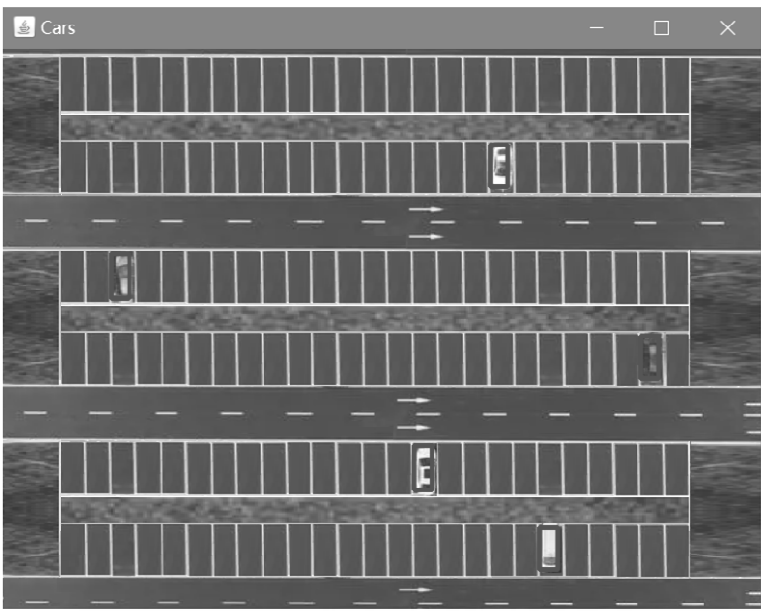

图 5-5 车位中是否停车的检测结果

5.4 完整代码

最后，给出本案例的完整代码：

```
//第 5 章/ParkingLot.java

import org.opencv.core.*;
import org.opencv.highgui.HighGui;
import org.opencv.imgcodecs.Imgcodecs;
import org.opencv.imgproc.Imgproc;
```

```
public class ParkingLot {

    public static int left;
    public static int right;

    public static void main(String[] args) {
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

        //读取图像并提取边缘
        Mat src = Imgcodecs.imread("Parking.png");
        Mat binary = runCanny(src);

        //获取水平和垂直方向的直线
        int[] p0 = matchLineH(binary);
        int[] py = combineLine(p0);
        int[] p1 = matchLineV(binary);
        int[] px = combineLine(p1);

        //验证车位边界值是否适当
        boolean pass = checkLines(px, py);
        if (!pass) {
            System.out.println("未找到适合的车位边界,请确认后重试!");
            System.exit(0);
        }

        //绘制所有车位
        left = px[0];
        right = px[1];
        drawBlocks(src, py);

        //测试所有车位是否有车并绘制测试结果
        testAll(src, py);

        System.exit(0);
    }

    public static Mat runCanny(Mat src) {
        //对图像进行边缘检测
        Mat gray = new Mat();
        Mat canny = new Mat();
        Imgproc.cvtColor(src, gray, Imgproc.COLOR_BGR2GRAY);
        Imgproc.Canny(gray, canny, 50, 200, 3, false);
```

```
        //显示检测结果
        HighGui.imshow("Canny", canny);
        HighGui.waitKey(0);
        return canny;
    }

    public static int[] matchLineH(Mat src) {
        //在白色背景上画一条水平的黑线
        Mat template = new Mat(3, 100, CvType.CV_8UC1, new Scalar(255));
        Point pt1 = new Point(0, 1);
        Point pt2 = new Point(99, 1);
        Imgproc.line(template, pt1, pt2, new Scalar(0), 1);

        //模板匹配
        Mat result = new Mat();
        Imgproc.matchTemplate(src,template,result,Imgproc.TM_CCOEFF_NORMED);

        //搜索匹配值>0.7 的直线,标记 y 坐标的值
        int rows = src.rows();
        int[] pt = new int[rows];
        for (int y = 0; y < result.rows(); y++) {
            for (int x = 0; x < result.cols(); x++) {
                double p = result.get(y, x)[0]; //匹配值
                if (p > 0.7) {
                    pt[y] = 1;
                }
            }
        }

        return pt;
    }

    public static int[] matchLineV(Mat src) {
        //在白色背景上画一条垂直的黑线
        Mat template = new Mat(100, 3, CvType.CV_8UC1, new Scalar(255));
        Point pt1 = new Point(1, 0);
        Point pt2 = new Point(1, 99);
        Imgproc.line(template, pt1, pt2, new Scalar(0), 1);

        //模板匹配
        Mat result = new Mat();
        Imgproc.matchTemplate(src,template,result, Imgproc.TM_CCOEFF_NORMED);
```

```
        //搜索匹配值>0.7 的直线,标记 x 坐标的值
        int cols = src.cols();
        int[] pt = new int[cols];
        for (int y = 0; y < result.rows(); y++) {
            for (int x = 0; x < result.cols(); x++) {
                double p = result.get(y, x)[0]; //匹配值
                if (p > 0.7) {
                    pt[x] = 1;
                }
            }
        }

        return pt;
    }

    public static int[] combineLine(int[] p0) {
        int last = -5;
        int num = p0.length;
        int[] p1 = new int[num];
        int n = 0;                    //有效数组长度
        for (int i = 0; i < num; i++) {
            if (p0[i] == 1) {
                if (i - last > 3) {   //坐标接近的只取 1 个
                    p1[n] = i;
                    n++;
                    last = i;
                }
            }
        }

        //复制数组（数组长度已确定）
        int[] p = new int[n];
        for (int i = 0; i < n; i++) {
            p[i] = p1[i];
        }

        return p;
    }

    public static boolean checkLines(int[] px, int[] py) {
        if (px.length!=2) return false;
        if (py.length!=12) return false;
        int rows = py.length / 2;
```

```
        if (px[1] - px[0] < 300)         //车位不够宽
            return false;
        for (int i = 0; i < rows; i++) {
            if (py[2*i+1] - py[2*i] < 30)//车位不够高
                  return false;
        }

        return true;
    }

    public static void drawBlocks(Mat src, int[] py) {
        //将原图像复制到 draw 中
        Mat draw = new Mat();
        src.copyTo(draw);
        Scalar color = new Scalar(0, 0, 255);
        double width = (right - left) / 25.0;

        //将所有车位用矩形标出
        for (int i = 0; i < 25; i++) {
            int x = left + (int) (width * i);
            int rows = py.length / 2;
            for (int j = 0; j < rows; j++) {
                Rect rec=new Rect(x, py[2*j],(int) width, py[2*j+1]- py[2*j]);
                Imgproc.rectangle(draw, rec, color, 3);
            }
        }

        //在屏幕上显示所有车位
        HighGui.imshow("Blocks", draw);
        HighGui.waitKey(0);
    }

    public static boolean carColor(Mat bgr, Rect rec) {
        //截取车位大小区域
        Mat sub = new Mat();
        Mat roi = bgr.submat(rec);
        roi.copyTo(sub);

        //获取均值和标准差
        MatOfDouble matMean = new MatOfDouble();
        MatOfDouble matStd = new MatOfDouble();
        Core.meanStdDev(sub, matMean, matStd);
        double std = matStd.get(0, 0)[0]; //标准差
```

```
            double b = matMean.get(0, 0)[0];
            double g = matMean.get(1, 0)[0];
            double r = matMean.get(2, 0)[0];
            double avg = (b + g + r) / 3.0;

            if (std > 50)                    //色彩不均匀的
                return true;

            if ((avg < 30) || (avg > 90))    //色彩太亮或太暗的
                return true;

            if ((Math.abs(b - avg) > 5) || (Math.abs(g - avg) > 5)
                    || (Math.abs(r - avg) > 5))
                return true;                 //非灰色系的

            return false;

        }

        public static void testAll(Mat src, int[] py) {
            Mat draw = new Mat();
            src.copyTo(draw);
            Scalar color = new Scalar(0, 0, 255);
            double width = (right - left) / 25.0;
            for (int i = 0; i < 25; i++) {
                int x = left + (int) (width * i);
                int rows = py.length / 2;
                for (int j = 0; j < rows; j++) {
                    Rect rec = new Rect(x + 3, py[2 * j] + 5, (int) width - 6,
                            py[2 * j + 1] - py[2 * j] - 10);
                    if (carColor(src, rec)) {
                        Imgproc.rectangle(draw, rec, color, 3);
                    }
                }
            }

            HighGui.imshow("Cars", draw);
            HighGui.waitKey(0);
        }

    }
```