**华南理工大学第十届智能控制设计大赛**

**初级组•三连棋人机交互视觉系统**

[](javascript:dyn.onClickNextTbn())

队长：牛雅儒（15级智能班）

组员：黄德东（15级智能班）

舒畅（15级计算机全英班）

## 设计任务

基于图像识别，实现对井字棋及中国直棋游戏的实时识别、显示以及人机对战功能。

## 方案设计及实现

### 井字棋

##### 确定区域

通过划线及角点确定区域的方式较为繁琐，而井字棋游戏只有九个区域，内容及形式较为简单，因此最后采用固定摄像头及棋盘、通过对摄像头获得图像进行区域划分来确定棋盘各区域的方法。因此最终将获取的640\*480的图像九等分。

##### 棋子识别

通过颜色识别是较为简单易行的方法。

## 彩色模型

数字图像处理中常用的采用模型是RGB（红，绿，蓝）模型和HSV（色调，饱和度，亮度），RGB广泛应用于彩色监视器和彩色视频摄像机，我们平时的图片一般都是RGB模型。而HSV模型更符合人描述和解释颜色的方式，HSV的彩色描述对人来说是自然且非常直观的。

## HSV模型

HSV模型中颜色的参数分别是：色调（H：hue），饱和度（S：saturation），亮度（V：value）。由A. R. Smith在1978年创建的一种颜色空间, 也称六角锥体模型(Hexcone Model)。

* 色调（H：hue）：用角度度量，取值范围为0°～360°，从红色开始按逆时针方向计算，红色为0°，绿色为120°,蓝色为240°。它们的补色是：黄色为60°，青色为180°,品红为300°；
* 饱和度（S：saturation）：取值范围为0.0～1.0，值越大，颜色越饱和。
* 亮度（V：value）：取值范围为0(黑色)～255(白色)。

## RGB转HSV

设 (r, g, b) 分别是一个颜色的红、绿和蓝坐标，它们的值是在 0 到 1 之间的实数。设 max 等价于 r, g 和 b 中的最大者。设 min 等于这些值中的最小者。要找到在 HSV 空间中的 (h, s, v) 值，这里的 h ∈ [0, 360）是角度的色相角，而 s, v ∈ [0,1] 是饱和度和亮度，计算为：

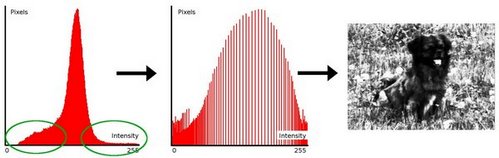
1. max=max(R,G,B)
2. min=min(R,G,B)
3. **if** R = max, H = (G-B)/(max-min)
4. **if** G = max, H = 2 + (B-R)/(max-min)
5. **if** B = max, H = 4 + (R-G)/(max-min)
7. H = H \* 60
8. **if** H < 0, H = H + 360
10. V=max(R,G,B)
11. S=(max-min)/max

OpenCV下有个函数可以直接将RGB模型转换为HSV模型，注意的是OpenCV中H∈ [0, 180）， S ∈ [0, 255]， V ∈ [0, 255]。我们知道H分量基本能表示一个物体的颜色，但是S和V的取值也要在一定范围内，因为S代表的是H所表示的那个颜色和白色的混合程度，也就说S越小，颜色越发白，也就是越浅；V代表的是H所表示的那个颜色和黑色的混合程度，也就说V越小，颜色越发黑。经过实验，识别蓝色的取值是 H在100到140，S和V都在90到255之间。一些基本的颜色H的取值可以如下设置：

1. Orange  0-22
2. Yellow 22- 38
3. Green 38-75
4. Blue 75-130
5. Violet 130-160
6. Red 160-179

## 直方图均衡化

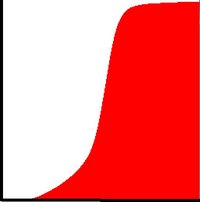
* 直方图均衡化是通过拉伸像素强度分布范围来增强图像对比度的一种方法.
* 说得更清楚一些, 以上面的直方图为例, 你可以看到像素主要集中在中间的一些强度值上. 直方图均衡化要做的就是 拉伸 这个范围. 见下面左图: 绿圈圈出了 少有像素分布其上的 强度值. 对其应用均衡化后, 得到了中间图所示的直方图. 均衡化的图像见下面右图.



* 均衡化指的是把一个分布 (给定的直方图) 映射 到另一个分布 (一个更宽更统一的强度值分布), 所以强度值分布会在整个范围内展开.
* 要想实现均衡化的效果, 映射函数应该是一个 累积分布函数 (cdf) (更多细节, 参考\*学习OpenCV\*). 对于直方图 H(i), 它的 累积分布H^{'}(i)是:

H^{'}(i) = \sum_{0 \le j < i} H(j)

要使用其作为映射函数, 我们必须对最大值为255 (或者用图像的最大强度值) 的累积分布 H^{'}(i)进行归一化. 同上例, 累积分布函数为:



* 最后, 我们使用一个简单的映射过程来获得均衡化后像素的强度值:

equalized( x, y ) = H^{'}( src(x,y) )

相关代码：

//在HSV空间做直方图均衡化

split(imgHSV, hsvSplit);

equalizeHist(hsvSplit[2],hsvSplit[2]);

merge(hsvSplit,imgHSV);

## 选定颜色范围并转化为二值图像

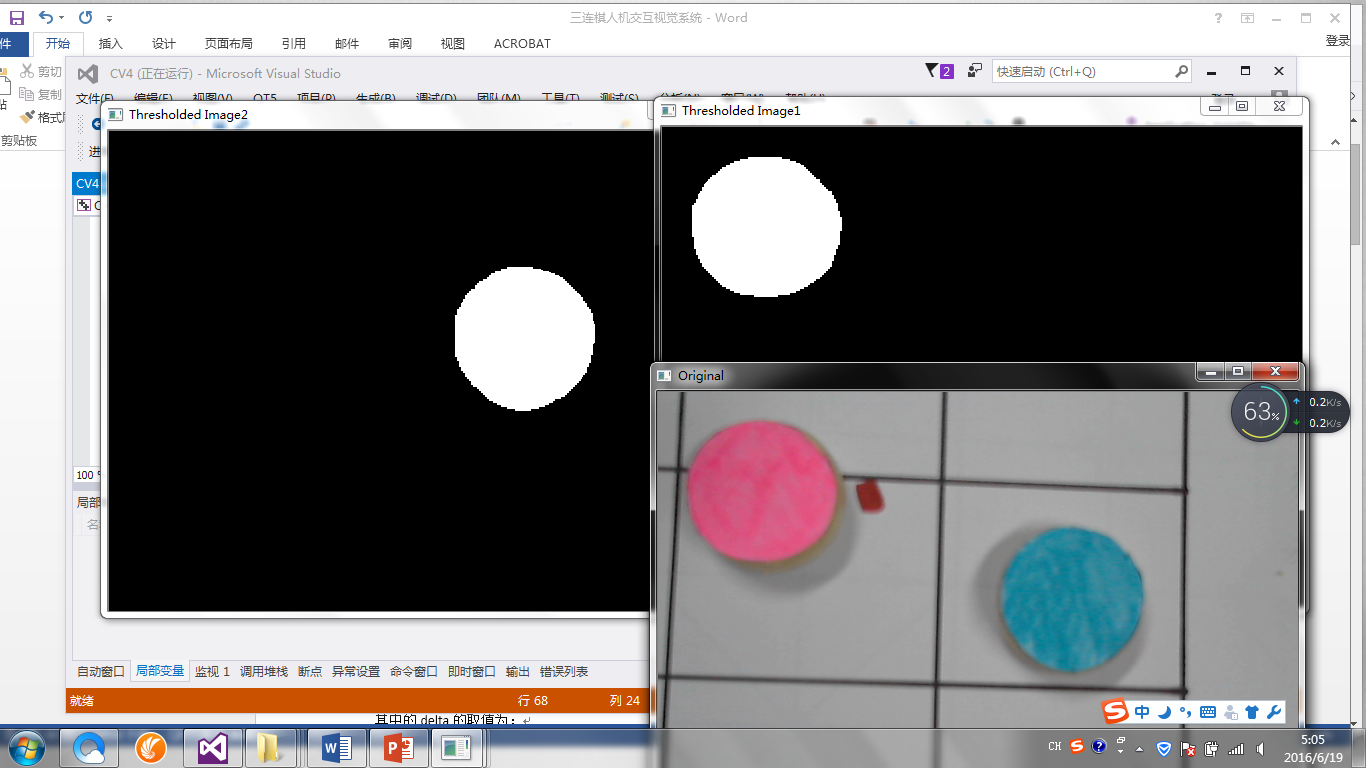
用void inRange(InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst);函数进行颜色检测，这个函数的作用就是检测src图像的每一个像素是不是在lowerb和upperb之间，如果是，这个像素就设置为255，并保存在dst图像中，否则为0。

inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV), imgThresholded); //Threshold the image

通过上面的函数我们就可以得到目标颜色的二值图像，接着我们先对二值图像进行开操作，删除一些零零星星的噪点，再使用闭操作，连接一些连通域，也就是删除一些目标区域的白色的洞。

1. //开操作 (去除一些噪点)
2. Mat element = getStructuringElement(MORPH\_RECT, Size(5, 5));
3. morphologyEx(imgThresholded, imgThresholded, MORPH\_OPEN, element);
5. //闭操作 (连接一些连通域)
6. morphologyEx(imgThresholded, imgThresholded, MORPH\_CLOSE, element);

最终我们的到两幅只显示红棋和蓝棋的二值图像。输出如图所示。



##### 检测

最终分别检测两个二值图像每个区域是否存在像素值为255的点即可确定两种颜色棋子的位置。

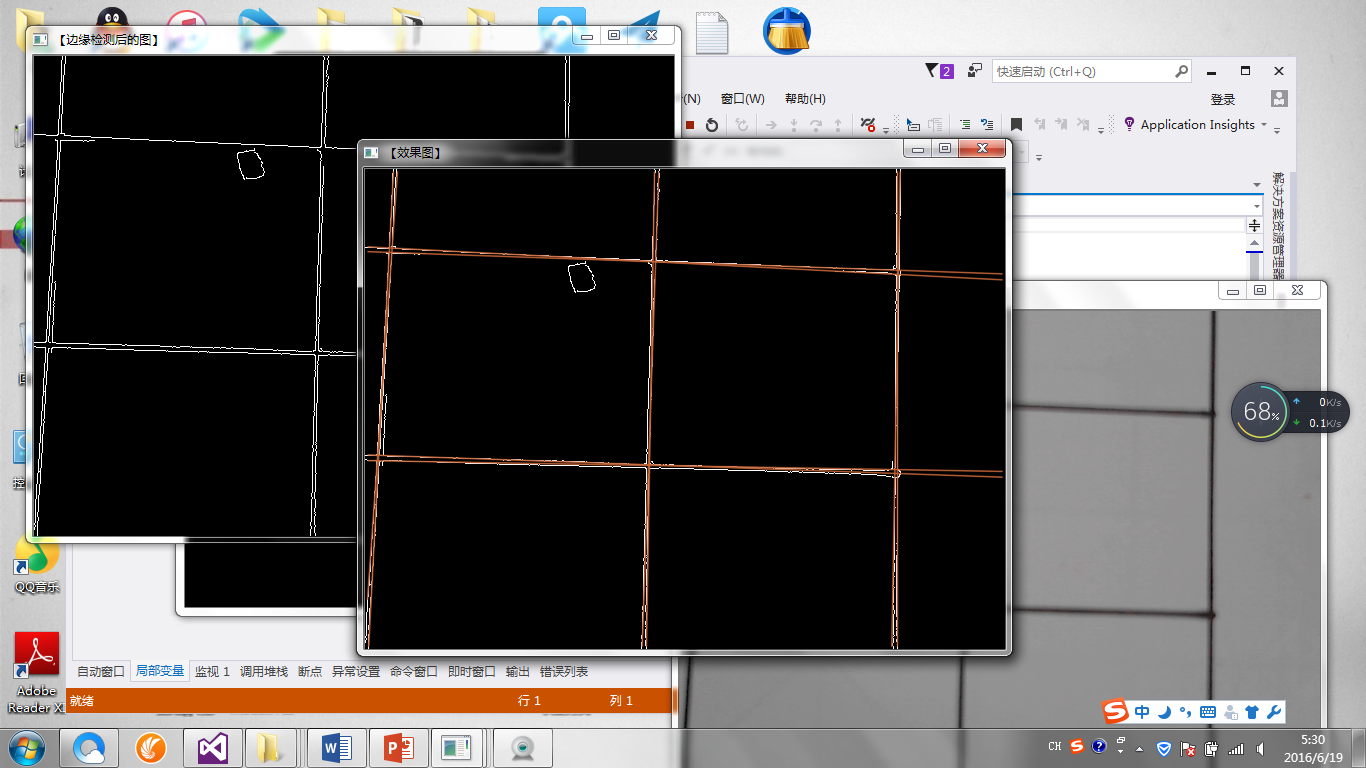
### 中国直棋

1. **方案比较**
2. 哈里斯角点法

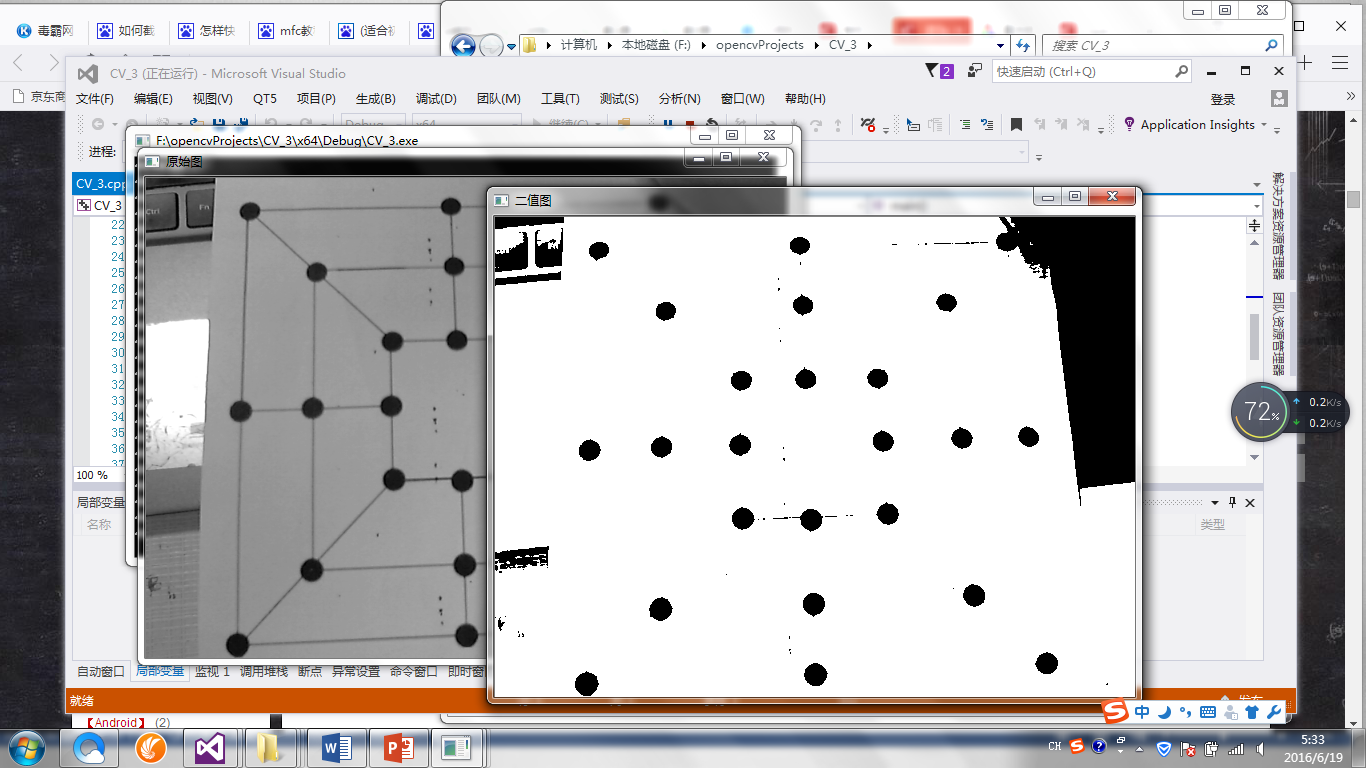
哈里斯角点检测是通过计算灰度的梯度来确定点位的，因此调用该函数后两条直线的交点会确定出四个角点。并且该算法对直线的要求很高，必须细而直，对棋盘的整洁程度也要求很高，否则会出现许多噪点，不便于处理。

1. 霍夫线变换确定交点法

霍夫线变换可以确定出图中的直线，但该方法非常不稳定，一条直线很容易处理得到多条直线，因此确定交点的时候比较麻烦。经霍夫线变换处理得到图像如图所示：



1. 使用灰度图转为二值图像识别点位法

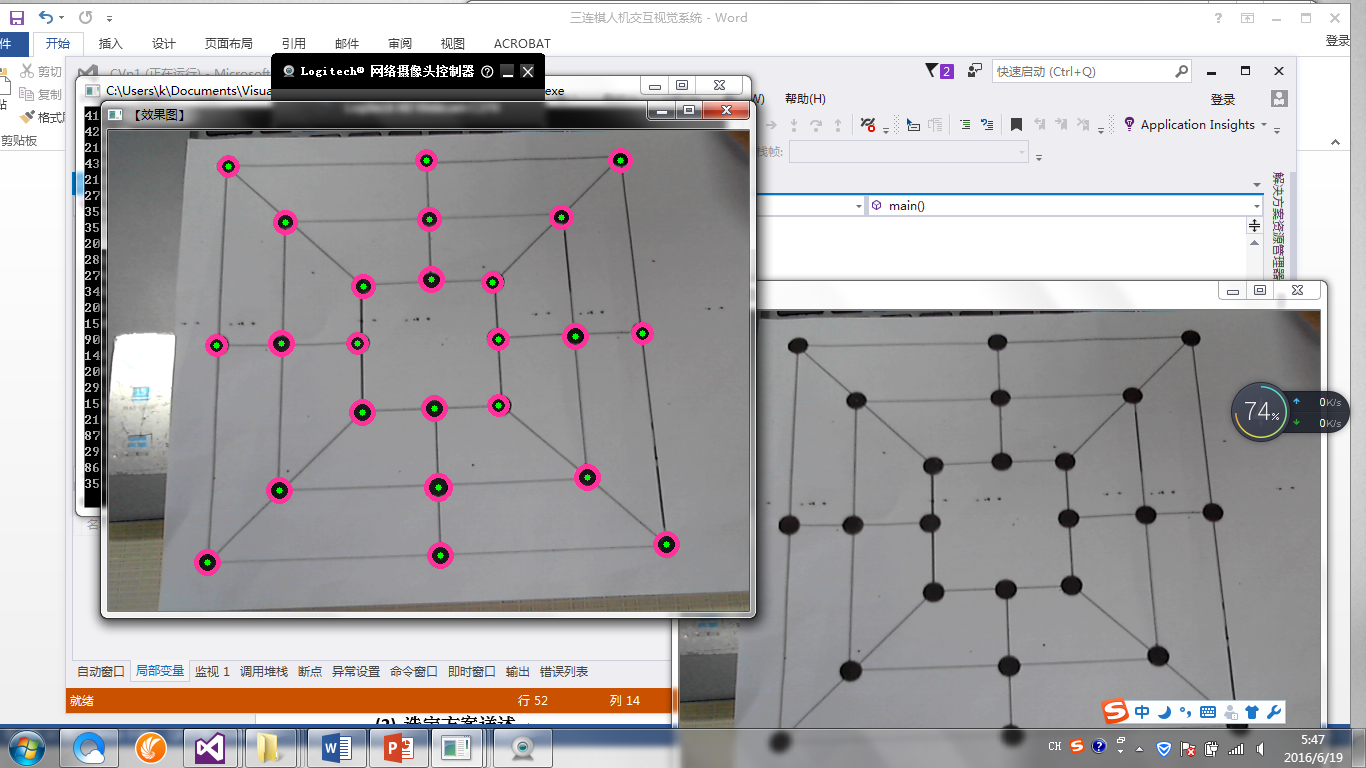
该方法需要将棋盘角点画得大一些并且黑一些，之后将读取摄像头的彩色图像转化为灰度图，进行直方图均衡化和去噪处理，最后获得一幅二值图像，如图所示：

该方法不能要求图片中出现其他相近灰度的背景，因此需要摄像头完全对准棋盘或要求白色底板棋盘背景足够大。

除此之外，得出的点位是像素点的点集，因此需要将所有黑色点位横坐标或纵坐标作差，在一定范围之内的点为一个点集内的点，最后将所有点的横纵坐标求均值，算出点集近似的中心点坐标。

这种确定点位的办法可以有较高的准确度，但是较为繁琐。

1. 霍夫圆变换求圆心确定点位法

霍夫圆变换可以检测出图像中的圆或近似为圆的图形，并且确定出其圆心，因此我们打算将各角点画成圆形，这样便于检测出各个角点的位置并且得到圆心。检测圆的图像如图所示：

因为直接得到了圆心，所以可以直接对点位进行处理。

综上所述，第四个方案有比较明显的优势，被我们采纳。

1. **选定方案详述**
2. 霍夫圆变换法

首先对图像应用边缘检测，比如用canny边缘检测。

然后，对边缘图像中的每一个非零点，考虑其局部梯度，即用Sobel（）函数计算x和y方向的Sobel一阶导数得到梯度。

利用得到的梯度，由斜率指定的直线上的每一个点都在累加器中被累加，这里的斜率是从一个指定的最小值到指定的最大值的距离。

同时，标记边缘图像中每一个非0像素的位置。

然后从二维累加器中这些点中选择候选的中心，这些中心都大于给定阈值并且大于其所有近邻。这些候选的中心按照累加值降序排列，以便于最支持像素的中心首先出现。

接下来对每一个中心，考虑所有的非0像素。

这些像素按照其与中心的距离排序。从到最大半径的最小距离算起，选择非0像素最支持的一条半径。8.如果一个中心收到边缘图像非0像素最充分的支持，并且到前期被选择的中心有足够的距离，那么它就会被保留下来。

**C++: void HoughCircles(InputArray image,OutputArray circles, int method, double dp, double minDist, double param1=100,double param2=100, int minRadius=0, int maxRadius=0 )**

* 第一个参数，InputArray类型的image，输入图像，即源图像，需为8位的灰度单通道图像。
* 第二个参数，InputArray类型的circles，经过调用HoughCircles函数后此参数存储了检测到的圆的输出矢量，每个矢量由包含了3个元素的浮点矢量(x, y, radius)表示。
* 第三个参数，int类型的method，即使用的检测方法，目前OpenCV中就霍夫梯度法一种可以使用，它的标识符为CV\_HOUGH\_GRADIENT，在此参数处填这个标识符即可。
* 第四个参数，double类型的dp，用来检测圆心的累加器图像的分辨率于输入图像之比的倒数，且此参数允许创建一个比输入图像分辨率低的累加器。上述文字不好理解的话，来看例子吧。例如，如果dp= 1时，累加器和输入图像具有相同的分辨率。如果dp=2，累加器便有输入图像一半那么大的宽度和高度。
* 第五个参数，double类型的minDist，为霍夫变换检测到的圆的圆心之间的最小距离，即让我们的算法能明显区分的两个不同圆之间的最小距离。这个参数如果太小的话，多个相邻的圆可能被错误地检测成了一个重合的圆。反之，这个参数设置太大的话，某些圆就不能被检测出来了。
* 第六个参数，double类型的param1，有默认值100。它是第三个参数method设置的检测方法的对应的参数。对当前唯一的方法霍夫梯度法CV\_HOUGH\_GRADIENT，它表示传递给canny边缘检测算子的高阈值，而低阈值为高阈值的一半。
* 第七个参数，double类型的param2，也有默认值100。它是第三个参数method设置的检测方法的对应的参数。对当前唯一的方法霍夫梯度法CV\_HOUGH\_GRADIENT，它表示在检测阶段圆心的累加器阈值。它越小的话，就可以检测到更多根本不存在的圆，而它越大的话，能通过检测的圆就更加接近完美的圆形了。
* 第八个参数，int类型的minRadius,有默认值0，表示圆半径的最小值。
* 第九个参数，int类型的maxRadius,也有默认值0，表示圆半径的最大值。

通过调试，我们将检测到圆的半径范围设置为8-13。

1. 处理点位：

从上到下依次检测圆心，分成七行，再分别从左到右检测对每行的点进行排序，并将每一点在经过霍夫圆变换处理过的vector中的序号存入数组mark当中当中。相关代码如下：

int line1[3];

count = 0;

for (int i = 0; i < midImage.rows&&count < 3; i++)

{

for (int j = 0; j <= 23; j++)

if (i == cvRound(circles[j][1]))

{

line1[count] = j;

count++;

}

}

count = 0;

for (int i = 0; i < midImage.cols; i++)

for (int j = 0; j < 3; j++)

if (i == cvRound(circles[line1[j]][0]))

{

mark[count] = line1[j];

count++;

}

int line2[3];

count = 0;

for (int i = 0; i < midImage.rows&&count < 6; i++)

{

for (int j = 0; j <= 23; j++)

if (i == cvRound(circles[j][1]))

{

count++;

if (count >= 4)

{

line2[count - 4] = j;

}

}

}

count = 0;

for (int i = 0; i < midImage.cols; i++)

for (int j = 0; j < 3; j++)

if (i == cvRound(circles[line2[j]][0]))

{

mark[count + 3] = line2[j];

count++;

}

int line3[3];

count = 0;

for (int i = 0; i < midImage.rows&&count < 9; i++)

{

for (int j = 0; j <= 23; j++)

if (i == cvRound(circles[j][1]))

{

count++;

if (count >= 7)

{

line3[count - 7] = j;

}

}

}

count = 0;

for (int i = 0; i < midImage.cols; i++)

for (int j = 0; j < 3; j++)

if (i == cvRound(circles[line3[j]][0]))

{

mark[count + 6] = line3[j];

count++;

}

int line4[6];

count = 0;

for (int i = 0; i < midImage.rows&&count < 15; i++)

{

for (int j = 0; j <= 23; j++)

if (i == cvRound(circles[j][1]))

{

count++;

if (count >= 10)

{

line4[count - 10] = j;

}

}

}

count = 0;

for (int i = 0; i < midImage.cols; i++)

for (int j = 0; j < 6; j++)

if (i == cvRound(circles[line4[j]][0]))

{

mark[count + 9] = line4[j];

count++;

}

int line5[3];

count = 0;

for (int i = 0; i < midImage.rows&&count < 18; i++)

{

for (int j = 0; j <= 23; j++)

if (i == cvRound(circles[j][1]))

{

count++;

if (count >= 16)

{

line5[count - 16] = j;

}

}

}

count = 0;

for (int i = 0; i < midImage.cols; i++)

for (int j = 0; j < 3; j++)

if (i == cvRound(circles[line5[j]][0]))

{

mark[count + 15] = line5[j];

count++;

}

int line6[3];

count = 0;

for (int i = 0; i < midImage.rows&&count < 21; i++)

{

for (int j = 0; j <= 23; j++)

if (i == cvRound(circles[j][1]))

{

count++;

if (count >= 19)

{

line6[count - 19] = j;

}

}

}

count = 0;

for (int i = 0; i < midImage.cols; i++)

for (int j = 0; j < 3; j++)

if (i == cvRound(circles[line6[j]][0]))

{

mark[count + 18] = line6[j];

count++;

}

int line7[3];

count = 0;

for (int i = 0; i < midImage.rows&&count < 24; i++)

{

for (int j = 0; j < 24; j++)

if (i == cvRound(circles[j][1]))

{

count++;

if (count >= 22)

{

line7[count - 22] = j;

}

}

}

count = 0;

for (int i = 0; i < midImage.cols; i++)

for (int j = 0; j < 3; j++)

if (i == cvRound(circles[line7[j]][0]))

{

mark[count + 21] = line7[j];

count++;

}

}

1. 检测：

在两张二值图像中分别对确定的24个点进行检测，从而判定每个点是红棋还是蓝棋。相关代码如下：

int array1[24];

int array2[24];

for (int i = 0; i < 24; i++)

{

if (imgThresholded1.at<uchar>(cvRound(circles[mark[i]][1]), cvRound(circles[mark[i]][0])) == 255)

{

array1[i] = 1;

}

else

{

array1[i] = 0;

}

}

for (int i = 0; i < 24; i++)

{

if (imgThresholded2.at<uchar>(cvRound(circles[mark[i]][1]), cvRound(circles[mark[i]][0])) == 255)

{

array2[i] = -1;

}

else

{

array2[i] = 0;

}