South China University of Technology

课程设计报告书

题目：三连棋人机交互系统

学院	吴贤铭智能工程学院
专业	智能制造/智能制造/机器人工程
学生姓名	李金鹏/徐希辰/宋雨辰
学生学号	201830640187/201830062026/201830360498
指导教师	
课程编号	
课程学分	
起止日期	2020.05.01-2020.6.12

教师评语	教师签名： 日期：
成绩评定	
备注	

三连棋人机交互系统

一、选题背景

1.主要问题

在此项课题中，我们应着力解决的主要问题是利用非编程摄像头摄取图像，利用机器视觉库开发上位机软件识别棋子落位，实现云玩家井字棋对战，并设计程序判断胜负。

2.技术要求

2.1 基础功能

2.1.1 识别棋子落位

采用摄像头获取图像数据，当两人对弈时，能准确识别出棋盘 A 上两种棋子的位置，利用摄像头持续读取图像，完成对落子位置和颜色的检测，要求不能有误差。

2.1.2 判断胜负

具有判断双方胜负的功能，当任意玩家的胜利条件满足时，程序将自动暂停，否则，当棋盘下满时，程序应提示玩家和棋。

2.1.3 电脑显示

玩家在现实世界的落子应相应的反映在上位机程序界面中，电脑显示识别结果的界面良好。

2.2 拓展功能

2.2.1 人机博弈

针对棋盘 A，具有人机对弈的功能，由电脑输出下棋位置（输出下棋位置后允许人工代下棋子）

2.2.2 对棋盘 B 的落子检测

在保证识别精度的基础上，选用棋盘 B，可实现对棋盘棋子的识别，包括棋

子颜色和落位。

2.2.3 其他拓展功能

在拓展功能 1 的基础上，开发出不同难度等级的人机博弈。

二、方案论证

1.基础功能的设计原理

1.1 基本识别功能实现：机器视觉库选择

随着机器视觉的发展，如今已经形成了许多机器视觉库。如高度开源并且广泛流行的 oepnCV、高度专业化的康耐视公司（Cognex ®）推出的 VisionPro ® 系统、美国 NI 公司的应用软件 LabVIEW 机器视觉软件、MATLAB 机器视觉相关工具箱等。考虑到本项目内容涉及机器视觉的基础领域，因此没有必要采用特别专业的库。MATLAB 程序存在效率和环境问题，混合编程出了问题很难调试。综合多方面考虑，为了更好地提高应用能力，我们选择了开源的 openCV 的 C++接口。openCV 的优势在于免费、基础、开源、多接口，为之后项目的统构提供了方便。
棋子检测及分类识别判定

本项目的机器视觉部分全部基于 openCV3.0 实现。为了方便 openCV3.0 在本项目中的应用，同时简洁快速高效地实现功能，棋子的选用很重要。本项目采用黑白五子棋作为双方棋子，五子棋有以下三个优点。一，容易获得；二，圆形棋子便于应用 openCV 中内置的圆形检测函数进行检测，便于应用颜色处理函数进行颜色的判定从而实现棋子的分类；三，便于将项目进行升级拓展，例如升级开发五子棋识别算法和围棋识别算法等。
棋盘检测

本项目棋盘利用生活中的废旧快递箱制作而成，使用快递箱有以下优势。一，减少经费开支的同时废物利用；二，褐色底色很好地区分开了黑色和白色，使得棋子在棋盘上更易被捕捉到。棋盘检测主要使用了 openCV 中的灰度处理、二值处理、轮廓检测等算法。

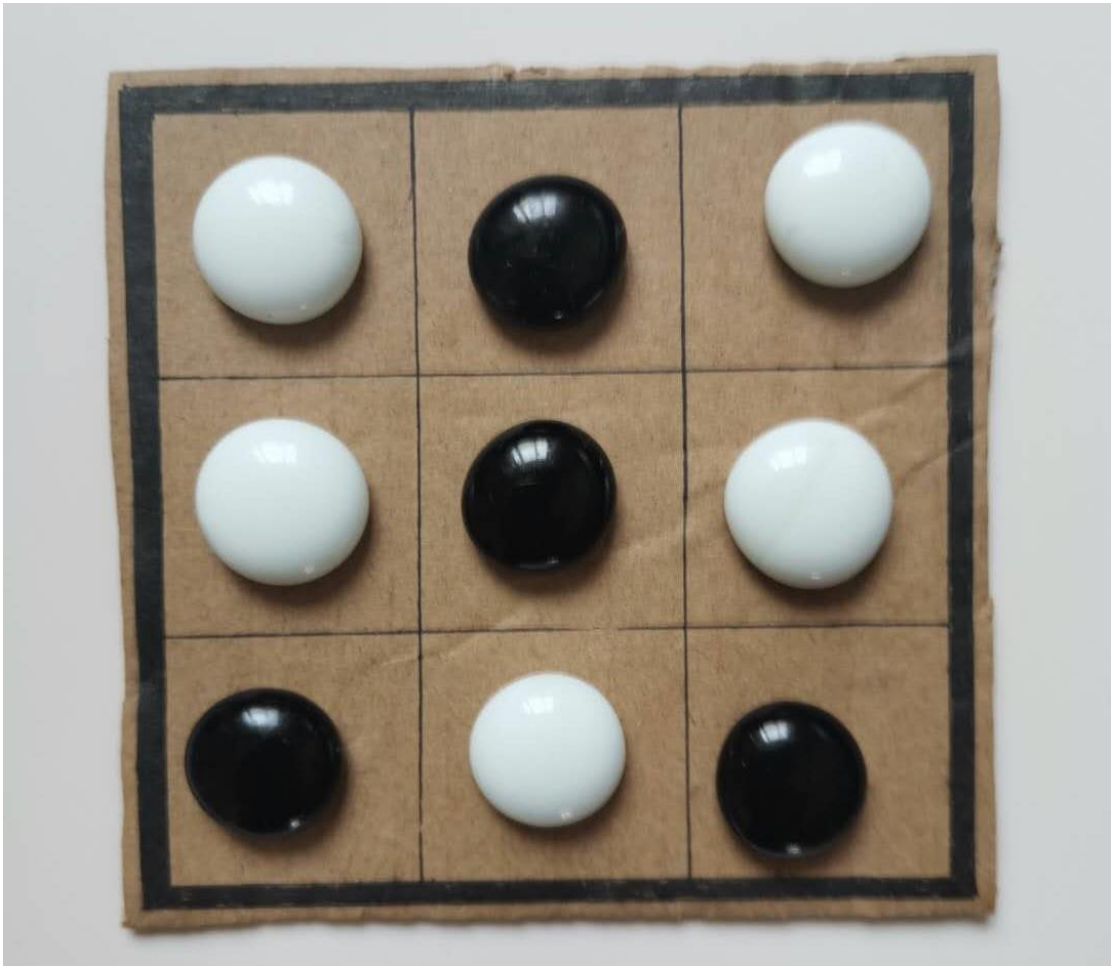


图 1 棋盘与棋子

1.2 胜负识别功能的实现

胜/负

基础功能中的胜负判断规则看似比较简单，实则需要进行一定转换，其核心思想是将棋盘的位置作为一个二维数组进行赋值（在代码中将其定义为 arr[][]），从而便于后续代码处理和判断。同时，用 1 和 2 分别代表不同的颜色，即代表不同的玩家的棋子。然后分别进行行、列和对角线的检查，看有无三连的情况。在优化代码的过程中，我们发现，只需进行行列求和判断即可，而如果将赋值转变为正负相反的的两个数会更加便捷 0。

平局

最初我们的思想是设置一个 step 变量，每进行一步，此变量的值便+1，最终计数到 9，若至此无任何玩家胜出，即判断平局。后来发现这个思想的兼容性和可移植性欠佳，于是新增加一个 isFull 函数，用于判断棋盘是否下满，配合之前已有的胜负判断函数进行平局的判断。

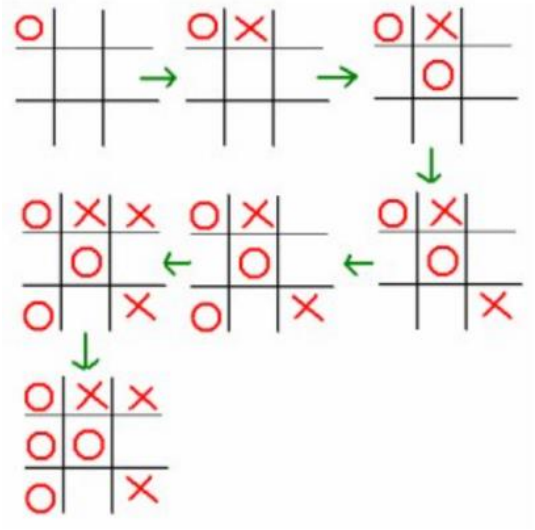


图 2 对战过程

2 拓展功能的设计原理

2.1 人机博弈算法功能的实现

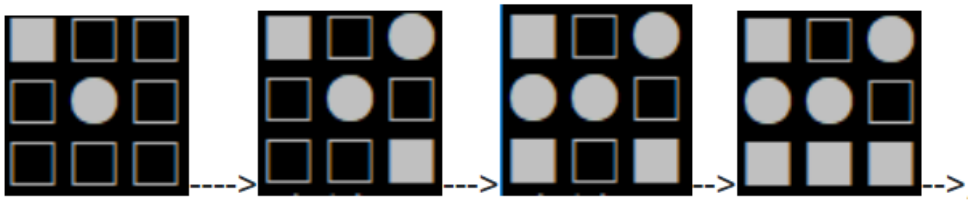
人机博弈的算法广义上分为智能和非智能算法两类，智能算法需要涉及机器学习的相关知识，在此我们小组采用了非智能算法。

本游戏为回合制游戏，不需要考虑实时的、复杂的情形。

本游戏为零和博弈，需要利用极大极小值。即：在敌方最不利的情况下使得己方优势最大[2]。

本算法采用了一个权重机制[3]，相当于强化学习中的奖赏机制，对于不同的棋局，将空位赋上不同的权重值，由电脑计算得出最大的权重值，进行决策。

对于人机博弈的算法，我们设计了四个难度等级，分别为：简单、普通、困难、专家，前二者并未使用权重体系，只是进行随机/找空位落子，后二者我们将权重值进行了改变，权重插值大的算法，优化程度更高，更难以战胜。



2.2 拓展功能识别的实现

原理与基础功能识别相同。

三、过程论述

项目整体架构示意简图：

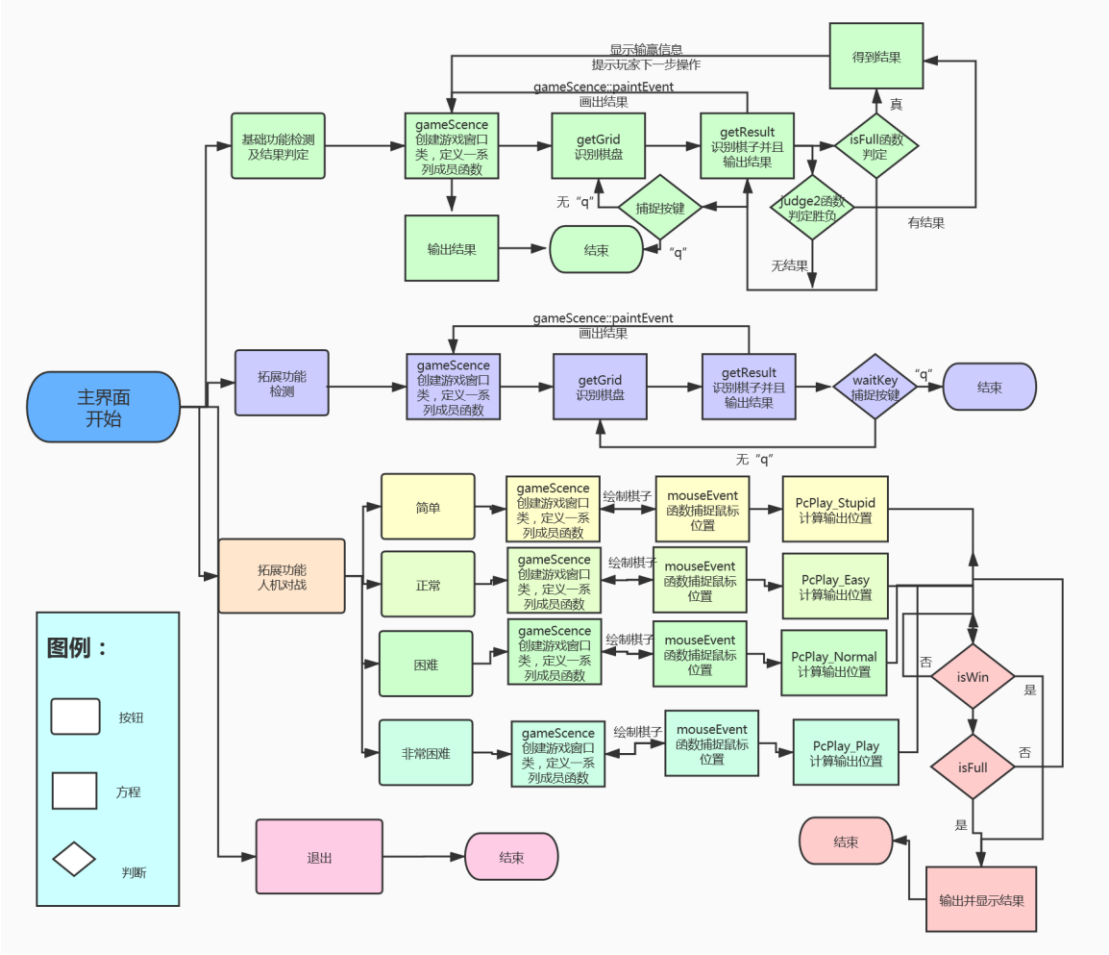


图 3 项目整体架构简图

1.系统界面实现

系统的界面一共可分为 4 个部分：分布图如下

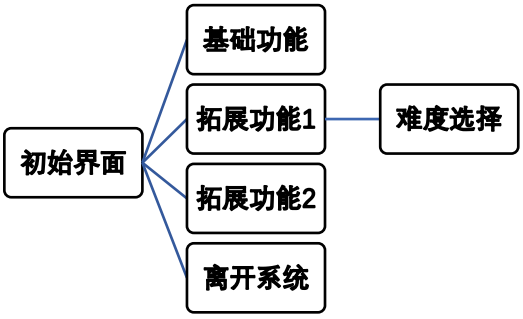


图 4 界面框架

初始化界面会为玩家提供 4 个选择，分别是"Start the game", "man vs com",

“chessboard B”, “exit the game”, 代表的功能分别是基础功能实现，拓展 1 功能实现，拓展 2 功能实现以及离开系统，玩家只需点击不同的按钮就可以进入不同的系统模式

1.1 基础部分：

1.1.1 识别部分

对于基础功能以及拓展功能 2，当玩家选择了模式，系统会自动调用 paintEvent(QPaintEvent *)[4]函数对于界面（除棋子之外的部分）进行绘制，当得到摄像头传输的数据之后，系统会采用不同的哈希算法来将传值转换成系统对应的棋子坐标，并打开绘制函数的“开关”，从而对于棋子进行定位并描绘。

1.1.2 人机对战部分

针对拓展功能 1 的实现，为了增加系统的趣味性，我们设立了四种不同的难度，分别对应系统中的”easy”, “normal”, “hard”, “very hard”按钮, 四种难度对应 4 种不同的算法，玩家可以根据自身水平来选取系统难度，当玩家在棋盘上落白子时候，电脑会计算并调整自己的结果，落下黑子，达到对战的效果，与此同时，为了保证用户的体验，我们在拓展功能 1 里面也保留了下文优化部分中的对战框，返回按钮，以及胜负显示等功能。

1.2 优化部分

1.2.1 对战信息记录

为了优化界面设计以及提高用户的体验感，我们在界面加入了对战信息记录框，前者通过调用 Listarea()函数来对对话框进行绘制，并调用 ItemColor1(bool , int)向对战框中自动根据用户落子位置输入对战信息，从而有效避免作弊。也更有利于比赛的回顾。

1.2.2 返回按钮

系统的右下角始终提供一个返回按钮，用于返回系统的上一页面，从而方便玩家自主中断系统，防止玩家错误点击按钮却无法退出的降低体验感的局面。 基础功能的“back”实现的是退回到初始页面, 并自动调用 deleteAllSlot（）清除上一局系统的信息。具体游戏检测的界面如图所示。

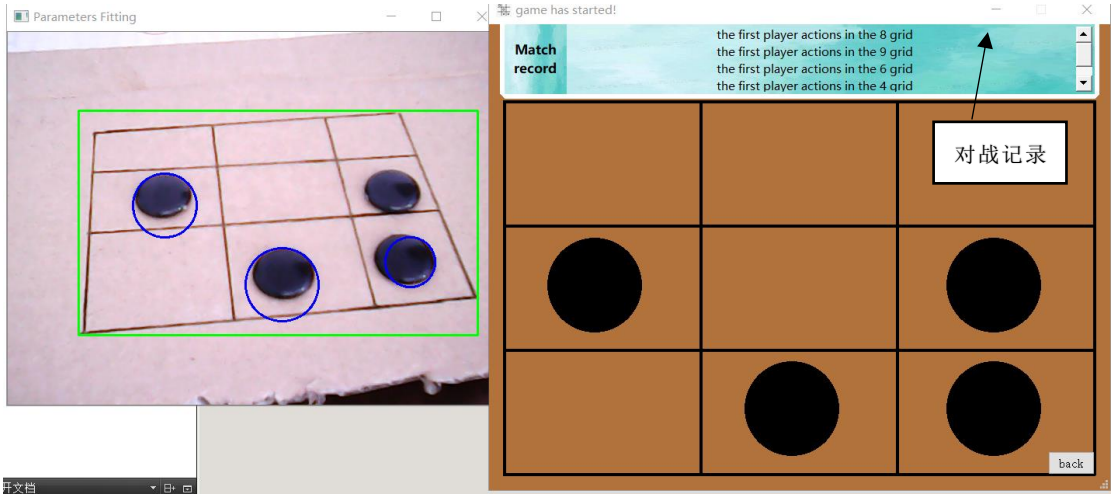


图 5 游戏检测界面

1.2.3 声明对话框

在根据算法得到胜负判断的结果后，系统调用 Dialog(int)来弹出对话框，声明胜利的玩家，并进行是否再次进行系统的询问，点击 ok 按钮则会向主界面发送信号再次刷新系统界面，点击 No 按钮则系统默认退出，自动关闭。

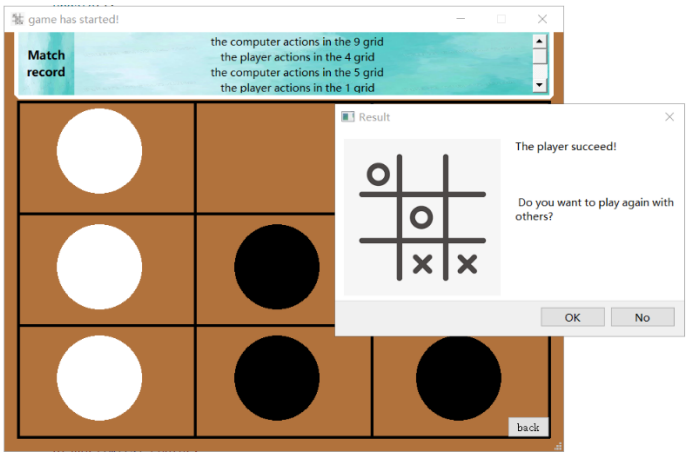


图 6 对话框实例

2.基本功能识别及判定的实现

2.1 视频帧读取及图片预处理

OpenCV 中有为视频内置的专属类 VideoCapture。使用前应先创建对象，之后调用成员函数 read 进行读取。

为了为棋子和棋盘检测做铺垫，需要在视频帧读取之后进行一些预处理。常见的预处理有灰度处理、模糊化处理、二值处理等。本项目中使用 cv::cvtColor 函数将 RBG 图像转化为灰度图像，再使用 cv::GaussianBlur 函数进行高斯模糊。在检测轮廓之前使用 cv::adaptiveThreshold 做自适应阈值处理并且使用 cv::Canny 函数完成最终的轮廓检测[5][6]。

2.2 棋子检测

基于二值预处理，采用 HoughCircles 函数进行棋子检测，该函数参数如下表。

HoughCircles(input_img,output_circles,cv2::HOUGH_GRADIENT,1,20,param1==100,param2=30,minRadius=0,maxRadius=100)

其中关键的参数有 param2，minRadius 和 maxRadius。Param2 是控制圆形检测精度的变量，表示在检测阶段圆心的累加器阈值。该值越小，判定为圆形的阈值就会降低，即会有更多不是完美的圆被检测到，越大则通过检测的圆更加接近完美的圆形。minRadius 和 maxRadius 是控制圆形大小的变量，分别表示检测到圆的半径下限和上限，不在范围内的圆将不予检测。检测后的结果以数组形式储存在变量 output_circles 中，每一个数组都代表一个圆，数组为三维，分别存储圆心的横纵坐标和半径。

2.3 棋子判定

检测到棋子之后要基于颜色对棋子进行分类。本项目中的颜色判定方法是创立黑色和白色蒙版，检测两个蒙版中圆心坐标处像素的值，基于此值对棋子颜色做出分类。

常见的颜色混合模式为 RGB 和 HSV 两种。RGB 模式通过对红(R)、绿(G)、蓝(B)三个颜色通道的变化以及它们相互之间的叠加来得到各式各样的颜色；HSV 是根据颜色的直观特性由 A. R. Smith 在 1978 年创建的一种颜色空间，这个模型中颜色的参数分别是：色调（H），饱和度（S），明度（V）。

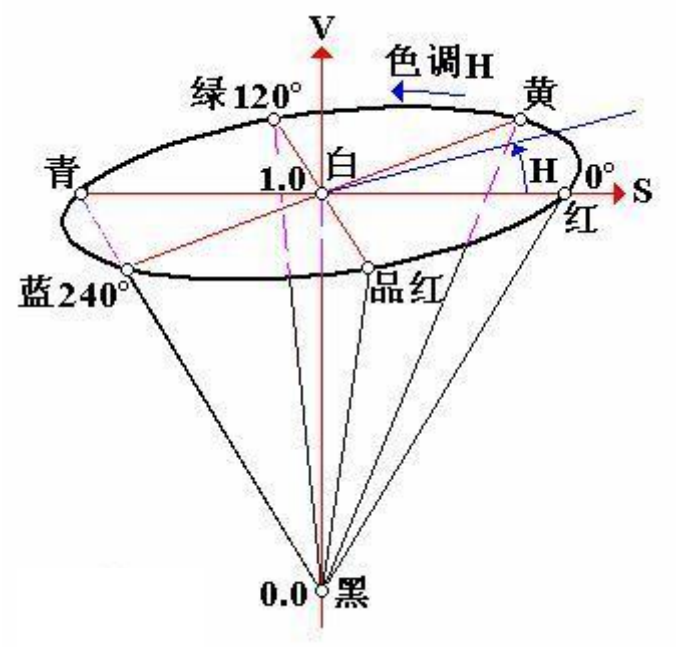


图 7 HSV 模型概念图

基于本项目检测的实际要求，最重要的是实现黑白颜色的区分。由 HSV 颜色空间可知，黑色与白色的 H 与 S 值都有很大自由度，决定因素为 V 值，然而 RGB 颜色混合过于复杂，在实际情况中难以确定精确的阈值，且受到光照强度、光线色调等环境影响较大，故在本项目中选用 HSV 颜色模型，且需要调整的参数仅为 V 值。

建立白色和黑色的 HSV 模型阈值

```
Scalar hsvWhiteLow(0, 0, 120);
Scalar hsvWhiteHigh(255, 255, 255);
Scalar hsvBlackLow(0, 0, 0);
Scalar hsvBlackHigh(255, 255, 50);
```

使用 inRange 函数对颜色进行区分，inRange 函数会生成二值图像，在范围内的部分为白色（255），否则为黑色（0）。

```
cv::inRange(hsv_image, hsvWhiteLow, hsvWhiteHigh, white_hue_image);
cv::inRange(hsv_image, hsvBlackLow, hsvBlackHigh, black_hue_image);
```

用 inRange[5][6]函数将原图分解为两个二值图像，存储在两个变量中，代表图像中检测到的黑色和白色。

读取检测到圆的横纵坐标，并且储存在 Point 类型变量 center 中。

```
if (white_hue_image.at<uchar>(Point(center.x, center.y)) == 255){
认定为白色（用 2 表示）
}
if (black_hue_image.at<uchar>(Point(center.x,center.y))==255){
认定为黑色（用 1 表示）
}
```

2.4 棋盘检测

基于二值预处理的结果，使用 openCV 中的 cv::findContours 函数实现棋盘检测。棋盘检测的原理是，先对图像预处理，之后利用 findContours 函数进行闭合轮廓检测，然后遍历每一个轮廓，利用 cv::contourArea 函数计算每个闭合轮廓的面积，找到面积最大的轮廓作为棋盘轮廓[5][6]。

findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point());

重要的参数为 mode，method 两个参数。Mode 为检测模式,CV_RETR_EXTERNAL 只检测最外围轮廓；CV_RETR_LIST 检测所有的轮廓；CV_RETR_CCOMP 检测所有的轮廓，但所有轮廓只建立两个等级关系，外围为顶层，若外围内的内围轮廓还包含了其他的轮廓信息，则内围内的所有轮廓均归属于顶层；CV_RETR_TREE，检测所有轮廓，所有轮廓建立一个等级树结构。Method 参数定义轮廓的近似方法。CV_CHAIN_APPROX_NONE 保存物体边界上所有连续的轮廓点到 contours 向量内；CV_CHAIN_APPROX_SIMPLE 仅保存轮廓的拐点信息[7]。

具体实现代码如图 8 所示。

```
if (contours.size() != 0) {//Make sure that there are contours
    for (int i = 0; i < contours.size(); i++) {
        int max_area = 50000;//Initialize a value for maximum area
        int area = cv::contourArea(contours[i]);//Compute all the areas of each contours
        if (area > max_area) {
            max_area = area;
            int best_cnt = i;
            Rect rect;
            rect = cv::boundingRect(contours[i]);//Use boundingRect function to analog using a rectangle
            x = rect.x;
            y = rect.y;
            w = rect.width;
            h = rect.height;
            cv::rectangle(img, rect, cv::Scalar(0, 255, 0), 2);//Draw out the detected grid
            return x, y, w, h;//Return the coordinate of the topleft point and its width and height
        }
    }
}
else return 0,0,0,0;
```

图 8 棋盘检测示例源码

根据项目需要，mode 参数我们选择 CV_RETR_EXTERNAL，method 参数我们选

择 CV_CHAIN_APPROX_NONE。

2.5 结果判定

经过棋子检测和棋盘检测，我们已经得到了棋盘位置有关参数 x，y，w，h；圆有关参数 center.x，center.y，color。我们采用简单数学计算的方式判断位置并且输出结果。我们将棋盘的九个格编号为 0-8，建立一个数组存储颜色（1 为黑色，2 为白色），最后返回的结果将是一个携带有不同位置棋子颜色信息的数组。如图 9 图 10 所示。

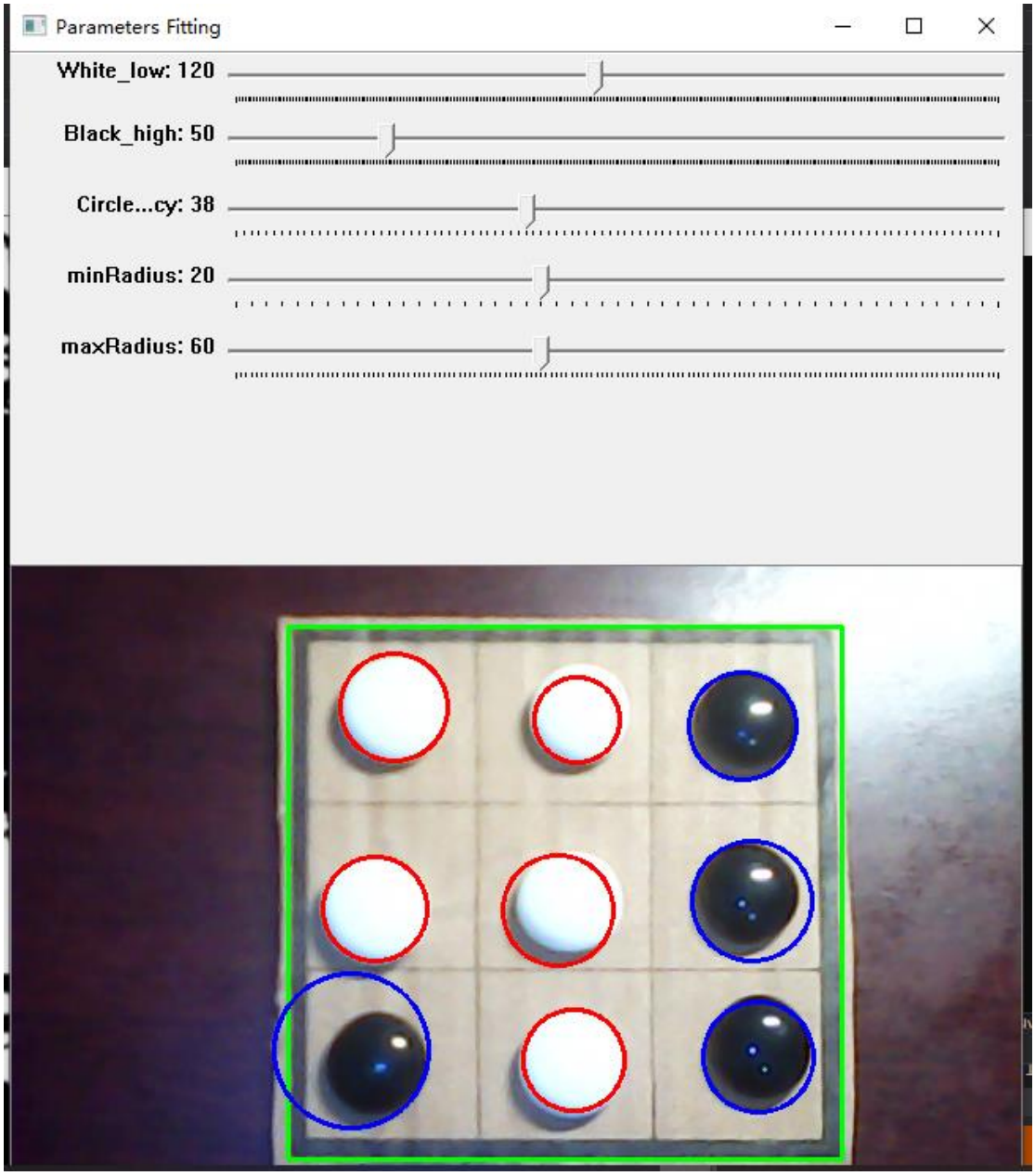


图 9 基本功能检测——摄像头界面

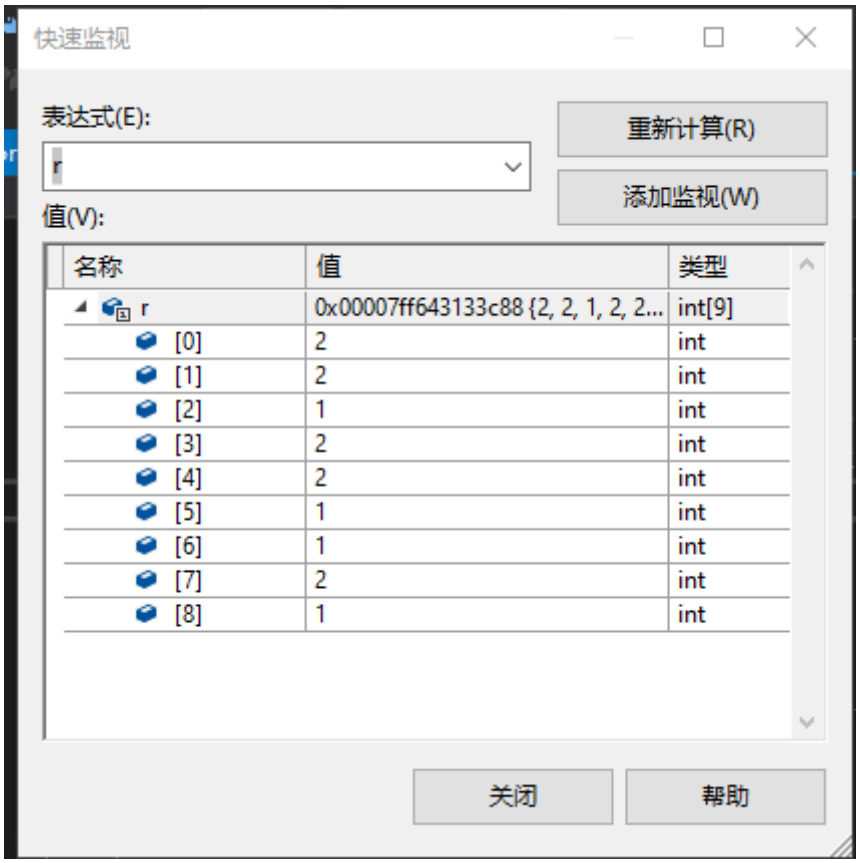


图 10 基本功能检测——检测结果界面

2.6 参数调试

由于非智能算法的限制，各个参数对检测的准确性以及鲁棒性影响很大，项目实施过程中对各个参数的要求非常高。本项目为了解决参数不易调试的问题，利用回调函数设计了基于 openCV 的 GUI 参数调试界面。本项目中重要的参数有决定着颜色区分的正确性的黑色明度（V）的最高阈值和白色明度的最低阈值；决定圆形检测的精度阈值的 HoughCircle 中的 param2 参数；决定检测圆的半径范围的 minR 和 maxR 参数等。利用 createTraceBar 函数，将这些参数与回调函数联系起来，实现了参数的实时动态调整，大大提高了调试的效率，同时提升了功能的稳定性。调试界面已经在图 9 中给予展示。

3.拓展功能识别及判定的实现

拓展功能与基本功能识别的区别主要是输出数据维度问题，将落子可能区域逐一编号将带来程序编写和读取的困难，因此拓展功能在基于基础功能的原理上对判定条件和输出形式进行了改进。

我们将拓展功能的棋盘设计为七行七列的等距棋盘，利用坐标的形式确定位

置。输出的结果存储在长度为 27（可能的落子位置为 27）的向量中，向量元素的数据类型为 openCV 中特有的 Vec3d。因此每一个元素都代表一个棋子，其中的第一个值为行数，第二个值为列数，第三个值为颜色。利用前文中提到的圆心与 x, y, w, h 的数学关系确定行数和列数，采用同样的原理进行颜色的检测。

图 11 为某次测试中的结果。

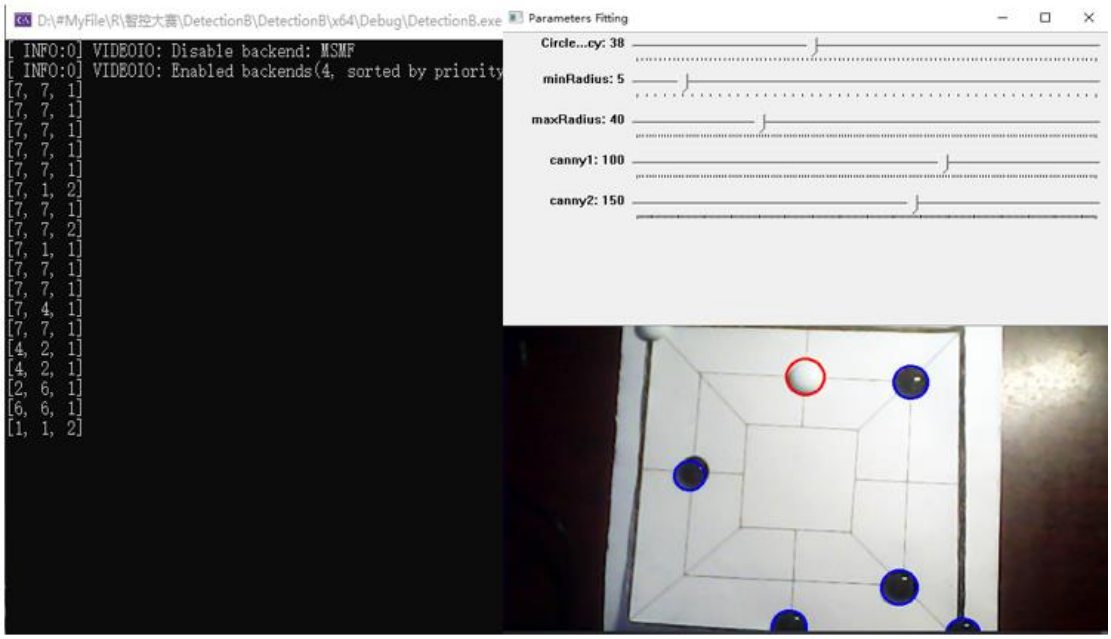


图 11 拓展功能的调试界面

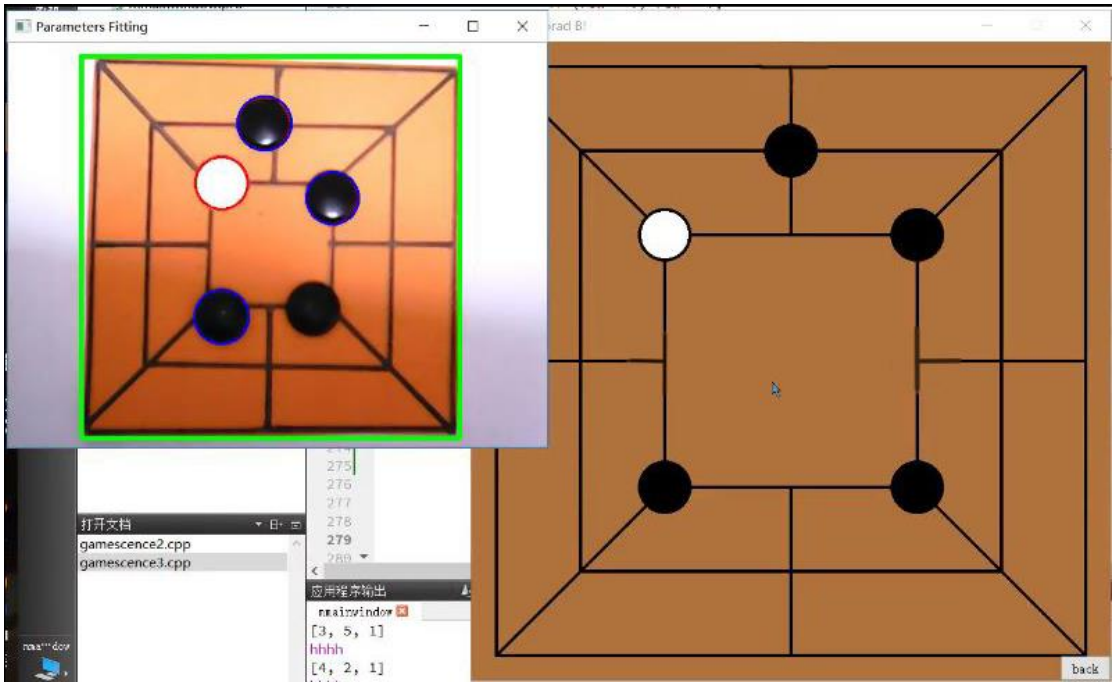


图 12 拓展功能棋盘检测界面

4.胜负判断的实现

胜负判断函数接受两个参数，其一为棋盘位置，另一为玩家号，在胜负判断函数内部，首先将对应格子表示的值转变为玩家序号，然后分别检测行、列、对角线、反对角线，如有三连情况，便返回此玩家序号（只有当前落子玩家才有可能胜出），若无，则检测棋盘是否已经下满，如果下满，则返回和棋的信号，否则将返回正常的信号，等待下一步操作。

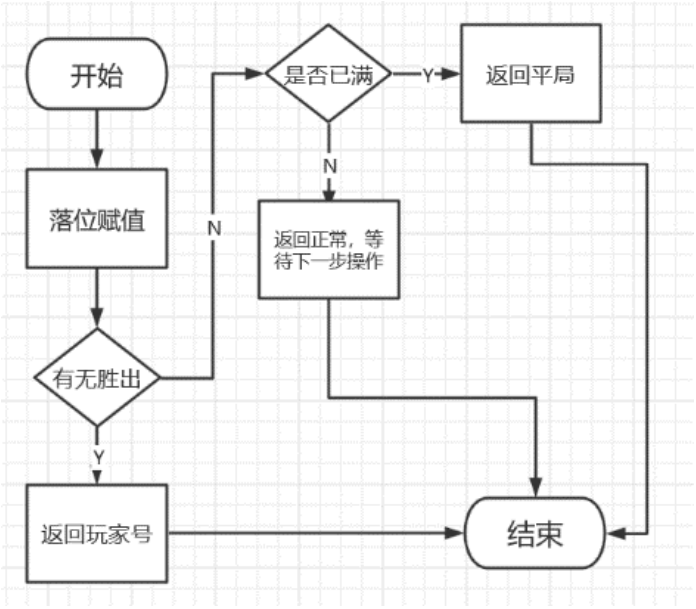


图 13 胜负判断逻辑

人机对战

电脑在自己的回合内，先对局势进行“分析”：求得场上剩余空位的权重值，得到最大的权重，在相应的位置进行落子（赋值）。

权重的具体细节如下：

对于每行/列/对角线，我们有：

---3 个为空，重要性最低，权值设置为 1　　//视为暂时不管

---2 个空 1 个对方，重要性次低，权值为 10　　//一下三个区别不大，可比较随意的设置

---1 个空格 1 个对方 1 个己方，重要行较低，权值 50

---2 个空格 1 个己方，重要性较高，权值为 100

---1 个空格 2 个对方，重要性次高，权值 500　　//不堵住会输

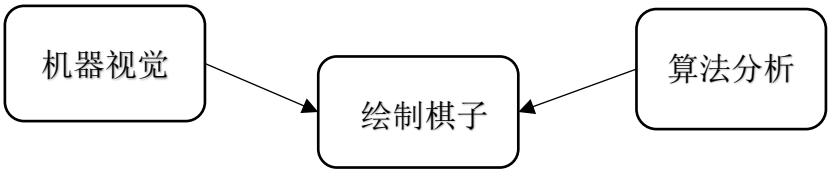
---1 个空格 2 个己方，重要性最高，权值 1000　　//可以直接赢

权重的细节我们根据难度进行了调整，对每个空位置，权值等于行权值+列权值+对角线权值+反对角线权值，这 4 种权值都可以用上面的估算。

四、结果论证

1.输出数据的含义

由于设计的系统的框架可以分为以下 3 个部分：



在基础功能中：机器识别到结果之后会向界面返回一个长度为 9 的二维数组，第一维度代表的结果是位置，对应三连棋的 1~9 格，第二维度代表颜色，用 0，1，2 来代替无子，黑子，白子等结果，之后界面文件会采用算法来将其换算为棋子圆心的坐标点，从而对要绘制的棋子进行定位，我们通过赋予这些数据不同的意义，巧妙将界面文件与机器视觉文件结合在一起，实现了识别功能。以下所示图中的"grid and color"为界面的得到的数组值，"judge"是对颜色的再次确定，通过以下输出可见结合效果令人满意。

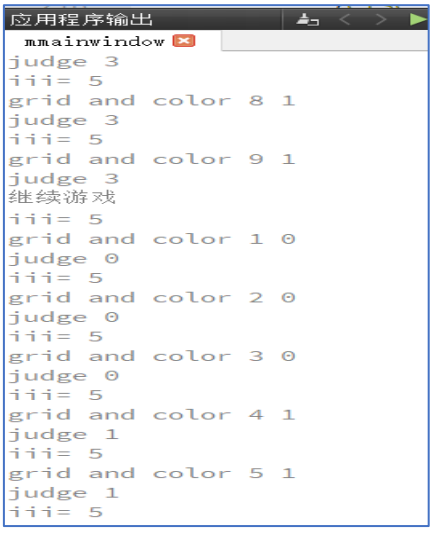


图 14 Qt 界面中的输出结果

在拓展功能 1 中，对于棋盘设立 3x3 的数组来存储落子点的权位，玩家的每一次落子都会对自身棋子以及周围位置的权重产生影响，每一步都会让数组中存储的数据进行调整与优化，规避了常见人机算法中电脑处理“死板”的问题。

在拓展功能 2 中，机器视觉首先会将棋盘分为 7x7 的坐标系，考虑到落子点的颜色，将平面的棋盘引入颜色作为 Z 轴，从三维角度处理问题，将识别的结果返回一个三维的数组，第一维度代表的是行数，取值为 1~7，第二维度代表的是列数，取值为 1~7，第三维度则代表了颜色，具体的识别结果已在图 11 中展示出。

从图 14 可以很清楚看到数组的取值，界面之后会继续采用哈希算法将它换算为坐标点，对棋子进行绘制。

2.重要参数对检测结果的影响

2.1 HoughCircle 函数中 param2 对检测圆精度的影响

由前文叙述可知 HoughCircle 中的 param2 参数控制着圆的精度检测。如果该参数过小（如图 15 左图所示），该检测判定为圆的阈值过低，则过多本不存在的棋子被误测到；如果该参数过大（如图 15 右图所示），则存在部分棋子难以检测到的情况；只有将该参数调整的较为合适（如图 15 中图所示）才能保证检测的准确性。这体现了简单算法对参数要求较高的弊端，也警示我们在开发过程中应当注意重要参数调试的便捷性。

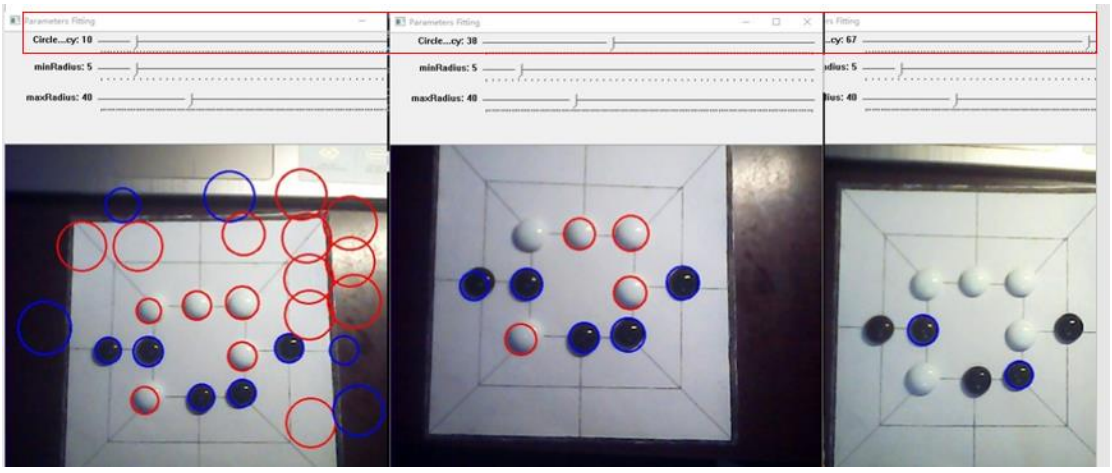


图 15 不同参数对圆精度检测影响

2.2 inRange 函数中黑色与白色明度阈值对颜色检测的影响

inRange 函数起到将 RGB 彩色图像转化为黑白二值图像的作用，对黑白棋子的判定至关重要。然而在调试过程中我们发现，白色因其自身明度（V）值较高，受到光线影响较小，即使有反光现象，因为反光之后仍然会被判定为白色，对最终的判定结果也没有影响。如图 16 所示，即使有反光现象，白色棋子仍然

会被判定为白色。且在光线较暗处（如图 16 左下角棋子）也能有不错的识别。

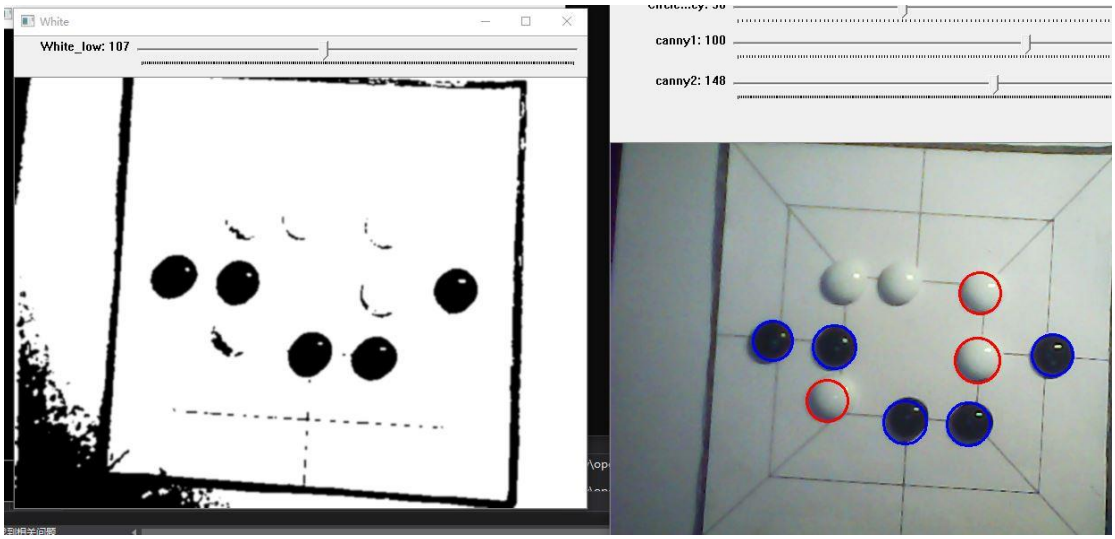


图 16 白色棋子受到光线影响较小

然而，由于黑色棋子明度低，黑色棋子的检测受到环境影响较大。如果参数设置不合理（如黑色的最高明度阈值设置得过低）较强的光线可能导致黑色棋子被误判为白色，且黑色棋子上的白色反光点可能对黑色棋子的判定造成影响。

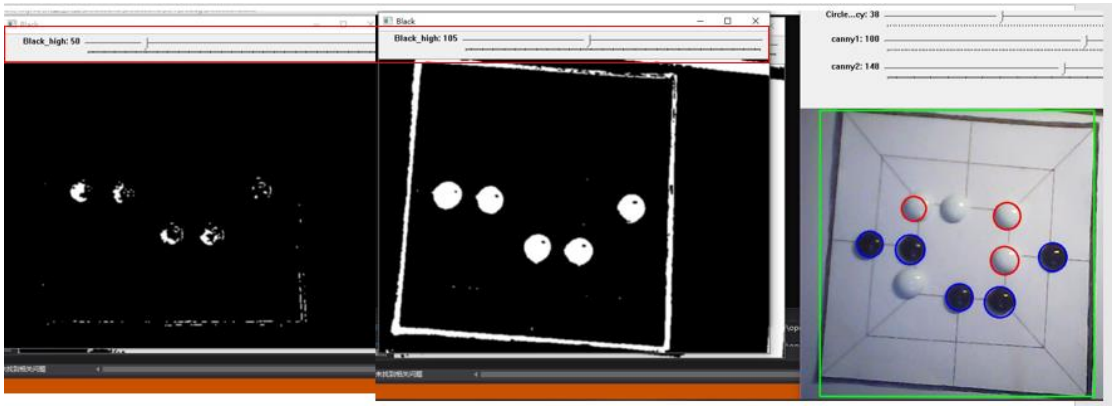


图 17 黑色明度阈值调试过程

图 17 是在室内强光环境下完成的检测实验。左图黑色明度阈值为 50，这一值在白天自然光状态下表现良好（如图 5 所示），然而在强光条件下表现欠佳，可以看到图 17 左图中被认定为黑色的部分非常少。只有将参数值调整为图 17 中图所示时检测才较为理想。

这一问题的出现再次佐证了选择 HSV 颜色模型的优越性。HSV 颜色模型在对白色和黑色的区分上只需要区分明度（V）一个值，因为黑色与白色的色调（H）与饱和度（S）均可在（0，255）范围内调节，没有区分度；也即，采用 HSV 模型，黑色与白色的区分只会受到光照强度一个因素的影响，而与光的冷暖、色调等其他因素无关。这将在降低环境影响的同时增加准确性以及调试的便捷性。

3.结果以及推论：

1.通过采用算法分析，机器视觉识别，界面绘制的分别编写，整合处理设计框架，使三连棋人机交互视觉系统具有很高的健壮性与可维护性，方便进行日后的开发以及优化。

2.人机对战中每一步落子都会让系统使用算法对每一个的权重进行调整，有效的实现电脑对于用户落子的智能化处理，增加了趣味性以及难度

3.对于抽象数据进行现实意义赋值并对二维的游戏引入颜色轴进行升维度处理，很好解决了如何对于位置以及颜色信息的处理的问题。

五、课程设计总结

1.项目总体情况

本项目较好地实现了所有基础功能，即基础棋盘的检测、显示、胜负判断。并且实现了拓展功能中的不同难度人机对弈和棋盘 B 的实时检测、显示。经过实验，准确度和稳定性都较好。由于时间原因，未能完成所有拓展功能，但是以目前的实现基础，拓展功能中的棋盘 B 的胜负判定如果时间稍加宽松也能够实现。且经过合理的方案论证和规划，经费开支仅为一个 USB 摄像头的费用，其他均利用生活中常见物品 DIY 制作，体现了节能环保的理念。

本项目受疫情影响，只能采取线上合作的形式，客观上降低了项目开发的效率。但是由于合理的分工以及任务计划制定和各位成员的共同努力，团队还是较好地完成了所有基本功能并且实现了两项拓展功能。项目实施期间，利用微信、腾讯会议等平台开会十余次，不断汇报进度、及时调整计划，确保时间的合理利用，没有造成时间大量浪费的现象。项目成果令团队的每位成员都感到欣慰。本项目的机器视觉部分主要由李金鹏同学和宋雨辰同学实现，软件的界面搭建与各个功能的连接由徐希辰同学实现。统一使用 C++作为编程语言，编译器使用了微软公司的 Visual Studio，GUI 设计主要使用了 Qt。

2.遇到的问题以及解决方案

由于缺乏项目实战经验，前期的机器视觉部分选用了 python 作为主要语言，但是在混合编程中由于 openCV 与 Qt 基于不同位数的操作系统，产生的问题短

期无法解决。在机器视觉中，棋子颜色识别开始选用 RGB 彩色混合模式，造成识别效果受光线因素影响很大，并不稳定。在拓展功能中，开始采用角点检测的思路，效果并不理想，对各个落子位置的排序和判断成了难以解决的问题。另外，Qt 的项目基于面向对象编程，然而具体功能的实现采用面向过程编程，在整合的过程中遇到了一些没有料想到的问题。

但是经过队员们的通力合作和大量资料的查找和反复调试，最终将问题逐一克服。针对语言不兼容的问题，最后统一使用了 C++；针对机器视觉颜色识别问题，学习并使用了 HSV 色彩模式，并且使用 openCV 内置的 GUI 功能，大大降低了参数调试的难度，提高了识别的精确性和鲁棒性；拓展功能的棋盘识别最终借鉴了基础功能识别的思想，用简单的图块划分方法实现了功能要求；对于一些没有料想到的问题和 bug，同学们充分利用自己的资源，大量查找文献和资料，最终将问题逐一解决。

3.收获及体会

通过本次项目，我们获得了从未接触过的领域开始探索最终实现项目开发的能力，这种能力将对之后的工程实践大有帮助。与此同时，我们收获了项目开发的经验，了解了项目开发的基本流程和一些应当注意到的问题。另外，特殊的合作形式也锻炼了我们团队合作的能力，这一能力也必将对未来的科研学习生活产生积极影响。作为项目的负责人，我最大的体会就是一定要合理规划时间、合理分工和安排任务，尽量发挥每位同学的优势。要及时跟进项目的实施情况，全面了解项目遇到的困难，要有一定前瞻意识，在项目前期就尽量避免未来可能遇到的困难，并且对每种方案有基于全局的分析，做出更加合理的选择。遇到困难时，要积极与队友合作，共同克服困难。我也要借此机会郑重地感谢我的两位队友对我工作的配合以及对项目的所有付出和努力，感谢大家！

六、参考文献

[1] https://blog.csdn.net/qq_28051453/article/details/51843130

[2] Artificial Intelligence: A Modern Approach by Stuart Russell (UC Berkeley) and Peter Norvig (Google)

[3] https://www.cnblogs.com/lfri/p/9880328.html

[4] Johan Thelin.Foundations of Qt Development[M].Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York,:2855 Telegraph Avenue, Suite 600, Berkeley,2009:，

[5] Baggio, Daniel L é lis. Mastering OpenCV with practical computer vision projects[J]. tailieu vn, 2012.

[6] Brahmbhatt, Samarth. Practical OpenCV ||[J]. 2013, 10.1007/978-1-4302-6080-6.

[7] https://blog.csdn.net/dcrmg/article/details/51987348