摘要

机器人在人们的日常生活中扮演着越来越多的角色，桌面 3D 打印机、扫地机器  
人……这些轻便、低成本的小型机器人虽然不具备工业级的高精度、高承载能力，但应  
对生活中特定的任务已经足够，但是在人交互娱乐领域仍有空缺。

在当今时代，随着互联网的发展和智能手机的应用，人们很少用传统的真实的棋子下棋，更多的是实时的在电脑端或者移动端下棋，方便而且快速，还能够跨地域下棋。我们课题针对于此，希望能够制作出一台能够自主下棋的机器，通过计算机计算，控制Delta机械手实现人机对弈，将虚拟棋盘再次搬上桌子上。

本课题主要介绍在采用delta机械手下象棋的过程，主要是通过机器视觉将棋盘上棋子的位置，然后将其移动的位置变换通过计算机象棋算法的函数接口得到计算机的下一步走法，然后将G代码控制指令主要包括视觉识别，象棋算法的实现，将G代码命令发送到下位机控制机械手，然后进行相互通信完成一整套动作。

**关键词**：Marlin，视觉，机电控制，象棋，Delta机械手

目录

[说明 I](#_Toc60571520)

[摘要 II](#_Toc60571521)

[目录 III](#_Toc60571522)

[1 视觉提取 1](#_Toc60571523)

[1.1 棋盘提取 1](#_Toc60571524)

[1.1.1 轮廓提取 1](#_Toc60571525)

[1.1.2 仿射变换 2](#_Toc60571526)

[1.2 棋子提取 3](#_Toc60571527)

[1.2.1 霍夫圆变换 3](#_Toc60571528)

[1.2.2 二值化阈值分割 4](#_Toc60571529)

[1.3 坐标提取 4](#_Toc60571530)

[2 串口通信 7](#_Toc60571531)

[2.1 GUI界面设计 7](#_Toc60571532)

[2.2 通信实现 8](#_Toc60571533)

[2.3 通信协议 8](#_Toc60571534)

[3 象棋算法 12](#_Toc60571535)

[3.1 搜索引擎 12](#_Toc60571536)

[3.1.1 最大最小值搜索引擎 12](#_Toc60571537)

[3.1.2 极小窗口搜索引擎 13](#_Toc60571538)

[3.1.3 迭代深化搜索引擎 15](#_Toc60571539)

[3.2 评估方法 15](#_Toc60571540)

[4 总结与致谢 17](#_Toc60571541)

[附录 18](#_Toc60571542)

1. 视觉提取

我们采用摄像头对棋盘实时图像进行采集，然后进过视觉处理获取我们感兴趣的区域，再逐步分析棋子的特征。视觉所需要的环境为VS2019,opencv。在这个过程中，我们采用了多线程进行处理，一个线程持续调用摄像头，连续显示拍摄的图像，另一个线程是在人每走完一步后，轮番扫描所有棋子，检测棋子数量的变化情况和棋子的位置变化情况，然后作为参数输入到象棋算法，让机器人思考下一步的移动。

* 1. 棋盘提取

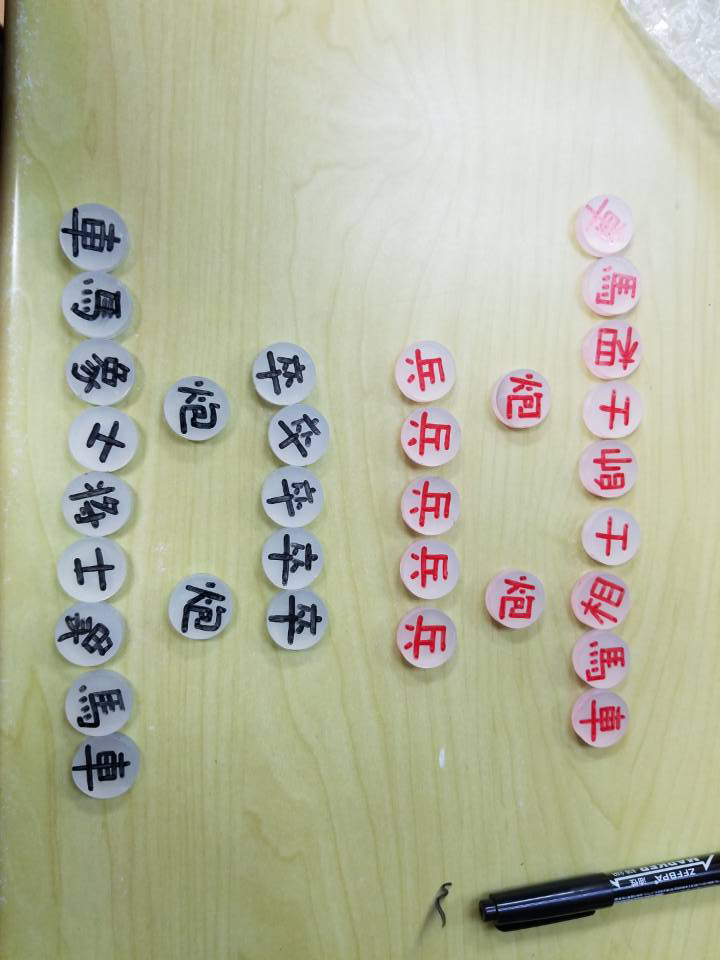
对于视觉部分的处理，主要受到Delta机械手的工作空间的限制，所以在市面上买到的棋子的尺寸不合适，所以我们用亚克力板自制了直径20mm的棋子，用一张A3纸重新打印了棋盘，不仅解决了尺寸问题，棋子的颜色更加便于分析，如图1-1所示。

图1-1

* + 1. 轮廓提取

鉴于Delta机械手的形状和工作空间的大小已经确定，所以摄像头的空间位置不能摆在了机械手上或者是机架的正中心区域，因为这样摄像头的视角不够大，无法覆盖整个棋盘。所以我们只能将摄像头安装在机架的侧上位置。由于摄像头倾斜角度的存在，拍摄出来的照片是一个不规则四边形。

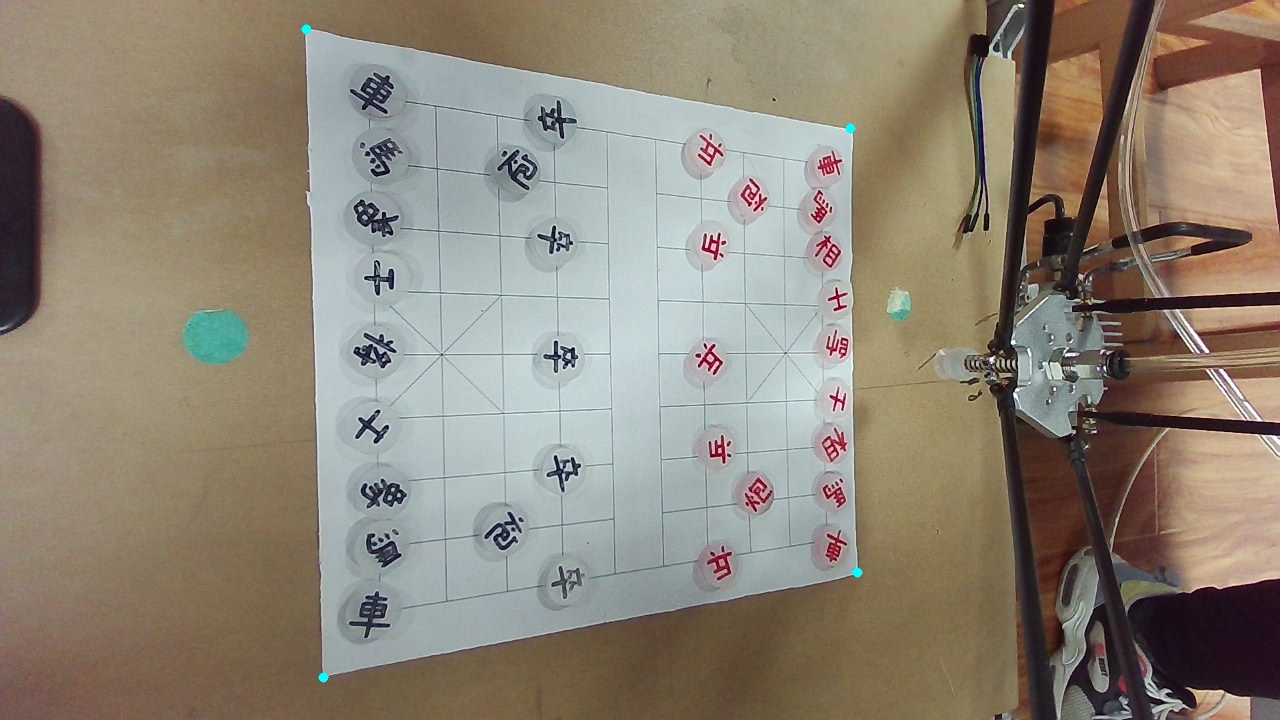
主要的轮廓提取思路为，首先对1280\*720图像的图像进行灰度化，然后进行高斯滤波去噪声，去完噪声后，因为白纸与环境色差相差比较大，所以直接用Canny算子进行边缘提取，然后用findContours()函数获取外部轮廓，筛选出最大面积的区域，那就是我们自己用A3纸打印的棋盘，得到相应的区域后，我们采用convexHull（）进行多边形闭包拟合，得到的是一个不规则四边形，然后就得到如下图1-2，四个蓝点标注的图像。

图1-2

* + 1. 仿射变换

由于得到的图像是一个不规则四边形，所以要对他进行仿射变换，将他的形状重新变为矩形。由轮廓提取后的四边形有四个角点，所以需要对其进行排序，按照左上、右上、左下、右下四个点进行排序，排序算法暂且不详系阐述。然后自己定义一个二维数组，里面存储我们目标位置的四个点，通过从opencv内置函数得出仿射变换的变换矩阵，直接得到变换后的图像。

获取仿射变换后的图像，通过棋盘与边缘的比例尺寸获取棋子方格的位置。如图1-3所示，通过上一部分讲到的四个角点和我们规定图像的大小的四个点进行仿射变换就能够将其变为理想的矩形区域。

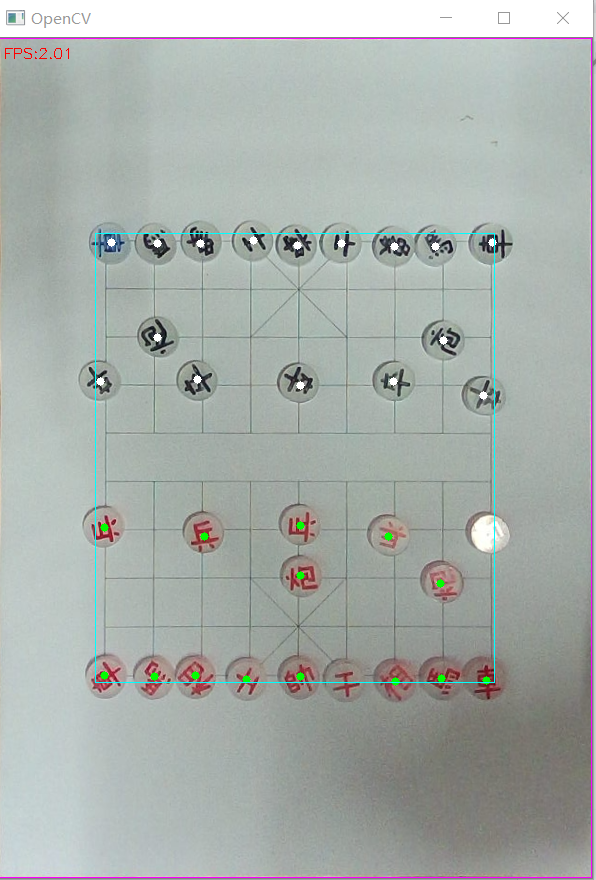


图1-3

* 1. 棋子提取
     1. 霍夫圆变换

通过得到的ROI区域，首先先进行灰度处理，再用中值滤波去除噪声，然后用霍夫圆变换提取圆形。此时，我们会得到棋子的坐标和半径的集合，通过函数复制出来一一分析棋子。其中的霍夫圆变换函数主要需要调试参数，比如最大最小圆半径，最小的圆心距离

* + 1. 二值化阈值分割

得到每一个棋子的坐标和半径，然后遍历包含每一个棋子的区域，进行灰度化处理，给定一个阈值，然后遍历像素点，超过阈值就判定为红色，这一步主要是统计红棋和黑棋的棋子个数，并且填充到我们预先定义的坐标数组中去，为后面的棋子移动做准备。

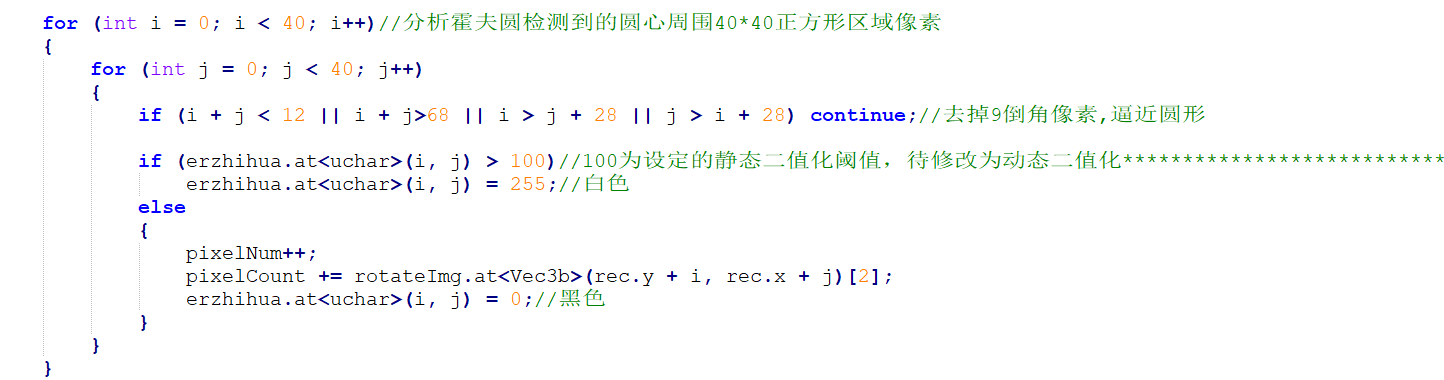


图1-4

* 1. 坐标提取

之前我们打算用深度学习的方式来识别每一个棋子的文字和颜色，然后再来分析每一个棋子的位置变化情况。但是发现训练的效果不是很好，因为每一次的开局棋子的位置是固定的，所以我们采用将棋子的初始坐标位置存储在二维数组chessBoardRTScan[10][9]里面。我们对所拍摄到的图片建立坐标系，，通过检测到棋子的坐标后,与格子像素的运算取模后转换为0~9或者0~8的整数。数组的状态值有三种，空，黑棋，红棋。

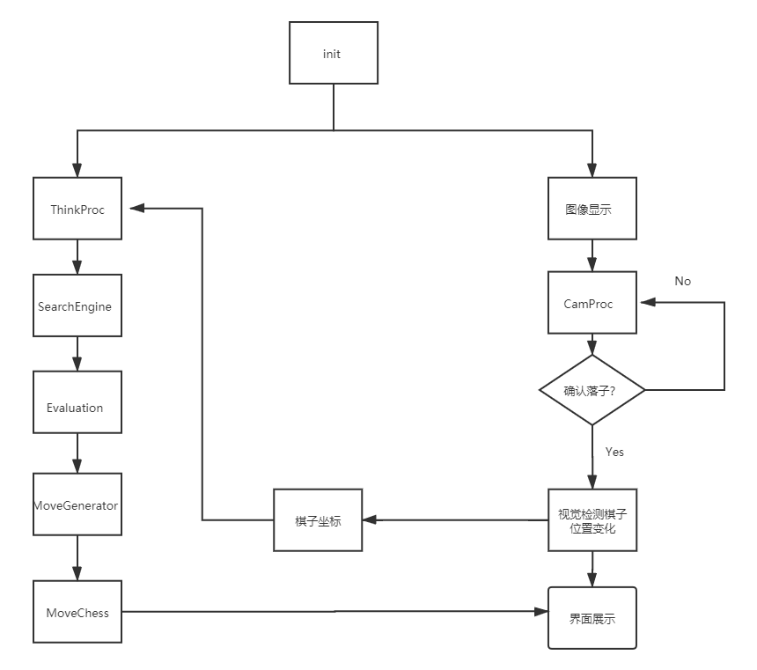
然后将得到的人下的得到的棋子变换通过视觉将其识别并且将其作为参数输入到棋子移动函数MoveChess()，计算机开始进入思考状态，思考下一步的走法。主要的算法流程图如下图1-5所示：

图1-5

在识别的时候我们发现，并不是每一次棋子的识别都能将所有的棋子都是别出来，所以必须在让计算机计算出下一步的最佳位置时，先要在前面加一个判定，必须让检测出来的红棋数量与他上次走完红棋的数量一致，才算人走完了一步棋。才能继续进行下一步。

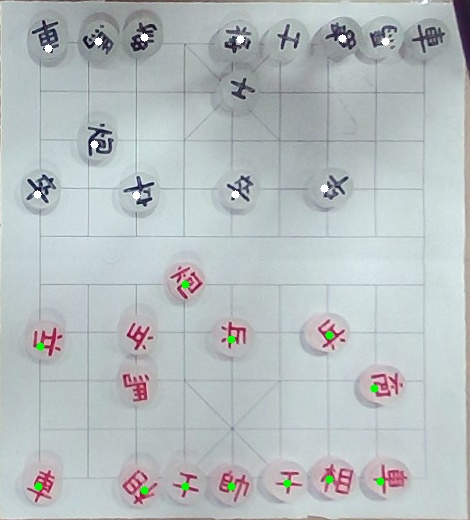


图1-6

1. 串口通信
   1. GUI界面设计

我们的GUI界面所运行的环境是VS2019，采用的是用经典MFC编写而成。界面如下图2-1：

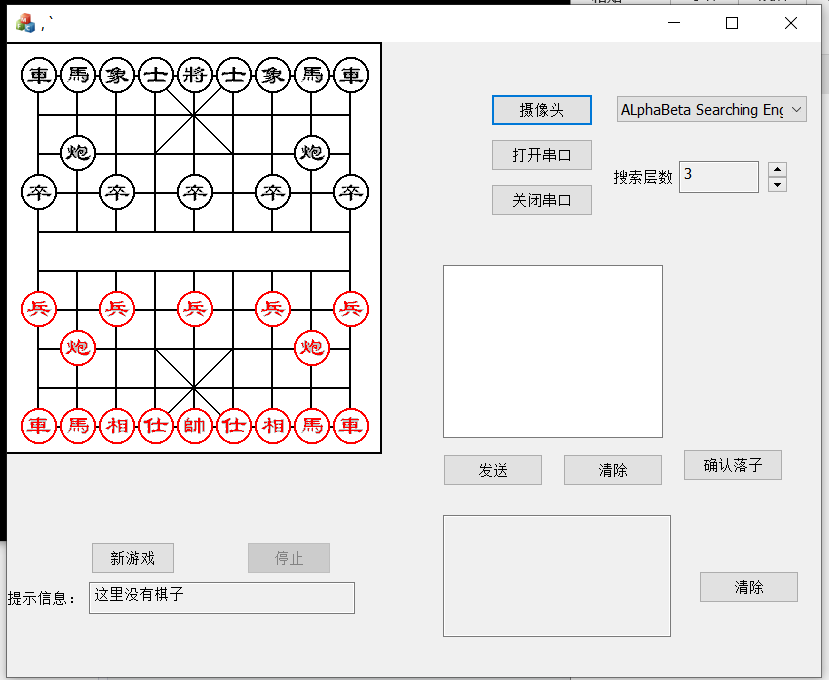


图2-1

在这样一个GUI界面， 左边部分是一个辅助棋盘，他能够针对于棋子移动能够实时反映，让我们能够看到棋子的移动，作为辅助功能。左边区域便是通信区域。编辑框1是串口信息发送区域，可以直接在发送框发送G代码，便于后期上下位机通信。而接受框能够实时显示接受到的字符情况，也是为了调试做准备。在这个界面，我们还可以选择象棋算法搜素层数以及搜索引擎，可选择的搜索引擎有三个，极小窗口搜索引擎、迭代深化搜索引擎，最大最小搜索引擎，能够选择的搜索层数从2层到5层。但是由于运算量会根据搜索层数而成倍增长，所以一般三层搜索耗费的时间在毫秒内。

为了编写程序的快速性，减少不必要的工作，所以我并没有将波特率、串口的选择设置在界面上。所以直接设置串口的波特率为115200，串口号设置为COM3 。

* 1. 通信实现

在通信过程中，我采用的是一个开源的串口通信封装库，SerialPort.c和SerialPort.h,它是Remon Spekreijse写的一个串口类，通过在基于对话框的GUI界面程序就能直接调用其方法和属性。具体而言就是在ChessDlg.c新建消息处理函数，接受的缓存区有数据存在时，就会直接调用ReceiveChar()函数，对收到的数据进行轮番读取单字符，如果想读取字符串的话，需要直接对收到的字符串直接相加。而对于发送数据，直接调用WriteToPort(char\* string)，直接将字符数组发送到下位机。具体的收发流程图如下图2-2：

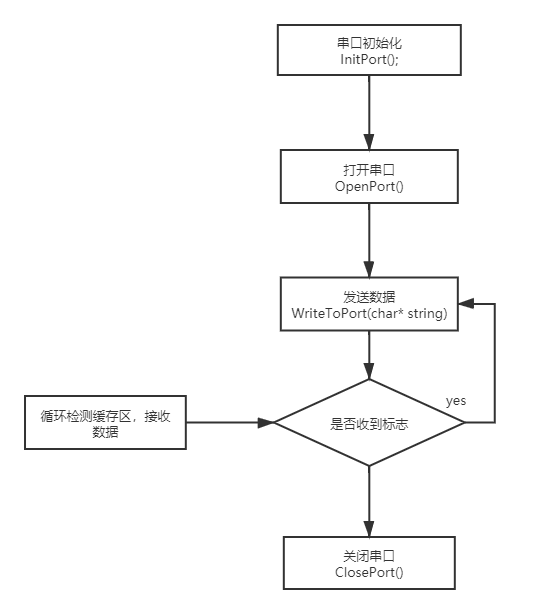


图2-2

电脑传给我们的数据是两个点坐标值，代表着他会从一个起始点移动到终点，但是下位机需要的是世界坐标系位置的移动，所以我们需要进行坐标变换，把数组值转化为实际坐标值，然后再通过串口通信发送给下位机，控制机械臂移动。

* 1. 通信协议

在收到上位机的控制命令后，下位机会发送一个动作完成的标志，这时候再发送下一行指令进行连续控制。本意想的是直接将所有的关于移动控制的G代码一次性发送给下位机，但是有于Marlin固件的限制，它每次只能接受一行G代码，然后产生中断，执行完动作后才能继续接受下一行。后多行同时接收的话会使其最后一行G代码覆盖第一行，使得之前发送的G代码失效。

在通信过程中，下位机要收到’\n’才算完成一行G指令的结束，所以，在向下位机发送G代码的时候，要在每一行G指令后面加上这样一个换行字符才算完成一个指令的输入。

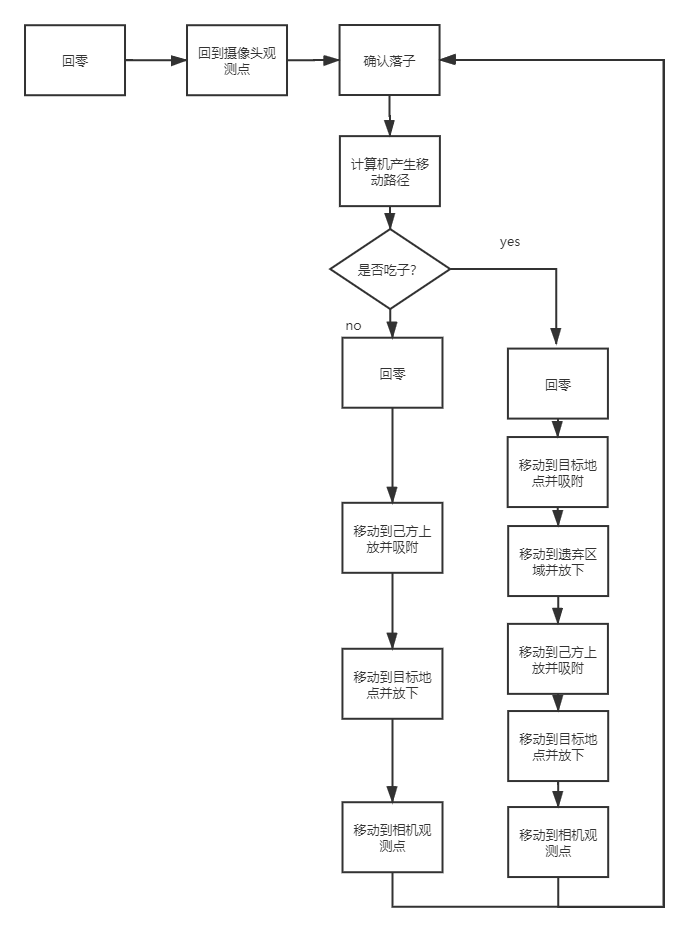
对于象棋而言，下的棋无非两种，一是直接将棋子移动到目标地点，另一种就是想将对方的棋子吃掉，然后移动到别的地方。针对于这两种情况，我们做了如下的逻辑示意图。

图2-3

我们对上位机象棋指令分为了两个部分，针对于吃子的部分，我们多出了一个将吃掉的棋子移动到棋盘区域外的位置，所以他的步骤要多一点。这样的两个循环我们用if条件语句来判断使用哪一个。

对于下位机的控制，G代码解析已经将大部分移动都已封装。这是在棋子没有吃子的情况下移动的指令发送，在这个过程中，每执行一步，因为下位机执行完一行G代码都会发送一个“ok”，此时的程序的计数器index都会加一，然后按照顺序执行，而样一个index在我们按下“确认落子”的按钮后，都会重置为零，准备进行下一个过程的循环，代码的实现如下：

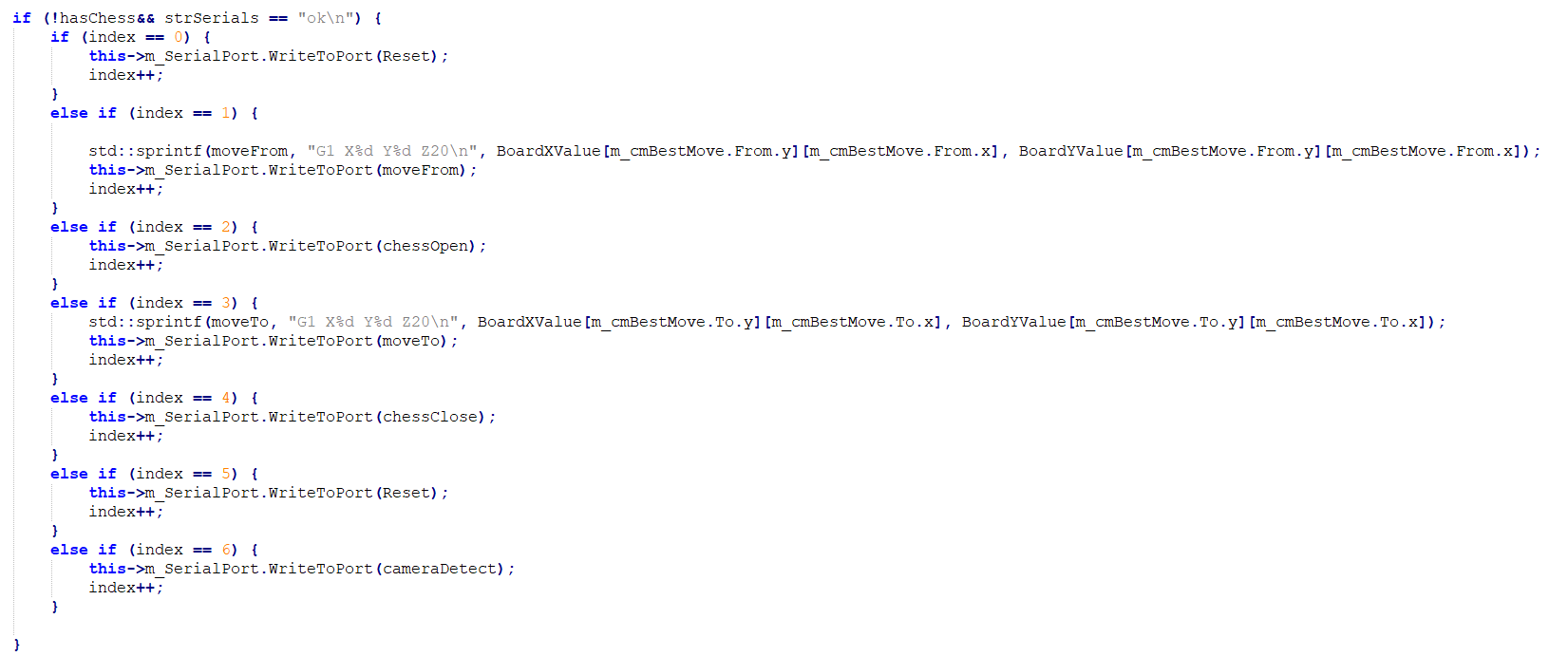


图2-4

吃子过程的代码与之类似，代码的实现如下图：

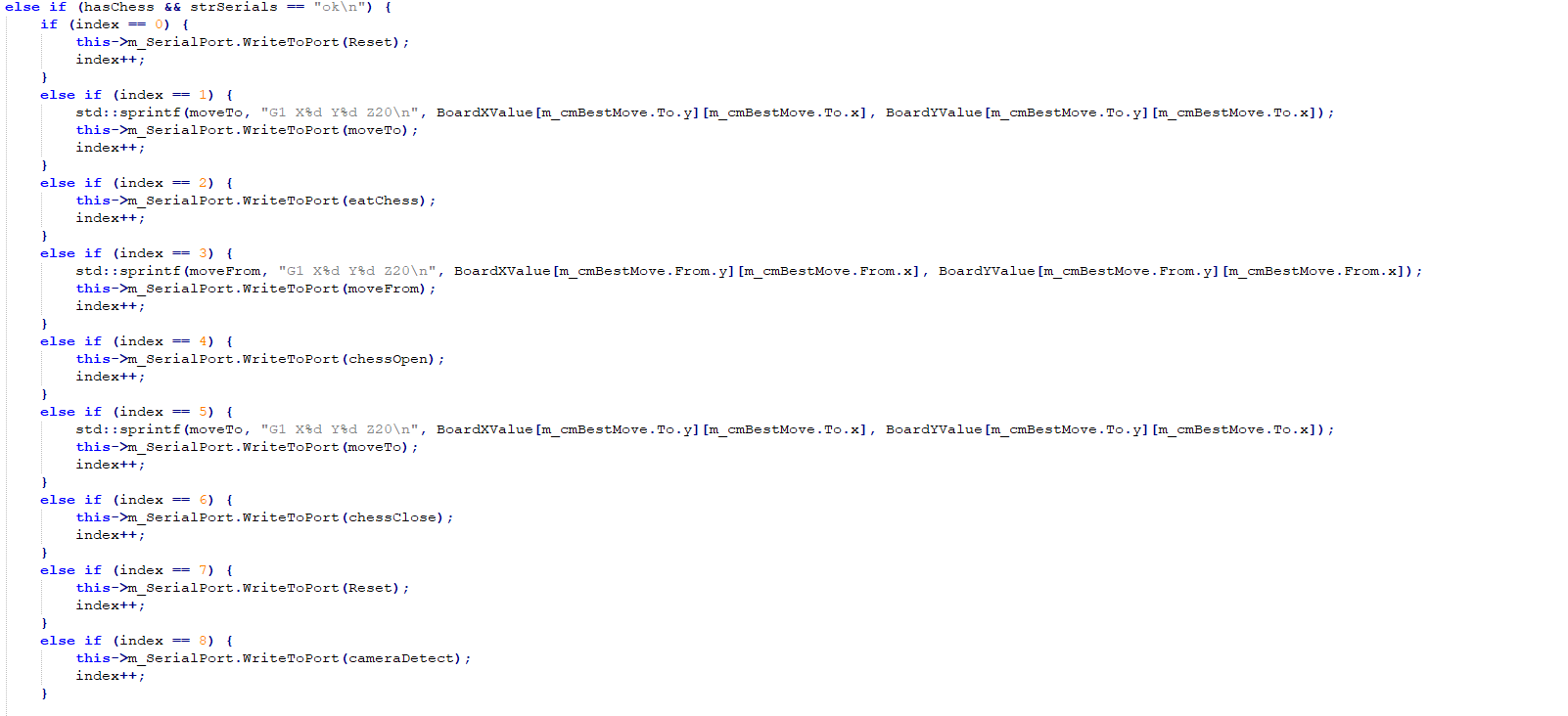


图2-5

其中一些指令解释，“cameraDetect”，移动到相机观测点，指令为“M13\n”，这是自己定义的一个的位置坐标点。“Reset”，回零，指令为“G28\n”。“chessOpen”，打开气泵吸附棋子，指令为“M11\n”。“chessClose”，关闭气泵翻下棋子，指令为“M15\n”。“eatChess”，移动吃掉的棋子并且移动到遗弃区域，指令为“M15\n”。

1. 象棋算法
   1. 搜索引擎
      1. 最大最小值搜索引擎

Minimax算法 又名极小化极大算法，是一种找出失败的最大可能性中的最小值的算法（即最小化对手的最大得益）。通常以递归形式来实现。其伪代码如下图所示：



图3-1

但是，实际问题中的所有局面所产生的博弈树一般都是非常庞大，非常庞大的多叉树~，并不能依靠暴力搜索来寻找最佳解法。因此需要用到一些剪枝手段。常用的比较初级的有 alpha-beta 剪枝。

AlphaBeta剪枝算法是一个搜索算法旨在减少在其搜索树中，被极大极小算法评估的节点数。这是一个常用人机游戏对抗的搜索算法。它的基本思想是根据上一层已经得到的当前最优结果，决定目前的搜索是否要继续下去。

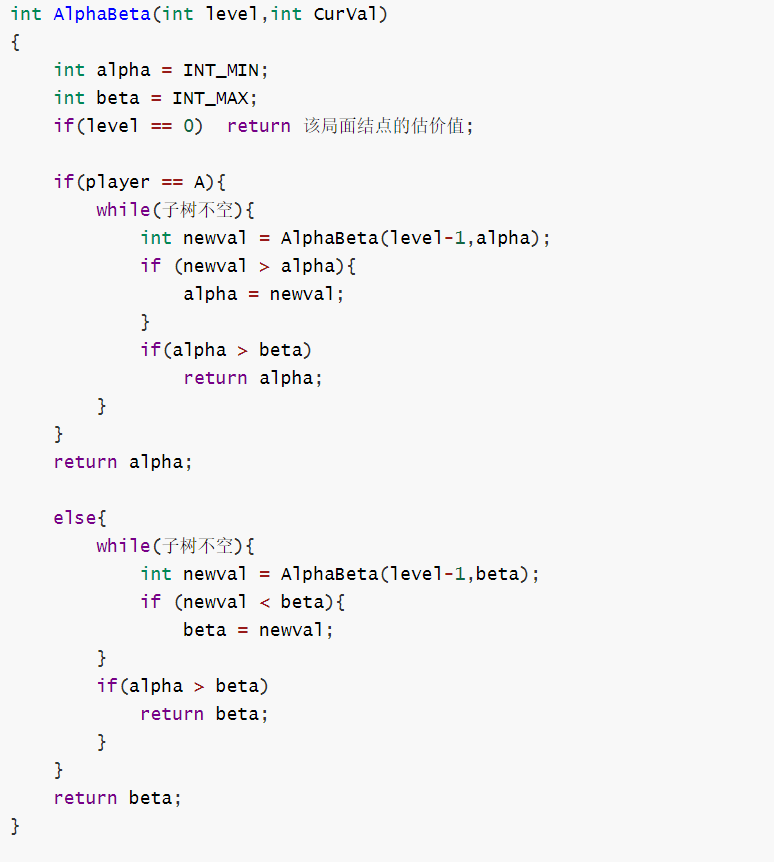


图3-2

* + 1. 极小窗口搜索引擎

主变量导向搜索pvs(PrincipalVariationSearch)又称最小窗口搜索（minimal window search），是alpha-beta pruning的一个变种，其区别在于除主变量节点外的其他所有节点都用一个零窗口(alpha,beta)且alpha=beta 进行搜索,其理念是对浅层的节点进行整理使其基本有序，并假设第一个节点是最好的，做为主变量，进行全窗口搜索。通过零窗口搜索其他节点，判断是否存在这些节点是否比当前最优值要好。假如符合alpha-beta剪枝则进行剪枝，假如高值失败则证明当初的节点不是主变量，对当前节点重新进行一次全窗口搜索，作为新的主变量。例如对于一个（alpha，alpha+1）的零窗口，假设返回的值为alpha+1，这说明该节点存在比alpha要大的值，因此需要对其进行一次重新搜索，这次就会得到一个比alpha大的值，去更新alpha的值，假如alpha>beta则进行剪枝。假如返回值<=alpha,则说明该节点的价值较低，可以忽略。

算法通过使用小窗口，增加了剪枝率，提高了alphabeta剪枝的效率，但当节点以随机排序时，其效率可能会比alphabeta剪枝要低。

极小窗口搜索的流程可以分为五步：

(1)对于第一个节点,我们按照原来的范围进行搜索，我们会得到一个最优解bestvalue。

(2)我们用(besvtalue,bestvalue+1)作为窗口进行测试。

(3)如果得到的值大于bestvaufe并且小于beta时，就说明有更好的方法，需要对(bestvalue,beta)进行测试。

(4)如果不是，再判断得到的值value大于bestvalue，就说明value是一个更好的行动，应该用它来代替原来的bestvalue。

(5)如果得到的值value小于bestvalue，说明这种策略还不如以前的策略，不同再分析。

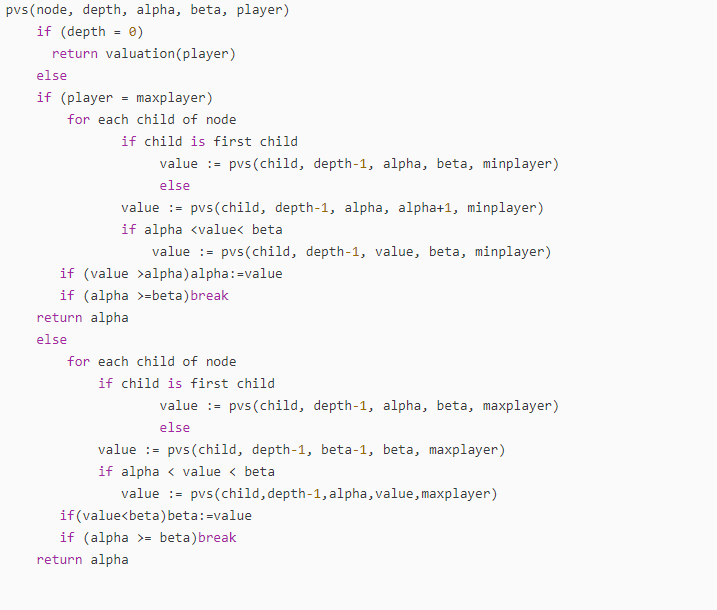


图3-3

* + 1. 迭代深化搜索引擎

迭代深化搜索本质上就是深度优先搜索，IDS结合了DFS的空间优势与BFS的时间优势。对搜索的深度进行了限制，使得在搜索到限制深度后必须开始新的搜索路径。

在这个搜索策略中，一个具有深度限制的深度优先搜索算法会不断重复地运行，并且同时放宽对于搜索深度的限制，直到找到目标状态。IDDFS 与广度优先算法是等价的，但对内存的使用会少很多；在每一步迭代中，它会按深度优先算法中的顺序，遍历搜索树中的节点，但第一次访问节点的累积顺序实际上是广度优先的。

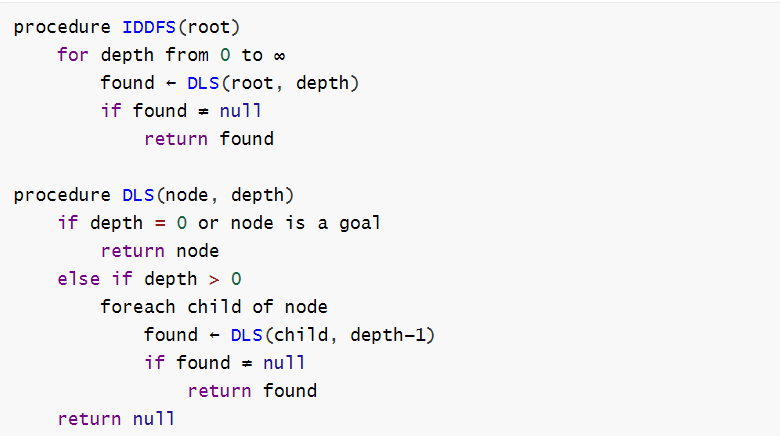


图3-4

* 1. 评估方法

象棋博弈者在走棋时,他需要考虑全部有价值的走法,并且通过前瞻若干步,才能形成博弈者的当前决策。在这个过程中,如何对棋局进行评估﹐以及如何让棋局朝着有利的方向进行呢﹖为了解决这个问题人们提出了评估函数。下面将介绍几种常用的构造评估函数的方法。

(一)传统评估函数

传统评估函数也叫静态评估函数,它是通过计算博弈双方评估值的差值得到的。我们假设现在是红方走棋,那么此时的评估函数就可以表示下式

Evaluation(CurrentBoard)=Evaluation(Red)—Evaluation(Black)

其中 Evaluation(x)表示的是评估函数,Evaluation(Red)表示红方状态的评估值;Evaluation(Black)表示黑方状态的评估值。

1.评估函数的组成

评估函数通常从六个方面评价博弈双方的局势:棋子的价值、棋子位置的价值、棋子对棋盘的控制、棋子的灵活性、棋子之间的威胁和保护以及一些对棋局影响较大的特征。

棋子的价值代表的棋子的重要程度。根据中国象棋规则,每个棋子都有自己独特的走法,这样导致了每种类型棋子的作用和重要程度都不相同。

棋子位置的价值表示的是不同棋子在棋盘不同位置上会有不同的价值,这里的价值主要是指对博弈对手的威胁程度。

棋子的灵活性是指每个棋子可以“走动”的范围。一般来说,我们认为棋子能“走动”的范围越多,它对防守或进攻的贡献越大。

中国象棋讲究的是棋子之间的配合,需要让棋子之间构成一个相互联系的整体,棋子之间的威胁与保护正是这方面的体现。棋子的安全程度取决于它的保护者和威胁者的个数、类型和当前轮到哪方走等多种因素。

棋子对棋盘的控制与棋子之间的威胁和保护在处理方法上有很多相似之处,它们之间的区别是前者是对棋盘中某些位置的控制,而后者是对棋子的控制。棋子对棋盘的控制这个特征在棋局开始和中盘的时候都是很有价值的,但是当棋局进入到残局阶段,棋盘上的棋子比较少,每个棋子控制的范围自然增多，这时棋子对棋盘的控制这个特征在评估函数中的作用就下降了。

下棋时的一些战术和棋子之间配合的价值也需要在评估函数中体现。下棋时的战术主要包括:捉双、牵制和抽子等。棋子间的配合主要是指兵、车、马和炮之间的配合﹐比如:过河兵牵手、连环马和霸王车等。这些棋子间配合的价值往往大于配合棋子的价值之和。

我们主要考虑的方面并不是象棋算法的实现，更多的是在网上找到一个开源的象棋算法应用，然后使用其接口来应用到我们的象棋机器人中。所以在象棋算法的实现过程中花费的时间较短，花费时间最多的方面可能就是在象棋源码的阅读上，要找的其移动步法生成器的接口函数比较麻烦。

附录

上位机程序源码说明，程序相关类说明如下：

1.CEveluation类:估值类,对给定的棋盘进行估值.

2.CMoveGenerator类:走法产生器,对给定的棋盘局面搜索出所有可能的走法.

3.CSearchEngine类:搜索引擎基类.

4.CAlphaBetaEngine类:采用了Alpha-Beta剪枝技术的搜索引擎.

5.CHistoryHeuristic类:历史启发类.

6.CAlphabeta\_HHEngine类:带历史启发的Alpha-Beta搜索引擎.

7.CIDAlphabetaEngine类:迭代深化搜索引擎.

8.CTranspositionTable类:置换表.

9.CPVS\_Engine类:极小窗口搜索引擎.

10.CSerialPort类:串口通信类

视觉处理部分的源程序都在Opencv.c和Opencv.h中。