2024 年“TI”杯全国大学生电子设计竞赛

三子棋游戏装置（E 题）

【本科组】

2024 年 8 月 1 日

目录

三子棋游戏装置（E 题）

【本科组】

摘要

本三子棋游戏装置采用仿 ABB 工业机的三自由度机械臂结构；使用以 ATmega1280 为主控芯片的 Arduino MEGA 开发板模块；ZP20D 总线舵机；OpenMV Cam H7 Plus；串口屏；真空气泵；配合 MIN-MAX 算法，完成了对棋子摆放的控制与人机对弈功能。该装置能够实现对棋盘与棋子的视觉识别; 可以完成指定棋子的指定位置摆放; 具有人机对弈功能; 能够检测出作弊行为并加以纠正等，可以达到题目所需要求。

关键字： Arduino MEGA，OpenMV，ABB 机械臂，MIN-MAX 算法，三子棋，真空气泵

Abstract

This tic-tac-toe game device adopts a three degree of freedom robotic arm structure imitating ABB industrial machines; Use Arduino MEGA development board module with ATmega1280 as the main control chip; ZP20D bus servo motor; OpenMV Cam H7 Plus；Serial port screen; Vacuum pump; Combined with the MIN-MAX algorithm, the control of chess piece placement and human-machine game functions have been completed. This device can achieve visual recognition of chessboard and chess pieces;complete the placement of designated chess pieces in designated positions;have human-machine game functions;detect cheating behavior and correct it, etc., which can meet the requirements of the problem.

Keywords: Arduino MEGA, OpenMV, ABB robotic arm, MIN-MAX algorithm, tic-tac-toe, vacuum pump

1 引言

本作品设计并制作三子棋游戏装置，能够控制机械臂或其他机构放置棋子，实现人机对弈。图 1 所示的三子棋棋盘和棋子位置示意图中，棋盘由黑色实线围成 9 个方格，人、机分别从棋子放置处拾取棋子并放置到方格中，先将己方的 3 个棋子连成一线（横连、竖连、斜连皆可）即获胜。

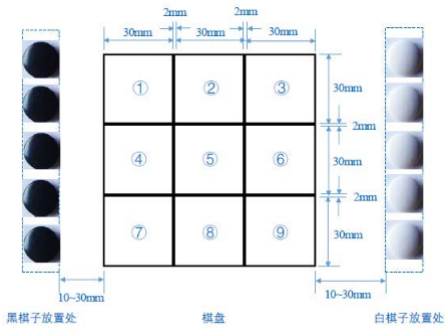

图 **1** 三子棋棋盘和棋子示意图

2 系统方案设计与论证

2.1 机械臂构型

方案一：SCARA 型机械臂

SCARA 型机械臂拥有 2 个旋转关节和 1 个垂直运动关节，主要在水平面内运动。具有水平面的运动灵活性较高，速度快，结构相对简单，成本较低等优点。但其垂直方向的运动范围有限，不适用于需要复杂三维运动的任务。

方案二：关节型机械臂

关节型机械臂拥有多个旋转关节，可以实现复杂的三维运动。具有高度灵活，能够完成复杂的路径规划和多自由度的操作等优点 [8]。但控制算法复杂，成本较高，对操作环境要求较高 [4]。

综上可知，相比于 SCARA 机械臂，尽管关节型机械臂尽管成本较高，但其高度灵活性和适应复杂运动的能力使其在任务精度和复杂性要求较高的情况下非常有用，在经过讨论后选择了三自由度机械臂。因此选用方案二。

2.2 处理器模块

方案一：STM32F407

STM32F407 基于 ARM Cortex-M4 内核，有 192KB 内存，多达 140 个 I/O 接口。具有高性能 MCU，低功耗，实时性强等优点。但编程和调试需要一定的专业知识，开发

难度较高，而且成本较高。

方案二：Arduino Mega 1280

Arduino Mega 1280 基于 AVR 微控制器，多达 54 个 I/O 接口。具有易于使用，丰富的开源资源和社区支持等优点。但性能较低，适用于简单控制任务，无法满足高性能需求。

综上可知，最终选择 Arduino Mega 作为处理器，主要是基于成本、功耗和开发简单性考虑。虽然 STM32F407 性能更强，但 Arduino Mega 完全能够满足需求，且成本更低，适合进行此项目开发。因此选用方案二。

2.3 视觉模块

方案一：Intel RealSense

Intel RealSense 有 720p 分辨率，60 帧每秒的帧率，支持深度感知。但价格较高，适合预算充足的项目，对硬件资源要求较高。

方案二：OpenMV Cam H7 Plus：

OpenMV Cam H7 Plus 320x240 分辨率，30 帧每秒的帧率，内置机器视觉算法。具有支持多种视觉算法，如颜色识别、运动检测等 [7]。但分辨率较低，适用于基本的视觉任务，图像质量中等。

综上可知，最终选择 OpenMV Cam H7 Plus 作为视觉模块，它能够提供足够的视觉处理能力和编程灵活性 [5]，满足任务需求，同时保持良好的性价比和系统集成性。因此选用方案二。

2.4 电机模块

方案一：ZP20D 总线舵机

ZP20D 总线舵机是一种高性能舵机，具有总线控制功能，可以通过串行通信同时控制多个舵机。它的精度高，响应速度快，具有良好的定位精度。但价格相对较高，而且总线通信对抗干扰能力要求高，需要良好的布线和屏蔽措施。

方案二：MG995 标准舵机

MG995 是一种常见的标准舵机。它价格低廉，易于使用和调试，适合简单的控制系统。但控制方式简单，缺乏总线控制功能，适用于单个或少量舵机的控制。定位精度和扭矩相对较低，不适合高精度和高负载的任务。

综上可知，最终选择 ZP20D 总线舵机作为机械臂的电机模块。相比于其他类型的舵机，ZP20D 总线舵机在性能和功能上更适合复杂的多舵机控制系统。因此选用方案一。

2.5 系统整体方案

本系统主要由电源模块、总控制模块 Arduino Mega、OpenMV 识别模块、三自由度关节机械臂组成，其系统框图如图 2 所示:

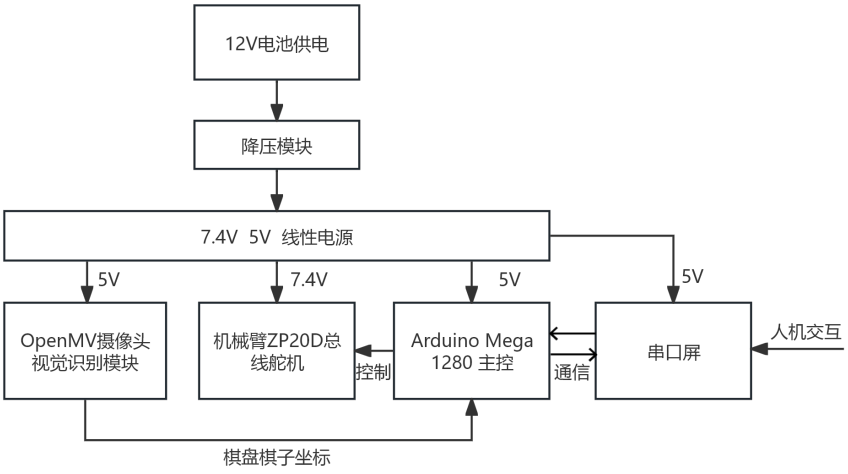

图 **2**　系统总体框图

3　理论分析与计算

3.1　棋盘棋子视觉识别

对于棋盘与棋子的识别，识别棋盘格与棋子都采用色块识别的方式，在代码中设置颜色的阈值，将实时获取到的图像进行颜色过滤并在图像中绘制识别到的色块，获取色块的尺寸大小及位置信息，计算得到该色块的中心坐标并将其发送给机械臂控制系统的主控。

3.2　机械臂运动学解算

为了完成所要求的棋子摆放，我们需要机器人能够在给定了一个末端坐标是，机器人能够自动调整姿态。我们假设机械臂最初的姿态 [4]：与 $\boldsymbol{X}$ 轴重合，已知：$\boldsymbol{x}$,$\boldsymbol{y}$,$\boldsymbol{z}$,$\boldsymbol{L1}$,$\boldsymbol{L2}$；求 $\boldsymbol{\theta 1}$、$\boldsymbol{\theta 2}$、$\boldsymbol{\theta 3}$，解析如下图 3 所示：

根据图三，可得

$$\boldsymbol{w} = \sqrt{\boldsymbol{x^2} + \boldsymbol{z^2}} \tag{1}$$

$$\boldsymbol{A} = \sqrt{\boldsymbol{w^2} + \boldsymbol{y^2}} \tag{2}$$

$$\cos \boldsymbol{\theta 3} = \frac{\boldsymbol{x}}{\boldsymbol{w}} \tag{3}$$

根据余弦定理有 $L^2 = {L_1}^2 + W^2 - 2\,L1\,W\cos(\theta 1 - \theta T)$，$\theta 2$ 同理。

于是有

$$\theta 1 = \arccos\frac{L1^2 + W^2 - L2^2}{2L1W} + \theta T \tag{4}$$

$$\theta 2 = \arccos\frac{W^2 - L1^2 - L2^2}{2L1L2} \tag{5}$$

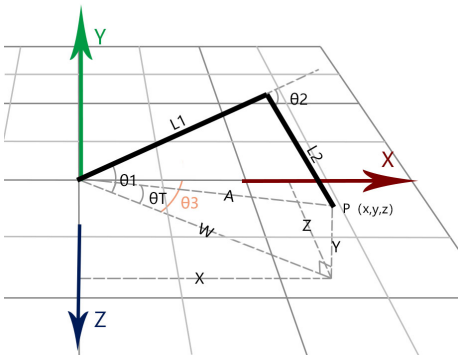

图 3　三自由度机械臂运动学解算

这样就得到了偏转角度 $\theta 1$、$\theta 2$ 与 $\theta 3$。于是在得到所需的棋盘棋子的位置坐标后，主控可以发送对应的角度数据，使机械臂能够通过舵机自动移动到指定位置。

3.3　MIN-MAX 算法博弈

MIN-MAX 算法是一种经典的决策算法，广泛应用于零和博弈中 [2]。其基本思想是为游戏中的每一步寻找最佳策略，使得即使对手做出最优反应，自己的损失也最小 [3]，示例博弈树如图 4 所示。

在本装置中，我们使用 MIN-MAX 算法通过构建博弈树，递归求解每个节点的评估值，选择最佳策略 [1]，能够有效地用于三子棋游戏的决策过程。

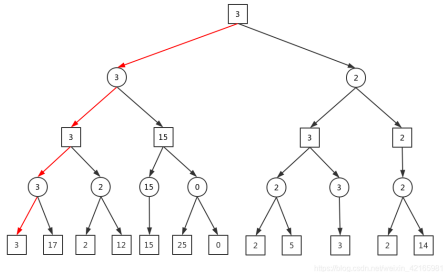

图 4　MIN-MAX 算法博弈树

4　硬件与电路设计

4.1　电源电路模块

在本装置中，我们设计了两款电源模块，采用 XL4016 DC-DC 降压芯片，分别设计了 $7.4V$ 与 $5V$ 的 DC-DC 降压电源模块，全装置采用 $12V$ 航模电池供电。电路图如图 5 所示：

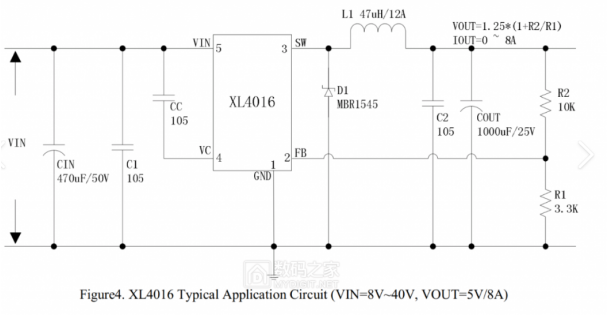

图 **5**　电源模块原理图

4.2　机械臂结构

为了完成对棋子的摆放动作，我们设计了一款三自由度的 ABB 关节机械臂 [8]，三个关节分别由三个 ZP20D 总线舵机控制，使气泵吸盘能够精准移动到指定位置。以完成三子棋游戏的棋子摆放动作。机械臂结构如图 6 所示：

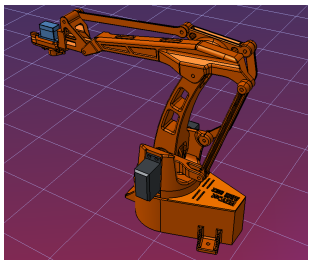

图 **6**　机械臂结构示意图

4.3　主控与人机交互模块

本装置采用 Arduino Mega 1280 为主控模块，完成对机械臂的总线舵机与真空气泵的控制 [6]；接受来自 OpenMV Cam 的坐标位置 [7]；链接串口屏实现装置所需的人机交互功能。模块原理图如图 7 所示：

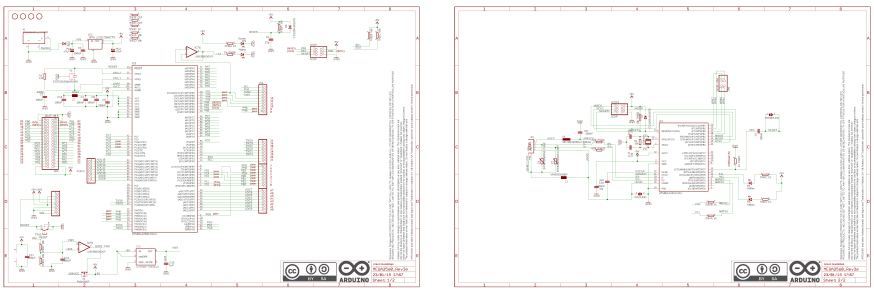

图 **7**　主控模块原理图

4.4　视觉模块

视觉识别部分，本装置使用 OpenMV Cam H7 Plus 对棋盘与棋子进行视觉识别，能够实时通过摄像头将棋盘格与棋子的中心坐标发送到主控上，完成对装置对棋盘格与棋子的定位 [8]。棋盘识别效果图如图 8 所示：

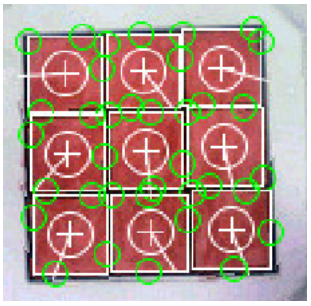

图 **8**　棋盘识别效果图

5　软件设计

5.1　视觉识别与舵机控制

见图 9，相关代码见附录。

5.2　人机交互

见图 10.

6　装置功能测试

6.1　各项功能测试方案

6.1.1. 测试目的

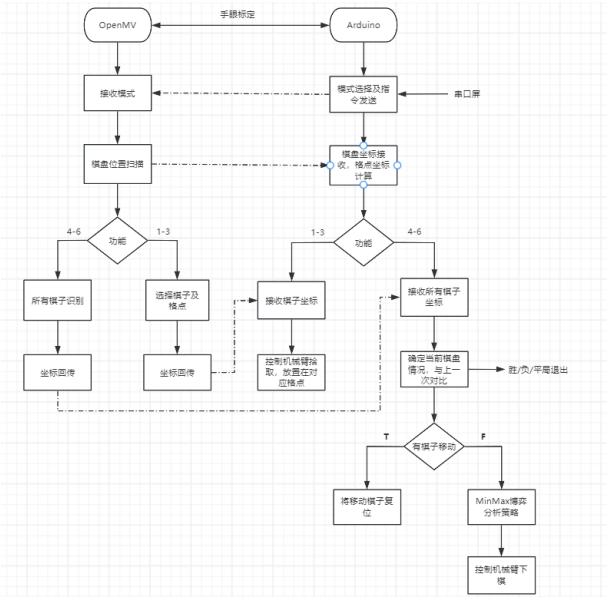

图 **9**　视觉识别与舵机控制程序流程图

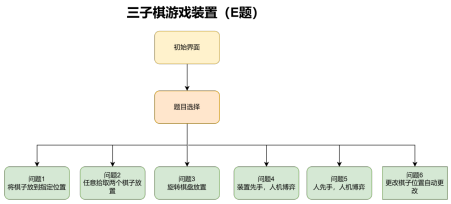

图 **10**　人机交互程序流程图

测试三子棋游戏装置的功能性、稳定性和精确性，确保其能够满足比赛要求，实现人机对弈并正确放置棋子。

6.1.2. 测试环境

标准三子棋棋盘和棋子，棋盘尺寸及棋子直径按照比赛要求。正常室内照明条件，无特殊照明要求。安装并调试好的三子棋游戏装置，包括机械臂、控制系统和视觉模块。

6.1.3. 测试工具

三子棋棋盘，棋子（黑白各五颗），计时器，摄像设备（记录测试过程）。

6.1.4. 测试步骤

步骤一：基础功能测试启动装置，指示其将黑棋子放置到 5 号方格中。观察并记录棋子的放置过程，确保棋子准确放置在指定方格内。

步骤二：多棋子放置测试依次指示装置将棋子放置到指定方格中，观察并记录每颗棋子的放置过程，确保准确性和稳定性。将棋盘绕中心 ±45° 范围内旋转，再次执行相同的放置操作，确保装置能够正确放置棋子。

步骤三：人机对弈测试人类玩家和装置交替下棋，记录每一步棋的放置过程和结果。确保装置在第 1 步棋子放置后，能正确响应人类玩家的操作，检测到人类玩家的错误并

获胜。测试装置在不同起始方格的对弈能力。

步骤四：反作弊功能测试。人类玩家在对弈过程中将装置已放置的 1 颗棋子移动到其他方格。观察装置是否能自动检测到棋子被移动，并在 15 秒内将棋子放回原位置。

6.2 各项功能测试结果与分析

6.2.1 测试结果分析标准

成功标准：装置能够在所有测试中准确放置棋子，响应及时，对人类玩家的操作做出正确反应，且具有稳定性。

失败标准：装置在放置棋子过程中出现错误，未能在规定时间内完成操作，或对人类玩家的操作未能正确响应。

6.2.2 测试结果记录与分析

表 **1** 装置功能测试结果记录表

黑棋子编号	1	2	3	4	5	2	4	1	3	5
指定位置	5	5	5	5	5	5	5	5	5	5
放置位置	5	5	5	5	5	5	5	5	5	5

指定棋子	黑 1	白 2	白 3	黑 4	黑 3	黑 5	白 1	白 4	黑 2	黑 4	白 5	白 3
指定位置	1	3	7	9	2	4	1	9	5	8	6	2
放置位置	1	3	7	9	2	4	1	9	5	8	6	2

由上表可知装置能够完成对棋子的定位与抓取，并具备将其放置到棋盘指定位置的功能。

经过多次测试，装置能够完成人机对弈步方格可设置），若人应对的第 1 步白棋有错误，装置能获胜。而且人执黑棋先行时，装置能正确放置白棋子以保持不输棋。并且在对弈过程中，若人将装置下过的 1 颗棋子变动位置，装置能自动发现并将该棋子放置回原来位置。

7 结论

本项目成功设计并实现了一个功能完善的三子棋游戏装置，能够满足比赛要求并具备良好的用户体验。装置的硬件选择合理，算法设计高效，系统稳定性和精确性均得到验证。

参考文献

[1] Elwyn R Berlekamp, John H Conway, and Richard K Guy. *Winning ways for your mathematical plays, volume 4*. AK Peters/CRC Press, 2004.

[2] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. *How to design programs: an introduction to programming and computing*. MIT Press, 2018.

[3] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

[4] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giuseppe Oriolo, et al. *Robotica–Modellistica, Pianificazione e Controllo*. McGraw-Hill libri Italia, 2008.

[5] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[6] 张一通, 孙玲, 张远, and 张尧. 基于 pid 控制策略的机械臂轨迹跟踪控制. *Artificial Intelligence and Robotics Research*, 11:164, 2022.

[7] 张宇豪, 庞辰骅, 宋子韩, 钱嘉毅, and 蒋书波. 基于 openmv 的五自由度机械臂控制方法研究. *Machine Tool & Hydraulics*, 52(5), 2024.

[8] 殷孝睢, 周莉, 孙志强, 郑闯, and 李波. 基于视觉识别的随动机械臂实验装置设计. *Experimental Technology & Management*, 40(11), 2023.

附录

```
def get_chessboard():
global point_temp
global points
while True:
clock.tick()
img = sensor.snapshot().lens_corr(1.7).histeq(adaptive=True, clip_limit=2)
img.laplacian(1, sharpen=True)
#   img.gaussian(1, unsharp=True)
lcd.display(img)
count = 0
for blob in img.find_blobs([board_threshold], roi = (65, 65, 165, 165), pixels_threshold =
    800, area_threshold=800):
# These values depend on the blob not being circular - otherwise they will be shaky.
if blob.elongation() > 0.5:
img.draw_edges(blob.min_corners(), color=(255, 0, 0))
img.draw_line(blob.major_axis_line(), color=(0, 255, 0))
img.draw_line(blob.minor_axis_line(), color=(0, 0, 255))
# These values are stable all the time.
img.draw_rectangle(blob.rect())
img.draw_cross(blob.cx(), blob.cy())
# Note - the blob rotation is unique to 0-180 only.
img.draw_keypoints(
[(blob.cx(), blob.cy(), int(math.degrees(blob.rotation())))], size=20
)

point_temp[count] = (blob.cx(), blob.cy())
count = count + 1
print(blob)
print(clock.fps())

if count == 9:
points = sort_points(point_temp)
break
else:
point_temp = [(0,0)] * 9
continue

def get_chess():
global points
global chess
count = 0
while True:
clock.tick()
img = sensor.snapshot().lens_corr(1.7)
#   img.gaussian(1, unsharp=True)
```

```
lcd.display(img)
for i in range(9):
_blobs = img.find_blobs([black_threshold, white_threshold], roi = (points[i][0]-23,
    points[i][1]-23, 46, 46))
if _blobs == [] and chess[i] == 0:
chess[i] = 0

for blob in _blobs:
# 滤除棋盘边框
if blob.w() > 40 or blob.h() > 40:
continue
if blob.w() < 20 or blob.h() < 20:
continue

# 验证色块是否为圆形
aspect_ratio = blob.w() / blob.h()
if aspect_ratio < 0.6 or aspect_ratio > 1.4: # 假设棋子的宽高比接近1
continue
if blob.code() == 1: # 黑色
cir = blob.enclosing_circle()
img.draw_circle(cir[0], cir[1], cir[2], color=(0, 255, 0))
chess[i] = 1

if blob.elongation() > 0.5:
img.draw_edges(blob.min_corners(), color=(255, 0, 0))
img.draw_line(blob.major_axis_line(), color=(0, 255, 0))
img.draw_line(blob.minor_axis_line(), color=(0, 0, 255))
# These values are stable all the time.
img.draw_rectangle(blob.rect())
img.draw_cross(blob.cx(), blob.cy())
# Note - the blob rotation is unique to 0-180 only.
img.draw_keypoints(
[(blob.cx(), blob.cy(), int(math.degrees(blob.rotation())))], size=20
)

elif blob.code() == 2: # 白色
cir = blob.enclosing_circle()
img.draw_circle(cir[0], cir[1], cir[2], color=(0, 0, 255))
chess[i] = 2

if blob.elongation() > 0.5:
img.draw_edges(blob.min_corners(), color=(255, 0, 0))
img.draw_line(blob.major_axis_line(), color=(0, 255, 0))
img.draw_line(blob.minor_axis_line(), color=(0, 0, 255))
# These values are stable all the time.
img.draw_rectangle(blob.rect())
img.draw_cross(blob.cx(), blob.cy())
```

```
# Note - the blob rotation is unique to 0-180 only.
img.draw_keypoints(
[(blob.cx(), blob.cy(), int(math.degrees(blob.rotation())))], size=20
)
count = count + 1
if count == 20:
print(chess)
sending_chess()
chess = [0] * 9
count = 0
break
#       print(chess)
#       chess = [0] * 9
display(img) # Take a picture and display the image.
#   sending_center()
```