

**智能控制系统课程设计报告**

**学 院： 计算机与信息安全学院**

**专业班级： 17003601**

**学 号： 1600110610**

**1700301117**

**1700300311**

**姓 名： 李旭 李闯 黄金荣**

**指导教师： 陈辉**

智能控制系统课程设计 任务书

Ⅰ、智能控制课程设计题目

自由选择所需的元器件，组队设计并制作基于机器视觉的自动报靶系统设计

Ⅱ、智能控制系统课程设计使用的原始资料(数据)及设计技术要求

要求报靶系统能在对射击靶进行图像提取并识别弹孔，同时识别出的靶数应能显示在显示屏上，用于提高传统打靶训练的效率。

Ⅲ、智能控制系统课程设计工作内容

（1）完成以OPENMV为主控模块的智能控制系统总体系统设计；

（2）完成智能控制系统的硬件环境搭建；

（3）完成智能控制系统的弹孔识别软件设计；

（4）完成智能打靶系统的环数判断、输出以及打靶历史记录的软件设计；

（5）完成智能控制系统的硬件模块与软件模块的串口通信模块；

（6）最终的设计要求能较为准确识别弹孔位置、成功识别是否打中胸型靶、返回对应得环数值，按要求完成智能控制等各项性能。

# 摘 要

图像处理技术的应用领域己经涉及到人类生活的各方面，数字图像处理技术相比模拟图像处理技术，有着处理精度高、内容丰富、 方法灵活等特点。本设计研究了军事射击中的自动报靶方向，使用OpenMV进行图像采集并进行二值化等处理，进行弹孔提取分析定位。并将结果传送至与PC端相连的LCD屏幕上，本设计成功完成了从硬件系统设计到软件算法设计、从图像采集到成绩输出多个环节的研究。

**关键词：**自动报靶；OpenMV；弹孔识别；图像识别；

# Abstract

The application field of image processing technology has involved all aspects of human life. Compared with analog image processing technology, digital image processing technology has the characteristics of high processing accuracy, rich content, and flexible methods. This design studies the automatic reporting of the target direction in military shooting, uses openmv for image acquisition and binarization and other processing, and performs bullet hole extraction, analysis and positioning. The results are transmitted to the LCD screen connected to the PC. The design has successfully completed the research of multiple links from hardware system design to software algorithm design, from image acquisition to score output.

**Keywords:** automatic target reporting; OpenMV; bullet hole recognition; pattern recognitio

目 录

[摘 要 1](#_Toc61376254)

[Abstract 2](#_Toc61376255)

[第一章 绪论 1](#_Toc61376256)

[1.1设计目的 2](#_Toc61376257)

[1.2设计要求 2](#_Toc61376258)

[第二章 系统组成及工作原理 3](#_Toc61376259)

[2.1 设计原理 3](#_Toc61376260)

[2.2系统组成 3](#_Toc61376261)

[2.3系统总体设计 3](#_Toc61376262)

[第三章 硬件电路方案确定 5](#_Toc61376263)

[3.1方案比较与确定 5](#_Toc61376264)

[3.2整体方案设计 6](#_Toc61376265)

[3.3器件选择 7](#_Toc61376266)

[3.3.1 OpenMV模块的比较与选择 7](#_Toc61376267)

[3.3.2受击靶模块的比较与选择 7](#_Toc61376268)

[3.3.3 STM32显示/通信模块 9](#_Toc61376269)

[第四章 软件设计 10](#_Toc61376270)

[4.1弹孔识别设计 10](#_Toc61376271)

[弹孔识别方案测试及选择 10](#_Toc61376272)

[4.2环数识别算法设计 11](#_Toc61376273)

[4.2.1环数识别方案选择 11](#_Toc61376274)

[4.2.2 环数识别算法（距离算法） 12](#_Toc61376275)

[4.3串口通信及LCD显示算法设计 13](#_Toc61376276)

[4.3.1串口通信方案选择 13](#_Toc61376277)

[4.3.1串口通信部分代码 13](#_Toc61376278)

[4.4历史记录读取算法设计 14](#_Toc61376279)

[4.5程序总体流程图 15](#_Toc61376280)

[第五章 实验、调试和测试结果与分析 16](#_Toc61376281)

[5.1测试方案 16](#_Toc61376282)

[5.2测试条件 16](#_Toc61376283)

[5.3测试结果 16](#_Toc61376284)

[5.4结果分析 17](#_Toc61376285)

[结论 18](#_Toc61376286)

[参考文献 19](#_Toc61376287)

[附录A 20](#_Toc61376288)

[附录B 21](#_Toc61376289)

# 第一章 绪论

OpenMV是一个开源，低成本，功能强大的机器视觉模块。以STM32F427CPU为核心，集成了OV7725摄像头芯片，在小巧的硬件模块上，用C语言高效地实现了核心机器视觉算法，提供Python编程接口。使用者们（包括发明家、爱好者以及智能设备开发商）可以用Python语言使用OpenMV提供的机器视觉功能，为自己的产品和发明增加有特色的竞争力。OpenMV采用的STM32F427拥有丰富的硬件资源，引出UART，I2C，SPI，PWM，ADC，DAC以及GPIO等接口方便扩展外围功能。USB接口用于连接电脑上的集成开发环境OpenMVIDE，协助完成编程、调试和更新固件等工作。TF卡槽支持大容量的TF卡，可以用于存放程序和保存照片等。而OpenMV的这一些特性都符合了本次课程设计的要求。

智能控制是具有智能信息处理、智能信息反馈和智能控制决策的控制方式，是控制理论发展的高级阶段，主要用来解决那些用传统方法难以解决的复杂系统的控制问题。图像识别技术是一门交叉学科，是模式识别的一个分支，图像的目的就是指利用计算机对图像进行处理、分析和理解，以识别各种不同模式的目标和对象的技术。图像技术作为智能计算机研究的关键和智能控制的关键技术之一，已经引起了越来越多的关注。

随着我国科技的不断进步，科技在军事上的重要性也有了显著的提升，智能打靶系统应运而生。本研究将智能控制技术与图像识别技术相结合，以实现智能打靶的目的，可以有效的减少人力、时间的损耗，具有重要的研究意义。

## 1.1设计目的

使系统能够实现识别图像、按要求完成智能打靶、报靶以及读取历史记录等功能，掌握基本控制为设计目的。  
（1）了解OpenMV部件结构与IDE使用；  
（2）了解电子元器件的基本形状及焊接过程；  
（3）了解OpenMV内部结构与程序编制方法；  
（4）了解图像识别原理，掌握智能控制技术；  
（5）通过本课设系统学习，全面掌握基于图像识别的节能智能控制系统设计方法。

## 1.2设计要求

使系统能够识别弹孔位置并判断弹孔所在环数，将成绩输出并保存。  
（1）完成以OpenMV为主控芯片的智能打靶系统总体系统设计；  
（2）完成智能打靶系统的总体结构搭建；  
（3）完成智能打靶系统的软件框架设计；

（4）完成智能打靶系统的软硬件之间的串口通信模块设计；  
（4）完成智能控制系统的硬件调试和软件调试；  
（5）最终的设计要求能完全表现智能打靶/报靶,成功识别弹孔并输出成绩、按要求完成智能控制等各项性能。

# 第二章 系统组成及工作原理

## 2.1 设计原理

智能打靶系统以OpenMV为主控模块，它是一款OpenMV是一个开源，低成本，功能强大的机器视觉模块。以STM32F427CPU为核心，集成了OV7725摄像头芯片，在小巧的硬件模块上，用C语言高效地实现了核心机器视觉算法，提供Python编程接口。OpenMV采用的STM32F427拥有丰富的硬件资源，引出UART，I2C，SPI，PWM，ADC，DAC以及GPIO等接口方便扩展外围功能。USB接口用于连接电脑上的集成开发环境OpenMVIDE，协助完成编程、调试和更新固件等工作。

使用OpenMVIDE进行程序的编写以及改进，对摄像头识别的图像进行处理，以达到识别弹孔的效果，并对于识别出的弹孔进行分析，判断此弹孔所在位置，与中心点位置进行距离计算得到当前成绩，并将成绩保存入档，达到实现基于OpenMV的图像识别的智能打靶系统的目的。

## 2.2系统组成

系统主控模块：OpenMV

图像识别：python OpenMV

通信模块：STM32与OpenMV的串口通信

显示模块：STM32连接的LCD屏幕

## 2.3系统总体设计

机器视觉技术是涉及多个领域的一门新兴技术，近年来随着人工智能技术的 发展 ，迎来了一轮新的产业技术革命 ，在工业、农业 、军事等领域得到了较为广 泛的应用，尤其在生产效率和精度方面相比人工方式具有很大的优势。

计算机用于发送指令 、统计打靶成绩、存储数据等 ，安放在首长控制台上， 可以用普通的 220V 交流供电，也可用计算机专用 UPS 等电源。

系统的操作和使用流程是：打靶训练开始前的场地准备阶段 ，将摄像机安放 在受击靶前端，开启计算机，打开计算机上的打靶控制软件，待系统启动完成后，打靶正式开始 ，战士按常规训练进行打靶。每打完一次后，若LCD屏幕上显示打靶成绩。

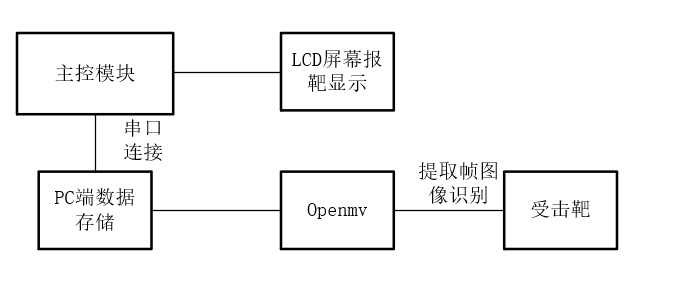
系统的总体框图如图 1.2.1 所示，工作流程是：当打靶结束后，终端机进行图像识别，摄像机对靶面进行拍摄 ，计算机收到视频流后开始提取其中的一l帧图像，并对这一帧靶 面图像进行预处理，如靶位分割、几何校正等。预处理后的靶面图像再进行弹孔 提取、环线检测、坐标计算 、成绩统计后，计算机自动将该组射击成绩等数据进行汇总 ，最后将成绩输出到显示屏。

图2.1 模块连接图

# 第三章 硬件电路方案确定

## 3.1方案比较与确定

（1）方案一：采用树莓派控制识别主控

将机器视觉方案搭载在树莓派上，使用树莓派控制电灯的开关以及摄像头进行图像识别。是一款基于ARM的微型电脑主板，以SD/MicroSD卡为内存硬盘，卡片主板周围有1/2/4个USB接口和一个10/100 以太网接口，可连接键盘、鼠标和网线，同时拥有视频模拟信号的电视输出接口和HDMI高清视频输出接口，以上部件全部整合在一张仅比信用卡稍大的主板上，但是其成本较高，并存在维护困难等问题，不具有一定推广普适性。

（2）方案二：采用PC处理图像数据，LCD显示

当打靶结束后，使用终端机进行图像识别，摄像机对靶面进行拍摄 ，计算机收到视频流后开始提取其中的一l帧图像，并对这一帧靶 面图像进行预处理，如靶位分割、几何校正等。预处理后的靶面图像再进行弹孔 提取、环线检测、坐标计算 、成绩统计后，计算机自动将该组射击成绩等数据进行汇总 ，最后将成绩输出到显示屏。PC端处理数据相比于单独应用树莓派更具有一定稳定性，这样符合了实际应用场景，具有一定普适性，有较大的实用价值。

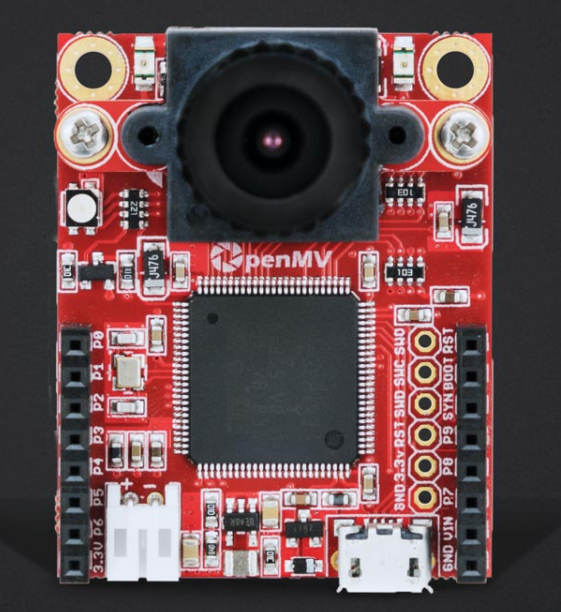
为了切合本次的实际场景及所有物件，考虑到进行图像处理与识别所需的算力以及时间问题，以及OpenMV内存大小的硬件问题等，我们选择方案一，使用STM32与OpenMV进行串口通信，可以节省OpenMV的内存，不影响到图像的识别、处理速度，更好地处理图像相关问题。

图 3.1 树莓派实物图

## 3.2整体方案设计

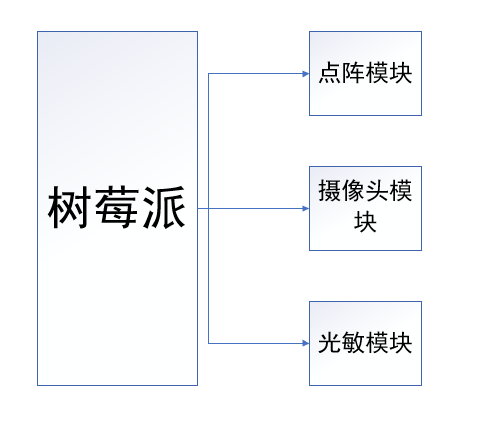
本设计的整体方案主要由OpenMV，STM32的LCD显示模块，受击靶模块等模块组成。本系统组成模块如下图3.2所示。本系统实物图如图3.3所示。

图 3.2 系统组成模块图

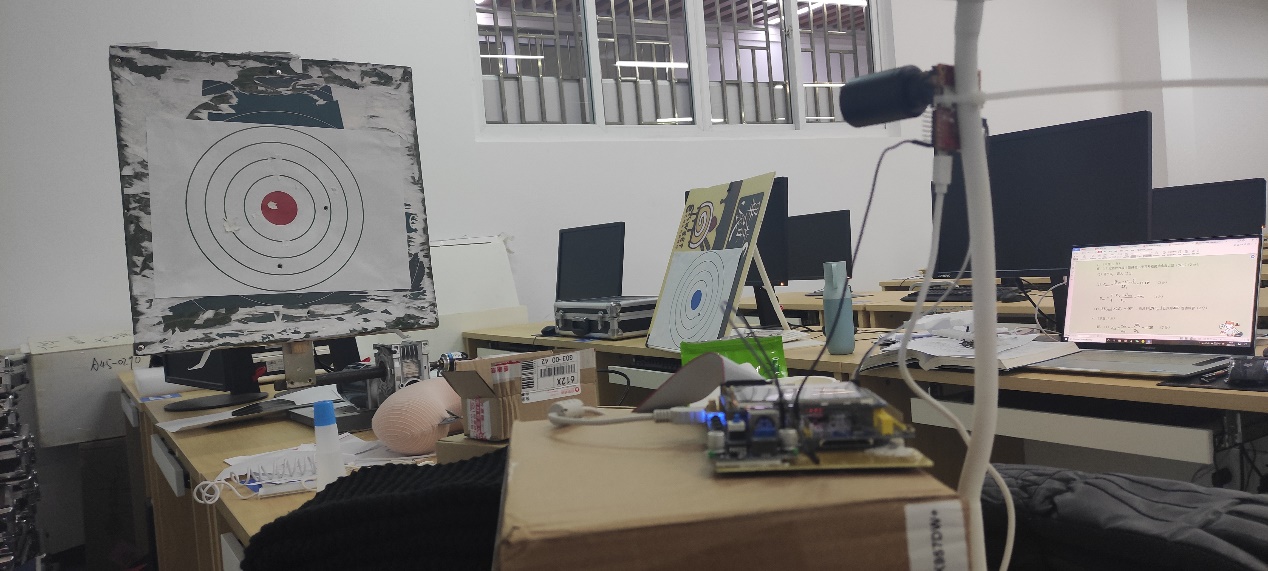


图 3.3 系统实物图

## 3.3器件选择

### 3.3.1 OpenMV模块的比较与选择

首先考虑到每个版本的OpenMV的CPU大小，内存不同，而OpenMV除了利用摄像头进行图像的收集外，还需要对所收集的图像进行识别处理。本着提高精确度，提高运算速度的目的，所以我们选择了OpenMV H7系列，并搭载长焦摄像头。使用实物图如图3.4所示。

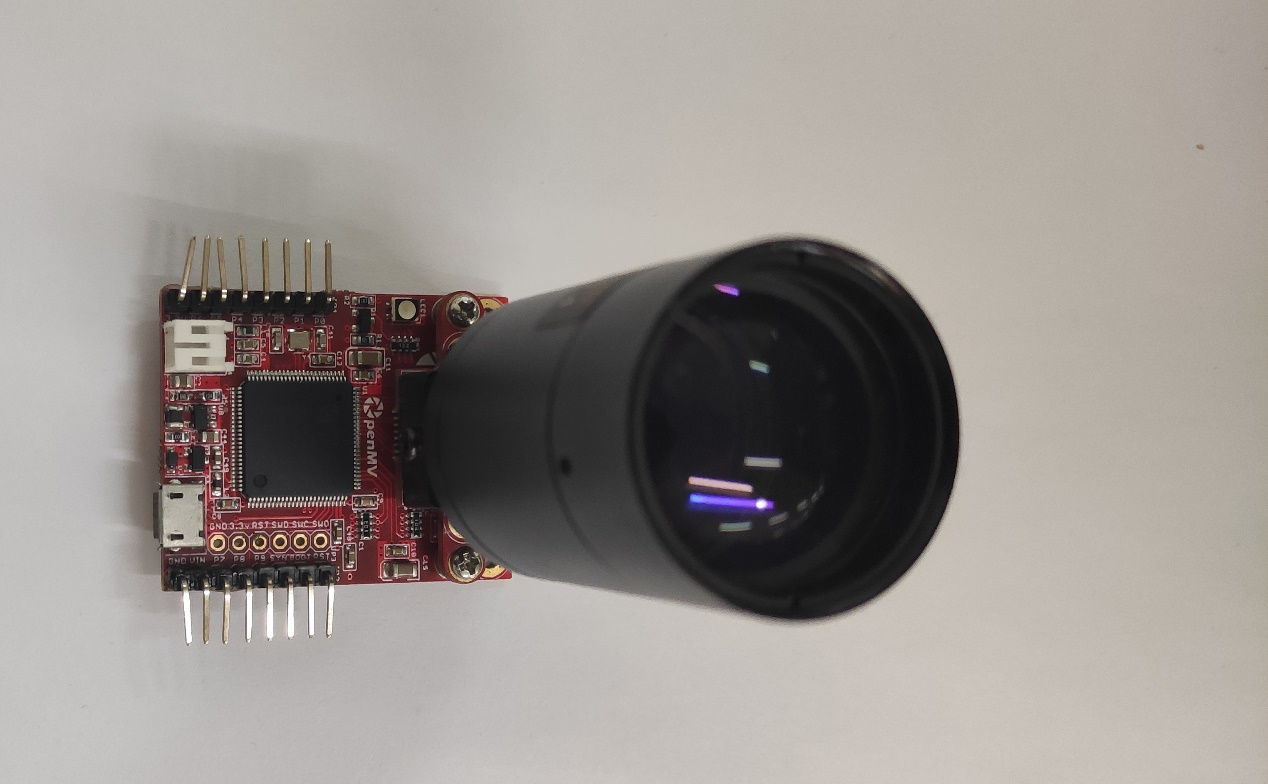


图 3.4 OpenMV H7搭载长焦镜头实物图

### 3.3.2受击靶模块的比较与选择

方案一：使用胸环靶作为受击靶。

方案二：使用圆环靶作为受击靶。

在本次设计中，我们考虑到光线，OpenMV摄像头分辨率等图像采集的问题，并通过实际测试检验，选择使用方案二的圆环靶作为受击靶。由于硬件问题，以及技术水平，使用方案一进行测试时，摄像头采集到的图像模糊不够清晰，无法比较准确的锁定真正的弹孔位置，会将其他阈值相近的非弹孔feature识别为弹孔，所以放弃方案一，而采用比较清晰的方案二。

其中，方案一测试图（小红圈位置为弹孔位置）、方案二受击靶实物图如下图3.5、图3.6所示。

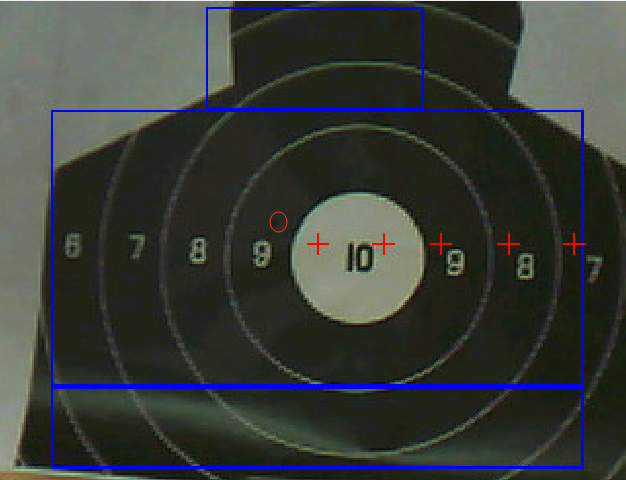
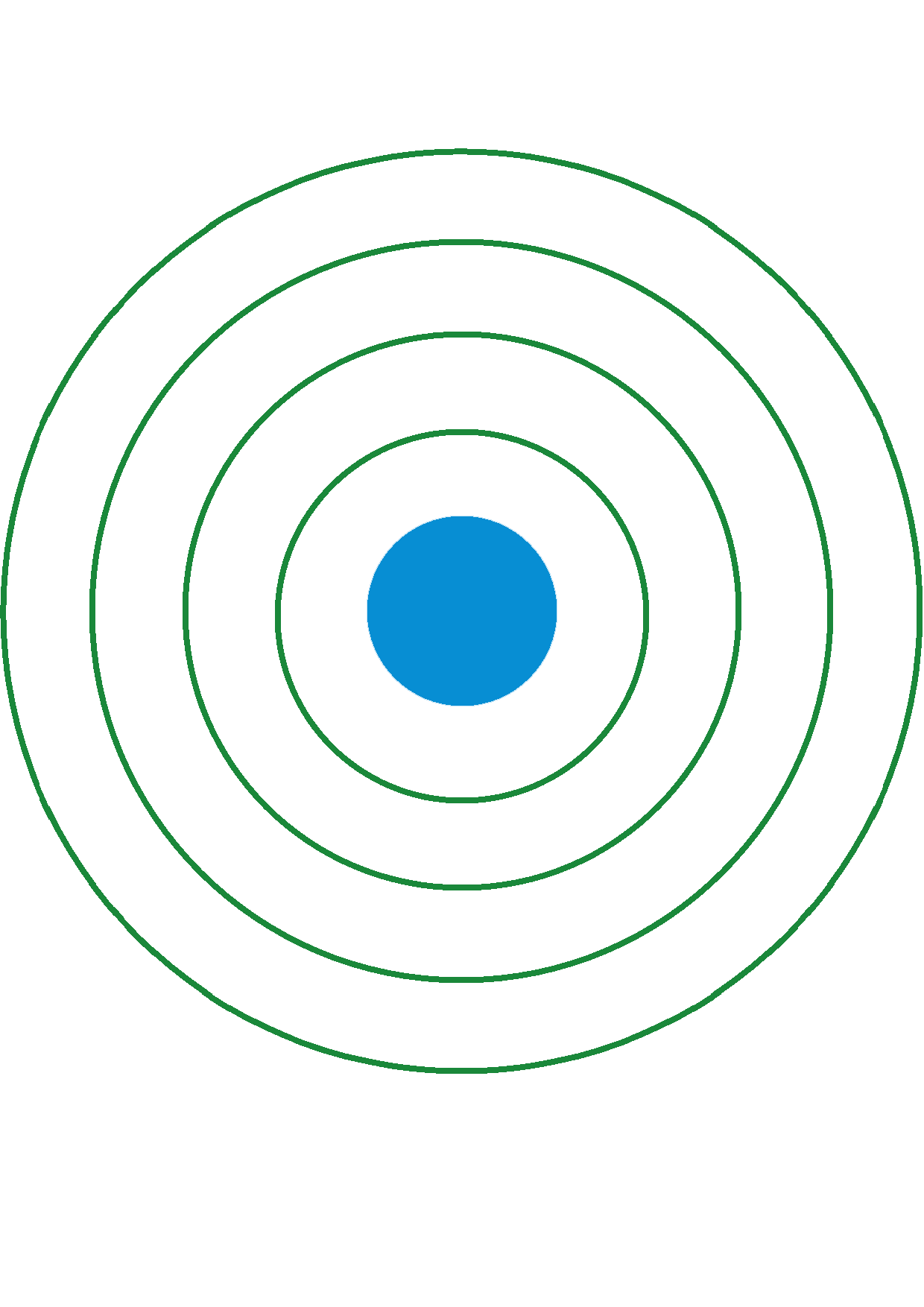
 图 3.5 方案一测试图

图 3.6 圆环靶实物图

### 3.3.3 STM32显示/通信模块

在本次系统设计中，加入STM32模块是因为本系统需要输出打靶成绩，并将其显示出来。STM32搭载的LCD显示屏作为成绩输出显示位置，而成绩的传输则使用STM32的串口与OpenMV进行通信，设置相应的传输格式，以保证数据传输正确无误。STM32原理图与实物图如下图3.7、图3.8所示。

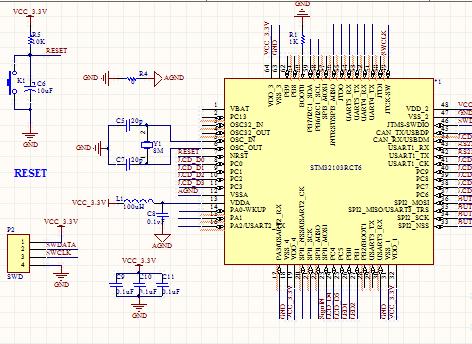


图 3.7 STM32原理图

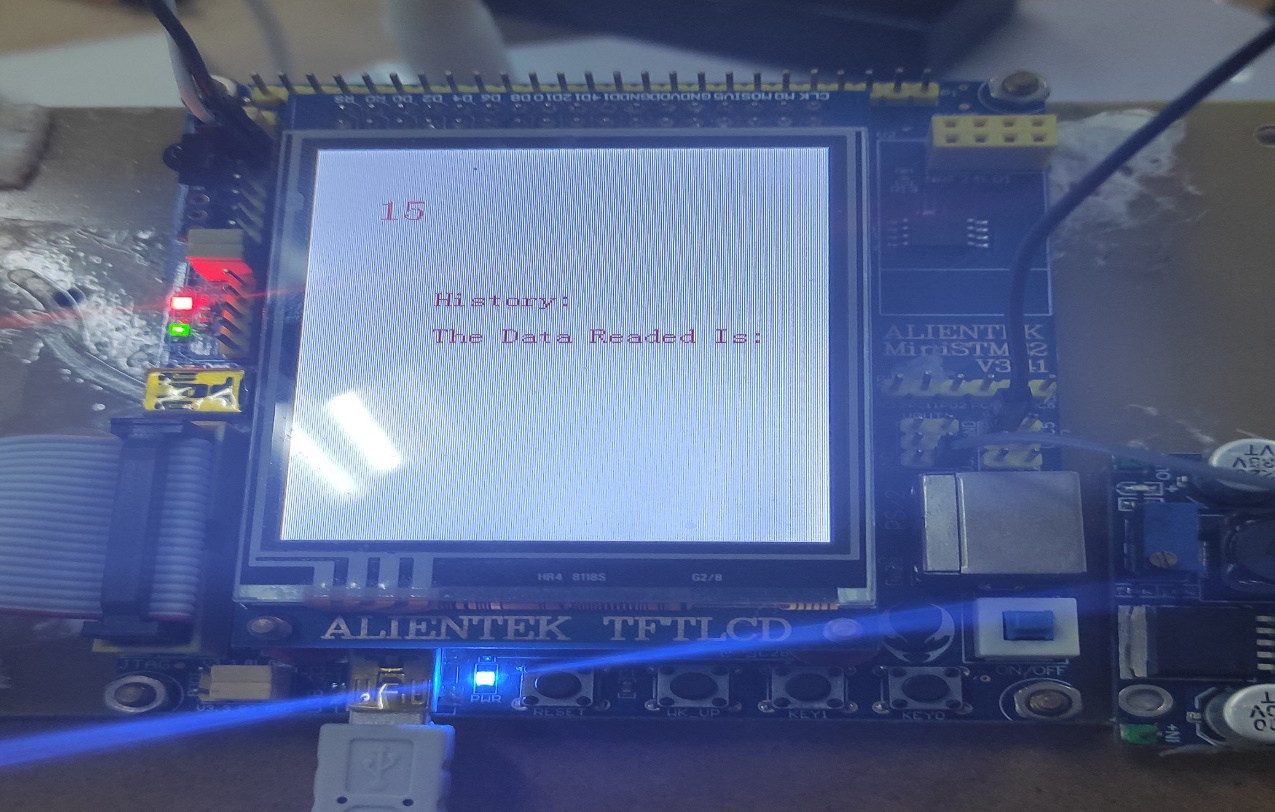


图 3.8 STM32搭载LCD实物图

# 第四章 软件设计

## 4.1弹孔识别设计

### 弹孔识别方案测试及选择

弹孔识别是自动报靶系统的关键环节之一，以往的方法都存在一些不足，如基于差影法的弹孔检测需要采集子弹中靶前后的两张图像，并且对这两幅图像中的靶面位置精度要求较高，任何微小的晃动都会导致配准失败，这也意味着，所需要的硬件要求也比较高，所需要花费的金额更大。实际上子弹击中靶面引起的微小晃动无法避免。因此，在弹孔识别方面，需要进行一定优化假设。

本次课程设计，经过讨论，我们采用两套方案进行测试筛选。

方案一：对识别的图像进行二值化处理，二值化处理后的图像将呈现出环境与feature的分化现象，从而识别出弹孔位置；在测试过程中发现，二值化后的图像中，环与环之间的线条与弹孔划分为一类，而由于摄像头的精确度问题，即使设置了像素大小过滤器也无法剔除干净线条的影响，摄像头仍然会将小部分的线条作为一串或一个长条性的feature；

方案二：对识别的图像，进行分区域像素阈值分析，若靶的像素阈值范围与弹孔的像素阈值范围有差异，则可以通过调用OpenMV的阈值编辑器，选出弹孔的像素阈值区，以及周围环境的像素阈值区，从而达到识别弹孔的效果。

经过测试检验，以及实际硬件问题，我们选择使用方案二作为弹孔识别的最终方案。以下是方案一以及方案二的测试图像。

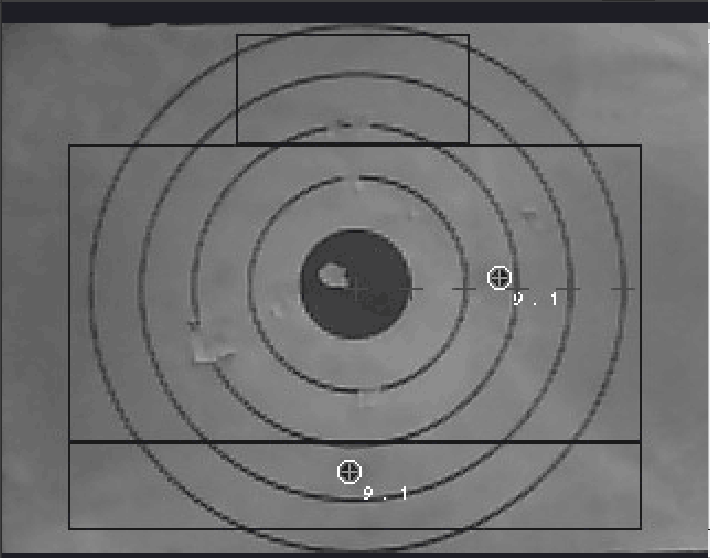
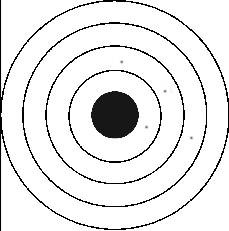


图 4.1 方案一/二测试图

## 4.2环数识别算法设计

### ****4.2.1环数识别方案选择****

方案一：使用二值化及轮廓发现算法进行区分，轮廓发现是基于图像边缘提取的基础寻找图像轮廓的方法。所以边缘提取的阈值选定会影响最终轮廓发现的结果。API介绍：

findContours——发现轮廓； drawContours——绘制轮廓

cv.RETR\_TREE，外轮廓内部的轮廓也会标记

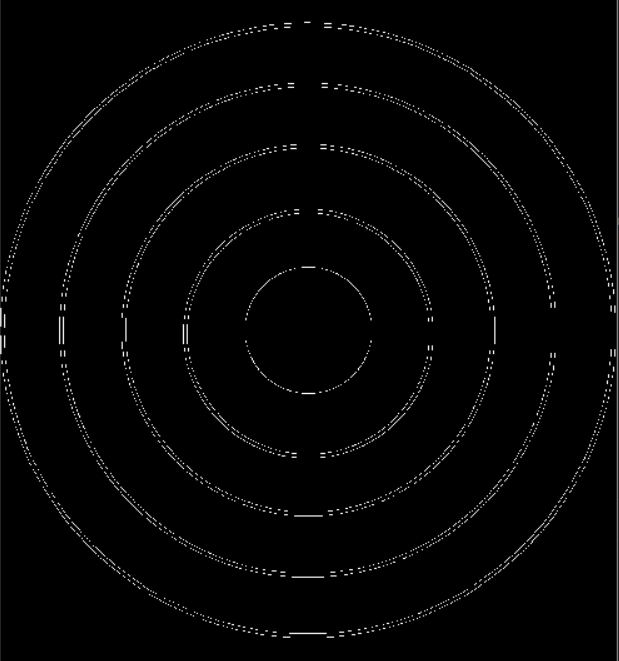
cv.RETR\_EXTERNAL，只标记外轮廓

图 4.2 轮廓发现算法测试图

方案二：使用中心点偏移距离算法进行区分，使用OpenMVIDE将中心点位置锁定，再设计测距算法返回两点之间的距离值，对比每个环区到中心点的范围，得到最终环数。

假设中心点位置为（x1,y1），弹孔位置为（x2,y2），使用勾股定理

length = sqrt((x1 – x2)² + (y1 – y2)²)

考虑到弹孔识别算法的方案，我们选择方案二的距离算法最为环数识别算法的最终方案。

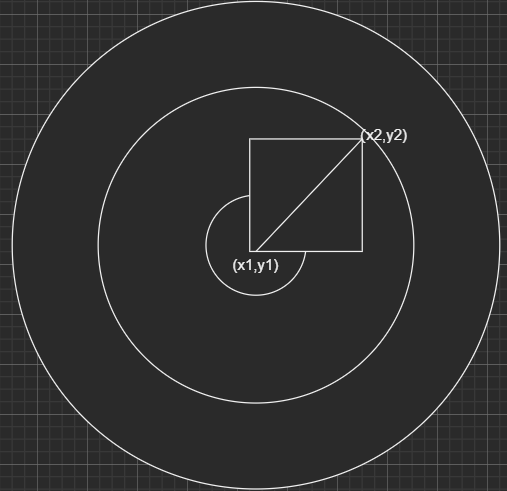


图 4.3 距离算法原理图

### 4.2.2 环数识别算法（距离算法）

1. **def** distance\_sq\_between(x1, y1, x2, y2):
2. x = x1 - x2
3. y = y1 - y2
4. **return** x \* x + y \* y

## 4.3串口通信及LCD显示算法设计

### ****4.3.1串口通信方案选择****

方案一：使用串口通信（TTL），至少需要3根线，RXD、TXD、GND；

方案二：使用915mhz的无线数传；

这里我们使用方案二，无线数传的适应性强、扩展性好、开通快，且在本次课程设计中，不用担心STM32与PC之间的距离，可以更好的对整个系统框架进行布局。以下为无线数传的实物图：

图 4.4 915mhz无线数传实物图

### ****4.3.1串口通信部分代码****

1. #串口通信：
2. u8 t=0;
3. u16 num = 0;
4. **static** u8 num\_for\_addr = 0;
5. delay\_init();
6. uart\_init(9600);
7. USART2\_Init(115200);
8. LED\_Init();
9. LCD\_Init();
10. KEY\_Init();
11. AT24CXX\_Init();
12. POINT\_COLOR=RED;
14. **for** (**int** i = 8; i < 16; i++) {
15. recv\_buf[i] = ' ';
16. }
17. recv\_buf[15] = 0;

## 4.4历史记录读取算法设计

历史记录读取算法在STM32上编写运行，具体流程为：将OpenMV传入的数据保存在单片机内存中，按下按键1，将其保存至闪存eeprom中作为存档，由于OpenMV一直在发送数据，STM32这边使用写入功能进行每一次的成绩读入，并保存至eeprom中，按下按键2可查看之前的历史记录，按下按键3可查看往后的记录，即按键2/3为翻页功能按键，以供使用者查看以往成绩单，按键4按下后为重启单片机。

**历史记录读取部分算法：**

1. **for** (**int** i = 8; i < 16; i++) {
2. recv\_buf[i] = ' ';
3. }
4. recv\_buf[15] = 0;
6. history\_write\_ptr = AT24CXX\_ReadOneByte(EEPROM\_LAST\_SAVE\_POS);
7. **if** (history\_write\_ptr == 0xffu) {
8. history\_write\_ptr = 0;
9. on\_button\_clear\_press();
10. }
12. history\_rotated = 0x80u & history\_write\_ptr;
13. history\_write\_ptr &= 0x1fu;
15. eeprom\_bulk\_load();
16. LCD\_ShowString(60,150,200,16,16, "History loaded");

## 4.5程序总体流程图

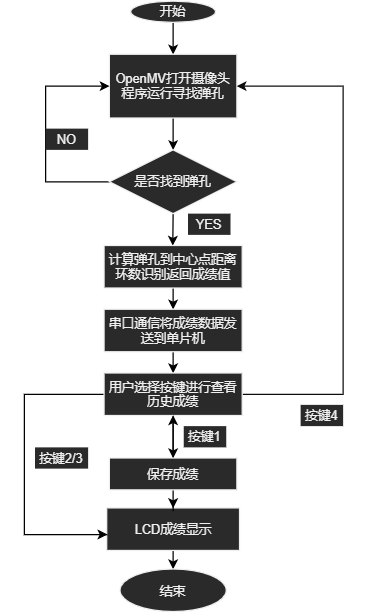


图 4.5 系统总体流程图

# 第五章 实验、调试和测试结果与分析

## 5.1测试方案

(1)选择好所需要的元器件后，完成外部硬件系统框架搭建。

(2)将OpenMV通上电后，通过使用OpenMVIDE，测试OpenMV是否能正常运行。

(3)利用连接好的OpenMV获取摄像头数据，测试摄像头是否正常。

(4)运行弹孔识别模块与环数识别模块，测试系统能否进行弹孔位置以及环数成绩的判断。

(5)将STM32与OpenMV进行无线串口通信，测试连接好的STM32的LCD显示屏能否正常显示成绩。

(6)所有硬件都连接好的情况下，通过现场测试是否能正确的识别弹孔的位置，判断出环数成绩，将成绩显示在STM32的LCD屏上，并查看是否能显示历史成绩等功能

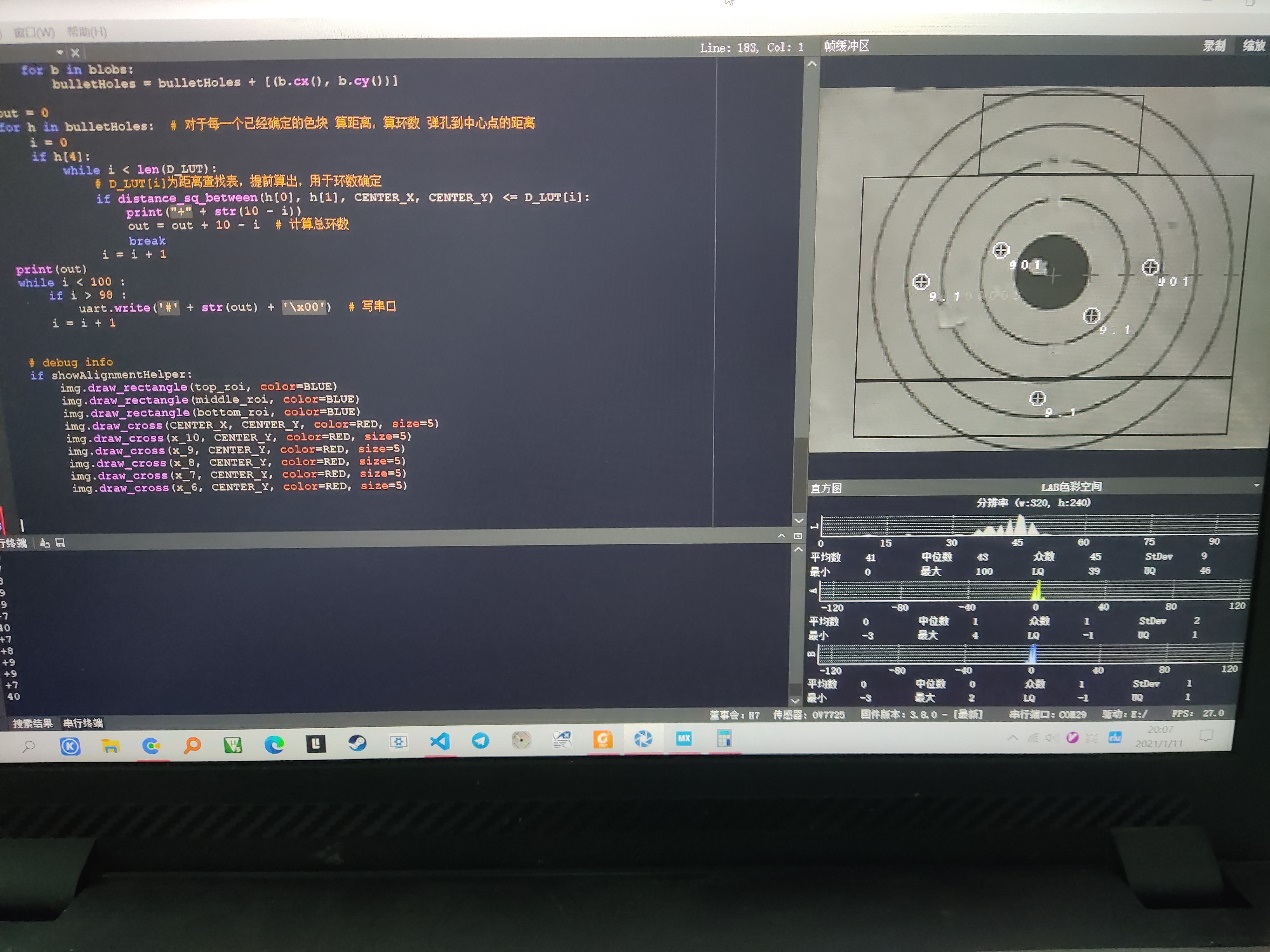
## 5.2测试条件

在OpenMV，摄像头、STM32等固硬件没有损坏，有一定光亮的情况下，测试智能打靶系统的弹孔识别、环数识别以及LCD屏显示结果。

## 5.3测试结果

 测试结果如下图所示：

图5.1 智能打靶系统成绩输出图

图5.2 弹孔识别、环数识别实测图

## 5.4结果分析

从结果中可以看出，系统能有效的识别到弹孔的位置，较为正确的返回打靶的成绩，并能在STM32单片机上进行成绩的显示，以及历史成绩的查询。

在测试的过程中发现，系统对于外部光学，硬件条件有一定的依赖，光亮充足，摄像头分辨率足够清晰的情况下，系统的准确性、稳定性都比较高，而在光学条件较暗，摄像头清晰度不足的时候，系统虽然仍能识别，但是会将一些非弹孔的feature读取进来，从而大大降低了系统的准确性和稳定性。较于实际应用来说，本系统不足之处还有很多，比如靶与OpenMV之间的距离稍微远一点，识别的准确性就会降低、图像的处理速度较慢、系统的识别时间还不够快等。

总的来说，系统提升改善的余地较大，提高系统识别的时间，改善系统在不同环境下的控制逻辑将成为下一步改进的方向，使系统够更好地节约能源。

## 5.5方案论证及成本分析

### ****5.5.1方案论证****

### ****5.5.2成本分析****

# 结论

本文基于OpenMV的图像识别模块以及STM32实现了基于OpenMV图像识别的智能打靶系统，主要研究了弹孔识别算法、环数识别算法在不同环境下对于弹孔识别的准确性和快速性，利用OpenMV和STM32的串口通信连接，实现了将成绩输出到LCD屏上的目的，完成了智能打靶系统的目的。

本文的实验数据都是在环境比较简单的教室中采集的，在图像识别的过程中没做过多的进行滤波和除杂，在简单环境中能够较好地进行图像识别并完成弹孔识别、环数识别及成绩输出等功能。但实际运用中，在环境较为复杂的情况比如黑暗中并不能发挥很好的作用。因此，怎样提高系统在不同环境下的识别准确率和速度，使系统更好地适用于多种多样的环境下并发挥智能打靶系统的作用，将是下一步的重点之一。

同时，本系统具有良好的可扩展性，运用舵机云台控制的方式可同时控制OpenMV进行多方位的识别，比如识别现有打靶中的移动靶。所以如何将智能打靶系统扩展为智能多方位打靶系统或者智能多位姿打靶系统，也是下一步需要重点研究的方向。

# 参考文献

[1]Liu Yisi,Zhao Gaowei. Simulation of swimming sports image recognition based on multi-core processor and dynamic image sampling[J]. Microprocessors and Microsystems,2021,81.

[2]Tang Nana. Image recognition algorithm for exercise fatigue based on FPGA processor and motion image capture[J]. Microprocessors and Microsystems,2021,81(prepublish).

[3]Miao Rui,Jiang Zihang,Zhou Qinye,Wu Yizhou,Gao Yuntian,Zhang Jie,Jiang Zhibin. Online inspection of narrow overlap weld quality using two-stage convolution neural network image recognition[J]. Machine Vision and Applications,2021,32(1).

[4]廖小春,华莹珂.一种图像识别系统与远传技术的应用研究[J].铜业工程,2020(06):83-85+100.

[5]彭昕昀,熊正辉,黄伟强,何振环,欧森源,唐武翰,陈锦平.基于姿态解算和图像识别算法的四旋翼自主飞行器系统[J].韶关学院学报,2020,41(06):33-38.

[6]施敏虎,栗云鹏,庄曙东,符正帆,王齐鑫.基于OpenMV的智能搬运车型机器人的设计[J].机械工程师,2020(02):20-22+25.

[7]王泽川,曾维鹏,黄果.基于OpenMV图像识别智能小车的设计与制作[J].电子世界,2019(24):143-144.

[8]李晋.基于openmv智能机械手臂创新实践课程探索[J].电脑知识与技术,2018,14(31):197-199.

[9]陈洪容,覃智广,王洪,张锐丽.一种智能打靶装置的设计[J].现代职业教育,2016(31):134.

[10]姜海明. 光纤Raman放大器增益谱平坦化研究[D].电子科技大学,2011.

[11]杨中国.基于实弹打靶训练系统的智能标靶硬件设计[J].通信技术,2010,43(06):189-190+193.

# 附录A

小组分工：

李闯： 1、系统整体思路敲定；

2、eeprom的写入、读取以及擦除算法；

3、按键读取历史成绩选择算法的硬件选型及算法实现；

4、STM32通过模拟LLC与eeprom通信（串口通信）

李旭： 1、软件图像处理部分思路敲定；

2、系统整体处理调试，优化系统准确率；

3、负责软硬件环境搭建，协调硬件优化实际效果；

4、LCD成绩显示功能实现；

黄金荣： 1、弹孔识别算法设计实现；

2、环数识别算法设计实现；

3、识别算法整合；

# 附录B

1. **Openmv代码**：  
      
   **from** machine **import** UART
2. **import** sensor, image, time, pyb
4. sensor.reset()
5. sensor.set\_framesize(sensor.QVGA)
6. sensor.set\_pixformat(sensor.RGB565)
7. sensor.skip\_frames(time = 2000)
8. clock = time.clock()
9. #串口通信
10. uart = UART(1, 115200)
11. uart.init(115200, bits=8, parity=None, stop=1)
13. # Program switches
14. useGlobalSeeking = True
15. useColor = True
16. useRoundnessSift = False
17. showAlignmentHelper = True
18. showDebugMarkers = False
19. useConfidenceFilter = True
20. useInvertingFieldScan = False
22. # Thresholds
23. darkThreshold = (0, 32)
24. brightThreshold = (137, 255)
25. colorThresholdHigherBright = (64, 100, 17, 45, 5, 31)
26. colorThresholdLowerBright = (90, 100, -15, 4, -3, 16)
27. roundnessThreshold = 0.55
28. confidenceDistance = 5
29. confidenceFrameCount = 5
31. # ROIs
32. attention\_roi = (0, 0, 320, 240)
34. # Constants
35. RED = (255, 0, 0)
36. BLUE = (0, 0, 255)
37. GREEN = (0, 255, 0)
38. BLACK = (0, 0, 0)
39. CENTER\_X = 160
40. CENTER\_Y = 120
41. x\_10 = 192
42. x\_9 = 220
43. x\_8 = 253
44. x\_7 = 285
45. x\_6 = 315
46. D\_LUT = [x\_10 - CENTER\_X, x\_9 - CENTER\_X, x\_8 - CENTER\_X, x\_7 - CENTER\_X, x\_6 - CENTER\_X]
48. # Global ground
49. field = True
50. bulletHoles = []
51. removeList = []

54. **def** distance\_sq\_between(x1, y1, x2, y2):
55. x = x1 - x2
56. y = y1 - y2
57. **return** x \* x + y \* y
59. # init
61. i = 0
62. **while** i < len(D\_LUT):
63. D\_LUT[i] = D\_LUT[i] \* D\_LUT[i]
64. i = i + 1
66. **while**(True):
67. blobs = []
68. img = sensor.snapshot()
69. **if** useGlobalSeeking:
70. **if** useColor:
71. blobs = blobs + img.find\_blobs([colorThresholdHigherBright], roi=attention\_roi, pixels\_threshold=3, merge=True)
72. **else**:
73. # TODO
74. **pass**
75. **else**:
76. # this branch is left intentionally blank
77. **pass**
79. **if** useRoundnessSift:
80. blobs = [b **for** b **in** blobs **if** b.roundness() > roundnessThreshold]
82. **if** useConfidenceFilter:
83. **for** b **in** blobs:
84. found = False
85. **if** showDebugMarkers:
86. img.draw\_circle(b.cx(), b.cy(), 3, BLACK)
87. **for** h **in** bulletHoles:
88. **if** showDebugMarkers:
89. img.draw\_circle(h[0], h[1], confidenceDistance)
90. img.draw\_string(h[0] + confidenceDistance, h[1] + confidenceDistance, str(h[2]))
91. **if** distance\_sq\_between(h[0], h[1], b.cx(), b.cy()) < confidenceDistance \* confidenceDistance:
92. h[2] = h[2] + 1
93. found = True
94. **if** h[2] > confidenceFrameCount:
95. **if** h[2] > 2 \* confidenceFrameCount:
96. h[2] = 2 \* confidenceFrameCount
97. img.draw\_cross(h[0], h[1], size=3, color=GREEN)
98. h[3] = True
99. **break**
100. **if** **not** found:
101. bulletHoles = bulletHoles + [[b.cx(), b.cy(), 1, True],]
102. removeList.clear()
103. **for** h **in** bulletHoles:
104. **if** **not** h[3]:
105. h[2] = h[2] - 1
106. **if** h[2] < 1:
107. removeList = removeList + [h]
108. h[3] = False
109. **for** x **in** removeList:
110. bulletHoles.remove(x)
111. **else**:
112. bulletHoles.clear()
113. **for** b **in** blobs:
114. bulletHoles = bulletHoles + [(b.cx(), b.cy())]
116. i = 0
117. out = 0
118. **for** h **in** bulletHoles:
119. **while** i < len(D\_LUT):
120. **if** distance\_sq\_between(h[0], h[1], CENTER\_X, CENTER\_Y) <= D\_LUT[i]:
121. out = out + 10 - i
122. **break**;
123. i = i + 1
124. **print**(out)
125. uart.write("$" + str(out) + "\x00")
127. # 辅助标点框图
128. **if** showAlignmentHelper:
129. img.draw\_rectangle(top\_roi, color=BLUE)
130. img.draw\_rectangle(middle\_roi, color=BLUE)
131. img.draw\_rectangle(bottom\_roi, color=BLUE)
132. img.draw\_cross(CENTER\_X, CENTER\_Y, color=RED, size=5)
133. img.draw\_cross(x\_10, CENTER\_Y, color=RED, size=5)
134. img.draw\_cross(x\_9, CENTER\_Y, color=RED, size=5)
135. img.draw\_cross(x\_8, CENTER\_Y, color=RED, size=5)
136. img.draw\_cross(x\_7, CENTER\_Y, color=RED, size=5)
137. img.draw\_cross(x\_6, CENTER\_Y, color=RED, size=5)

**STM32代码：**

1. #include "led.h"
2. #include "delay.h"
3. #include "sys.h"
4. #include "usart.h"
5. #include "lcd.h"
6. #include "usart2.h"
7. #include "key.h"
8. #include "24cxx.h"
9. #include "myiic.h"
10. #include <string.h>
12. **extern** **volatile** u8 buf01;
13. **extern** **volatile** u8 frame\_available;
14. **extern** **volatile** u8 recv\_buf[16];
15. #define Max\_num 32
16. **char** history[Max\_num][4]={0};
17. **static** u8 history\_read\_ptr = 0, history\_write\_ptr = 0, history\_rotated = 0;
18. **static** u32 hold\_time\_len = 0;
19. **static** u8 triggered\_buttons = 0;
21. #define EEPROM\_LAST\_SAVE\_POS 0
22. #define EEPROM\_SAVE\_POS 1
24. **void** eeprom\_init() {
25. **for** (**int** i = 0; i < 256; i++) {
26. AT24CXX\_WriteOneByte(i, 0);
27. }
28. }
30. **void** eeprom\_bulk\_load() {
31. **for** (u8 i = 0; i < (history\_rotated ? Max\_num : history\_write\_ptr); i++) {
32. AT24CXX\_Read(EEPROM\_SAVE\_POS + i \* 3, (u8 \*) history[i], 3);
33. history[i][3] = 0;
34. }
35. }
37. **void** on\_button\_one\_press() {
38. LCD\_ShowNum(60,100,history\_read\_ptr,16,16);
39. LCD\_ShowString(60,120,200,16,16,"History:");
40. LCD\_ShowString(60,170,200,16,16,"   ");
41. LCD\_ShowString(60,170,200,16,16,(u8\*)history[history\_read\_ptr]);????????
42. **if** (history\_rotated) {
43. **if** (++history\_read\_ptr >= Max\_num) {
44. history\_read\_ptr = 0;
45. }
46. } **else** {
47. **if** (++history\_read\_ptr >= history\_write\_ptr) {
48. history\_read\_ptr = 0;
49. }
50. }
51. }
53. **void** on\_button\_two\_press() {
54. LCD\_ShowNum(60,100,history\_read\_ptr,16,16);
55. LCD\_ShowString(60,120,200,16,16,"History:");
56. LCD\_ShowString(60,170,200,16,16,"   ");
57. LCD\_ShowString(60,170,200,16,16,(u8\*)history[history\_read\_ptr]);//??????????????
58. **if** (history\_rotated) {
59. **if** (history\_read\_ptr-- == 0) {
60. history\_read\_ptr = Max\_num - 1;
61. }
62. } **else** {
63. **if** (history\_read\_ptr-- == 0) {
64. **if** (history\_write\_ptr == 0) {
65. history\_read\_ptr = 0;
66. } **else** {
67. history\_read\_ptr = history\_write\_ptr - 1;
68. }
69. }
70. }
71. }
73. **void** on\_button\_clear\_press() {
74. LCD\_ShowString(60,150,200,16,16,"Clearing FLASH data.... ");
75. eeprom\_init();
76. LCD\_ShowString(60,150,200,16,16,"MEMORY CLEARED          ");
77. }
79. **void** on\_button\_save\_press() {
80. strncpy(history[history\_write\_ptr], (**char** \*) recv\_buf, 3);
81. history[history\_write\_ptr][3] = 0;
82. LCD\_Fill(0,0,239,319,WHITE );
83. LCD\_ShowString(60,150,200,16,16,"Start Write 24C02....");
84. AT24CXX\_Write(EEPROM\_SAVE\_POS + history\_write\_ptr \* 3, (u8\*) history[history\_write\_ptr], 3);
85. **if** (++history\_write\_ptr >= Max\_num) {
86. history\_rotated = 1;
87. }
88. AT24CXX\_WriteOneByte(EEPROM\_LAST\_SAVE\_POS, (history\_rotated ? 0x80u : 0) | history\_write\_ptr);
89. LCD\_ShowString(60,150,200,16,16,"24C02 Write Finished!");
90. }
92. **int** main(**void**)
93. {
94. u8 t=0;
95. u16 num = 0;
96. **static** u8 num\_for\_addr = 0;
97. delay\_init();
98. uart\_init(9600);
99. USART2\_Init(115200);
100. LED\_Init();
101. LCD\_Init();
102. KEY\_Init();
103. AT24CXX\_Init();
104. POINT\_COLOR=RED;
106. **for** (**int** i = 8; i < 16; i++) {
107. recv\_buf[i] = ' ';
108. }
109. recv\_buf[15] = 0;
111. history\_write\_ptr = AT24CXX\_ReadOneByte(EEPROM\_LAST\_SAVE\_POS);
112. **if** (history\_write\_ptr == 0xffu) {
113. history\_write\_ptr = 0;
114. on\_button\_clear\_press();
115. }
117. history\_rotated = 0x80u & history\_write\_ptr;
118. history\_write\_ptr &= 0x1fu;
120. eeprom\_bulk\_load();
121. LCD\_ShowString(60,150,200,16,16, "History loaded");
123. **while**(1)
124. {
125. t = KEY\_Scan(0);        // ??????? ????????
126. **if** (triggered\_buttons == t) {
127. // button holding
128. **switch** (t) {
129. **case** KEY0\_PRES:
130. **case** KEY1\_PRES:
131. ++hold\_time\_len;
132. **break**;
133. **default**:
134. // holding not allowed
135. hold\_time\_len = 0;
136. **break**;
137. }
138. **if** (hold\_time\_len > 100u) {
139. LED0 = !LED0;
140. **switch** (t) {
141. **case** KEY0\_PRES:
142. on\_button\_one\_press();
143. **break**;
144. **case** KEY1\_PRES:
145. on\_button\_two\_press();
146. **break**;
147. **default**:
148. **break**;
149. }
150. hold\_time\_len = 0;
151. }
152. } **else** {
153. LED0 = !LED0;
154. // new triggered button
155. **if** (t > 0) {
156. // new button press
157. **switch** (t) {
158. **case** KEY0\_PRES:
159. on\_button\_one\_press();
160. **break**;
161. **case** KEY1\_PRES:
162. on\_button\_two\_press();
163. **break**;
164. **case** KEY2\_PRES:
165. on\_button\_save\_press();
166. **break**;
167. **case** WKUP\_PRES:
168. on\_button\_clear\_press();
169. **break**;
170. **default**:
171. **break**;
172. }
173. } **else** {
174. hold\_time\_len = 0;
175. LED1 = !LED1;
176. }
177. triggered\_buttons = t;
178. }
179. POINT\_COLOR=RED;
180. **if** (frame\_available) {
181. LCD\_ShowString(30,40,200,24,24, (u8\*) recv\_buf);
182. frame\_available = 0;
183. }
184. }
185. }