**《电路与电子III》项目结题报告**

机械臂汽车自动加油装置的研制与开发

学 院： 未来技术学院

专业班级： 智能机器平台一班、二班

姓 名： 周艺梵、刘长远、黄纯竹

学 号： 3021207192、3021202132、3021232075

**天津大学**

**未来技术学院**

**二零二三 年 六 月**

**摘要**

随着物流产业的兴起以及科技的发展，智能搬运机器已经成为人们研究的热点话题。如何让智能机器在完成避障、定位、取件、返回等多个任务的同时，实现全自动控制，并兼顾轻便性和环境友好性，是该项目研究的一大挑战。

本组项目以Tankbot智能搬运履带机器人为原型，模拟加油站中加油这一工作的执行。Tankbot智能搬运履带机器人是一款以STM32单片机为主控制板，集合红外传感器、超声波传感器、声音传感器、机械臂等多个模块的机器人。经过一个完整的项目设计周期，包括对于项目背景的查找，制定项目规划，硬件设计与实现，软件设计与编程；本组完成了对于智能履带机器人的设计工作，并通过集成与功能测试等环节，探讨如何将其应用于市场。本组着重于软件设计与实现阶段，以C语言为基础，Keil5为编程软件，利用轮巡控制与并行控制的基本思想编写代码，并进行符合实际测试环境的调试工作。最终，实现了预期的全部功能，产品具有极佳的测试效果与应用前景。

**关键词：智能搬运 STM32单片机 软件设计 轮巡控制**

目录

[摘要 II](#_Toc4009)

[第一章 研究背景 1](#_Toc6570)

[1.1 机械臂汽车自动加油装置的研究背景和意义 1](#_Toc22729)

[1.2 机械臂汽车自动加油装置存在的技术难题和实际问题 2](#_Toc11123)

[1.3 OpenMv视觉系统及其特点和优势 3](#_Toc19731)

[1.4 OpenMv视觉系统在机械臂汽车自动加油装置中的应用前景和研究意义 3](#_Toc32335)

[第二章 项目规划 4](#_Toc16246)

[2.1 研究目标 4](#_Toc30686)

[2.2 实施计划 5](#_Toc11199)

[2.2.1 项目作品需求表达 5](#_Toc18109)

[2.2.2 项目作品的计划实施 6](#_Toc10260)

[2.3 成员及分工 7](#_Toc10892)

[2.4 进度及安排 8](#_Toc14169)

[第三章 硬件设计与实现 9](#_Toc32066)

[3.1 硬件设计与器件选型 9](#_Toc11208)

[3.1.1 Tankbot智能搬运履带机器人整体功能参数 9](#_Toc4066)

[3.1.2 OpenMv视觉系统 10](#_Toc834)

[3.1.3 总线舵机 11](#_Toc15883)

[3.1.4 语音识别模块 11](#_Toc13087)

[3.1.5 4路循线传感器 12](#_Toc30474)

[3.2 原理图设计 13](#_Toc18310)

[3.2.1 4路循线传感器原理图 13](#_Toc12357)

[3.2.2 STM32单片机原理图 13](#_Toc6815)

[3.2.3 小车主控制器原理图 14](#_Toc5182)

[3.2.4 OpenMv视觉系统原理图 14](#_Toc18959)

[3.3 仿真及调试结果 15](#_Toc6787)

[第四章 软件设计与编程 16](#_Toc1442)

[4.1 软件设计 16](#_Toc22595)

[4.2 程序/算法设计 17](#_Toc26648)

[4.2.1 算法流程框图 17](#_Toc5728)

[4.2.2 OpenMv视觉系统程序 18](#_Toc21348)

[4.2.3 循迹与抓取程序 19](#_Toc24883)

[4.2.4 STM32与OpenMv实现串口通信 24](#_Toc21476)

[4.3 仿真及测试结果 30](#_Toc10085)

[第五章 产品集成及功能测试 32](#_Toc27226)

[5.1 外观及结构设计 32](#_Toc26752)

[5.2 功能测试 33](#_Toc23215)

[5.2.1 颜色识别功能测试 33](#_Toc14637)

[5.2.2 循迹功能测试 33](#_Toc19909)

[5.2.3 抓取放下功能测试 34](#_Toc21046)

[参考文献 36](#_Toc18149)

[附录 37](#_Toc18630)

[附录一：OpenMv获取颜色信息代码 37](#_Toc12680)

[附录二：循迹抓取代码 39](#_Toc16768)

[附录三：OpenMv与STM32连接代码 43](#_Toc12946)

# 

# 第一章 研究背景

## 1.1 机械臂汽车自动加油装置的研究背景和意义

随着现代工业技术的不断发展和机械化水平的提高，机械设备和工业机器人的应用范围越来越广泛。其中，机械臂汽车自动加油装置是机械化生产中不可缺少的部件之一，广泛应用于汽车、航空、航天、机械制造等领域。

机械臂汽车自动加油装置的作用是实现对机器设备或车辆的油量补给，从而保证机器设备或车辆的正常运转。传统的加油方式通常需要人工进行操作，存在人工误差和劳动强度大等问题。而机械臂汽车自动加油装置可以自动完成加油过程，不仅提高了生产效率，而且还能减少人工干预和事故风险。

然而，机械臂汽车自动加油装置在应用中还存在一些技术难题和实际问题。例如，机械臂的自动控制和加油枪位置的自动调整等问题。此外，传统的油口识别方法主要依靠传感器或者视觉识别，效果不稳定且成本高昂。这些问题严重制约了机械臂汽车自动加油装置的应用和发展。

因此，如何实现对油口位置的高速、精准识别和机械臂的高速、精准控制成为了当前机械臂汽车自动加油装置研究的重要方向。近年来，越来越多的学者和企业开始关注这一领域，并提出了许多新的解决方案和技术。如今比较广泛采用的是一种基于深度学习和计算机视觉技术的机械臂汽车自动加油装置，该系统采用OpenCV和Python等技术实现对油口的识别和机械臂的控制，取得了较好的识别和控制效果。[1]

同时，国内外的许多企业和研究机构也在积极研究机械臂汽车自动加油装置相关技术，如德国的KUKA机器人、日本的Fanuc机器人、美国的ABB机器人等等，这些企业在机械臂汽车自动加油装置技术方面积累了大量的经验和技术优势，为该领域的发展提供了有力支持。

总之，机械臂汽车自动加油装置是现代机械化生产中的重要组成部分，具有很大的应用和发展前景。通过引入先进的技术和方法，如机器视觉和机器人控制技术等，可以实现对油口位置的高速、精准识别和机械臂的高速、精准控制，提高机械臂汽车自动加油装置的加油精度和速度，减少人工干预和事故风险，具有很高的实际应用价值和社会经济效益。

## 1.2 机械臂汽车自动加油装置存在的技术难题和实际问题

机械臂汽车自动加油装置是现代机械化生产中的重要组成部分，可以实现对油口位置的高速、精准识别和机械臂的高速、精准控制，提高机械臂汽车自动加油装置的加油精度和速度，减少人工干预和事故风险。但是，机械臂汽车自动加油装置在应用中存在一些技术难题和实际问题，主要表现在以下几个方面。

(1) 油口位置的自动识别问题

机械臂汽车自动加油装置需要对油口位置进行自动识别，以便进行机械臂的控制和加油枪的自动控制。传统的油口识别方法主要依靠传感器或者视觉识别，效果不稳定且成本高昂。因此，需要采用更先进的机器视觉技术，如深度学习、神经网络等方法，以提高油口位置的自动识别精度和稳定性。[2]一种基于机器视觉的汽车加油机械臂识别系统采用图像处理和识别技术，通过对油口位置和加油枪角度进行识别和控制，实现了对汽车加油机械臂的自动控制，取得了良好的识别效果。

(2) 机械臂的自动控制问题

机械臂汽车自动加油装置需要实现对加油枪的自动控制和油口位置的自动调整，这需要对机械臂的自动控制技术进行深入研究。传统的机械臂控制方法通常采用PID控制等基础控制方法，这些方法存在精度不高、响应速度慢、难以适应复杂环境等问题。因此，需要采用更先进的控制方法和技术，如模型预测控制、自适应控制等，以提高机械臂汽车自动加油装置的控制精度和速度。

(3) 加油枪的自动控制问题

机械臂汽车自动加油装置需要实现对加油枪的自动控制，以便精准地控制油口位置和油枪角度。加油枪的自动控制需要考虑多种因素，如控制精度、控制速度、控制精度稳定性等。因此，需要对加油枪的控制方法和技术进行深入研究，以实现对加油枪的自动控制。

(4) 成本和安全性问题

机械臂汽车自动加油装置的制造成本和安全性问题也是需要考虑的重要因素。机械臂汽车自动加油装置需要采用高性能的机械臂和加油枪等部件，成本较高。同时，加油过程中存在油渗漏和火灾爆炸等安全问题，需要采取相应的安全措施和控制方法，以保证加油过程的安全性和可靠性。陈海波等人针对机械臂汽车自动加油装置的安全问题进行了分析和探讨，提出了一种基于自适应控制和安全监测技术的机械臂汽车自动加油装置，该装置通过加油枪的自动控制和油渗漏监测，实现了对机械臂自动加油过程的安全控制。[3]

综上所述，机械臂汽车自动加油装置在应用中存在着许多技术难题和实际问题，需要采用先进的技术和方法进行解决。通过对这些问题进行深入研究和分析，可以不断提高机械臂汽车自动加油装置的加油精度和速度，减少人工干预和事故风险，具有很高的实际应用价值和社会经济效益。

## 1.3 OpenMv视觉系统及其特点和优势

OpenMv视觉系统是一种基于微控制器的计算机视觉系统，由OpenMv公司推出，具有良好的性能和应用优势。下面将介绍OpenMv视觉系统的特点和优势。

(1) 硬件特点

OpenMv视觉系统采用了STM32微控制器作为主控芯片，具有较高的运算速度和计算能力。同时，OpenMv视觉系统还采用了高性能的CMOS图像传感器和丰富的外设接口，可以实现对多种传感器和执行器的控制和读取。此外，OpenMv视觉系统还采用了丰富的开发板和扩展模块，可以实现对不同应用场景的定制和扩展。

(2) 软件特点

OpenMv视觉系统提供了基于Python语言的编程接口，使得用户可以通过简单的Python脚本实现对视觉传感器的读取和处理。同时，OpenMv视觉系统还提供了丰富的图像处理和计算机视觉算法库，如图像滤波、特征检测、目标识别等算法，使得用户可以快速实现各种视觉任务。[4]

OpenMv视觉系统具有以下几个优势：

(1) 低成本：OpenMv视觉系统采用了低成本的微控制器和图像传感器，成本较低，具有很高的性价比。

(2) 易用性：OpenMv视觉系统提供了简单易用的编程接口和图像处理算法库，使得用户可以快速实现各种视觉任务。

(3) 定制性：OpenMv视觉系统提供了丰富的开发板和扩展模块，可以实现对不同应用场景的定制和扩展。

(4) 灵活性：OpenMv视觉系统具有较高的灵活性，可以实现对多种传感器和执行器的控制和读取。

因此，OpenMv视觉系统具有较高的性能和应用优势，可以实现对多种视觉任务的快速处理和实现。可利用OpenMv视觉系统设计了一种基于移动机器人的色块识别系统，该系统可实现对不同颜色的色块进行精确识别。[5]

## 1.4 OpenMv视觉系统在机械臂汽车自动加油装置中的应用前景和研究意义

OpenMv视觉系统也是一种基于计算机视觉技术的新兴系统，具有良好的性能和应用优势。OpenMv视觉系统可以实现对图像的高效读取和处理，且具有较高的灵活性和易用性，可以满足机械臂汽车自动加油装置对精准定位和自动化控制的需求。

因此，将OpenMv视觉系统与机械臂汽车自动加油装置相结合，可以实现对加油作业的自动化控制和精准定位，从而提高加油效率、降低安全风险和提升用户体验。下面将从两个方面进行介绍。

(1) 自动化控制

OpenMv视觉系统可以实现对图像的高效读取和处理，可以识别机械臂汽车自动加油装置的目标位置和加油口方位，并通过控制机械臂的运动轨迹和运动速度，实现对加油作业的自动化控制。利用OpenMv视觉系统实现对加油作业的自动化控制，可以提高加油效率，减少加油时间，并减轻工作负担。

(2) 精准定位

OpenMv视觉系统具有较高的图像处理和计算机视觉算法库，可以实现对目标位置和方位的精准定位和识别。利用OpenMv视觉系统实现对机械臂汽车自动加油装置的精准定位，可以提高加油的精准度和准确性，并减少加油过程中可能出现的误操作。

综上所述，将OpenMv视觉系统与机械臂汽车自动加油装置相结合，可以实现对加油作业的自动化控制和精准定位，从而提高加油效率、降低安全风险和提升用户体验。通过利用OpenMv视觉系统和机械臂技术，实现一种基于机械臂汽车自动加油装置的加油站自动化管理系统，该系统能够实现对多个加油作业的自动化控制和精准定位，提高了加油作业的效率和精准度。[6]

# 第二章 项目规划

## 2.1 研究目标

项目组基于Tankbot智能搬运履带机器人和OpenMv视觉系统实现相关功能的自动化控制。

本组的最终实现目标为利用机械臂自动摘取加油枪并找到汽车完成加油工作。具体功能可分为以下三点。

1. 将OpenMv和STM32联合使用，利用视觉系统找到加油枪的位置，并摘取加油枪，运动到汽车油箱处完成加油工作。

2. 语音播报。

3. 车牌识别。

功能分解图如图2-1所示。

图示

描述已自动生成

图 2-1 功能分解图

## 2.2 实施计划

### 2.2.1 项目作品需求表达

(1) 需求列举

经过查找资料与实际分析，我们列举了以下几条用户对于机械臂汽车自动加油装置的不同方面的需求可能。

a) 识别加油枪的正确性。

b) 装置的快速性与准确性。

c) 可运送货物的重量范围尽可能大。

d) 装置自身的质量不能过大。

e) 装置制作所耗费的能量要尽可能少。

f) 完成加油任务的准确性和快速性。

(2) 组织需求的层级

货物需求：识别加油枪的准确性和抓取加油枪过程中的稳定性；

装置本身的功能性：运行速度快、循迹准确、承重大；

装置自身性质：智能搬运机器人自身质量轻便且能耗较少为佳。

(3) 建立需求的相对重要性

①　货物需求（☆☆☆☆☆）：识别加油枪的准确性和抓取加油枪过程中的稳定性（☆☆☆☆☆）。

②　装置本身的功能性（☆☆☆）：运行速度快（☆☆☆），循迹准确（☆☆☆），承重大（☆☆）。

③　装置自身性质（☆☆）：智能搬运机器人自身质量轻便且能耗较少（☆☆）。

注：☆多少代表对应需求重要性程度，☆☆☆☆☆较重要，☆☆较不重要

### 2.2.2 项目作品的计划实施

在进行项目规划之前，我们首先对于用户的具体需求进行了简单的调研。调研对象主要面向各个高校的师生以及的科研工作者，共收集到100份有效数据，汇总如表2-1所示。

表2-1 用户需求统计表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 需求 | 重要性 |  |  |  |
| 1 | 完成摘取加油枪任务 | 1 |  |  |  |
| 2 | 减小产品质量以及空间体积，方便较小空间运行 | 2 |  |  |  |
| 3 | 减震、抗震设计 | 2 |  |  |  |
| 4 | 科学设计自动控制系统 | 1 |  |  |  |
| 5 | 提高电池容量，增强续航 | 2 |  |  |  |
| 6 | 保证制动能力 | 3 |  |  |  |
| 7 | 降低能耗 | 1 |  |  |  |

基于上述对于项目作品需求的分析与调研，我们制定了如下项目计划。

1. 项目实施的第一阶段：对产品的建设进行初步的认识与学习，明确最终实现目标和具体的任务分配。

2. 项目实施的第二阶段：对产品的硬件进行组装和搭建。在组装和搭建的过程中，特别注意OpenMv视觉系统的安装和放置，视觉系统的放置决定了加油枪识别的准确性。其次还应该注意减小不必要的装置从而达到减轻质量的效果。

3. 项目实施的第三阶段：对产品进行软件编程。在程序中需要稳定运行并循迹、视觉系统识别加油枪的位置、完成摘取加油枪的动作、准确到达汽车位置等多个功能。

4. 项目实施的第四阶段：首先，我们将对相关硬件，即对传感器的灵敏度进行调整，使其达到符合运行轨道的目的。之后，对视觉系统和循迹模块之间的配合度进行调试，使其达到最佳配合度。其次我们将对我们的程序进行调试，力求采用轮巡控制与并行控制相结合的方式，实现整个过程的全自动控制。同时，对于设定的电机与舵机的控制方式与速度参数等，也将根据实际情况进行相应的调整。最后我们将通过实际的运行检验设计成果。

## 2.3 成员及分工

本组的组长是周艺梵，组员为刘长远和黄纯竹。经过组内会议讨论，拟定在接下来的项目中分工如下：

周艺梵：项目整体规划与进度管理；OpenMv与STM32实现串口通信；撰写课程报告。

刘长远：OpenMv视觉系统程序设计与组装；硬件设计与器件选型；产品硬件的组装和搭建。

黄纯竹：原理图设计；外观及结构设计；循迹与转向模块程序设计与功能测试；抓取模块程序设计与功能测试。

在整个项目研究与进行的前期阶段，各个成员分工明确，着力研究自己的分配目标，完成自己的任务模块。在项目进行的后期阶段，各个成员负责的部分进行相互配合和相互调试，达到实现最终研究目标的目的。

## 2.4 进度及安排

按本项目的实际规模和建设要求，大致的项目进度安排表见表2-2所示，按任务需求调研安排大致需要100个工作日，以3月1日为工作启动，基本将在6月10日左右结束。

表 2-2 项目进度安排表



按照本期项目的实际规模和建设要求，计划大致的项目进度甘特图如图2-2所示。

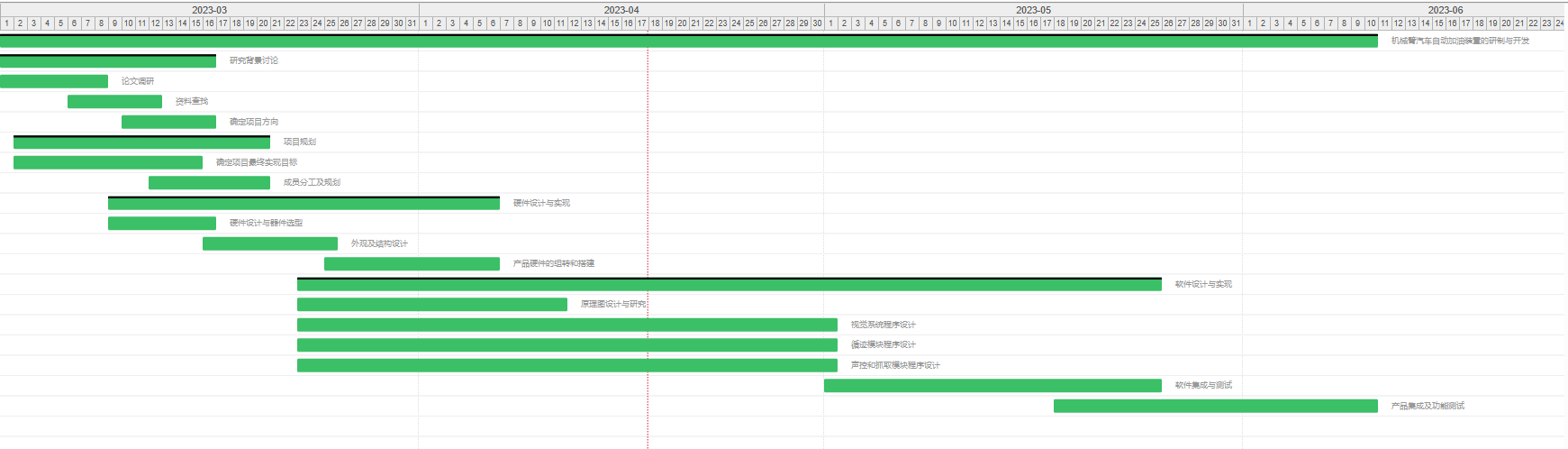


图2-2 项目进度甘特图

其中，软件设计及编程部分为该项目的核心内容和主要目标，因此本组将在此方面投入大量时间与精力。在项目进行过程中，本组还将定期进行组会，总结前一阶段出现的不足和问题，制定改进计划。同时对于后一阶段进行细化分工和适当调整项目目标计划，为其顺利开展做出展望和规划。

# 第三章 硬件设计与实现

## 3.1 硬件设计与器件选型

硬件的设计和器件的选型根据项目需要实现的功能主要分为两个方面：一是具有循迹功能的Tankbot智能搬运履带机器人的相关器件选型；二是具备识别加油枪功能的OpenMv视觉系统的相关器件选型。

### 3.1.1 Tankbot智能搬运履带机器人整体功能参数

硬件部分主要由厂商提供，组装后Tankbot智能搬运履带机器人的整体功能参数如表3-1所示。

表3-1 Tankbot规格参数

|  |  |  |
| --- | --- | --- |
| 名称 | Tankbot智能搬运履带机器人 | |
| 机体参数 | 产品尺寸（长\*宽\*高） | 230\*300\*570mm |
| 机体重量（未含电池） | 2.16kg |
| 机体材料 | 硬铝合金 |
| 供电部分 | 正常工作电压 | 6.9V-7.4V |
| 电池 | 7.4V锂电池 |
| 充电时间 | 3-4h |
| 满电续航能力 | 100分钟 |
| 主控部分 | 控制板 | 开源控制板 |
| 单片机 | STM32单片机模板 |
| 功能参数 | 承重能力 | 0.5kg |
| 极限车速 | 0.27m/s |
| PS2手柄控制 | 至少10m（空旷20m） |
| 手机app控制 | 至少10m（空旷15m） |
| 爪子举重能力 | 0.22kg |
| 爬坡能力 | 25° |
| 超声波探测距离 | 2cm-400cm（精度3mm） |
| 控制部分 | 见下文所述 | |

### 3.1.2 OpenMv视觉系统

OpenMv视觉系统以OpenMv摄像头为核心部件，实物图如图3-1所示。OpenMv摄像头是一款小巧，低功耗，低成本的电路板，能够帮助该项目实现并完成机器视觉的应用。OpenMv是通过高级语言Python脚本实现相关功能的，Python的高级数据结构能够使本项目组较为容易地在机器视觉算法中处理复杂的输出。同时，可以使用外部终端触发拍摄或者执行算法，将算法的结果控制IO引脚。

OpenMv摄像头具备多种特点，与本项目组的任务配合度很高，其特点包括：

1. 帧差分算法

OpenMv Cam具备帧差分算法，通过此算法可以查看场景中的运动情况。帧差分算法也可以将OpenMv视觉系统用于安全应用。

2. 颜色追踪

OpenMv在图像中同时检测多达16种颜色，并且每种颜色都可以有任意数量的不同的色块。OpenMv能够清晰表示出每个色块的位置、大小、中心和方向。使用颜色跟踪，OpenMv Cam可以进行编程，以实现跟踪太阳，线跟踪，目标跟踪等功能。

3. 标记跟踪

OpenMv Cam可以检测颜色组的颜色，而不是单独的颜色。可以通过在对象上放置2种或多种颜色的标签，来获取标签对象的内容。



图3-1 OpenMv摄像头实物图

### 3.1.3 总线舵机

总线舵机包括LX-824和LX-1501两种舵机类型。LX-824舵机是由幻尔科技研发的双轴三端口串行总线舵机。舵机精度高，虚位小，扭矩大(17KG.cm)，具有反馈功能，三端口接线更便捷。LX-1501舵机是由幻尔科技研发的单轴三端口串行总线舵机。舵机精度高，虚位小，扭矩大(17KG.cm)，具有反馈功能，三端口接线更便捷。两个舵机分别安装在机械臂的不同位置，舵机实物图如图3-2所示。

图片包含 游戏机

描述已自动生成

图3-2 LX-824舵机实物图

### 3.1.4 语音识别模块

语音识别模块可通过编程语言控制，使用方法简单，用户无需深入了解语音识别原理，通过IIC通信，即可识别语音，输出结果，可直接应用于多种人机交互领域的开发实验中。语音识别模块的实物图如图3-3所示。

电子仪器

中度可信度描述已自动生成

图3-3 语音识别模块实物图

语音识别模块的规格参数表如表3-2所示。

表 3-2 语音识别模块规格参数表

|  |  |
| --- | --- |
| 工作电压 | 5V |
| 长度 | 48mm |
| 宽度 | 24mm |
| 通讯方式 | IIC |

### 3.1.5 4路循线传感器

4路循线传感器的实物图如图3-4所示。该传感器利用红外原理，能提前探测到大角度转弯的黑线，可以完成高难度巡线，且I2C输出不占用IO口。



图3-4 4路循线传感器实物图

4路循线传感器模块的规格参数表如表3-3所示。

表3-3 4路循线传感器规格参数表

|  |  |
| --- | --- |
| 工作电压 | 5V |
| 工作电流 | 10mA-50mA |
| 工作温度 | 0℃-50℃ |
| 检测距离 | 5mm-15mm（可调） |
| 安装孔径 | M4铜柱 |
| 长度 | 72.5mm |
| 宽度 | 44mm |
| 输出接口 | 4PIN接口 |
| 输出信号 | I2C总线 |

## 3.2 原理图设计

### 3.2.1 4路循线传感器原理图

4路循线传感器的电路原理图如图3-5所示。其中各路传感器灯的亮灭均采用二极管控制。

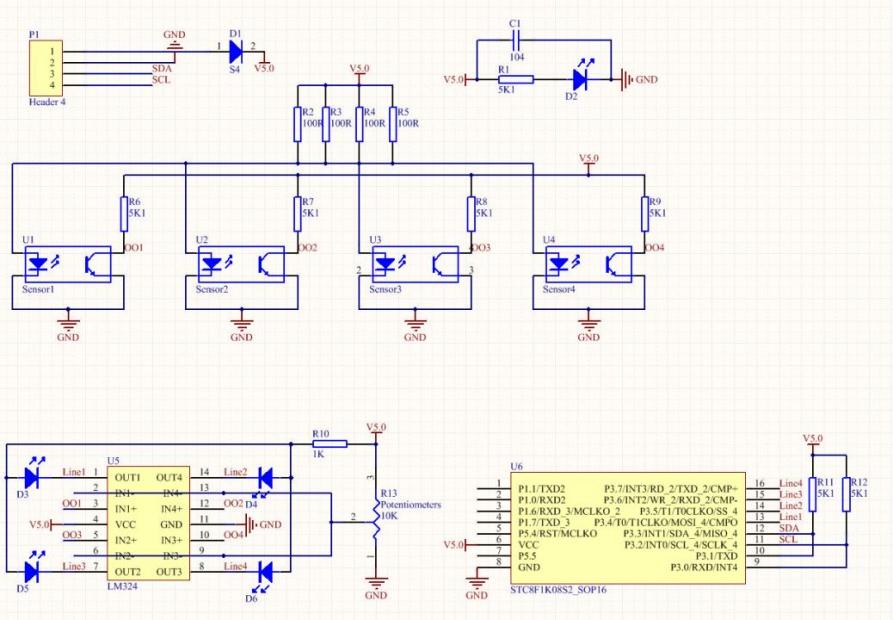


图3-5 4路循线传感器原理图

### 3.2.2 STM32单片机原理图

STM32单片机的电路原理图如图3-6所示。

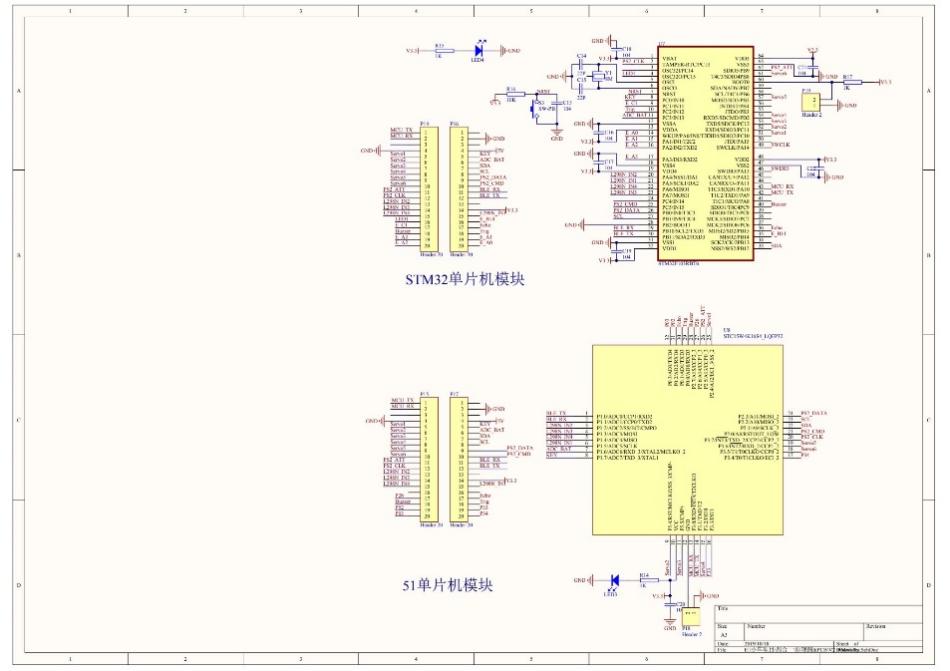


图3-6 STM32单片机原理图

### 3.2.3 小车主控制器原理图

小车主控制器的电路原理图如图3-7所示。

图示, 示意图

描述已自动生成

图3-7 小车主控制器原理图

### 3.2.4 OpenMv视觉系统原理图

OpenMv视觉系统的电路原理图如图3-8所示。

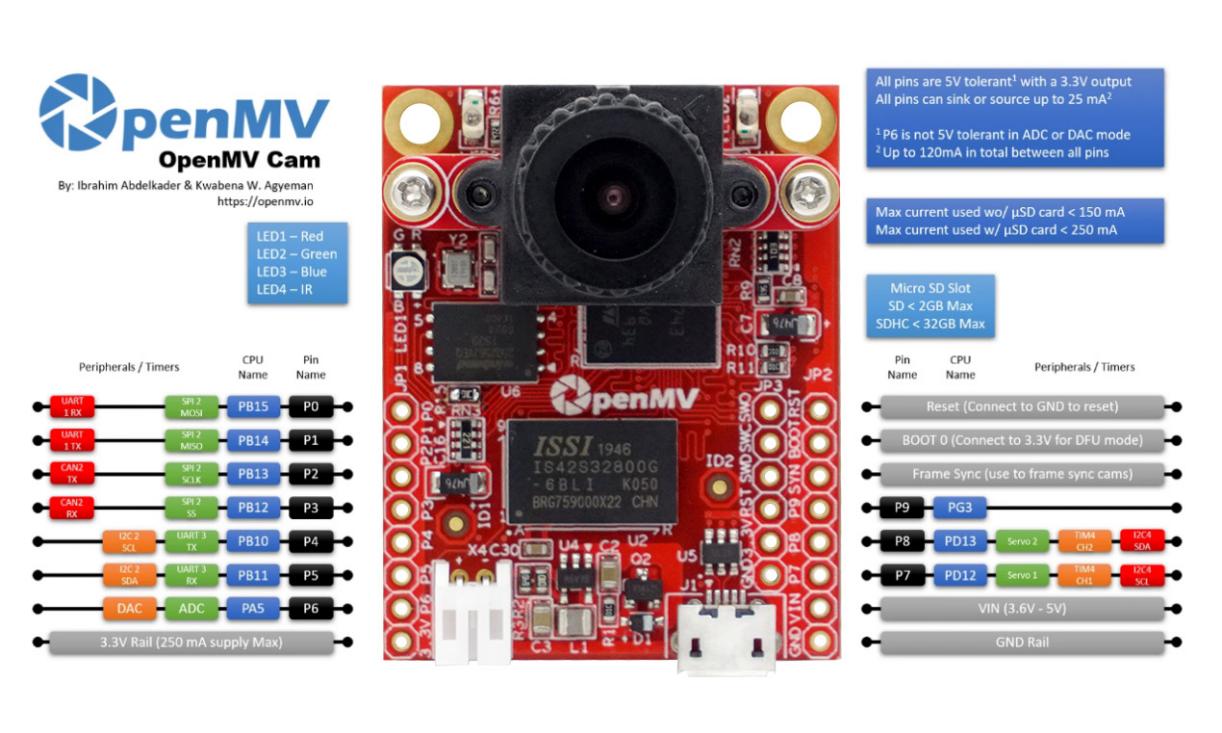


图3-8 OpenMv视觉系统原理图

## 3.3 仿真及调试结果

综合以上器件的选型和参数的对比，项目组对产品硬件完成了组装与搭建，并且进行了详细的调试过程。调试过程如下：

首先，对于4路循线传感器进行调试。根据传感器原理，当传感器检测到暗光时，应反馈灯灭，检测到亮光时，应反馈灯亮。根据循线传感器放置的位置，调节灵敏度旋钮，使得每路传感器在检测到黑线时，反馈灯灭；在未检测到黑线时，反馈灯亮。

其次，对视觉系统进行调试。项目的目标任务是成功识别加油枪并准确运行至加油枪位置，视觉系统检测到加油枪并能够做出相应的反馈为调试成功。

最后，对机械臂舵机进行调节，使其能够平稳地抓起并放下加油枪。

# 第四章 软件设计与编程

## 4.1 软件设计

本项目需要实现的最终研究目标为利用机械臂自动摘取加油枪并找到汽车完成加油工作。在整个过程中，我们需要以Tankbot智能搬运履带机器人为主体模拟该过程的具体操作步骤。在功能上，设计了自动循迹、颜色识别、视觉抓取这三大主要功能。在软件设计方面，STM32以C语言为主体，搭载了少量的汇编语言，支持模块化编程，OpenMv以Python语言为主体，搭载使用Micropython语言编写程序，可以在微控制器和嵌入式系统上运行，我们通过此来实现项目的具体功能。

软件设计功能图如图4-1所示。

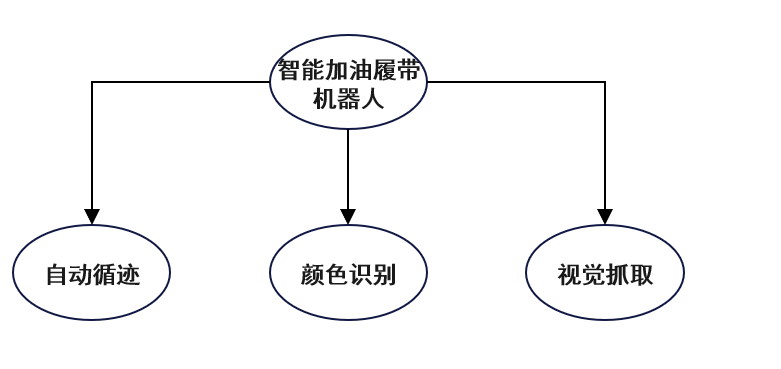


图4-1 软件设计功能图

## 4.2 程序/算法设计

### 4.2.1 算法流程框图

根据目标功能，设计算法流程框图如图4-2所示。图中，由三个闭环控制构成整个系统的全自动化，从而达到预期设定的目标。

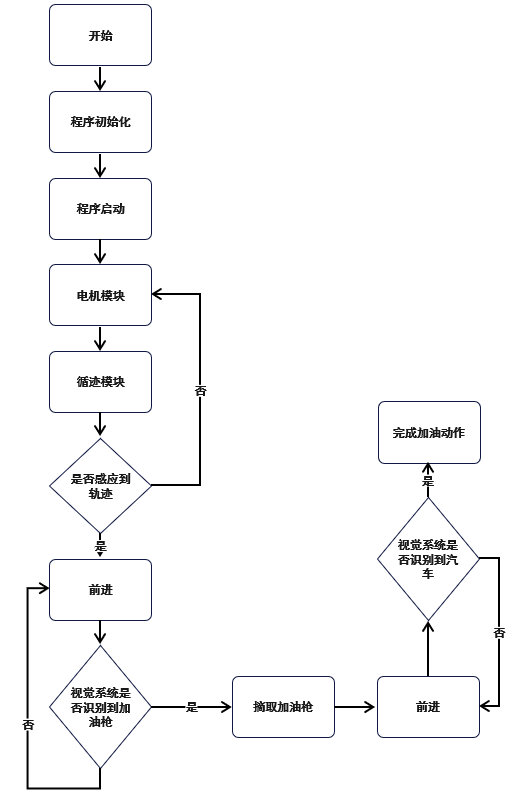


图4-2 算法设计流程图

### 4.2.2 OpenMv视觉系统程序

OpenMv视觉系统进行颜色的识别和物体具体坐标的获得。该视觉系统能够实现以下功能：

1、能实现实时对任意颜色和形状组合的自动捕捉，如图4-3所示。

2、能实时准确的定位到所需目标的具体坐标。

文本

描述已自动生成

图4-3 物块自动捕捉和分析图

OpenMv代码如下所示：

import sensor, image, time,math,pyb

from pyb import UART,LED

import json

import ustruct

sensor.reset()

sensor.set\_pixformat(sensor.RGB565)

sensor.set\_framesize(sensor.QVGA)

sensor.skip\_frames(time = 2000)

sensor.set\_auto\_gain(False) # must be turned off for color tracking

sensor.set\_auto\_whitebal(False) # must be turned off for color tracking

red\_threshold\_01=(10, 100, 127, 32, -43, 67)

yellow\_threshold = (70, 100, -15, 50, -20, 40)

clock = time.clock()

uart = UART(3,115200) #定义串口3变量

uart.init(115200, bits=8, parity=None, stop=1) # init with given parameters

def sending\_data(cx,cy):

global uart;

#frame=[0x2C,18,cx%0xff,int(cx/0xff),cy%0xff,int(cy/0xff),0x5B];

#data = bytearray(frame)

data = ustruct.pack("<bbhhb", #格式为俩个字符俩个短整型(2字节)

0x2C, #帧头1

0x12, #帧头2

int(cx), # up sample by 4 #数据1

int(cy), # up sample by 4 #数据2

0x5B)

uart.write(data); #必须要传入一个字节数组

while(True):

clock.tick()

img = sensor.snapshot()

cx=0;cy=0;

red\_blobs = img.find\_blobs([red\_threshold\_01], pixels\_threshold=200, area\_threshold=200)

yellow\_blobs = img.find\_blobs([yellow\_threshold], pixels\_threshold=200, area\_threshold=200)

if red\_blobs:

cx=1

print(cx,cy)

sending\_data(cx,cy)

elif yellow\_blobs:

cy=2

print(cx,cy)

sending\_data(cx,cy)

### 4.2.3 循迹与抓取程序

该部分程序实现智能履带机器人的循迹与抓取功能。通过从OpenMv中获得的颜色信息变量OpenMv\_X和OpenMv\_Y设置抓取放下的布尔值，从而判断是否执行抓取或者放下动作。若通过OpenMv识别到加油枪，则将抓取布尔值赋为1，执行抓取动作抓取加油枪；若未识别到加油枪，则进入循迹模块，小车将继续沿黑线行进；到达终点时，停止运动同时执行放下动作。在具体的抓取函数中，抓取与放下被定义为grab和put两个函数，在每个函数执行完之后，grab\_s或put\_s这两个布尔值被赋值为FALSE，防止重复执行。

循迹抓取算法流程图如图4-4所示。

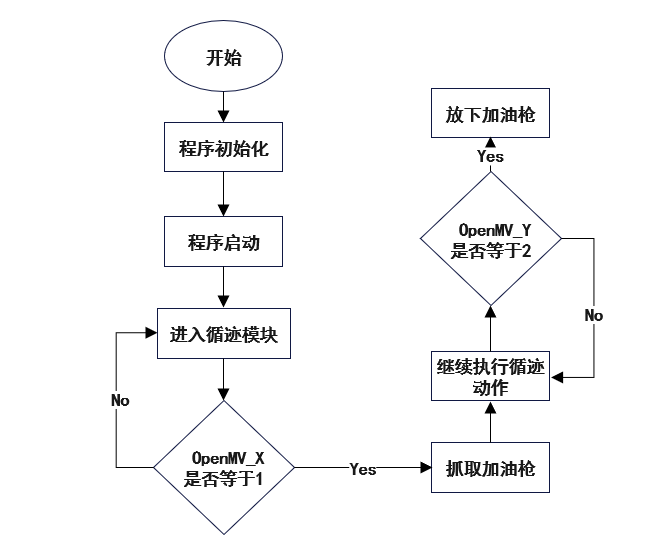


图4-4 循迹抓取算法流程图

循迹代码如下所示：

void Control(void)

{

u8 val;

if(grab\_s) //判断是否执行抓取动作组

{

Grab(); //抓取函数

DelayMs(50000);

return;

}

else if(put\_s) //判断是否执行放下动作组

{

put();//放下函数

return;

}

if( OpenMv\_X ==1 ) //识别物体，若识别则执行抓取动作

{

grab\_s=TRUE;

}

else

{

IIC\_start(); //起始信号

IIC\_send\_byte(0xF0); //发送写指令

IIC\_wait\_ack(); //应答

IIC\_send\_byte(0x01); //发送寄存器地址

IIC\_wait\_ack(); //应答

IIC\_start(); //重新启动

IIC\_send\_byte(0xF1); //发送读指令

IIC\_wait\_ack(); //应答

val = IIC\_read\_byte(0); //读取一个字节

IIC\_stop(); //停止

if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 0) & (((val>>2) & 0x01) == 0) & (((val>>3) & 0x01) == 1))

MotorControl(80, -80);

else if (((val & 0x01) == 1) & (((val>>1) & 0x01) == 0) & (((val>>2) & 0x01) == 0) & (((val>>3) & 0x01) == 0))

MotorControl(-80, 80);

else if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 0) & (((val>>2) & 0x01) == 1) & (((val>>3) & 0x01) == 0))

MotorControl(80, 0);

else if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 1) & (((val>>2) & 0x01) == 0) & (((val>>3) & 0x01) == 0))

MotorControl(0, 80);

else if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 0) & (((val>>2) & 0x01) == 0) & (((val>>3) & 0x01) == 0))

MotorControl(0, 0);

else if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 1) & (((val>>2) & 0x01) == 1) & (((val>>3) & 0x01) == 0))

MotorControl(80, 80);

else if (((val & 0x01) == 1) & (((val>>1) & 0x01) == 1) & (((val>>2) & 0x01) == 0) & (((val>>3) & 0x01) == 0))

MotorControl(-80, 80);

else if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 0) & (((val>>2) & 0x01) == 1) & (((val>>3) & 0x01) == 1))

MotorControl(80, -80);

else if (((val & 0x01) == 1) & (((val>>1) & 0x01) == 1) & (((val>>2) & 0x01) == 1) & (((val>>3) & 0x01) == 0))

MotorControl(-80, 80);

else if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 1) & (((val>>2) & 0x01) == 1) & (((val>>3) & 0x01) == 1))

MotorControl(80, -80);

else

{MotorControl(0, 0);//到达终点，执行放下函数

put\_s=TRUE;

}

}

抓取代码如下所示：

void Grab(void)

{

BusServoCtrl(1,SERVO\_MOVE\_TIME\_WRITE,300,1000);

BusServoCtrl(2,SERVO\_MOVE\_TIME\_WRITE,500,1000);

BusServoCtrl(3,SERVO\_MOVE\_TIME\_WRITE,350,1000);

BusServoCtrl(4,SERVO\_MOVE\_TIME\_WRITE,100,1000);

BusServoCtrl(5,SERVO\_MOVE\_TIME\_WRITE,300,1000);

BusServoCtrl(6,SERVO\_MOVE\_TIME\_WRITE,500,1000);

DelayMs(1000);

BusServoCtrl(1,SERVO\_MOVE\_TIME\_WRITE,140,1000);

BusServoCtrl(2,SERVO\_MOVE\_TIME\_WRITE,500,1000);

BusServoCtrl(3,SERVO\_MOVE\_TIME\_WRITE,810,1000);

BusServoCtrl(4,SERVO\_MOVE\_TIME\_WRITE,100,1000);

BusServoCtrl(5,SERVO\_MOVE\_TIME\_WRITE,765,1000);

DelayMs(1000);

DelayMs(1000);

BusServoCtrl(1,SERVO\_MOVE\_TIME\_WRITE,600,1000);

DelayMs(1000);

BusServoCtrl(3,SERVO\_MOVE\_TIME\_WRITE,350,1000);

BusServoCtrl(5,SERVO\_MOVE\_TIME\_WRITE,300,1000);

DelayMs(1000);

grab\_s = FALSE;

}

void put(void)

{

BusServoCtrl(6,SERVO\_MOVE\_TIME\_WRITE,130,1000);

DelayMs(1000);

BusServoCtrl(3,SERVO\_MOVE\_TIME\_WRITE,810,1000);

BusServoCtrl(5,SERVO\_MOVE\_TIME\_WRITE,765,1000);

DelayMs(1000);

DelayMs(1000);

BusServoCtrl(1,SERVO\_MOVE\_TIME\_WRITE,0,1000);

DelayMs(1000);

BusServoCtrl(3,SERVO\_MOVE\_TIME\_WRITE,350,1000);

BusServoCtrl(5,SERVO\_MOVE\_TIME\_WRITE,300,1000);

DelayMs(1000);

BusServoCtrl(1,SERVO\_MOVE\_TIME\_WRITE,300,1000);

BusServoCtrl(6,SERVO\_MOVE\_TIME\_WRITE,500,1000);

DelayMs(1000);

put\_s=FALSE;

}

### 4.2.4 STM32与OpenMv实现串口通信

STM32与OpenMv实现连接首先需要通过OpenMv对于加油枪的识别，实现用OpenMv识别出我们想要的颜色，其次再通过串口通信，将获取的颜色坐标通过串口发送给STM32，最后在STM32中根据OpenMv传递过去的内容，进行关键信息识别。OpenMv工作原理图如图4-5所示。

图示

描述已自动生成

图4-5 OpenMv工作原理图

OpenMv具体工作原理和代码实现如下所示。

**(1) 硬件的通信连接**

由图3-8OpenMv视觉系统原理图可知OpenMv4只有1个串口——USART。这里使用P4(USART3\_TX)与P5(USART3\_RX)脚与STM32的USART交叉联接，并且使得双方共地，如图4-6所示。

图片包含 折线图

描述已自动生成

图4-6 引脚连接图

通用同步异步收发器(USART)提供了一种灵活的方法与使用工业标准NRZ异步串行数据格式的外部设备之间进行全双工数据交换。接口通过三个引脚与其他设备连接在一起，USART串口通信系统流程图如图4-7所示。任何USART双向通信至少需要两个脚：接收数据输入(RX)和发送数据输出(TX)。

RX：接收数据串行输。通过采样技术来区别数据和噪音，从而恢复数据。

TX：发送数据输出。当发送器被禁止时，输出引脚恢复到它的I/O端口配置。当发送器被激活，并且不发送数据时，TX引脚处于高电平。在单线和智能卡模式里，此I/O口被同时用于数据的发送和接收。

图示, 示意图

描述已自动生成

图4-7 USART串口通信系统流程图

**(2) 项目设计中的注意事项**

1、OpenMv对于串口通信数据的传输是一个字节去传输，在传递过程中所要传递的内容较多，可以通过for循环的多次遍历去传输信息或者通过json字符串进行内容的传递。

2、在进行OpenMv与STM32的串口通信时，一方面要注意OpenMv引脚要选择3号并且保持波特率与STM32端口一致，此外还需要在信息传递时设计帧头和帧尾，来使得STM32端对于一组数据进行区分，使得结果准确，提高传输以及工作效率

**(3) OpenMv代码实现**

根据OpenMv对于颜色阈值进行识别，由于python语言的具有丰富的函数库，使得对于代码的实现较为简单，其中较为核心关键的部分在于需考虑对于镜头畸变、采样的间隔、图像分辨率、色块模式以及信息的处理速度等问题综合进行考量优化。OpenMv功能是进行色块识别并把颜色信息转换为整型变量传递给STM32，其中对数据的打包格式，用到了ustruct.pack这个函数。

OpenMv传输部分代码如下所示：

def sending\_data(cx,cy):

global uart;

#frame=[0x2C,18,cx%0xff,int(cx/0xff),cy%0xff,int(cy/0xff),0x5B];

#data = bytearray(frame)

data = ustruct.pack("<bbhhb", #格式为俩个字符俩个短整型(2字节)

0x2C, #帧头1

0x12, #帧头2

int(cx), # up sample by 4 #数据1

int(cy), # up sample by 4 #数据2

0x5B)

uart.write(data); #必须要传入一个字节数组

**(4) STM32代码实现**

打开串口的驱动，我们要打开的串口驱动为USART，同时设置OpenMv\_Receive\_Data函数对接收到的数据进行处理。

在STM32中接收数据采取接收中断的方法来实现，在STM32中，选择接收中断方式是因为它可以提高代码的响应速度和实时性。使用中断方式可以将处理器的时间分配给其他操作，而不是一直等待数据到达，从而降低了处理器的负载并提高了系统效率。

具体来说，在中断方式下，当有数据到达时，STM32会立即触发一个接收中断，这将引起一个中断服务程序(ISR)的执行。中断服务程序能够及时处理接收到的数据，而不会阻塞主程序，因此能够更好地满足实时性要求。总之，使用中断方式可以提高STM32的响应速度和实时性，同时也能够降低处理器的负载，提高系统效率。

STM32接收代码如下所示：

void uart\_init(u32 bound){

//GPIO端口设置

GPIO\_InitTypeDef GPIO\_InitStructure;

USART\_InitTypeDef USART\_InitStructure;

NVIC\_InitTypeDef NVIC\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_USART1|RCC\_APB2Periph\_GPIOA, ENABLE); //使能USART1，GPIOA时钟

//USART1\_TX GPIOA.9

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_9; //PA.9

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF\_PP; //复用推挽输出

GPIO\_Init(GPIOA, &GPIO\_InitStructure);//初始化GPIOA.9

//USART1\_RX GPIOA.10初始化

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_10;//PA10

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IN\_FLOATING;//浮空输入

GPIO\_Init(GPIOA, &GPIO\_InitStructure);//初始化GPIOA.10

//Usart1 NVIC 配置

NVIC\_InitStructure.NVIC\_IRQChannel = USART1\_IRQn;

NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority=3 ;//抢占优先级3

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 3; //子优先级3

NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE; //IRQ通道使能

NVIC\_Init(&NVIC\_InitStructure); //根据指定的参数初始化VIC寄存器

//USART 初始化设置

USART\_InitStructure.USART\_BaudRate = bound;//串口波特率

USART\_InitStructure.USART\_WordLength = USART\_WordLength\_8b;//字长为8位数据格式

USART\_InitStructure.USART\_StopBits = USART\_StopBits\_1;//一个停止位

USART\_InitStructure.USART\_Parity = USART\_Parity\_No;//无奇偶校验位

USART\_InitStructure.USART\_HardwareFlowControl = USART\_HardwareFlowControl\_None;//无硬件数据流控制

USART\_InitStructure.USART\_Mode = USART\_Mode\_Rx | USART\_Mode\_Tx; //收发模式

USART\_Init(USART1, &USART\_InitStructure); //初始化串口1

USART\_ITConfig(USART1, USART\_IT\_RXNE, ENABLE);//开启串口接受中断

USART\_Cmd(USART1, ENABLE); //使能串口1

}

中断函数：

if(USART\_GetITStatus(USART1, USART\_IT\_RXNE) != RESET) //接收中断(接收到的数据必须是0x0d 0x0a结尾)

{

Res =USART\_ReceiveData(USART1); //读取接收到的数据

if((USART\_RX\_STA&0x8000)==0)//接收未完成

{

if(USART\_RX\_STA&0x4000)//接收到了0x0d

{

if(Res!=0x0a)USART\_RX\_STA=0;//接收错误,重新开始

else USART\_RX\_STA|=0x8000; //接收完成了

}

else //还没收到0X0D

{

if(Res==0x0d)USART\_RX\_STA|=0x4000;

else

{

USART\_RX\_BUF[USART\_RX\_STA&0X3FFF]=Res ;

USART\_RX\_STA++;

if(USART\_RX\_STA>(USART\_REC\_LEN-1))

USART\_RX\_STA=0;//接收数据错误,重新开始接收

}

}

}

}

void OpenMv\_Receive\_Data(int16\_t data) //接收OpenMv传过来的数据

{

static u8 state = 0;

if(state==0&&data==0xb3)

{

state=1;

OpenMv[0]=data;

}

else if(state==1&&data==0xb3)

{

state=2;

OpenMv[1]=data;

}

else if(state==2)

{

state=3;

OpenMv[2]=data;

}

else if(state==3)

{

state = 4;

OpenMv[3]=data;

}

else if(state==4) //检测是否接受到结束标志

{

if(data == 0x5B)

{

state = 0;

OpenMv[4]=data;

OpenMv\_Data();

}

else if(data != 0x5B)

{

state = 0;

for(i=0;i<5;i++)

{

OpenMv[i]=0x00;

}

}

}

else

{

state = 0;

for(i=0;i<5;i++)

{

OpenMv[i]=0x00;

}

}

}

void OpenMv\_Data(void)

{

OpenMv\_X=OpenMv[2];//OpenMv加油枪反馈变量

OpenMv\_Y=OpenMv[3];//OpenMv车反馈变量

}

## 4.3 仿真及测试结果

在软件设计与编程及其仿真测试过程中，由于分工不同，各个成员负责的部分相互独立，需要一个很好的枢纽将其连接，在代码调试过程中仍然出现了许多意料之外的状况，但经过反复调整与试验，最终实验了所有预设目标，具体问题与调整措施如下：

1、在颜色识别阶段，最开始存在缺陷导致OpenMv会捕捉所有红色圆形，对算法进行进一步的优化，可以准确的定位到所需目标，并且准确获取所需加油枪的具体坐标位置。

2、在循迹阶段，总是会出现4路巡线传感器失灵的时候。我们通过调整传感器的灵敏度，并且适当降低4路巡线传感器的高度，使得其能更加精准地判定黑线，最终使我们的循迹准确又快速。

3、在循迹的转弯过程中，会出现履带机器人无法识别弯道而持续直行的问题。我们通过反复调整转弯速度的参数，设定不同参数测试使其到达一个理想的转弯速度，最终我们确定参数实现理想直行和转弯循迹。

4、在抓取阶段，我们的目标是抓取加油枪，并在指定位置将其放下。而原有的函数是抓取并放下物体一连贯的动作，我们反复调整每个舵机参数，最终确定舵机的数字编号，通过对函数的拆分和修改实现抓取加油枪和放下加油枪的分步动作。

5、实现OpenMv和STM32的连接是一个关键步骤，在连接过程中出现履带机器人无法执行动作的情况。我们通过对OpenMv发送数据和STM32接收数据这两个部分进行检验，在OpenMv中使用串口终端查看颜色检测信息，发现输出值“1 0”和“0 2”符合预期标准，说明OpenMv发送数据没有问题。这时，我们考虑到STM32和OpenMv的连接紧密问题，选择先加固连接二者连接方式，在不能焊接的情况下使得其尽量接触紧密，最终在串口助手中收到来自OpenMv的数据，实现了他们之间的连接。

# 第五章 产品集成及功能测试

## 5.1 外观及结构设计

Tankbot是一款基于STM32的智能搬运履带机器人。它的机体搭载了一个6自由度的机械臂，机械臂关节采用智能总线舵机来控制。

Tankbot的机体搭载了超声波、加速度、四路巡线、声音传感器等模块。通过简单的机械拼搭，不仅能实现手机、手柄、体感手套等多种控制，还可以完成巡线、抓取、侧翻恢复等趣味玩法。Tankbot外观图如图5-1所示。

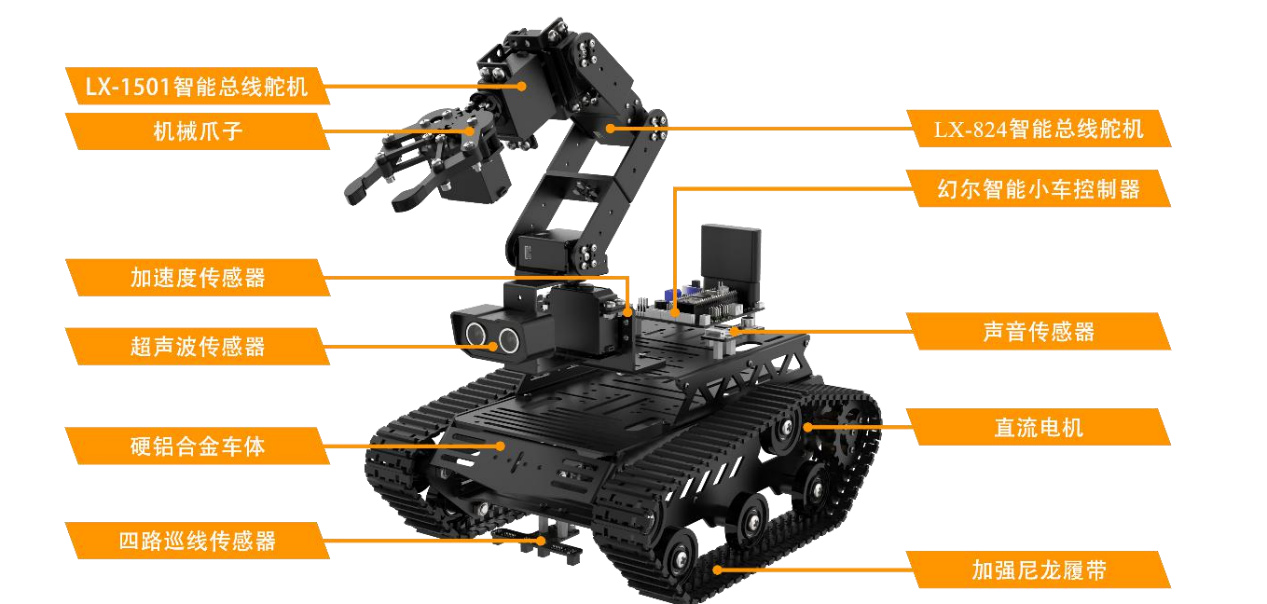


图5-1 Tankbot外观图

为了使OpenMv视觉系统更好的识别加油枪并且获取颜色信息，OpenMv在智能履带机器人上安装的位置尤为重要。通过调整OpenMv在履带机器人车体的位置，比较不同位置时获取颜色信息的准确性和获取加油枪位置的精确度，最终我们确定将OpenMv安装在硬铝合金车体的正前方以获取最佳加油枪颜色和坐标信息。

## 5.2 功能测试

### 5.2.1 颜色识别功能测试

通过在OpenMv视野中央放置不同颜色的物体，观察串行终端中输出的颜色信息，从而获取我们需要的目标物品——加油枪。串行终端显示图如图5-2所示。

图文本

描述已自动生成5-2 串行终端显示图

### 5.2.2 循迹功能测试

在测试阶段，我们解决了履带机器人转弯速度过快、直线行驶不灵敏等问题。

通过调节直行前进速度，避免直角转弯速度过快，来不及转弯冲出，经过调试转速从而使转速降低，使得履带机器人顺利转弯。

调节转弯逻辑，使其能顺利通过直角弯，从只使用中间两个传感器来控制循迹，到使用外面两个传感器来控制循迹。

使用外面两个传感器循迹，导致直线行驶调整不敏感，难以直线行走，速度慢，改用同时使用四个传感器来控制循迹。中间两个传感器主要作用在直线行走阶段，外面两个传感器主要作用在直角转弯阶段，能同时使用这两种策略的原因是，直线循迹阶段，外面两个传感器始终保持未检测到黑线状态，避免了两种逻辑产生冲突。

在判断停止阶段，判断逻辑中，黑线停止的条件达成的同时，一定会达成左右转条件，在修改循迹过程中，误把左右转判断放在停止判断之前，导致永远无法停止，后经过多次调试和代码的重新阅读，发现问题，通过修改条件判断的先后顺序解决问题。循迹测试如图5-3所示。

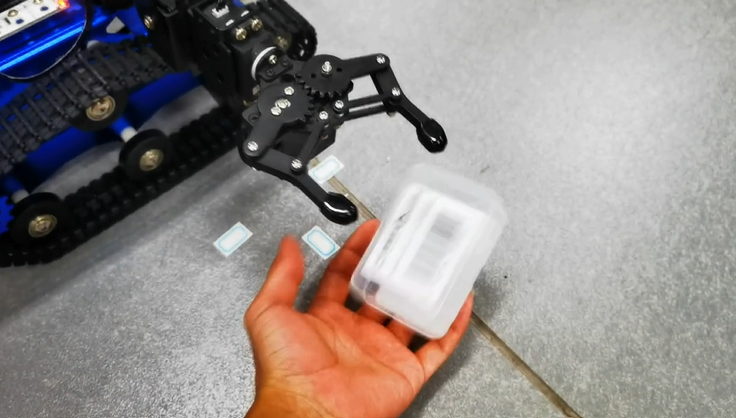
图5-3 循迹测试

### 5.2.3 抓取放下功能测试

图片包含 室内, 桌子, 盒子, 男人

描述已自动生成为了实现准确抓取加油枪，通过各种颜色信息获得难易程度的对比和抓取加油枪的难易程度，我们选取红色作为抓取对象的颜色，图5-4为抓取加油枪测试过程，图5-5为放下加油枪测试过程。

图5-4 抓取加油枪测试

图5-5放下加油枪测试

# 参考文献

[1] 王力, 陈峰, 罗海平. 基于深度学习的机械臂汽车自动加油装置研究[J]. 机械设计与制造, 2018(4): 71-75.

[2] 李德磊, 陈晋涛, 李新,等. 基于机器视觉的汽车加油机械臂识别系统设计[J]. 控制与自动化, 2019, 46(7): 129-134.

[3] 陈海波, 申志勇, 陈长安,等. 基于自适应控制和安全监测的机械臂汽车自动加油装置[J]. 机器人, 2018, 40(3): 344-348.

[4]付书添,查雪红,许超.基于OpenMv视觉系统的智能送药小车[J].工业控制计算机,2022,35(07):6-9.

[5]李虹,张宇晨,苗旭焘.OpenMv智能物料搬运方向的开发[J].电子技术与软件工程,2022(03):107-112.

[6] 李伟, 杨帅, 张海军,等. 基于机械臂技术的加油站自动化管理系统的设计与实现[J]. 河北科技师范学院学报, 2020, 34(3): 56-60.

# 附录

## 附录一：OpenMv获取颜色信息代码

import sensor, image, time,math,pyb

from pyb import UART,LED

import json

import ustruct

sensor.reset()

sensor.set\_pixformat(sensor.RGB565)

sensor.set\_framesize(sensor.QVGA)

sensor.skip\_frames(time = 2000)

sensor.set\_auto\_gain(False) # must be turned off for color tracking

sensor.set\_auto\_whitebal(False) # must be turned off for color tracking

red\_threshold\_01=(10, 100, 127, 32, -43, 67)

yellow\_threshold = (70, 100, -15, 50, -20, 40)

clock = time.clock()

uart = UART(3,115200) #定义串口3变量

uart.init(115200, bits=8, parity=None, stop=1) # init with given parameters

def sending\_data(cx,cy):

global uart;

#frame=[0x2C,18,cx%0xff,int(cx/0xff),cy%0xff,int(cy/0xff),0x5B];

#data = bytearray(frame)

data = ustruct.pack("<bbhhb", #格式为俩个字符俩个短整型(2字节)

0x2C, #帧头1

0x12, #帧头2

int(cx), # up sample by 4 #数据1

int(cy), # up sample by 4 #数据2

0x5B)

uart.write(data); #必须要传入一个字节数组

while(True):

clock.tick()

img = sensor.snapshot()

cx=0;cy=0;

red\_blobs = img.find\_blobs([red\_threshold\_01], pixels\_threshold=200, area\_threshold=200)

yellow\_blobs = img.find\_blobs([yellow\_threshold], pixels\_threshold=200, area\_threshold=200)

if red\_blobs:

cx=1

print(cx,cy)

sending\_data(cx,cy)

elif yellow\_blobs:

cy=2

print(cx,cy)

sending\_data(cx,cy)

## 附录二：循迹抓取代码

循迹代码如下所示：

void Control(void)

{

u8 val;

if(grab\_s) //判断是否执行抓取动作组

{

Grab(); //抓取函数

DelayMs(50000);

return;

}

else if(put\_s) //判断是否执行放下动作组

{

put();//放下函数

return;

}

if( OpenMv\_X ==1 ) //识别物体，若识别则执行抓取动作

{

grab\_s=TRUE;

}

else

{

IIC\_start(); //起始信号

IIC\_send\_byte(0xF0); //发送写指令

IIC\_wait\_ack(); //应答

IIC\_send\_byte(0x01); //发送寄存器地址

IIC\_wait\_ack(); //应答

IIC\_start(); //重新启动

IIC\_send\_byte(0xF1); //发送读指令

IIC\_wait\_ack(); //应答

val = IIC\_read\_byte(0); //读取一个字节

IIC\_stop(); //停止

if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 0) & (((val>>2) & 0x01) == 0) & (((val>>3) & 0x01) == 1))

MotorControl(80, -80);

else if (((val & 0x01) == 1) & (((val>>1) & 0x01) == 0) & (((val>>2) & 0x01) == 0) & (((val>>3) & 0x01) == 0))

MotorControl(-80, 80);

else if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 0) & (((val>>2) & 0x01) == 1) & (((val>>3) & 0x01) == 0))

MotorControl(80, 0);

else if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 1) & (((val>>2) & 0x01) == 0) & (((val>>3) & 0x01) == 0))

MotorControl(0, 80);

else if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 0) & (((val>>2) & 0x01) == 0) & (((val>>3) & 0x01) == 0))

MotorControl(0, 0);

else if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 1) & (((val>>2) & 0x01) == 1) & (((val>>3) & 0x01) == 0))

MotorControl(80, 80);

else if (((val & 0x01) == 1) & (((val>>1) & 0x01) == 1) & (((val>>2) & 0x01) == 0) & (((val>>3) & 0x01) == 0))

MotorControl(-80, 80);

else if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 0) & (((val>>2) & 0x01) == 1) & (((val>>3) & 0x01) == 1))

MotorControl(80, -80);

else if (((val & 0x01) == 1) & (((val>>1) & 0x01) == 1) & (((val>>2) & 0x01) == 1) & (((val>>3) & 0x01) == 0))

MotorControl(-80, 80);

else if (((val & 0x01) == 0) & (((val>>1) & 0x01) == 1) & (((val>>2) & 0x01) == 1) & (((val>>3) & 0x01) == 1))

MotorControl(80, -80);

else

{MotorControl(0, 0);//到达终点，执行放下函数

put\_s=TRUE;

}

}

抓取代码如下所示：

void Grab(void)

{

BusServoCtrl(1,SERVO\_MOVE\_TIME\_WRITE,300,1000);

BusServoCtrl(2,SERVO\_MOVE\_TIME\_WRITE,500,1000);

BusServoCtrl(3,SERVO\_MOVE\_TIME\_WRITE,350,1000);

BusServoCtrl(4,SERVO\_MOVE\_TIME\_WRITE,100,1000);

BusServoCtrl(5,SERVO\_MOVE\_TIME\_WRITE,300,1000);

BusServoCtrl(6,SERVO\_MOVE\_TIME\_WRITE,500,1000);

DelayMs(1000);

BusServoCtrl(1,SERVO\_MOVE\_TIME\_WRITE,140,1000);

BusServoCtrl(2,SERVO\_MOVE\_TIME\_WRITE,500,1000);

BusServoCtrl(3,SERVO\_MOVE\_TIME\_WRITE,810,1000);

BusServoCtrl(4,SERVO\_MOVE\_TIME\_WRITE,100,1000);

BusServoCtrl(5,SERVO\_MOVE\_TIME\_WRITE,765,1000);

DelayMs(1000);

DelayMs(1000);

BusServoCtrl(1,SERVO\_MOVE\_TIME\_WRITE,600,1000);

DelayMs(1000);

BusServoCtrl(3,SERVO\_MOVE\_TIME\_WRITE,350,1000);

BusServoCtrl(5,SERVO\_MOVE\_TIME\_WRITE,300,1000);

DelayMs(1000);

grab\_s = FALSE;

}

void put(void)

{

BusServoCtrl(6,SERVO\_MOVE\_TIME\_WRITE,130,1000);

DelayMs(1000);

BusServoCtrl(3,SERVO\_MOVE\_TIME\_WRITE,810,1000);

BusServoCtrl(5,SERVO\_MOVE\_TIME\_WRITE,765,1000);

DelayMs(1000);

DelayMs(1000);

BusServoCtrl(1,SERVO\_MOVE\_TIME\_WRITE,0,1000);

DelayMs(1000);

BusServoCtrl(3,SERVO\_MOVE\_TIME\_WRITE,350,1000);

BusServoCtrl(5,SERVO\_MOVE\_TIME\_WRITE,300,1000);

DelayMs(1000);

BusServoCtrl(1,SERVO\_MOVE\_TIME\_WRITE,300,1000);

BusServoCtrl(6,SERVO\_MOVE\_TIME\_WRITE,500,1000);

DelayMs(1000);

put\_s=FALSE;

## 附录三：OpenMv与STM32连接代码

STM32接收代码如下所示：

void uart\_init(u32 bound){

//GPIO端口设置

GPIO\_InitTypeDef GPIO\_InitStructure;

USART\_InitTypeDef USART\_InitStructure;

NVIC\_InitTypeDef NVIC\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_USART1|RCC\_APB2Periph\_GPIOA, ENABLE); //使能USART1，GPIOA时钟

//USART1\_TX GPIOA.9

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_9; //PA.9

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF\_PP; //复用推挽输出

GPIO\_Init(GPIOA, &GPIO\_InitStructure);//初始化GPIOA.9

//USART1\_RX GPIOA.10初始化

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_10;//PA10

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IN\_FLOATING;//浮空输入

GPIO\_Init(GPIOA, &GPIO\_InitStructure);//初始化GPIOA.10

//Usart1 NVIC 配置

NVIC\_InitStructure.NVIC\_IRQChannel = USART1\_IRQn;

NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority=3 ;//抢占优先级3

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 3; //子优先级3

NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE; //IRQ通道使能

NVIC\_Init(&NVIC\_InitStructure); //根据指定的参数初始化VIC寄存器

//USART 初始化设置

USART\_InitStructure.USART\_BaudRate = bound;//串口波特率

USART\_InitStructure.USART\_WordLength = USART\_WordLength\_8b;//字长为8位数据格式

USART\_InitStructure.USART\_StopBits = USART\_StopBits\_1;//一个停止位

USART\_InitStructure.USART\_Parity = USART\_Parity\_No;//无奇偶校验位

USART\_InitStructure.USART\_HardwareFlowControl = USART\_HardwareFlowControl\_None;//无硬件数据流控制

USART\_InitStructure.USART\_Mode = USART\_Mode\_Rx | USART\_Mode\_Tx; //收发模式

USART\_Init(USART1, &USART\_InitStructure); //初始化串口1

USART\_ITConfig(USART1, USART\_IT\_RXNE, ENABLE);//开启串口接受中断

USART\_Cmd(USART1, ENABLE); //使能串口1

}

中断函数：

if(USART\_GetITStatus(USART1, USART\_IT\_RXNE) != RESET) //接收中断(接收到的数据必须是0x0d 0x0a结尾)

{

Res =USART\_ReceiveData(USART1); //读取接收到的数据

if((USART\_RX\_STA&0x8000)==0)//接收未完成

{

if(USART\_RX\_STA&0x4000)//接收到了0x0d

{

if(Res!=0x0a)USART\_RX\_STA=0;//接收错误,重新开始

else USART\_RX\_STA|=0x8000; //接收完成了

}

else //还没收到0X0D

{

if(Res==0x0d)USART\_RX\_STA|=0x4000;

else

{

USART\_RX\_BUF[USART\_RX\_STA&0X3FFF]=Res ;

USART\_RX\_STA++;

if(USART\_RX\_STA>(USART\_REC\_LEN-1))

USART\_RX\_STA=0;//接收数据错误,重新开始接收

}

}

}

}

void OpenMv\_Receive\_Data(int16\_t data) //接收OpenMv传过来的数据

{

static u8 state = 0;

if(state==0&&data==0xb3)

{

state=1;

OpenMv[0]=data;

}

else if(state==1&&data==0xb3)

{

state=2;

OpenMv[1]=data;

}

else if(state==2)

{

state=3;

OpenMv[2]=data;

}

else if(state==3)

{

state = 4;

OpenMv[3]=data;

}

else if(state==4) //检测是否接受到结束标志

{

if(data == 0x5B)

{

state = 0;

OpenMv[4]=data;

OpenMv\_Data();

}

else if(data != 0x5B)

{

state = 0;

for(i=0;i<5;i++)

{

OpenMv[i]=0x00;

}

}

}

else

{

state = 0;

for(i=0;i<5;i++)

{

OpenMv[i]=0x00;

}

}

}

void OpenMv\_Data(void)

{

OpenMv\_X=OpenMv[2];//OpenMv加油枪反馈变量

OpenMv\_Y=OpenMv[3];//OpenMv车反馈变量

}