

# 设计报告

Author: innns

# 智能物流搬运机器人视觉及通信系统设计

\* innns

**摘要：**智能物流搬运机器人适用于各种仓库、车间中物料的搬运，能够大大减少仓库、车间中对人力的需求。由于成本低、生产周期短，因此能够大规模的批量生产，也能普遍应用。作者随队参加的第七届全国大学生工程训练综合能力竞赛-智能+赛道赛事中，我们队伍设计了一个智能物流搬运机器人，模拟工厂制造过程，实现了机器人利用陀螺仪、编码器、以及巡线模块自主定位导航；读取二维码或利用 WiFi 通信获取搬运任务；以及利用视觉识别自动地高精度地搬运、存放物料等功能。

**关键字：**搬运机器人、计算机视觉

## 目录

<b>一、 绪论</b>	<b>3</b>
1.1 智能机器人设计需求 . . . . .	3
1.1.1 功能要求 . . . . .	3
1.1.2 电控及驱动要求 . . . . .	3
1.1.3 机械结构要求 . . . . .	3
1.1.4 外形尺寸及载重要求 . . . . .	3
1.2 运行环境 . . . . .	4
1.2.1 机器人运行场地 . . . . .	4
1.2.2 物料 . . . . .	5
1.2.3 总体设计 . . . . .	7
<b>二、 设计方案论证</b>	<b>7</b>
2.1 视觉系统 . . . . .	7
2.2 WiFi 通信 . . . . .	8
2.3 选型 . . . . .	8
<b>三、 系统设计方案</b>	<b>9</b>
3.1 系统硬件环境 . . . . .	9
3.2 系统软件环境 . . . . .	9
3.2.1 相机端口绑定 . . . . .	10
3.3 开发环境 . . . . .	10
3.4 系统总体流程 . . . . .	11
3.5 通信系统 . . . . .	12
3.5.1 串口通信 . . . . .	12
3.5.2 WiFi 通信 . . . . .	15
3.6 视觉系统 . . . . .	15
3.6.1 相机模块 . . . . .	15
3.6.2 处理模块 . . . . .	15
<b>四、 系统测试</b>	<b>20</b>
<b>五、 总结</b>	<b>20</b>

## 一、绪论

随着电子信息技术的高速发展，机器人在越来越多的领域协助人们更好的工作。其中物流机器人在如今的仓库管理中的作用愈发凸显。物流机器人涉及的核心技术包括导航及定位技术、运动控制技术、计算机视觉技术等。

### 1.1 智能机器人设计需求

#### 1.1.1 功能要求

我们自主设计并制作一台按照给定任务完成物料搬运的智能机器人（简称：机器人）具有定位、移动、避障、读取条形码及二维码、Wi-Fi 网络通信、物料位置和颜色识别、物料抓取与载运、上坡和下坡、路径规划等功能；竞赛过程机器人可以自主运行，或采用无线人机交互手段操作，但必须首选自主运行方式，只有在自主运行方式出现故障时才可申请使用无线遥控运行方式。该机器人能够通过扫描二维码或 Wi-Fi 网络通信等方式领取搬运任务，在指定的工业场景内行走与避障，并按任务要求将物料搬运至指定地点并精准摆放（色环或条形码）。

#### 1.1.2 电控及驱动要求

机器人所用传感器和电机的种类及数量不限，在机器人的醒目位置安装有任务码显示装置，显示装置必须放置在机器人上部醒目位置，且不被任何物体遮挡，必须是亮光显示，字体高度不小于 8mm，该装置能够持续显示所有任务信息直至比赛结束，否则成绩无效。机器人各机构只能使用电驱动，采用电池（蓄电池除外）供电，供电电压限制在 12V 以下，随车装载，比赛过程中不能更换。

#### 1.1.3 机械结构要求

自主设计并制造机器人的机械部分，除标准件外，非标零件应自主设计和制作，不允许使用购买的成品套件拼装而成。机器人的行走方式、机械手臂的结构形式均不限制，机器人腕部与手爪的连接结构自行确定。

#### 1.1.4 外形尺寸及载重要求

机器人（含机械手臂）外形尺寸满足铅垂方向投影在边长为 300mm 的正方形内，高度不超过 400mm 方可参加比赛。允许机器人结构设计为可折叠形式，但出发之后才可自行展开。在初赛时机器人没有载重要求，而在决赛时机器人的总重

量不能小于规定重量，用于对集成在决赛场地中的桥梁进行测试。载重物块形状自定，运行时物块不能掉落。

## 1.2 运行环境

### 1.2.1 机器人运行场地

近水平铺设的赛场尺寸为  $4800 \times 2400 \text{ mm}$  长方形平面区域（如图 1），赛场

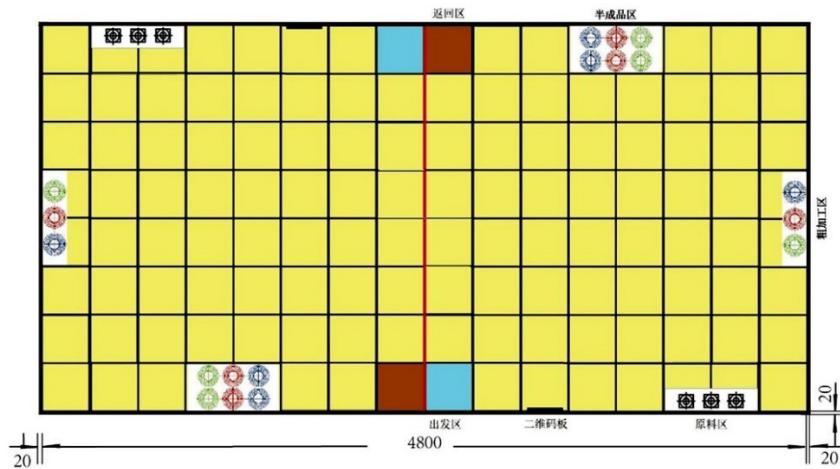


图 1: 场地示意图

周围设有一定高度的挡板，仅作为场地边界（颜色和高度不做任何要求），不宜作为寻边等其它任何用途。赛道地面为亚光白色或浅黄色等浅色底色，地面图案由线宽为  $20 \text{ mm}$ 、线中心距为  $300 \text{ mm}$  的黑色方格组成。在比赛场地内，设置出发区、返回区、原料区、粗加工区、半成品区、精加工区、库存区。

其中机器人初赛主要经过原料区、粗加工区和半成品区完成粗加工物料的搬运过程；机器人决赛主要经过半成品区、精加工区和库存区完成精加工物料的搬运过程。出发区和返回区的尺寸均为  $300 \times 300 \text{ mm}$ ，颜色分别为蓝色和褐色；原料区和库存区的尺寸（长  $\times$  宽  $\times$  高）为  $580 \times 145 \times (80 - 100) \text{ mm}$  白色亚光的双层货架（原料区的高度为  $100 \text{ mm}$ ，物料采用颜色识别，库存区的货架高度在  $80 - 100 \text{ mm}$  范围，采用条形码识别物料放置的位置）（如图 2 所示）粗加工区和精加工区的尺寸（长  $\times$  宽）为  $580 \times 150 \text{ mm}$ ；半成品区的尺寸（长  $\times$  宽  $\times$  高）为  $580 \times 150 \times 45$  及  $580 \times 148 \times 0 \text{ mm}$  的台阶区域（如图 3 所示）粗加工区、半成品区、精加工区顶面上均有用于测量物料摆放位置准确程度的色环，色环如图 17.5 所示，其中  $\phi$  为物料最大直径（单位： $\text{mm}$ ）， $\phi 1 - \phi 5$  为色环 1 – 5 环的外径，色

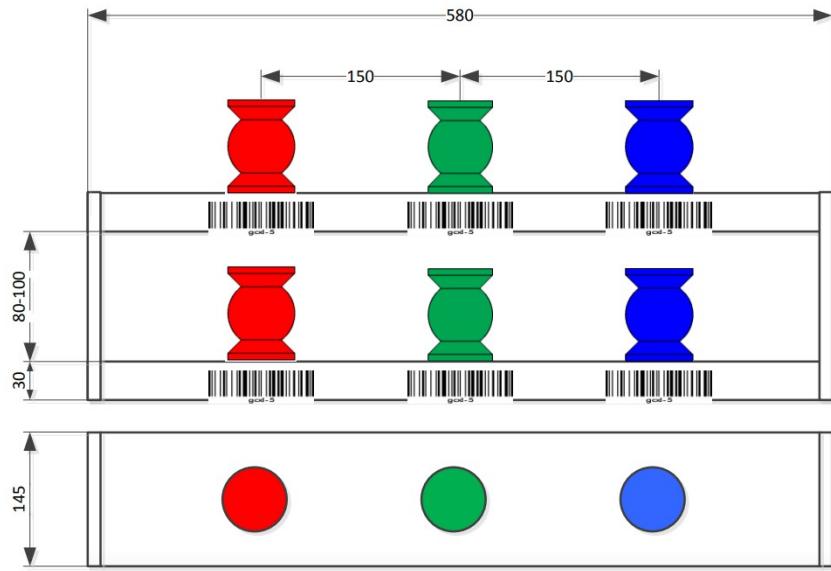


图 2: 原料区示意图

环线宽为  $1.5\text{ mm}$ 。除标注尺寸外，其余色环的直径差为  $10\text{ mm}$ 。库存区顶面有外径为  $\phi$  (物料直径)  $+15\text{ mm}$  的圆形区域，用于确定物料是否摆放到位。

### 1.2.2 物料

机器人初赛时待搬运的物料形状包络在直径为  $50\text{ mm}$ 、高度为  $70\text{ mm}$ 、重约为  $50\text{ g}$  的圆柱体中 (如图 6 所示)，夹持部分的形状为球体，物料的材料为 3D 打印 ABS，三种颜色为：红 (ABS/Red (C-21-03))、绿 (ABS/Green (C-21-06))、蓝 (ABS/Blue (C-21-04))。三种不同颜色的物料 (每种颜色两个) 随机放置在原料区的物料架上 (上层及下层红、绿、蓝物料各一个)，物料间距为  $150\text{ mm}$  (如图 2 所示)

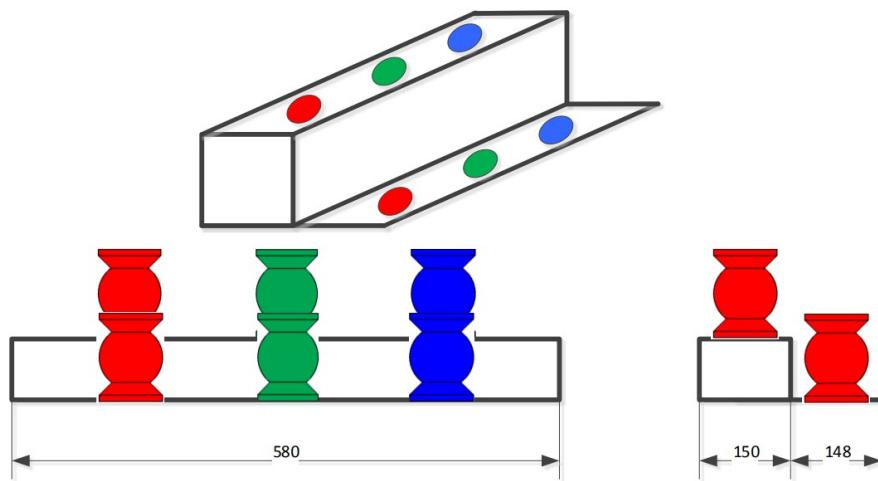


图 3: 半成品区示意图

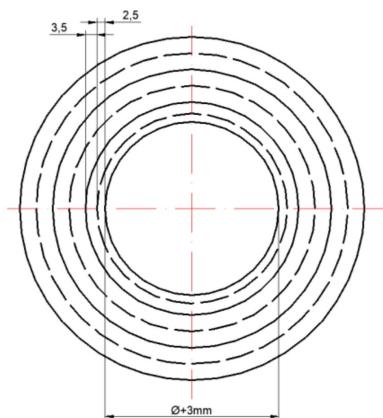


图 4: 色环尺寸图

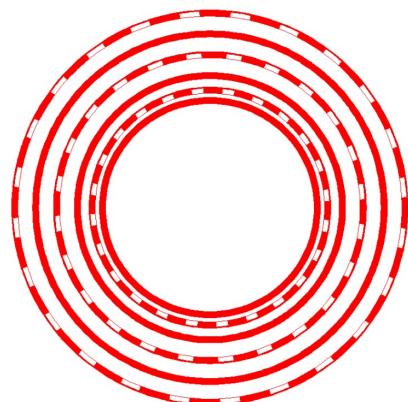


图 5: 色环示意图

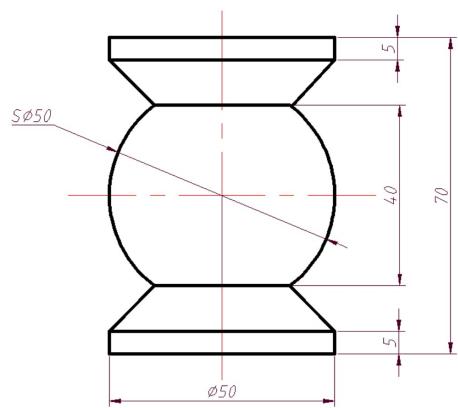


图 6: 原料示意图

### 1.2.3 总体设计

总体设计如图 7, 上层使用二维平台, 实现垂直、水平移动; 通过 DJI 3508 电机实现上层结构旋转; 通过 DJI 2006 电机实现手抓升降; 通过舵机实现手抓开闭; 其中机器人拥有 OV2710, OV5647 两个相机分别对准前方和地面。

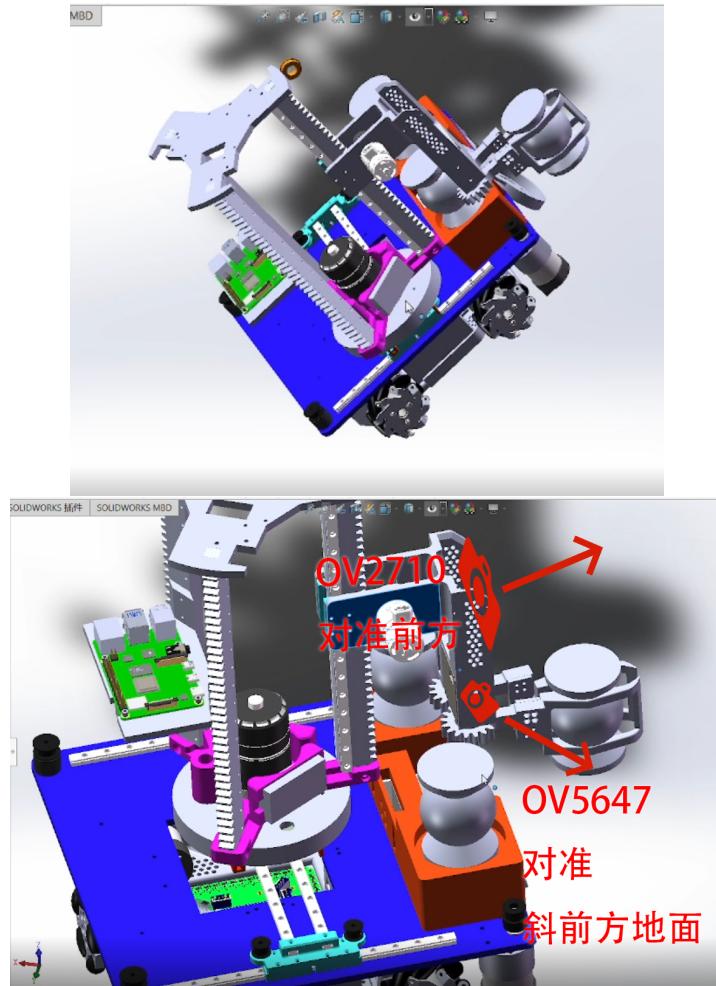


图 7: 总体设计图

## 二、设计方案论证

### 2.1 视觉系统

目前主流的较低成本（1000 元以下）的嵌入式视觉解决方案主要有 OpenMV、FPGA、树莓派、Nvidia Jetson Nano 等。其中：

- OpenMV, 简单来说, 它是一个可编程的摄像头, 摄像头本身内置了一些图像处理算法, 很容易使用 [1], 价格也较低（400-700 元）。

- FPGA 相比 CPU 处理图像而言具有并发处理的优势，但是实现较为困难，扩展性差。
- 树莓派（图 8）是一款基于 ARM 的微型电脑主板，以 SD/MicroSD 卡为内存硬盘。树莓派 4B 拥有内置无线网卡、40-pin GPIO、2 个 USB3.0 及 2 个 USB2.0、具备 HDMI 输出功能，可扩展性强，性能较好。价格较低（当时 2GB 版本 300 元，2022 年 1 月国内竟然卖到了 600 元以上...）
- Nvidia Jetson Nano Developer Kit（图 9 是英伟达对标树莓派出品的一款基于 ARM 的边缘计算设备。性能高于树莓派，可扩展性也稍微强于树莓派（拥有 4 个 USB3.0，且兼容树莓派 40-pin GPIO，可安装 m.2 接口外置天线的无线网卡）。但是由于各种原因，官网 2GB 版本 69 美元、4GB 版本 99 美元的价格到了国内（2022 年 1 月）变成了 850 元和 1300 元左右。

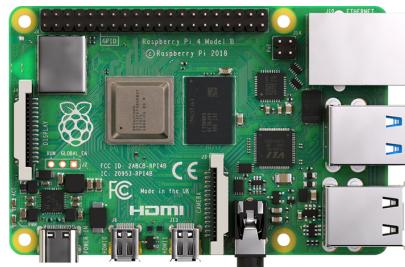


图 8: 树莓派



图 9: Jetson Nano

## 2.2 WiFi 通信

WiFi 通信可以选择的方案并不多，除了内置 WiFi 的树莓派或者可装 WiFi 模块的 Jetson Nano，还可以选择 WiFi 串口模块或带有 WiFi 功能的单片机如 ESP 系列实现 UDP 和服务器通信。

## 2.3 选型

综合以上考虑我们选择了基于树莓派的方案（图 8）。采用了一款体积较大，传感器为 OV2710，1/2.7 英寸 CMOS 的 USB 工业相机用于识别物料的颜色（图

10)；一款体积较小，传感器为 OV5647，1/4 英寸 CMOS 的 USB 相机用于识别并定位靶子（图 11）。



图 10: 颜色摄像头



图 11: 靶子摄像头

### 三、系统设计方案

#### 3.1 系统硬件环境

- Raspberry Pi 4B 2GB（图 8）
- OV2710 USB camera（图 10）
- OV5647 USB camera（图 11）

#### 3.2 系统软件环境

- 2021-05-27-raspios-buster-arm64
- OpenCV 4.5.2 + contrib
- C++
- CMake

### 3.2.1 相机端口绑定

由于重启之后相机端口号可能会改变，故使用 v4l/by-id 的软链接绑定相机端口号。

```
1 # e.g.
2 # ln /dev/v4l/by-id 再用软链接绑定
3 ln /dev/v4l/by-id
4 # $ tree /dev/v4l
5 # /dev/v4l
6 #   └── by-id
7 #     └── usb-OmniVision._USB_Camera-B4.04.27.1-video-index0 -> ../../video0
8 #   └── by-path
9 #     └── pci-0000:00:14.0-usb-0:2:1.0-video-index0 -> ../../video0
10 sudo ln -s /dev/v4l/by-id/usb-337b_HBVCAM_5M-AF-video-index0 /dev/camera1
```

代码 1: 相机端口绑定

### 3.3 开发环境

作者使用了 Windows 10 + VS Code 通过 SSH 连接树莓派开发，插件如下：

- ms-vscode.cpptools-extension-pack
- coenraads.bracket-pair-colorizer
- obkoro1.korofileheader
- christian-kohler.path-intellisense
- ms-python.python
- gruntfuggly.todo-tree
- visualstudioexptteam.vscodeintellicode
- donjayamanne.githistory
- eamodio.gitlens
- github.vscode-pull-request-github

并且也使用了 WSL 在 Ubuntu amd64 环境下验证代码。

### 3.4 系统总体流程

主程序会单独开一个监听线程实时监听 32 串口，当 32 发送命令时，若判断命令格式正确则会执行相应命令的功能。图 12 即为系统流程图。因为扫颜色使用 OV2710，扫靶子使用 OV5647，为减小性能损失一次只打开一个相机，故需要切换相机功能。其中串口代码使用了 Ge Zhenpeng 学长的 rmoss\_core，Github @gezp，Apache2.0 协议 [2]；WiFi 通信部分使用了 Github @eminfedar 的 async-sockets-cpp，MIT 协议 [3]，在此致谢。

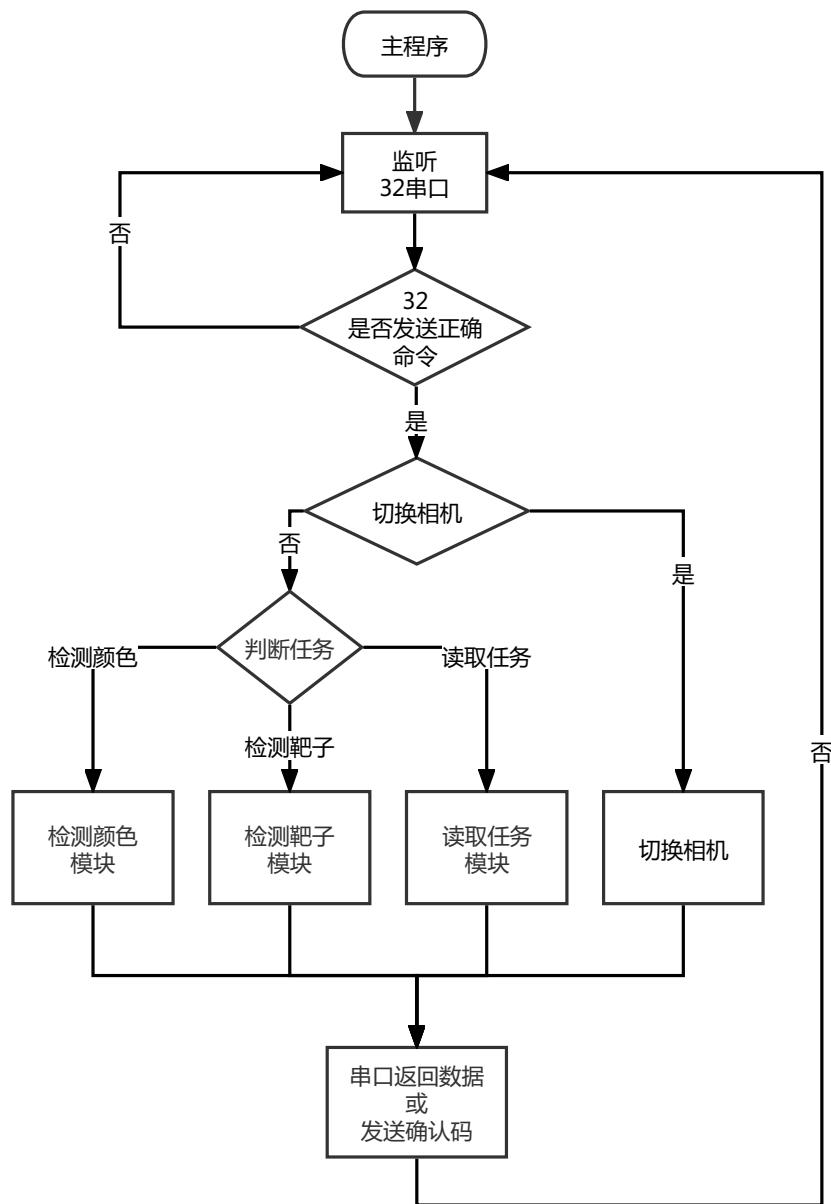


图 12: 系统流程图

### 3.5 通信系统

#### 3.5.1 串口通信

**数据帧封装** 采用 16 字节数据帧，采用小端模式，封装如下：

帧头 *1	命令位 *1	数据位 *12	校验位 *1	帧尾 *1
[0] 0xAA	[1] CMD	[2-13] DATA	[14] 0xF0	[15] 0xBB

**协议定义** 具体定义如下：

```

1  typedef enum : unsigned char
2  {
3      // >>>>> 发送用
4      CMD_CURRENT_COLOR = 0x01,    // 当前检测到的颜色 条形码读取的颜色
5      CMD_TARGET_POS = 0x02,       // 当前检测到的靶心
6      CMD_QR_MISSION = 0x03,      // 二维码读取的任务
7      CMD_ALL_COLOR = 0x04,        // 所有颜色读取
8      CMD_ACK = 0x05,             // 确认 切换相机 切换靶心等
9      CMD_STORE_POS = 0x06,       // 库存区域位置
10
11     // >>>>> 接收用
12     CMD_SET_MODE = 0x11,         // 检测颜色/检测靶心
13     CMD_SWITCH_CAM = 0x12,      // 切换相机
14 } SerialPortCMD;
15
16 //>>>>>树莓派端数据接口>>>>
17 bool current_color(char color_num);
18 bool all_color(char m1, char m2, char m3, char m4, char m5, char m6);
19 bool target_pos(int target_x, int target_y, int target_z);
20 bool QR_mission(char m1, char m2, char m3, char m4, char m5, char m6);
21 bool temp(int target_x, int target_y, int target_z);

```

代码 2: 相机端口绑定

#### Pi 发给 32

**当前颜色（单个颜色）** 条形码识别采用该形式发送：

帧头 *1	命令位 *1	数据位 *1	校验位 *1	帧尾 *1
[0] 0xAA	[1] 0x01	[2] (char) COLOR	[14] 0xF0	[15] 0xBB

其中，COLOR 为 char 型，定义如下：

NONE	RED	GREEN	BLUE
0x00	0x01	0x02	0x03

**靶心位置** 识别完靶心位置后, 发送给 32 告诉小车二维云台需要移动的距离, 以该形式发送:

帧头 *1	命令位 *1	数据位 *12	校验位 *1	帧尾 *1
[0] 0xAA	[1] 0x02	[2-13] (int) pos_x, pos_y, pos_z	[14] 0xF0	[15] 0xBB

其中, 我们定义正方向为:

- 左方向 pos\_x
- 上方向 pos\_y
- 逆时针 pos\_z

如图 13 所示:

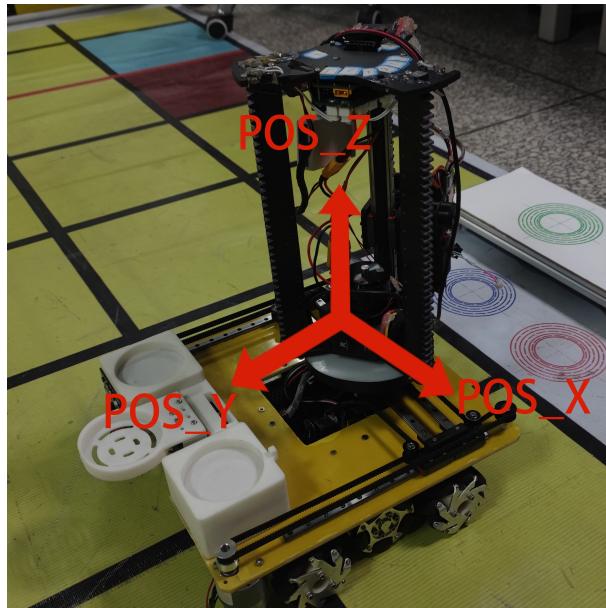


图 13: 坐标示意图

**颜色识别** 物料首先摆放在原料区域, 上层 3 个, 下层 3 个, 共六个。一次性识别 6 个之后, 以该形式发送:

帧头 *1	命令位 *1	数据位 *12	校验位 *1	帧尾 *1
[0] 0xAA	[1] 0x03	[2-7] (char) COLOR * 6	[14] 0xF0	[15] 0xBB

**任务顺序** 树莓派通过扫描二维码或者 WiFi 获取任务顺序，以该形式发送：

帧头 *1	命令位 *1	数据位 *12	校验位 *1	帧尾 *1
[0] 0xAA	[1] 0x04	[2-7] (char) COLOR * 6	[14] 0xF0	[15] 0xBB

### 32 发给 Pi

**切换相机** 切换相机功能，32 发送格式如下：

帧头 *1	命令位 *1	数据位 *12	校验位 *1	帧尾 *1
[0] 0xAA	[1] 0x12	[2] CAM_NUM	[14] 0xF0	[15] 0xBB

其中，我们定义相机为：

- OV2710 (char) 0x00
- OV5647 (char) 0x01

**设置任务** 决定程序进哪个任务执行模块，32 发送格式如下：

帧头 *1	命令位 *1	数据位 *12	校验位 *1	帧尾 *1
[0] 0xAA	[1] 0x11	[2] (char) DETECT_MODE	[14] 0xF0	[15] 0xBB
		[3] (char) TARGET_POS		

其中，

```

1 enum DETECT_MODE : unsigned char
2 {
3     DEFAULT_MODE = 0x00,      // 空模式 0
4     COLOR_MODE = 0x01,        // 读取颜色 1 国赛读取条形码
5     TARGET_MODE = 0x02,       // 寻找靶心 2
6     QR_MODE = 0x03,          // 读取二维码 3
7     ALL_COLOR_MODE = 0x04,    // 读取所有颜色 4
8     STORE_MODE = 0x06
9 };
10
11 // enum COLOR: N R G B
12 enum TARGET_POS : unsigned char
13 {
14     NONE_TAR = 0x00,
15     LEFT_TAR = 0x01,
16     CENTER_TAR = 0x02,
17     RIGHT_TAR = 0x03
18 };

```

代码 3: DETECT\_MODE 及 TARGET\_POS 定义

### 3.5.2 WiFi 通信

和服务器连接至同一个网段下后，使用 `async-sockets-cpp udp-client` 模块，开启 UDP Socket 监听端口，等待服务器广播即可 [3]。

## 3.6 视觉系统

视觉系统分为下面两个模块：

- 相机对象模块
- 处理模块

其中，相机对象原本打算也采用多线程实现，但是测试时相机经常会掉线（可能原因是模块 Bug 或者是多线程时，工作时间过长过热），故没有使用 `Cam_MT` 多线程模块（已实现，在源码文件中 `include` 文件夹下，可能有 Bug），使用了普通的单线程模块。处理模块为具体功能的实现。

### 3.6.1 相机模块

相机模块，主要功能为读取 `yaml` 文件配置相机的分辨率、帧率、白平衡等参数，并提供读取图像的接口，不再赘述。

### 3.6.2 处理模块

处理模块主要分为：

- 检测颜色模块
- 检测靶子模块
- 获取任务模块

### 检测颜色模块

**单个颜色** 单个颜色功能较为简单，统计各颜色面积输出最大值即可。获取HSV图片之后还需要经过一定的形态学处理，流程图中省略了。大致流程如图14所示。

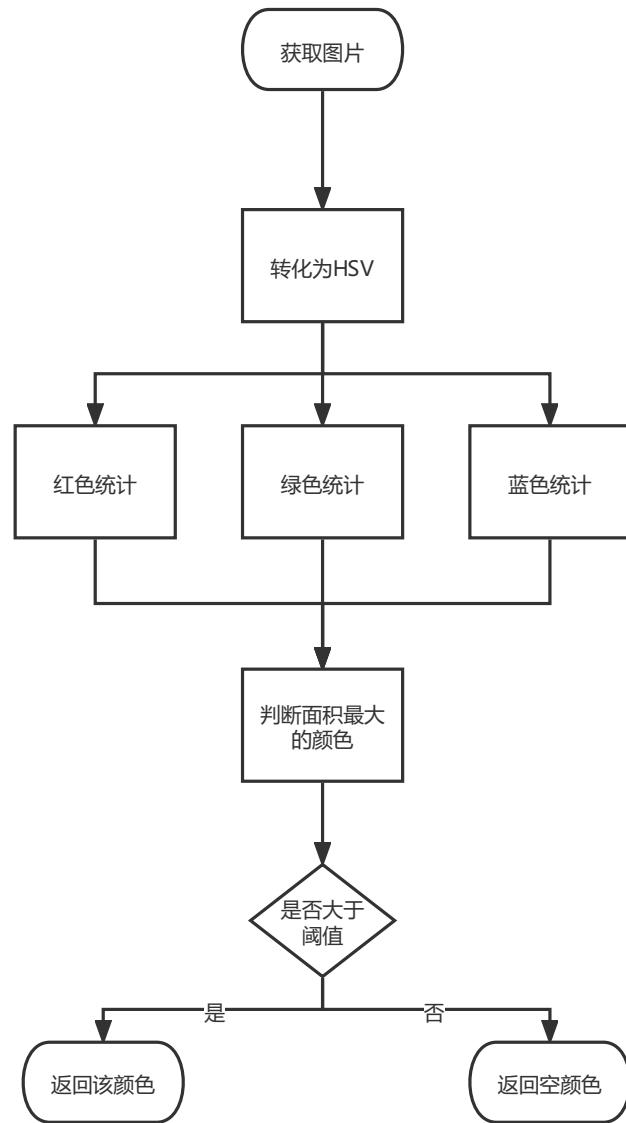


图14：单个颜色识别流程图

**多个颜色** 多个颜色功能以单个颜色的处理函数为基础，以各颜色连通区域的形状、大小、面积判断是否为物料，再根据位置输出顺序，流程如图 15 所示。

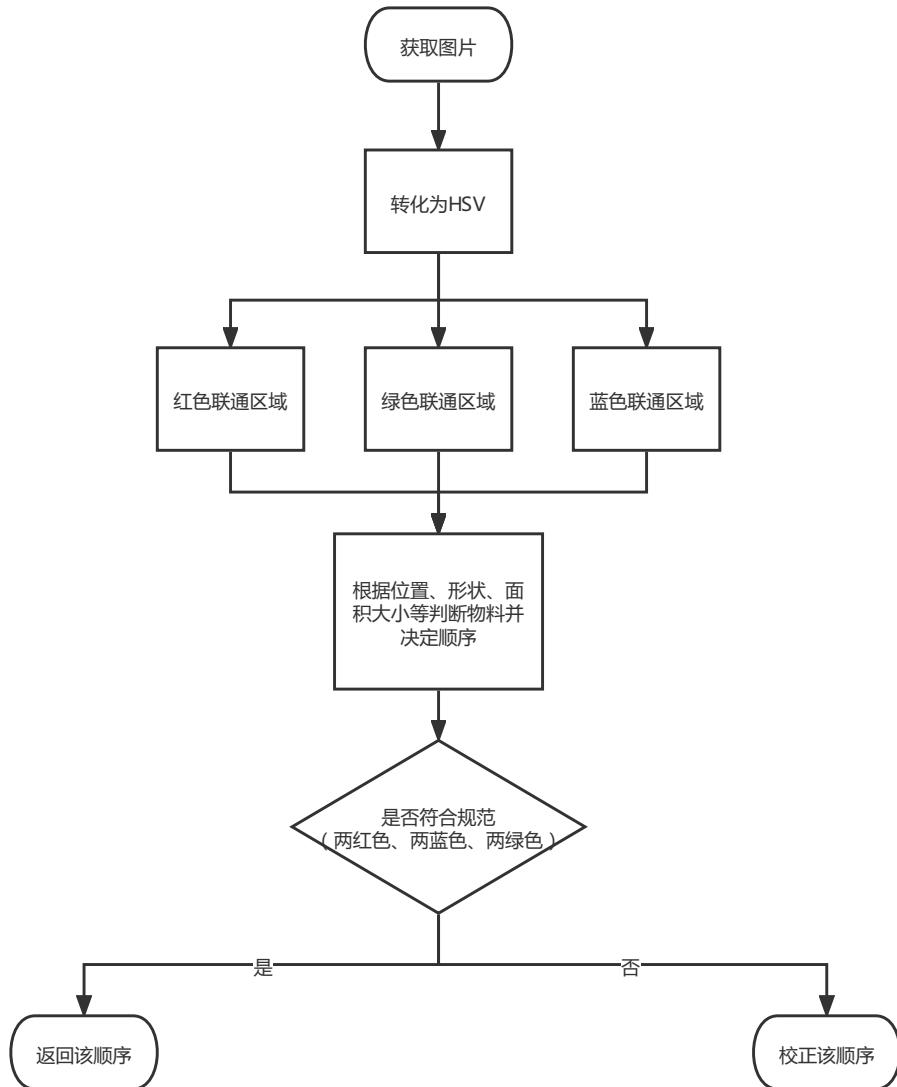


图 15: 多个颜色识别流程图

其中，校正顺序为判断该函数的输出最可能的符合的正确情况，如输出 120+213 则矫正为 123+213（每一层的正确输出共有 6 种，通过循环来判断最可能符合的正确情况）。

1 `int ALL_RIGHT_CO_LIST[6][3] = {{1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}, {3, 1, 2}, {3, 2, 1}};`

代码 4: 所有可能的正确情况

各颜色输出结果如图 16:

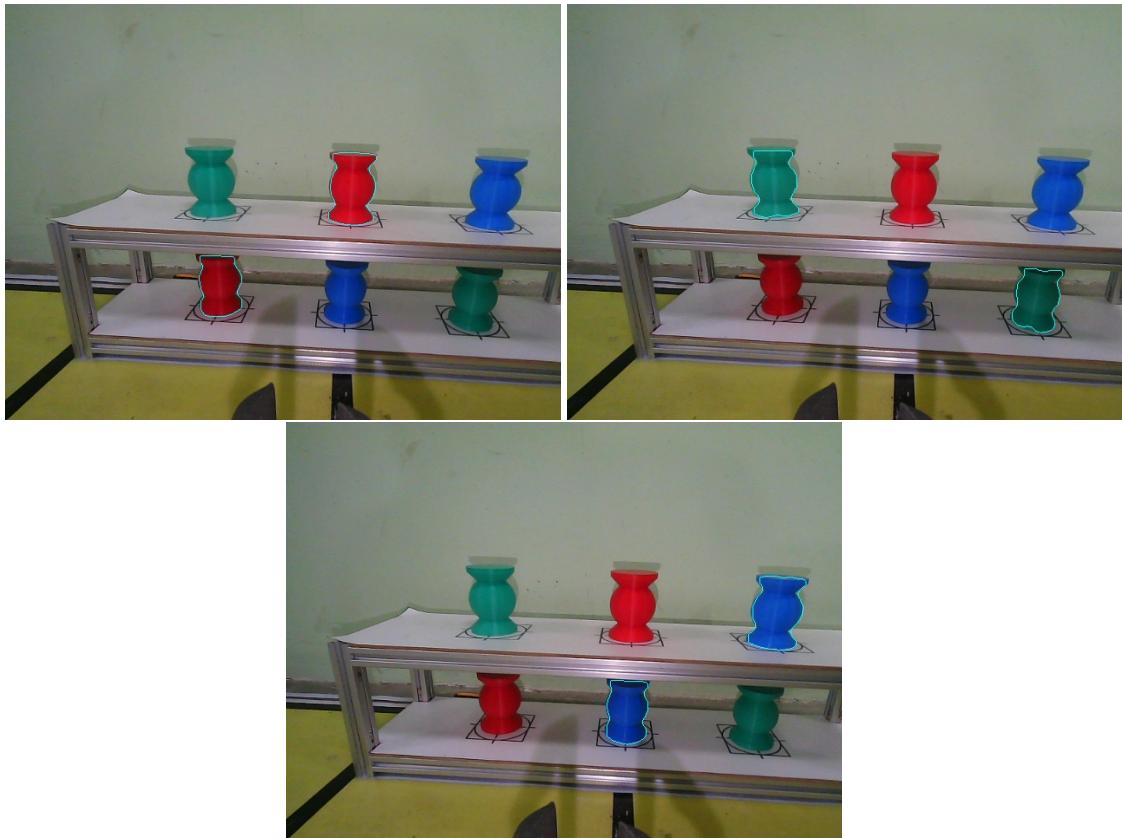


图 16: 多个颜色识别结果

**检测靶子** 检测靶子前需要切换至 OV5647，由于我们选择的 OV5647 结构较小可以放在手抓结构之下扫描斜前方靶子。通过视角校正为水平画面、得到靶心坐标后根据测试得到当前相机位置下像素坐标和世界坐标的对应关系。总体流程图如图 17，实际识别结果如图 18：

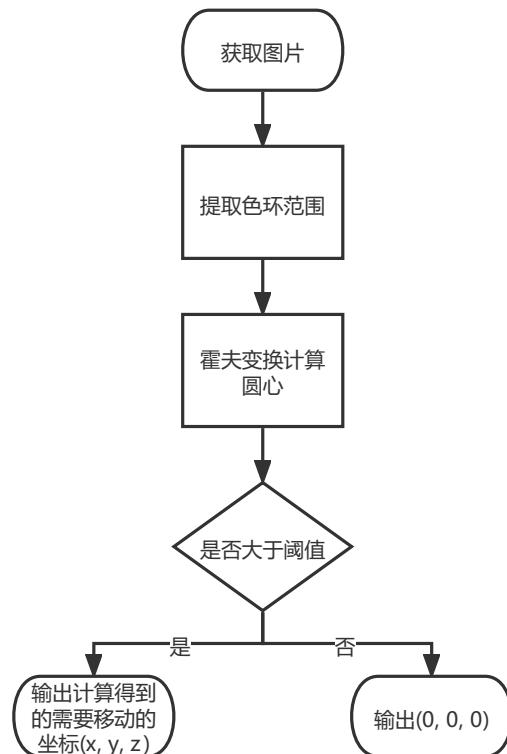


图 17: 识别靶心流程图

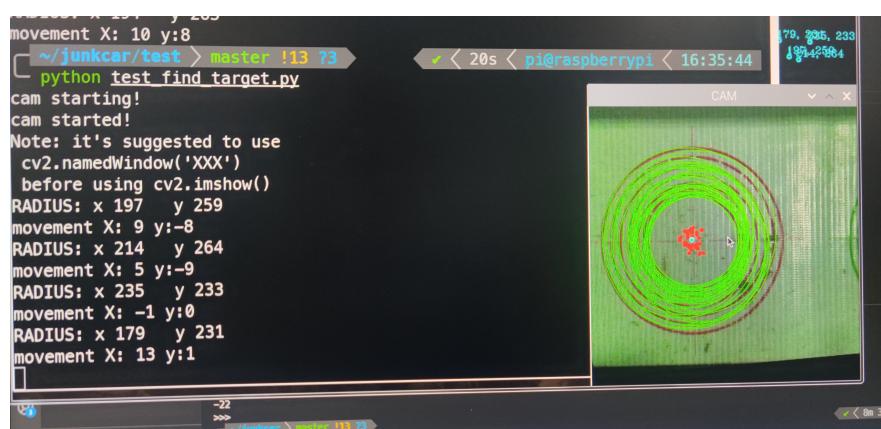


图 18: 靶心识别结果

## 任务获取

- WiFi 通信获取任务
- 摄像头模块解析二维码/条形码图片获取任务
- 二维码模块读取二维码/条形码内容获取任务

三者均实现了。在实际比赛中根据需求选择。

## 四、系统测试

在本校的场地测试时效果都很不错，能正确读取物料顺序，获取任务顺序以及在校正靶心之后可以达到 2mm 以内的放置精度；可以顺利完成流程。

在国赛赛场也顺利通过了国赛初赛，因为背景颜色问题以及 ROI 设置问题在读取颜色时差点翻大车；然而在国赛决赛的比赛过程中，车内部有一根电源线断开了，时间限制没有办法修复，真正翻车了。有些遗憾了，最终获得了银奖。

## 五、总结

系统整体代码已上传至 <https://github.com/innns/GCXL-Conveying-Robot-CV>，GPL 协议。

本项目林林总总共计写了 4000 行左右的代码；算上使用的开源模块的话，本项目有 5000 行代码，但是很大一部分注释以及无效代码。算上省赛的准备时间，以及由于国赛延期，耗时将近一年。期间也做了调试电控部分、焊电路版、设计 3D 打印件等等工作，让我对嵌入式设备、机器人的了解更深了。希望之后可以继续做相关的领域吧。

## 参考文献

- [1] 星瞳科技. Openmv 嵌入式图像处理. <https://book.openmv.cc/>.
- [2] geopol. rmoss\_base. [https://github.com/robomaster-oss/rmoss\\_core/tree/main/rmoss\\_base](https://github.com/robomaster-oss/rmoss_core/tree/main/rmoss_base).
- [3] eminfedor. Asynchronous sockets for c++. <https://github.com/eminfedor/async-sockets-cpp>.