南京大学

智能小车 Alpha-GO 设计文档

院系：电子科学与工程学院

专业：通信工程

年级：大三 学号：191180207

学生姓名：庄嘉鑫

指导老师：方元

日期：2022 年 6 月 18 日

目录

1 概要说明

本项目是基于树莓派 3B 搭建的智能小车 Alpha-Go，内置有直流电机、红外传感器、舵机、摄像头、蜂鸣器、控制杆、LED 灯等硬件资源。考虑到联网性和软件库的支持，系统采用的是树莓派官方的 64 位操作系统，并使用 Python3 进行开发。通过对硬件资源的驱动二次开发实现小车行驶和其他硬件控制，并使用 Socket 套接字编程实现网络控制小车，智能小车的摄像头使用 OpenCV 库进行视频数据获取以及压缩传输。上位机（电脑）使用 Socket 套接字和 OpenCV 获取和解析视频数据，并实现键盘输入通过网络控制小车行动。项目资料已上传至 GitHub：仓库地址。

关键词：树莓派 3B、智能小车、嵌入式系统、Socket、OpenCV

2 系统介绍

下面将从硬件资源和软件设计两个方面介绍本项目，硬件采用的是微雪电子 AlphaBot2 的硬件资源，相关资料可以查看官方教程。软件则是采用 AlphaBot2 提供的实例代码和自己的二次设计和开发完成的。

2.1 硬件资源介绍

（1）树莓派 3B

树莓派（RaspberryPi）是英国树莓派基金会开发的微型单板计算机，目前是以低价硬件和自由软件促进学校的基本电脑科学教育。树莓派系列计算机每一代均适用博通（Broadcom）生产的 ARM 架构处理器，如今生产的机型内存在 2 GB 到 8 GB 之间。主要使用 TF 卡作为系统存储的媒体，配备 USB 接口和 HDMI 视频音频输出，内置 Ethernet/WLAN/Bluetooth 等通讯硬件，并且可使用多种操作系统。虽然只有信用卡大小，但是其完备的功能和自由软件受到众多电子爱好者的喜爱。

本项目采用的是树莓派 3，其 SOC 为博通的 BCM2837，搭载 ARM Cortext-A53 64 位 1.2GHz 的 CPU，内存为 1GB。

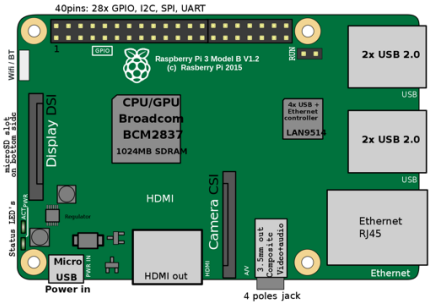


Figure 1 **树莓派** 3B（RaspberryPi 3B）**硬件资源**

（2）GPIO 和 PWM

树莓派系列提供一组 40Pin 的 GPIO 接口供用户连接外设使用，其中涵盖了输入、输出、I2C、SPI、UART 等接口功能。具体的分布和用法已经有人总结并制作成一个网站提供参考：Raspberry Pi Pinout。分布示意图如下图所示：

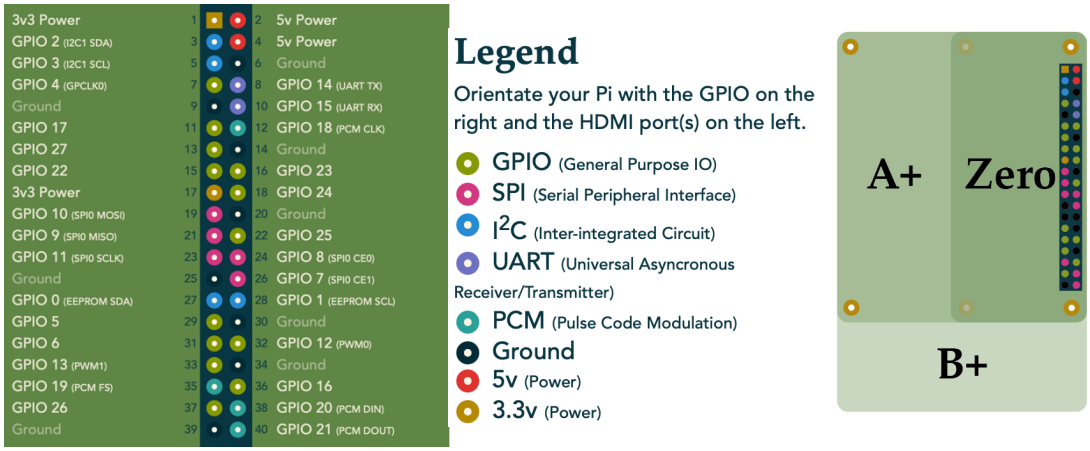


Figure 2 **树莓派引脚示意图**

在树莓派中，只有固定的接口支持硬件的 PWM 波形发生，但是几乎每一个 GPIO 接口都支持软件的 PWM，考虑到灵活性，这里推荐的也是软件 PWM。使用 PWM 的方式很简单，以 Python3 为例：

```
import RPi.GPIO as GPIO                 # 导入 GPIO
pwm = GPIO.PWM(Pin, frequency)          # 设置 PWM
pwm.start(duty)                         # 启动 PWM
pwm.ChangeDutyCycle(new_duty)           # 切换占空比
```

（3）直流电机

直流电机控制电路由两个功率输出端和一个控制开关组成。其结构图如下图所示，当开关接通时，输出电流方向由两个输入端的电压差决定，从而实现电机的双向转动控制。调整开关闭合比例可以调整直流电机的转速。

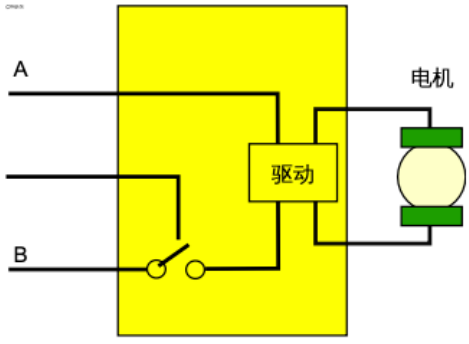


Figure 3 **直流电机示意图**

这里采用控制开关引脚采用 PWM 来控制开关的闭合速度，从而控制小车的速度，两个输出引脚控制速度。经过实际测试，得到了以下几种模式下的：（PWM 频率为 500Hz）

Mode	Left Duty	Right Duty
Forward / Backward	50	50
Turn Right / Left	30	30
Stop	0	0

Table 1 **直流电机 PWM 占空比控制**

由于小车直流电机驱动力与小车电池电量、地面摩擦等因素有关，这里仅作参考，实际运行中需要根据真实场景和期望的速度进行适当调节。

（4）红外传感器

红外传感器原理如下图所示。工作室，要求 Trig 引脚产生不低于 10us 的正脉冲，模块会自动发出 8 个周期的超声脉冲信号（40KHz），并在 Echo 引脚输出高电平。当检测到回波将 Echo 回置到低电平。通过测量 Echo 高电平维持周期，再根据声波速度，我们就可以粗略计算出发射器到障碍物之间的距离。

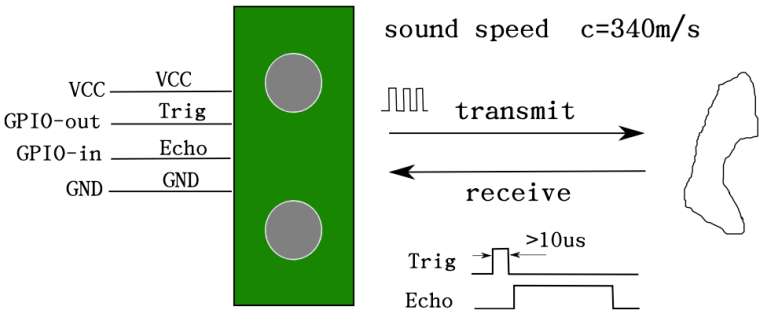


Figure 4 **超声波测距示意图**

对于车前面的红外传感器，我们只使用其数字信号引脚，通过高低电平判断是否在一定距离内由障碍物，而这个距离我们可以通过旋动传感器的滑动变阻器来调整。为了实现下文所提到的自动制动功能，需要结合直流电机驱动的小车速度和网络延时来调节距离以实现比较好的制动效果。

（5）摄像头和舵机

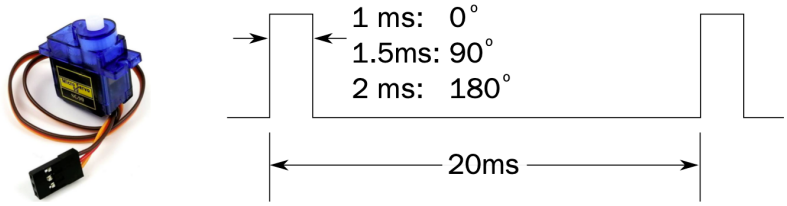
Alpha-Go 小车还配备摄像头（Raspberry Pi Camera Rev 2.0）以及两个舵机和支架实现摄像头的上下左右四个方向的移动。

舵机采用的是 Micro Servo SG90，是一款轻巧的微机电设备，重量只有 9 克，控制速度 0.1s/60 度。通过 PWM 的占空比可以精确控制。使用 20ms 周期，占空比 5-10%的 PWM，可以控制舵机 0-180 度的偏转。

Alpha-Go 是通过 PCA9685 芯片（芯片手册）产生的 PWM 脉冲，控制舵机的旋转角度。

PCA8685 是一个 16 通道的程控 PWM 发生器，采用 I2C 接口进行控制。我们在使用之前需要打开系统 I2C 接口功能，否则可能无法正确控制芯片工作。

根据 PCA9685 芯片手册和 AlphaBot2 硬件原理图我们可以看到 Alpha-Go 上的 PCA9685 的 LED0 和 LED1 引脚接在舵机上，因此我们只需要编写程序控制 LED0 和 LED1 两个引脚的 PWM 输出即可控制两个舵机的角度。



(a) 伺服机 SG90　　(b) 控制信号

Figure 5 舵机控制示意图

为了获取摄像头的数据，我们需要先打开系统的摄像头接口功能。这里我们使用比较流行的图像处理库 OpenCV 来获取摄像头的数据，OpenCV 的相关例程和资料可以在官网上找到。

（6）控制杆

控制杆由一个五个端口的开关组成，控制杆可移动的一段端接地，剩下的 A、B、C、D 以及中心 Center 分别接至输入端接口。计算机根据读到的端口状态判断控制杆当前处于什么位置。

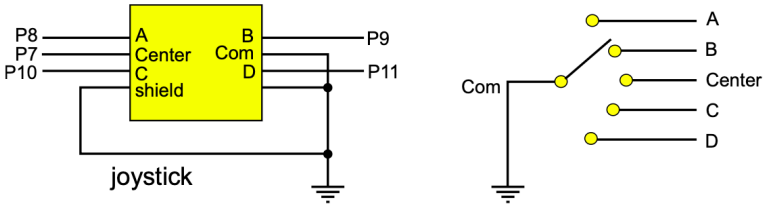


Figure 6 控制杆示意图

（7）LED 灯

LED 灯珠是由红绿蓝三个颜色的发光二极管组成的，三个颜色的发光二极管产生的不同亮度合在一起形成不同的颜色。每个发光二极管由一个 8 位的数字信号控制，实现 256 级的亮度调节。数字信号以串行方式输入，多个灯珠串联，形成一个灯带。等待的数字信号控制时间在微秒以下，对系统的时钟要求很高，普通的程序难以达到这样的精度，因此必须使用硬件控制。这里采用 rpi_ws281x 模块来控制。

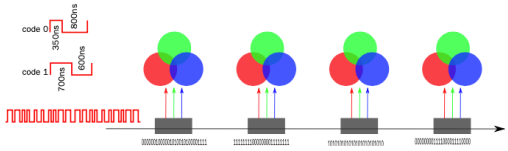


Figure 7 LED 彩灯控制原理

2.2 软件介绍

（1）操作系统和开发语言

考虑到软件库的安装，本项目使用官方的 64 位系统。在官网上下载官方的烧录工具 Raspberry Pi Imager。找到一张全新的 TF 卡连接到电脑上，在烧录工具中配置好需要烧录的系统、主机名、ssh 以及 WiFi 配置，点击烧录即可。

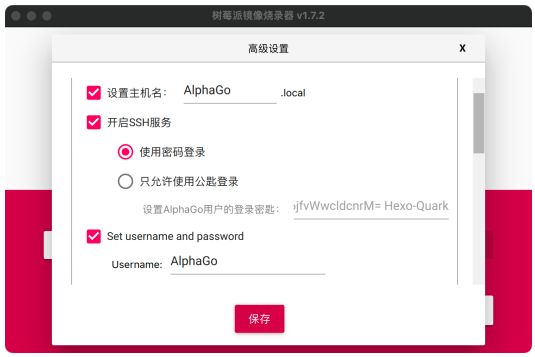


Figure 8 **系统烧录**

Raspberry Pi OS 是基于 Linux 的，系统自带 Python3 解释器。Python3 作为一个相对简单上手的编程语言，拥有 OpenCV 等功能强大的库。且一开始打算在树莓派上运行一些机器学习（后来发现树莓派 3B 的算力不足以支持这么复杂的计算），因此选择 Python3 作为开发语言。以下是本项目的软件环境：

- 硬件平台：树莓派 3B
- 系统：Raspberry Pi OS（64-bit）- Bullseye
- 开发语言：Python3.9

（2）软件结构介绍

下面介绍系统整体的软件框架，本项目是基于模块化和类封装的思想进行编程。在小车端，将每个硬件部分封装成一个类，最后通过 AlphaSystem 类来组合其他功能，并实现核心

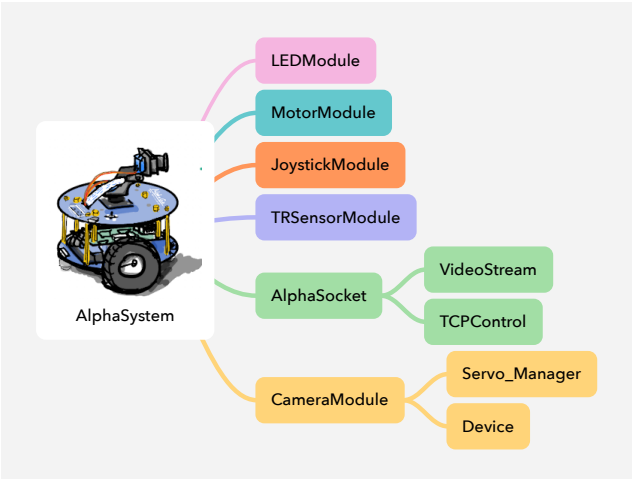


Figure 9 **小车端软件结构**

多线程控制和套接字 Socket 的网络连接功能。在上位机（电脑）端，采用多线程和 Socket 套接字来实现网络连接和发送指令，接受来自小车的摄像头数据。

LEDModule：小车 LED 彩灯部分，基于 rpi_ws281x 进行开发，实现控制小车上四个 LED 彩灯的亮度和颜色。内置的方法有：

- close(self)：在程序结束的时候清楚缓存的数据，让 LED 彩灯熄灭。
- numLED(self)：返回 LED 灯的数量（在初始化时设置）。
- setLED(self, seqLED, color, wait_ms=10)：设置序号为 seqLED 的彩灯，颜色为 color，wait_ms 用于阻塞一段时间确保芯片完全载入数据。
- setAll(self, color, wait_ms=20)：设置所有 LED 彩灯。
- show(self)：刷新硬件的缓存

TRSensorModule：小车前面红外传感器以及地盘四个循迹传感器的驱动。四个循迹传感器输出的是经过 AD 采样得到的模拟数据，而前面的两个避障传感器则是输出 01 数字信号。内置的方法有：

- AnalogRead(self)：读取循迹传感器的模拟数据
- Calibrate(self)：校准四个循迹传感器。
- readCalibrate(self)：读取校准的结果。
- readLine(self, white_line=0)：循迹。
- detect_front(self)：前方障碍检测。
- close_decte(self)：停止前方障碍检测。

MotorModule：小车直流电机驱动，基于 GPIO 和 PWM 开发，控制小车前进后退左右转弯以及智能制动。内置的方法有：

- Forward(self)：控制小车前进。
- Backward(self)：控制小车后退。
- TurnLeft(self)：控制小车原地左转。
- TurnRight(self)：控制小车原地右转。
- SetSpeed(self, speed)：设置小车的速度。
- Avoid_object(self)：小车前方的红外传感器不断检测，一旦检测到障碍物，将前进方向锁死以达到制动的目的，但是不会锁定左右转弯和后退。

JoystickModule：小车控制杆的驱动部分，其他部件可以调用它以实现控制杆控制的功能。通过方法 getkey(self)可以得到一个由五个元素组成的列表，分别对应 CTR，A，B，C，D 五个端口的电平，默认是 1，一旦触发对应的端口数据变为 0。可以通过轮询查看 0 的出现来判断控制杆状态。

CameraModule：摄像头部分，包括摄像头和控制方向的舵机。使用 OpenCV 开启和读取摄像头数据，舵机控制摄像头方向，同时将当前转动的角度显示在画面左上方方便校正和移动。内置的方法有：

- deivce：成员变量，存储摄像头对象
- up(self, delta)：控制舵机向上转动，转动 delta 度数。
- down(self, delta)：控制舵机向下转动，转动 delta 度数
- left(self, delta)：控制舵机向左转动，转动 delta 度数

- right(self, delta)：控制舵机向右转动，转动 delta 度数

在实践中发现，在不断检测指令状态来控制舵机运动的模式下，如果选择较大的 delta 角度摄像头在每次转动后都会有一定的振动。减小 delta 和适当提高刷新率可以缓解振动。根据实测，当刷新率为 80 时，delta=0.5 会是一个比较好的取值。

AlphaSocket：是系统所有 Socket 的集合，主要涉及有控制指令接受和处理的 TCPControl 以及负责图像传输 VideoStream，其中采用多线程的方式提高效率，其内置的方法有：

- openVideo(self, IP, Gray=1, qulity=50)：创建并开启图像传输流线程，IP 为上位机的地址，Gray 控制传输彩色还是灰度图，qulity 是每一帧图像的压缩程度。
- CloseVideo(self)：关闭图像传输线程
- CloseControl(self)：关闭总的网络控制

上位机的 AlphaGoClient 类负责建立连接，使用 Python3 的第三方库 pynput 实现键盘监听，并将指令传输给小车实现小车的网络控制功能。

3 功能实现

以上介绍了 Alpha-Go 的硬件和软件，下面着重介绍小车 Alpha-Go 的一些功能。这些功能有些是一开始设想好的，但是在实现的过程中受到系统资源的限制以及环境影响效果不尽人意，经过测试和优化，最终得以在性能和体验上折中后成功部署在小车上。

3.1 图像网络传输

（1）OpenCV+Socket 实现实时传输图像

使用摄像头需要在系统设置中打开摄像头接口，重启之后系统才会正确载入摄像头设备，我们通过访问 /sys 来查看是否成功载入摄像头设备。成功载入设备后，使用 Python3 和 OpenCV 读取摄像头数据（更详细的资料可以查阅 OpenCV 的官网）：

```
import cv2                          # 导入 OpenCV
cap = cv2.VideoCapture(0)           # 打开摄像头
ret, frame = cap.read()             # 读取一帧图像
cv2.imshow(“frame”, frame)          # 显示图像
k = cv2.waikey(10)                  # 监听键盘，输入 q 退出
if k==ord(‘q’):
    cv2.destroyAllWindows()
```

这里取出来的每一帧都是一个（480，640，3）的 uint8 矩阵，经过简单的计算我们的到一帧的原始数据量接近 1M，保证画面的流畅，我们应该保证 fps 保持在 30 以上，因此每秒的数据量接近 30M，这对于树莓派 3B 的处理能力来说有些吃力，后续经过实测也证明了直接传输将会带来很高的延时和很高的系统负担。由于我们是逐帧抽取，因此我们可以将每一

帧进行压缩再传输，这里是用 OpenCV 的 JPEG 函数来压缩每一帧。

```
encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), qulity]              # 设置参数
result, EncodeImg = cv2.imencode( '.jpg' ,frame, encode_param)  # 进行 JPEG 压缩
```

根据 qulity 设置的不同，其数据压缩比从 10:1 到 40:1，这样就大大减小了数据的传输量。但即使压缩了十几倍，我们打开视频流后仍会发现其帧率和时延仍在不可接受的范围内（平均帧率在 20 帧，画面延时在 400ms 浮动）。因此为了流畅的画面显示，还需要进一步缩小数据量。这里我们注意到虽然压缩成了 JPEG 大大减小了数据量，但是 JPEG 对于三通道的彩色图像是三个通道分别压缩再编码，如果我们转换成灰度图像，则可以进一步压缩数据量到原来的三分之一，最终测试结果满足预期，平均帧率达到 40 帧，画面平均延时控制在 150ms 左右。（这里的帧率和延时不仅仅是因为更换为灰度图，还有后续的客户端和服务器端的分配的选择）

（2）客户端还是服务器？

网络控制部分，由于我们电脑连接的是小车的 WiFi 热点，很容易得到小车的 IP 地址，因此我将网络控制的服务器端设置在小车上，电脑端作为客户端连接。由于控制部分需要比较可靠的指令传输和持续性的连接，因此建立 TCP 连接。

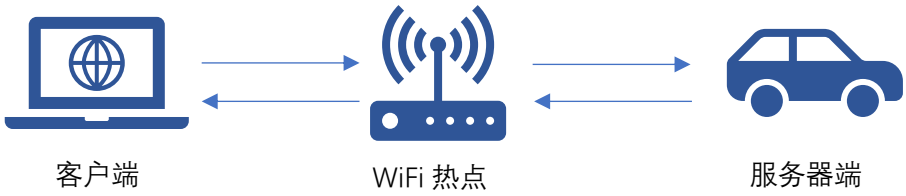


Figure 10 TCP **控制连接**

由于连接的是小车的 WiFi 热点，我设置 WiFi 的网络地址为 192.168.207.xxx，电脑端只需要通过 TCP 套接字连接 192.168.207.1 地址，就可以连接上小车的控制。为了提高效率和降低时延，TCP 单独创建一个线程用于接收指令。

网络连接中，除了要考虑 TCP 控制以外，还有一个图像传输连接。小车端负责数据获取、图像压缩和发送，电脑端则接收数据并解析，最终显示出来。这里考虑到 4 中模式：使用 UDP 协议，小车为客户端，电脑为服务器端以及小车为服务器端，电脑为客户端；使用 TCP 协议，小车为服务器端，电脑为客户端；小车为客户端，电脑为服务器端。分别测试帧率和时延，结果如下：图像均为灰度图。

协议	电脑端	小车端	平均帧率	平均时延
TCP	服务器	客户端	50fps	150ms
TCP	客户端	服务器	30fps	300ms
UDP	服务器	客户端	50fps	150ms
UDP	客户端	服务器	30fps	300ms

Table 2 **网络图像传输测试**

由于测试环境的变化以及系统负载的不同，可能数据有偏差，但呈现出来的趋势仍值得参考。从表中可以看到当小车作为服务器端时，总体的图像传输性能是比较差的，而作为服务器端性能有所增长。初步猜测是由于树莓派 3B 系统资源有限，在多线程情况下，开启图像服务器端有些吃力，使用客户端可以减小一些系统负载。由于我们连接小车的 WiFi 使用的是 DHCP 协议，并没有设置固定 IP，因此连接热点后分配到的 IP 地址并不一定每次都相同，把小车作为客户端则需要考虑如何让小车自动得到电脑端的 IP 地址。其实这里的解决办法十分简单，由于我们之前创建了 TCP 套接字作为总的控制，小车作为服务器持续监听，当产生连接时会返回电脑端接入的信息，我们可以让小车自动解析这部分信息得到电脑的 IP 地址（端口都是预先安排好的），这样就不用担心小车无法建立图像传输的问题。

Figure 11 **电脑端图像接收显示效果**

（3）优化控制指令传输

上面解决了网络控制和图像传输的大部分问题，但是在实际的操控中发现，当网络环境不好时，指令传达出现错误，或者是由于传输时延导致某些指令被跳过。因此在 TCP 接收指令部分额外增加一个缓冲区，并设置最大缓冲 8 个指令，超过 8 个指令则丢弃，以保证当网络出现问题时，指令仍按照顺序执行，并提高操控的可靠性。

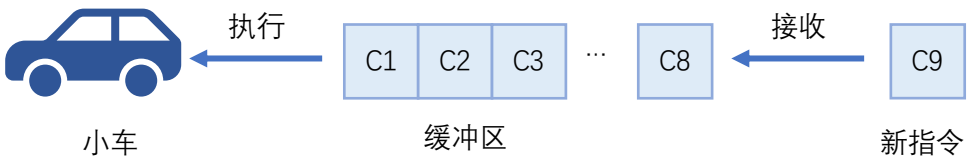


Figure 12 TCP **控制缓冲区示意图**

实践证明，增加一个缓冲区提高了小车操作的可靠性，在一次性输入多个指令的时候仍可以按顺序执行这些指令。也测试过使用 16 甚至更大的缓冲区，但是提升并不明显，初步猜测是 8 指令缓冲已经足够当前的网络环境使用。使用更多的缓存会增大系统的存储开销。

3.2 基于红外传感器的智能制动

在操控小车的过程发现，当小车驶出一定距离后，WiFi 信号较弱，指令传输的时延增大，视频传输也出现卡顿，摄像头的画面有较大的延时，此时若遇上障碍物则无法及时刹车产生碰撞，对于系统而言不够安全。正好小车前方有两个红外传感器，可以用于障碍物检测。为此机遇前面的两个红外传感器设计了智能制动功能。

该功能已经封装在 MotorModule 之中，其工作流程是开辟一个新的线程以一定的频率监测红外传感器的输出，如果感知到障碍物，则传感器输出一个高电平，此时锁住小车的前进功能，小车自动停止。由于在前面的两个传感器需要预留出一段距离供小车刹车，在比较狭窄的地方会因误检测导致车辆无法前进，因此采用可以手动开关的多线程而不是采用 GPIO 的电平边缘检测触发回调函数的方法。在比较狭窄的地方可以关掉检测，通过自己高超的操作技术控制小车。

3.3 基于摄像头的自动循迹

Alpha-Go 小车地盘自带了四个用于循迹的红外传感器，且已经提供了例程，因此这里不打算使用红外传感器的方法。正好小车配备了摄像头，因此考虑用摄像头来实现自动循迹。由于循迹是在白色地板上跟着黑线走，为了保持小车沿着线走，我们需要从摄像头的画面中提取出我们需要的特征。由于路线是白底黑线，因此我们可以对摄像头的画面进行二值化，将画面分成三个部分，分别是左白区，右白区以及中间的黑线区。其中黑线区域的宽度决定小车行驶的误差容忍程度，在图像二值化之后，在事先分配好的左/右白区域数白色像素的个数，若左边白色像素的点>右边白色像素的点，表明小车偏右行驶，此时应该让小车适当向左转向，在继续前行。反之同理。

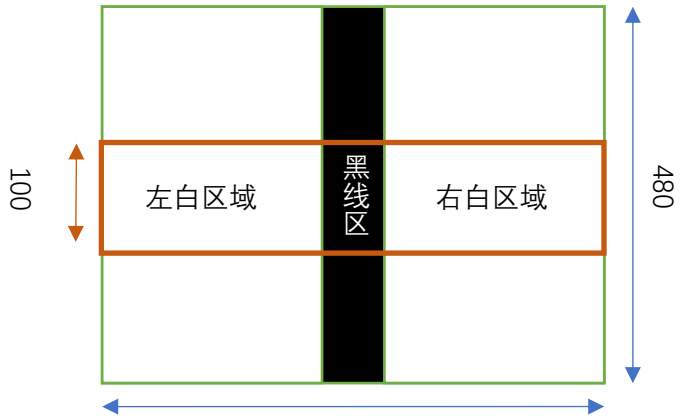


Figure 13 **画面分割示意图**

然而这个方法在实际运行的时候效果令人担忧，分析其原因有三点：1、小车本身的摄像头是歪的，导致简单分区域计算时，即便小车位于黑线中央，仍会出现误判的现象。2、树莓派 3B 在大规模矩阵的计数上速度不够快，导致小车系统负载高，运行卡顿。3、对于转向控制没有细化，存在过度转向无法回正的现象。针对第二点，我们采用裁取部分画面来进行处理，减小数据量。针对第一点，在我们裁取部分画面的基础上，扩大黑线区的范围，试图缓解摄像头歪带来的影响。针对第三点，我们则需要根据白色像素点个数的差异动态调整旋转的角度。

目前只解决了 1 和 2 的问题，然而效果仍不尽人意，并存在使用电池供电不足 WiFi 断掉的情况，因此认为目前的解决方法受到树莓派系统资源的限制无法流畅运行。今后会考虑使用新的方案以及实现动态调整转动角度。

4 系统的不足和改进

上面介绍了 Alpha-Go 智能小车的硬件资源、软件架构以及实现的一些功能。但是系统仍存在一些缺陷，下面给出系统的一些待改进的地方。

4.1 硬件和网络传输

上面具体介绍了小车的网络控制和图像传输功能，实际体验下来虽然满足基本的需求，但是仍有一些不足：

1、 在功能全部打开的情况下，系统负载高，导致接收处理指令、图像传输等需要比较消耗资源的功能出现卡顿。
2、 为了保持图像传输的流畅性和高帧率，只能传输灰度图。
3、 摄像头循迹仍有不足。
4、 设备交互有待完善，上位机没有界面用户体验较差。

后续可能采用的解决方案：优化系统架构，减小不必要的系统计算和存储的开销，减小摄像头画面大小，尽可能在不影响正常使用的情况下减小所需要处理和传输的数据量。优化多线程协同，加快处理速度。对于图像传输，考虑到画面传输中，如果没有变化巨大的画面，则大部分数据是冗余的，即使采用 JPEG 的压缩算法压缩每一帧的数据，帧与帧之间的信息大部分是重合的，因此考虑使用视频流的压缩格式来替换目前的逐帧压缩传输方案。但是可能会对上位机端提出更高的要求。摄像头循迹部分表现不佳的原因和改进想法已经在 3.3 中介绍了。

目前上位机只实现了核心功能，用户体验较差，将来考虑设计和开发一个对应的 UI 界面，以及拓展上位机的平台，让其能够用手机和平板来控制，大大提高设备的交互和用户体验。

除此之外，小车还有一些硬件资源（例如蜂鸣器、红外遥控器接收、串口、蓝牙等）没有充分利用起来，后续会继续完善和开发好玩有趣的功能来充分利用现有的硬件资源。

4.2 软件架构设计

当前的软件结构是根据模块化的思想，其结构图可参考 Figure 9。虽然在项目初期方便组装，但是在功能复杂情况下，各个类别数据不能交互，只能通过返回值和全局变量进行交互，这对于后续复杂功能的开发有些阻碍，因此未来考虑使用新的架构，方便功能的拓展。