基于 STM32 的线性 CCD 舵机循迹智能车的研究与探索

刘尚育

（东北大 16 级物联网工程， 沈阳）

摘要：

这是一篇披着论文外衣的博客，留存了论文朴素的版式，却省去了繁文缛节。它是对新型实验报告的尝试，同时也是对网站后台能同时支持 Markdown 语法和 Word 语法的一种炫耀。

在下文中我将主要介绍 18 年下半年来每个星期一次的研究：基于 STM32 微控制器的 CCD 舵机循迹智能车。在上半年光电传感器配合舵机循迹的深刻失败后，我意识到 CCD 传感器与舵机的配合是命中注定。高分辨率的 CCD 采集的数据可以近似看做连续量，与之相对应，舵机的转向角度也是完美的连续量，而四个突兀的红外对管的数字量就与之格格不入，下文中你将体会到这一点。

下文将从知识铺垫、部件调试和转向逻辑三大方面展开。知识铺垫将讲述研究开始前期对整个项目的评估与知识储备计划；部件调试将讲述各个部件的分部调试过程；转向逻辑将讲述万事俱备后组件之间的配合方法。最后还将谈到未来的改进方向和初步设想。通过本文，你可以了解这个项目的整体流程和相关知识，获得一些实践总结的非官方观点，但不会看到详细到每步代码的教程。

关键字 STM32 线性 CCD 舵机 循迹

一、知识铺垫

在学期开始的时候，我拿到这个题目，通过快速浏览相关论文（来自 CNKI），总结了以下几项需要储备的知识点以及学习它们的先后次序：

（1）无线串口通信
（2）ADC 模数转换原理与实现
（3）线性 CCD 传感器工作原理
（4）CCD 图像处理算法
（5）循迹算法
（6）编码电机的速度提取
（7）PD 舵机控制
（8）PID 电机控制

这是在默认你已经掌握了 STM32 的基本语法、舵机的控制原理和 TB6612FNG 驱动板的使用方法的前提下列出的。如果你还没有掌握以上知识，请关闭这篇文章并开始更基础的学习，遇到相关问题可以参考：基于 STM32 控制的舵机车循迹的研究及相关问题[1]这篇文章。

在以上列出的 8 点中，只要掌握了前 5 点就可以实现循迹功能，但要追求速度和平稳的极致，还需要对后 3 点以及第 4 点进行深入研究，并加入自己的一点小创造。博主就只研究完了基础功能的循迹，本小节中介绍前 3 点，第二节介绍第 4 点，第三节介绍第 5 节。

1.1 串口通信

之所以要调通串口，是因为这是未来研究中不可避免的基础知识。硬件调试不像软件能够输出中间结果信息，故需要通过串口返回 PC 端查看。硬件的串口可以看做 C 语言 printf 一样的功能和地位，而实际上实现过程中，串口的输出函数名也的确是 printf。

我们这里讨论的串口是指 USART（Universal Synchronous/Asynchronous Receiver/Transmitter 通用同步/异步串行接收/发送器）。实现过程主要用到了串口中断功能，可用的例程代码在这里：STM32 Usart 可用例程[2]。使用步骤：

（1）首先将文件包含到 keil 工程中。

（2）完成连线，使用引脚 USART1_TX-GPIOA.9，USART1_RX-GPIOA.10

（3）在 main 函数中加入 uart_init(9600);这里 9600 指的是波特率->

在信息传输通道中，携带数据信息的信号单元叫码元，每秒钟通过信道传输的码元数称为码元传输速率，简称波特率。波特率是指数据信号对载波的调制速率，它用单位时间内载波调制状态改变的次数来表示(也就是每秒调制的符号数)，其单位是波特（Baud,symbol/s）。波特率是传输通道频宽的指标。

因为信息在传输时会经过压缩（调制），所以每个调制状态可能对应多位，由此比特率会大于等于波特率。

比特率 = 波特率 * 单个调制状态对应的二进制位数

可以不使用 9600，但一定要保证程序设定值与串口助手中的波特率匹配，这好比编码和解码，如果不一致就会出现乱码。

（4）单片机向 PC 端输出使用：printf(“hello world”);使用方式与 C 语言大致相同。单片机接收 PC 端输入：Res = USART_ReceiveData(USART1);响应动作写在 usart.c 中的 modify_debug()函数中。

注意事项：

- ✧ 单片机和 USB 转串口 TTL 的 RXD 和 TXD 引脚要交叉相连。
- ✧ 在 JTAG 线连接到 PC 机后，USB 转串口的串口号可能发生变更，建议使用串口通信时不要同时连接其他 USB 设备
- ✧ **千万不要使用 USB3.0 接口**，多数电脑有多个 USB2.0 接口和一个 USB3.0 接口，因为连接了 USB3.0 而怀疑整个实验室的 TTL 都是坏的是一个惨痛的悲剧

有关蓝牙通信

通过使用 HC05 串口转蓝牙模块，我们可以在掌握串口通信的基础上轻松实现蓝牙通信。只需将 USART 的 TX、RX 引脚转接蓝牙对应的 RXD，TXD 即可。但需要注意的是，在第一次使用该模块时需要进行 AT 指令调试，设置蓝牙的名称、PIN 和主从工作模式。具体方法参考这篇文章：蓝牙的使用过程研究[3]

注意事项：

- ✧ 蓝牙的收发不是同一个串口

Python 串口辅助

python 是一门简练而多功能的编程语言，很适用于做各种辅助工具。python 的 pyserial 包提供了对指定串口的读写方法。可以参考这篇文章：python 串口通信实验[4]。由于 python 的 matplotlib 库提供了很好的可视化接口，numpy 库提供了很好的矩阵处理，所以 python 的串口实验为后来的 CCD 图像调试提供了重要铺垫。

1.2 模数转换

模数转换是传感器的模拟量与单片机数字量之间的桥梁。参考这篇文章：STM32 ADC 的相关研究[5]。注意代码中的 adc_init()函数需要放在程序的开头，但要在 system _init()和 delay_init()之后。

u16 get_Adc(u8)函数的参数为 ADC1 的通道（对应引脚可查表），返回值为

一个 u16 类型的整型数。由于 STM32 的 ADC 为 12 位,所以返回值范围为 0~4095。

将文章中的例程加入工程，完成接线，将 PA0 作为模拟电压输入端口，通过 get_Adc(u8)就能得到程序可以处理的整数了。输入电压范围在 0~3.6V，假设所接外设能达到的最高电压为 3.3V，则当前电压 V=(float)get_Adc(0)*(3.3/4096)。

接下来我进行了上学期的遗留课题的研究：外部调参器的实现与应用。简要回顾以下，外部调参器实现了外部调节电位器修改程序内部参数的功能，是程序内外的接口。主要应用在参数优化调试过程,以及需要根据环境调节参数的程序。在比赛中，赛道的材质、环境的照度和电池的电量都会影响参数，使用外部调参将大大简化了程序改参的流程。具体电路和实验过程可以参考刚刚那篇文章：STM32 ADC 的相关研究[5]。

1.3 线性 CCD

这是所有铺垫知识中最难以下手的部分。我们使用的是 TSL1401 线性 CCD 传感器,相关的资料只有一篇纯英文文档和几篇互相抄袭载着机翻中文的劣质博客，虽然生产厂家是中国企业。于是我仔细阅读并翻译了文档中的要点，见这篇文章：TSL1401 线性 CCD 传感器的原理研究[6]主要介绍了 TSL1401 的组成原理和工作时序，总结了输入时钟频率的约束条件等问题。

这部分的研究在后期的指导意义其实不大,因为后期所用的例程规定了频率的经验值。如果你不想看懂例程中的每一行代码，而只是想知道大概过程，就不必太留意这部分内容，但至少要知道线性 CCD 是串行输出 128 个电压值的，而每个电压值就代表对应位置的灰度值。

二、部件调试

2.1 CCD 图像处理

线性 CCD 传感器采集数据后，传送给 STM32 控制器处理，但在处理过程中的数据对于我们是不可见的。为了调试并了解 CCD 的工作状态，我们需要将它传回的数据展示在 PC 端。参考文章：Python 上位机接收 CCD 图像实验[7]，首先按照文章中介绍的进行接线，并在工程中加入 CCD 配置的代码，这样 CCD 的 AO 将输出到 STM32 的 ADC 引脚，STM32 内部将 128 位串行数据存入缓冲区数组 ADV 中。我们再利用之前讲过的串口通信方法，将 ADV 的数据 printf 到 PC 端。下面我们讨论 PC 端如何实现可视化。

可视化

我们利用 Python 的 matplotlib.pyplot 库进行图像绘制，这在上文已有所铺垫。具体例程在上文提到的参考文章[7]中，为了方便这里重新给出。

```
import numpy as np
import matplotlib.pyplot as plt
import serial

fig, ax = plt.subplots()
ser = serial.Serial("COM5", 9600, timeout=0.5)
img = [[0 for i in range(128)] for j in range(128)]
plt.ion()
while True:
    tmp = ser.readline().strip().split()
    if len(tmp) != 128:
        continue
    data = [1 - int(t.decode()) for t in tmp]
```

```
            # print(data)
            img.pop(0)
            img.append(data)
            ax.cla()
            ax.axis('off')
            ax.imshow(np.array(img), cmap=plt.cm.gray_r)
            plt.pause(0.001)
    ser.close()
```

可视化的难点在于如何将一个线性的图像转换成我们能够理解的矩阵图像，并动态刷新。我们需要转换思维方式：小车在赛道上沿着黑线行进时，会扫描垂直于行进方向的128个位置，在时间域上看，黑线在128个像素空间内左右晃动，但从行进域上看，每一条128像素的图像中的黑线部分共同组成了一个完整的黑线。所以，我们可以建立一个128x128的图像区域，维护含有128个灰度向量元素的队列。每有一个新的灰度向量被接收，我们就从队列里去除一个“过期”的灰度向量，并将新的加入进来。

具体的动态更新的实现有赖于代码中plt.ion()方法，它的大概意思是开启所谓的交互模式，在这个模式下，ion之后的循环，每次会擦除原有的图像，并绘制新的图像。这个模式下，使用imshow方法可以直接输出图像，而不需要原来的show函数。imshow可以将numpy数组中的内容直接当做图像数据进行绘制。

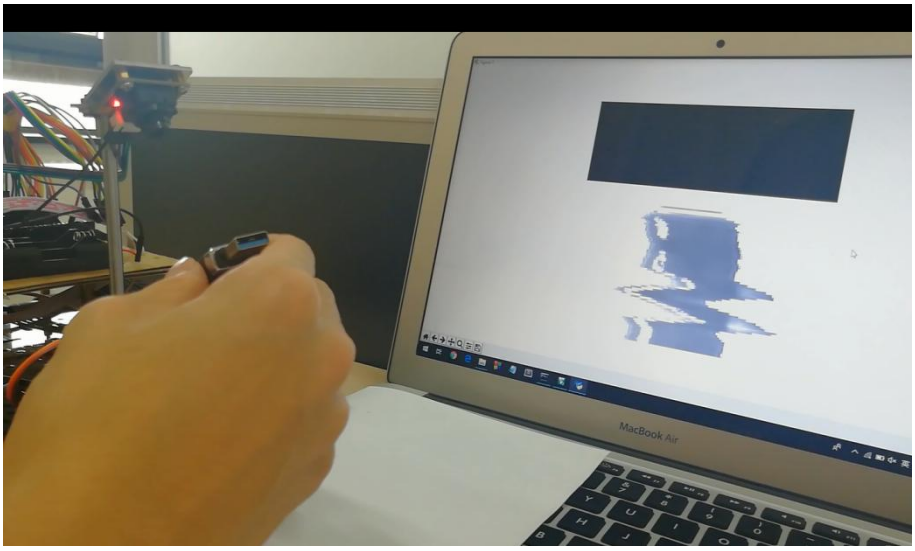

图2-1 灰度可视化效果

注意事项：

- ✧ 文中所用的代码的不足之处在于，有plt.ion()而没有plt.ioff()，也就是说如果关掉可视化窗口，程序不会终止，你需要从命令行强制终止。如果想实现的更完美，可以使用matplotlib自带的button，并为它绑定按键监听函数，在点击stop时，调用plt.ioff()
- ✧ 如果你选用的是蓝牙方式进行通信，要注意蓝牙的收发速度会比较慢。相对而言串口转USB更可靠和便捷
- ✧ 在调试结束后，上赛道测试时一定要将串口通信关闭。因为串口调用中断收发信息，会极大的影响小车的判断速度，小心翻车。

二值化

由于环境光线和赛道洁净度的不同，所得到的灰度值实际上是黑线和白色背

景的近似。我们需要将黑线与背景严格划分，从而抽象出黑线的具体位置。

图像二值化的方法有很多，这里采用最基本的方法：第一遍扫描求灰度向量的平均值，第二遍扫描将小于平均值的灰度值改写为 0，将大于平均值的灰度值改写为 255，完成。

```
// 二值化算法
for (j = 0; j < 128; j++) {
    if (ADV[j] == 0XFF) ADV[j] --;
    data[j] = ADV[j];
    //printf("%d ", data[j]);
    mean += data[j];
}
mean = mean >> 7;
for (j = 0; j < 128; j++) {
    if (data[j] < mean) data[j] = 0;
    else data[j] = 255;
    //printf("%d ", data[j]);
}
```

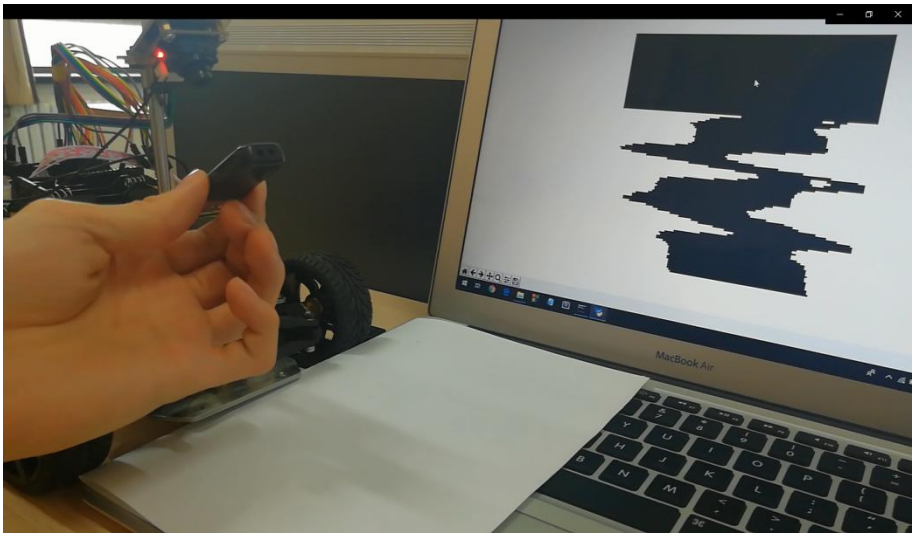

图 2-2 二值化可视化效果

注意事项：

- ✧ 二值化边界问题。为了降低硬件开销，程序中没有使用浮点数来表示平均值，这加剧了一个问题：在灰度向量的元素值相差不大时，程序仍然会对黑白进行划分。最极端的情况是，小车遇到全白的情况时，平均值为 255，与每一个元素值都相等，故 ADV[j] < mean 对于每一个 j 都不成立。所以程序中这里使用的是小于号，而不是小于等于号，要格外注意。并且在后文中对二值化结果进行处理时也应定要考虑全白的情况。至于不考虑全黑是因为黑线的宽度一般不会大于 CCD 采集总宽度。
- ✧ 在下载程序或 USB 串口调试时,需要注意这两种线都带有 5V 供电功能。这时虽然要使用电池给 CCD 供电,但要断开电池与 STM32 开发板的连接，否则可能电压倒灌烧坏板子和电源。

2.2 电源功率问题

在同时驱动多个电机或舵机时，电源电压值虽然足够，但功率供应不足，导致最终多个部件都不能正常工作的问题，就是所谓的电源功率问题。关于这个问题我已经在文章[1]中讨论过了。

我们使用两个电压要求 12V 以下的编码电机，一个 5V 的 MG995 舵机，一个 5V 的 CCD 传感器和一块 3.3V 的 STM32 开发版。整车使用钢结构和亚克力板，车重较大。在使用三节 14500 电池供电时，供电电压 10V，但在快速转向时很容易出现舵机停转然后电机停转，最后 STM32 电源指示灯灭掉的情况。只有在车速为 PWM60%的情况下能够勉强正常运转。上学期的研究指出，改进方法有三种：

（1）并联电源法：再加入几节电池和原来的电池并联以提高电流。但这个方法需要并联的电池的电压相近，否则会出现环流短路。假如两个电池的内阻很小，电流将很大，会把电池烧坏。环流将从较高电压的电池出来倒灌进入较低电压的电池。环流的大小 = 电压差 / 两电池内阻的和。显然每个电池的放电曲线都不相同，我们无法控制并联电池电压相同，所以尽量不要使用这种方法。

（2）独立电源法：使用两套电源分别给电机和舵机供电。根据上学期的试验，在电压低到 4.7V 时仍然可以正常工作。但这种方法虽然解决了并联电源不安全的问题，却和第一种方法一样增大了车重，消耗过多的资源，复杂化了车上的布局。

（3）更换电源吧：使用放电倍率高的航模电池。放电倍率指的是最大放电电流与电池容量的比值，常用 C 表示。原装循迹舵机车使用的就是航模电池。航模电池是一种锂聚合物电池，而 14500、18650 是锂离子电池（磷酸铁锂）。至于两者的区别，个人理解主要有以下：锂离子电池，使用电解液作为介质，钢壳包装，能量密度低，价格便宜，需要远离热源，避免过充过放；锂聚合物电池，使用胶态固体作为介质，层叠形式出现，可塑性好，铝塑膜包装，能量密度高，造价较贵，较为安全。但最终要的一点是，虽然两者的电压相当，但 14500 的放电倍率在 0.2C~0.5C,而航模电池能达到 10C 甚至几十 C。总体来说在同等情况下，锂聚合物电池性能要比锂离子电池高 10%。这是当前最有效的解决方案。

三、转向逻辑

3.1 巡线中点计算

在控制小车循迹时，我们通常设定一个理论前进方向，通过传感器获得小车的当前前进方向,然后使用控制算法使小车的实际前进方向不断地趋向理论前进方向，这就是巡线逻辑的本质。

首先根据小车的实际情况，我们确定理论前进方向。一方面我们要考虑传感器的安装位置，即小车直行时，黑线是否在 CCD 采集区域的正中央；另一方面我们要考虑动力问题，即小车左右轮转速是否天然一致，舵机左右转向能力是否一致。一般来说，理论中点要设置在靠近能力强的一边。这样一旦行进方向向能力弱的一方偏离黑线,将偏离理论中点更多,这样就有足够的时间让小车进行反应。

然后我们需要根据 CCD 采集的数据计算实际的中点位置。我们可以以小车为参考系，把小车脱离黑线的问题转为黑线脱离小车中点的问题。那么所谓的小车中点就是上述的理论中点，而当前黑线的中点就是实际中点。我们可以先假设当前全白，将黑线左边界和右边界都设置成 128。从左到右扫描，当第一次出现黑色，存入左边界；之后第一次出现白色，存入右边界。最后取左右边界的中值作为黑线中点。

```
flag = 0;
l_index = r_index = 128;
for (j = 0; j < 128; j++) {
    if (data[j] == 0 && flag == 0) {
        l_index = j;
        flag = 1;
```

```
        }
        if (data[j] == 255 && flag == 1) {
            r_index = j;
            break;
        }
    }
    mid = (l_index + r_index) / 2;
```

注意事项:

- ✧ 下文在处理时仍要留意全白的情况时,黑线的中点被判定在128位置上。我的实验结果在右转时会出现问题,主要是舵机转向时没有考虑这一点。实际上,全白的情况无非是转向来的太快,舵机没反应过来,这时只需要保持舵机原有角度不变即可。
- ✧ 如果要考虑采集到的“脏”数据,还要对原有灰度向量进行预处理。在赛道不干净的情况下,黑线上可能有少许灰尘导致灰度值上升,而白背景上的灰尘会导致灰度值下降。二值化可以处理掉大部分情况,但在赛道真的脏的不行的时候(到底是谁一天天总踩赛道),可能在二值化结果中出现黑线或白背景中断的情况,这时需要对断点进行填充。填充方法在此不详述(论文里都有)。

3.2 舵机转角计算

我们知道了理论中点和实际中点后,剩下的就是控制逻辑了。舵机的连续与CCD的近似连续的搭配就将在这里体现出来。我们可以知道,在两个中点偏离最多的时候,舵机转角最大,偏离最小的时候,舵机转角最小,而两个的变化过程都是连续的,那么就可以建立两者的函数映射关系。

```
angle = function(real_mid - theory_mid) + servo_mid;
angle = angle > 160 ? 160: angle;
angle = angle < 80 ? 80: angle;
set_servo_angle(angle);
```

其中real_mid是实际中点,theory_mid是理论中点,servo_mid是舵机中点角度。function是驱动函数。下面的两句是限制舵机转角不越界。最后一句是更新舵机角度。

关于舵机转角越界。我们使用的舵机是180度舵机,角度范围应该在[servo_mid-90, servo_mid+90]。但实际上,由于小车转向的机械结构限制,角度范围往往达不到那么大。如果不设置越界检测,那可能会烧坏舵机。

关于转向驱动函数。驱动函数是中点差值对角度变化的驱动,它决定了在多大的差值下应当转多大的角度。首先我们可以使用线性驱动:

```
function(real_mid - theory_mid):
(real_mid - theory_mid) * K / max_delta
```

其中K表示驱动系数,K=1时按照差值占最大差值情况的比例,等比例的转角。K越大,转动越迅速,对差值越敏感。max_delta是最大差值情况。比如理论中点设为63时,最大差值只能是63。

线性驱动方法在低速下运行稳定,简洁迅速,但无法处理高速情况。

在小车高速行进时,我们希望小的差值只需要微小转动,而大的差值要迅速反应。这时我们可以使用非线性的驱动函数,如三次函数:

```
function(real_mid - theory_mid):
(real_mid - theory_mid)^3 * K / max_delta^3
```

这里要格外注意的是如果使用偶数次幂,要保留差值的正负,以确定转向方

向是向左还是向右。具体的函数形式还有待进一步研究。对于特定的车况，还需要反复实验，以确定最优参数。

四、结语

这半年的研究算是取得了一定进展，学到了很多实用的技术。如果让我来评判这半年的工作效率，那么对于每周一次的时间而言，工作效率还是很高的。当然我希望能投入更多的时间和精力，学习更多的知识。中途被 FPGA 和 CC2530 的失败耽搁了一段时间,而后由于考试、实验和课设的步步相逼时间越来越紧张。我希望能够将小车调到更好，也希望能将我最失败的两个研究继续下去

小车的性能还没有达到最优。下一步的研究应该是更换电池、修改全白的特殊情况和试验最优的驱动函数。如果这些步骤进行完毕，小车仍然没有达到最理想的情况，还需要再多查阅一些相关的论文，查找解决方案。最后还要对小车的布线和重量布局进行优化，甚至可以给它做一个车壳来圆满收官。