

机器人概论 小组 4

# 刚体机器人

——以自平衡小车的制作为例

【小组成员】 陈泊帆 居星宇 王梓鉴 王章昊

2021-12-24

## 目录

一、	选题说明.....	2
二、	刚体机器人的骨架——自平衡小车的机械结构 .....	2
三、	刚体机器人的灵魂——自平衡小车的 PID 控制.....	3
	1. 平衡原理.....	3
	2. PID 简介 .....	5
	3. 串级与并行 PID 控制比较.....	6
	4. 自平衡小车的调参过程——以电机速度内环闭环 PID 控制为例 .....	8
四、	刚体机器人的小脑——STM32 单片机在自平衡小车上的应用.....	10
	1. 命名与用途 .....	10
	2. 如何使用 STM32 开发.....	11
	3. 小车控制代码.....	14
五、	刚体机器人的大脑——利用树莓派拓展小车的功能 .....	20
	1. 上位机简介——以树莓派为例 .....	20
	2. 如何建立树莓派与 stm32、PC 间的通信.....	20
	3. 二维码图像处理功能展示.....	21
六、	反思与总结.....	21

## 一、 选题说明

该选题的主题是刚体机器人。作为典型的刚体机器人之一，自平衡车既具有刚体机器人的特征，又在运动控制方面有着较高的要求。我们希望通过这次实践尝试自平衡车的制作，从而积累刚体机器人的经验，熟悉机器人控制常用软件，并尝试运用所学知识对小车进行改造，掌握机器人控制方法，提升实践能力。本文将从结构到控制，分别介绍自平衡车的机械结构、平衡 PID 控制方法、STM32 单片机控制的应用以及通过树莓派实现的扩展功能。

## 二、 刚体机器人的骨架——自平衡小车的机械结构

自平衡小车由主控制区、直流电机和轮胎组成。其中小车的车体部分包括了电池和主控板，主控板上包含了小车的各个基础模块和扩展模块（核心板、传感器模块、驱动模块、稳压模块、蓝牙模块、超声波模块、四路红外模块、OLED 模块等）。

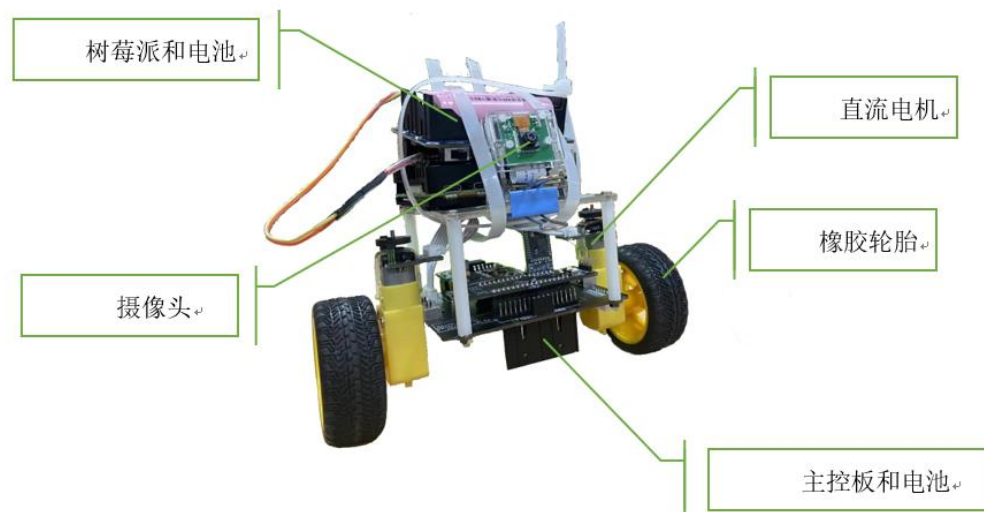


图 1 自平衡小车的结构

系统框架示意图如下：

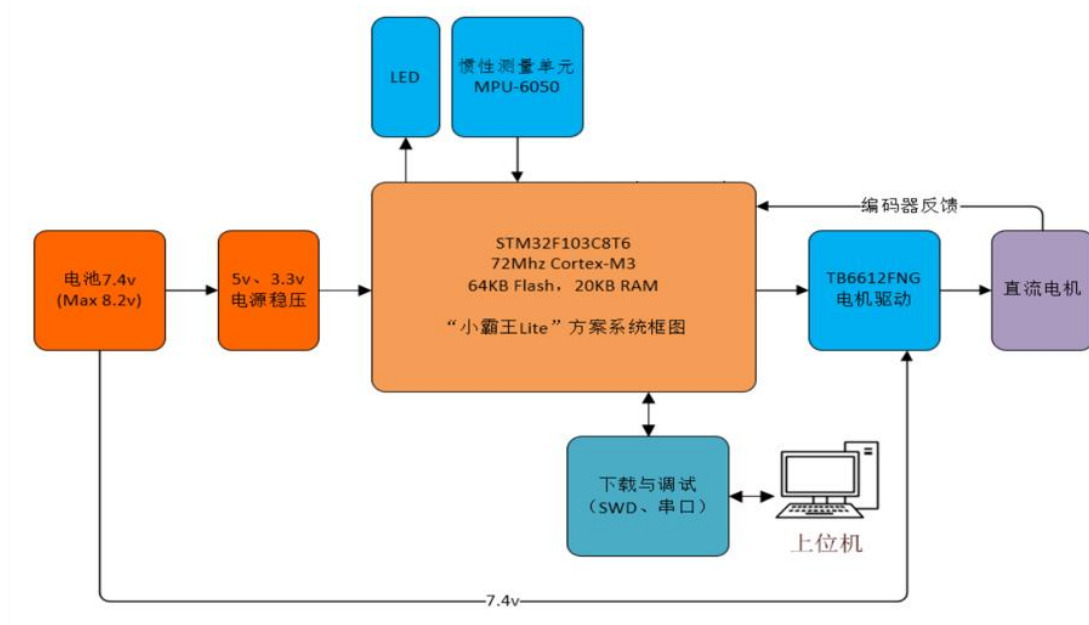


图 2 自平衡小车的系统框架示意图

### 三、 刚体机器人的灵魂——自平衡小车的PID控制

#### 1. 平衡原理

以下是自平衡小车的一个简化模型，我们可以看到，可以将它视作一个倒立的单摆。通过建立简单的微分方程可以得知，单摆的平衡主要由两个力来实现，一个是由重力分力提供的回复力，另一个是由空气阻力提供的阻尼力。然而，在倒立摆模型中，重力的分力与倒立摆偏移方向相同，无法为倒立摆提供回复力，因此我们需要通过小车上的电机产生额外的力来提供这个回复力。

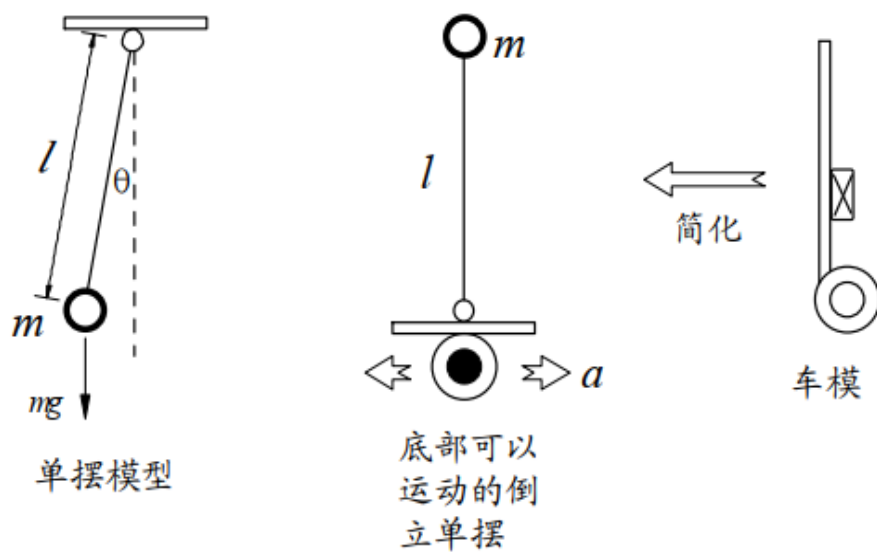


图 3 自平衡小车的简化模型——倒立摆

我们提供回复力的方式是让车轮向着小车倒下的方向加速运动，这样以车轮中心为参

考系，我们就可以发现倒立摆受到了一个惯性力，这个惯性力的分力可以用来平衡正向的回复力，即小车受到的总回复力为

$$F = mg \sin \theta - m a \cos \theta \approx mg \theta - m k_1 \theta$$

这里由于 $\theta$ 很小，所以作了线性化。 $k_1$ 为可以设定的系数。如果 $k_1 > g$ ，就可以得到与运动方向相反的回弹力。

由于空气阻力很小，我们还需要由电机提供额外的阻尼力以使小车尽快稳定下来。因此我们在上面的式子上再加一项与小车倾斜速度成比例的项，得到

$$F = m g \theta - m k_1 \theta - m k_2 \theta'$$

这样，自平衡小车的受力方式就与倒立摆相同，只要 $k_1 > g$ 并且 $k_2 > 0$ ，小车就可以稳定到垂直平衡位置，我们可以进一步调节这两个参数来让小车的平衡性能更好。

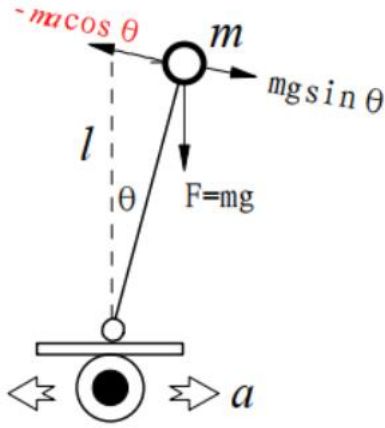


图 4 倒立摆的受力分析

下面给出更为具体的数学建模，将小车视为高度为  $L$ 、质量为  $m$  的倒立摆，并设车轮加速度  $a(t)$ 、外力引起的角加速度  $x(t)$ ，得出下列式子：

$$L \frac{d^2 \theta(t)}{dt^2} = g \sin[\theta(t)] - a(t) \cos[\theta(t)] + L x(t)$$

角度较小时可以简化为：

$$L \frac{d^2 \theta(t)}{dt^2} = g \theta(t) - a(t) + L x(t)$$

如果无外力冲击，静止时 $a(t) = 0$ ，所以方程为

$$L \frac{d^2 \theta(t)}{dt^2} = g \theta(t) + L x(t)$$

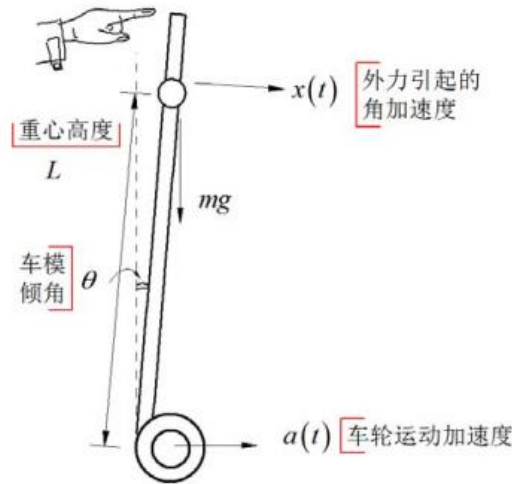


图 5 倒立摆的受力分析 2

由此可算出小车静止时传递函数为 $H(s) = (\theta(s)/x(s)) = 1/(s^2 - g/L)$ ，此时系统有两个极点 $s_p = \pm \sqrt{g/L}$ 。

该传递函数有两个零极点，有一个在  $s$  平面的右半平面。这说明两轮自平衡小车是不

稳定的。在引入比例和微分反馈后系统如图所示：

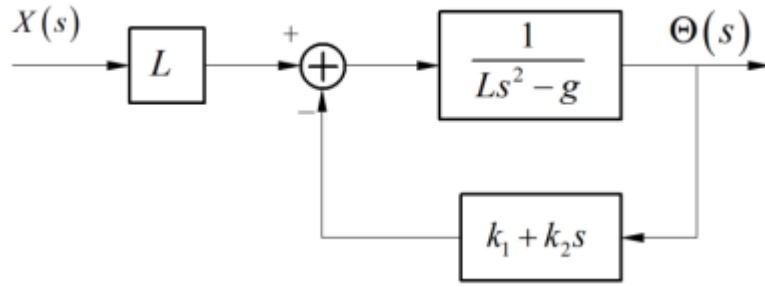


图 6 加入比例和微分后的系统框图

此时传递函数变为

$$H(s) = \frac{1}{s^2 + \frac{k_2}{L}s + \frac{k_1 - g}{L}}$$

得出系统两个极点位于  $s_p = \left( \frac{-k_2 \pm \sqrt{(k_2^2 - 4L(k_1 - g))}}{2L} \right)$  处。

若要求系统稳定，两个极点都应位于  $s$  平面的左半平面，则  $k_1 > g$ ,  $k_2 > 0$  此时系统传递函数的极点都已经分布在了  $s$  平面的左半平面，只要参数适合，极点的位置将会进一步远离 0 点，系统将会更稳定。

## 2. PID简介

PID( Proportional Integral Derivative)控制，即“比例、积分、微分控制”，是工业控制领域常用的自动控制方法，因其算法简单、稳健和可靠性高，被广泛应用于工业过程控制，我们这里的自平衡小车就采用这种控制方法。

PID 控制算法的原理图如下。其中， $r(t)$ 是我们想要得到的一个稳定的值（对于自平衡小车来说，是小车的行进速度和倾斜角度）。 $y(t)$ 是系统实际的输出值（这里是小车的实际速度和倾斜角）， $e(t)$ 是误差，即  $r(t) - y(t)$ ，PID 算法中将  $r(t)$ 经过比例、积分、微分三种处理得到的值叠加作为控制量输出给电机，驱动直流电机改变其转速。

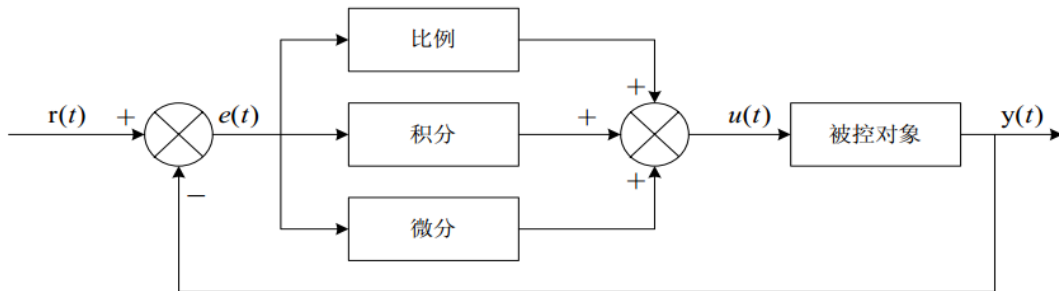


图 7 PID 控制算法示意图

模拟 PID 控制器的具体控制规律为

$$u(t) = kp(e(t) + \frac{1}{T_i} \int e(t)dt + T_d \frac{de(t)}{dt})$$

其中，比例项的作用是根据当前的误差值作出反应减小误差，积分项的作用是减小稳定误差，微分项的作用是对当前系统的瞬间状态作出反应，以在误差变得更大之前及时消除误差。简单来说，这三项分别是对现在、过去、未来的反应。在实际应用中，常常将公式离散

化，得到

$$u_k = K_p e_k + K_i \sum_{j=0}^k e_j + K_d (e_k - e_{k-1})$$

上面的控制算法输出的是全部的控制量，即全量，这种控制方法计算量较大，而且在实际应用中存在风险，因此实际中经常应用增量式 PID 算法，即输出的只是控制量的增量  $\Delta u_k$ 。我们来简单推导增量式控制算法的输出量公式：由前式可以推出第  $k-1$  个时刻的输出量为

$$u_{k-1} = K_p e_{k-1} + K_i \sum_{j=0}^{k-1} e_j + K_d (e_{k-1} - e_{k-2})$$

两式相减并整理，就可以得到增量式算法的控制式

$$\Delta u_k = u_k - u_{k-1} = A e_k + B e_{k-1} + C e_{k-2}$$

这里的 A, B, C 为常数。这种算法的计算量比全量式算法小得多，因此在实际中得到广泛应用。我们的自平衡小车使用的就是这种增量式 PID 控制算法。

### 3. 串级与并行PID控制比较

要实现两轮自平衡车的运动与平衡，除了基本的速度控制外，还同时需要角度控制。速度和角度的控制具有串级控制和并行控制两种控制方式。

#### (1) 自平衡车的并行控制

并行 PID 控制，简单来说就是分别对速度和角度进行 PID 控制，使其接近期望值，再将控制结果同时输出。在此过程中，角度控制与速度控制分开进行，联系较少。

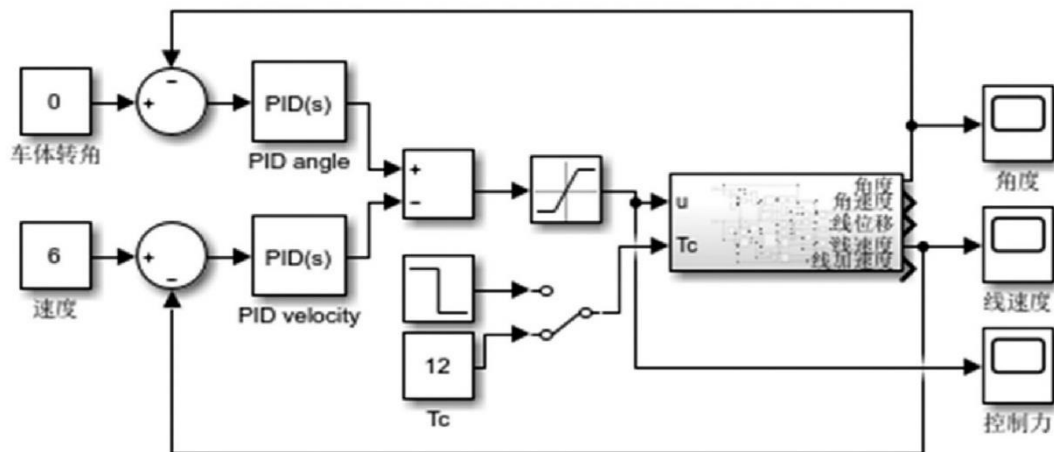


图 8 常用并行 PID 框图

#### (2) 自平衡车的串级控制

串级 PID 控制,也就是本自平衡车所用的控制方式，则是通过串联两个 PID 控制器，形成内外环控制，在干扰未能影响外环控制器时先使用内环控制器进行大致调节，再通过外环控制器实现更加精确的调节。在自平衡车的调节过程中，速度调节环作为外环，角度调节环作为内环。更加具体来说，当要求小车在原地平衡时，在小的干扰下小车指挥通过角度控制维持平衡，在扰动下才会出现移动，而当小车向前加速时，因为小车的运行速度与倾斜角有关，小车会先增大倾斜角度，之后再直立控制下车轮向前运动保持小车平衡，从而实现小车的加速。简单来说，串级 PID 调节就是先通过外环角度环进行粗调，再通过内环速度环进行细调的串级控制系统。

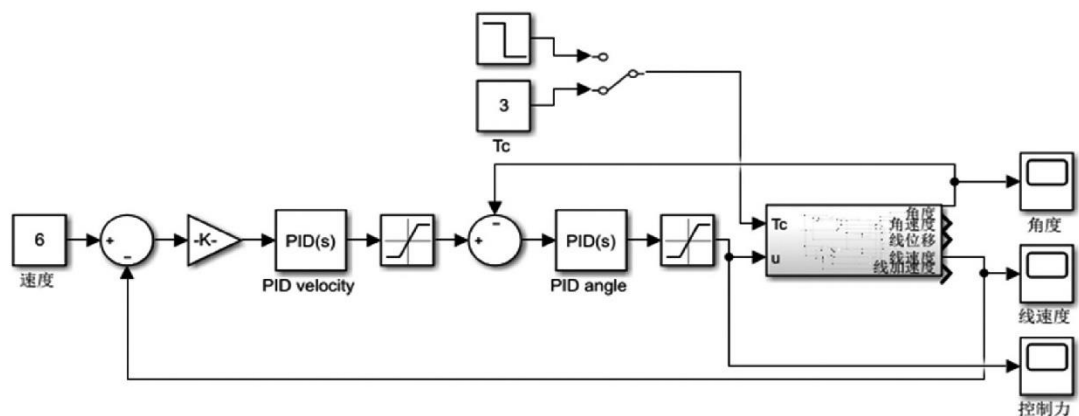


图 9 常用串级 PID 框图

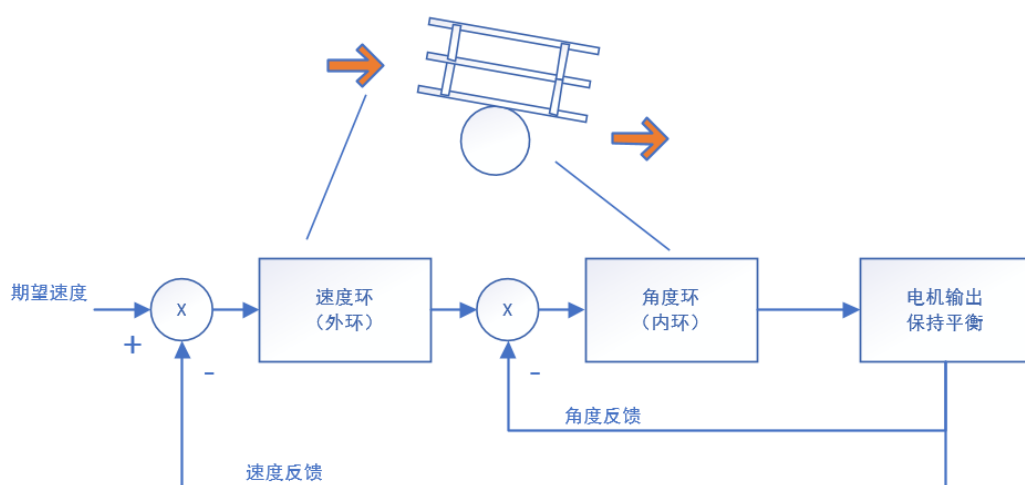


图 10 自平衡小车串级控制示意图

### (3) 串级和并行 PID 的比较

相较于并行 PID 控制简单的分别控制，串级 PID 能够实现分级控制，理论上能够实现较好的控制结果。实际上，在对自平衡车的两种控制方法进行对比研究后，南京林业大学的解文周、张子璇等人发现，在给定参数条件下，并行 PID 控制在平衡控制中表现更好，而在速度控制中，串级 PID 抗干扰能力更强，且控制更加及时，但在干扰较大时更容易失去稳定。



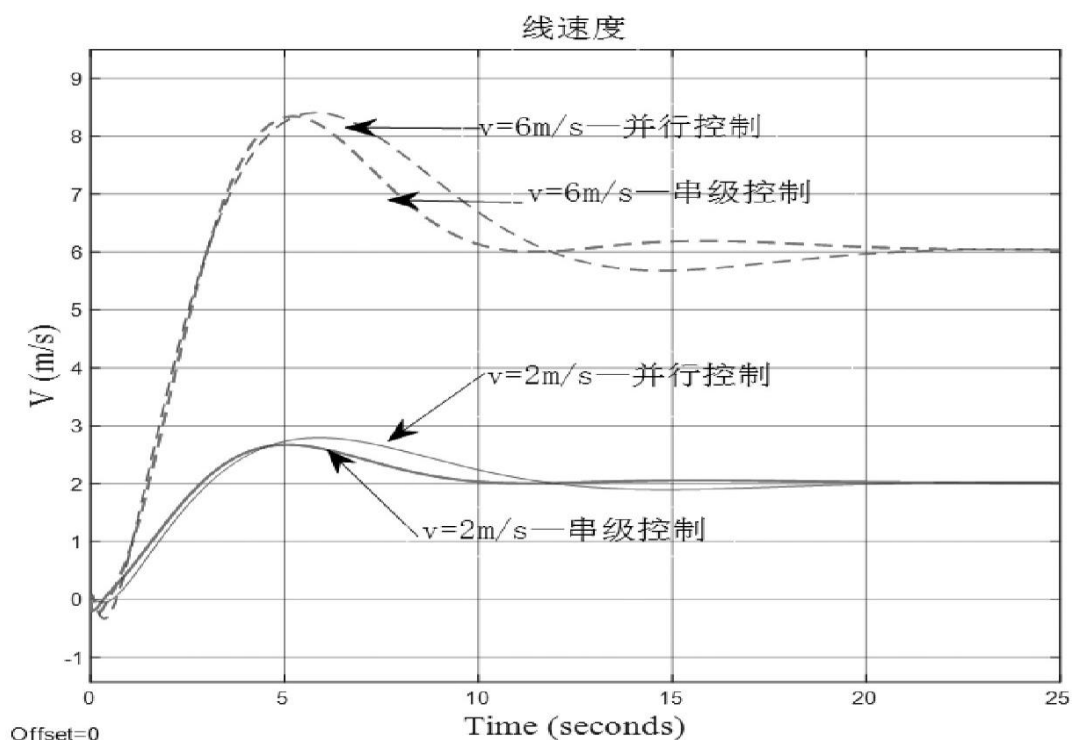


图 11 阶跃输入时响应曲线

实际上，在自平衡车的具体控制过程中，由于并行 PID 抗较小干扰的能力不强，大多数自平衡车采用的是串级 PID 控制方法。在串级 PID 控制外，自平衡车还有双闭环 PID、模糊 PID 等控制方法，限于篇幅，本篇在此不做赘述。

#### (4) 自平衡小车的调参过程——以电机速度内环闭环PID控制为例

##### (1) 参数选取

对于 PID 控制，参数的选取至关重要，然而 PID 控制参数并没有确定的最优解，很多时候参数的选取要依靠经验和技巧，但无论如何，参数的选取都需要调整的过程对结果进行优化，这个过程需要不断的尝试与微调。在这个自平衡车的调参过程中，为了方便调参，PID 控制中并未引入 D 参数，并且已经提前给定了一组较为理想的参数，且由于真实的 PID 调参过程中微调所带来的变化并不是很明显，本节将选取几组较为极端的 PID 参数，简单地大致介绍 PID 调参过程。为比较参数的控制效果，该实验设置阶跃目标速度，并将实际速度与其比较。

首先选取较小的参数  $f_p=0, f_l=0.1$ , 响应曲线如下：

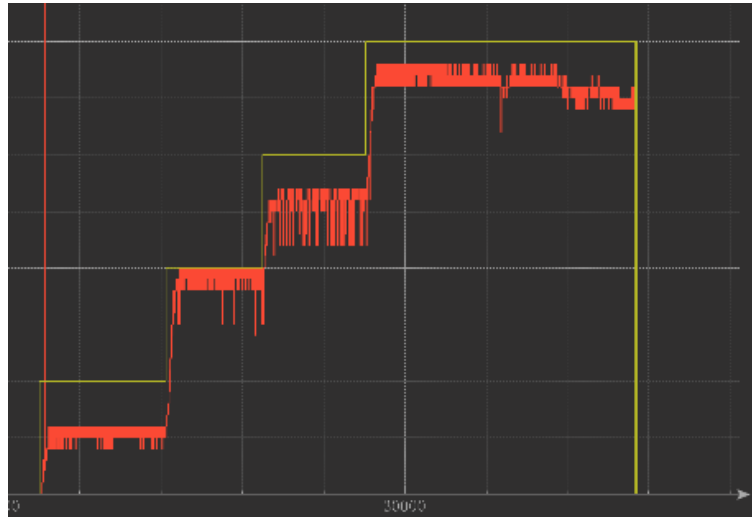


图 12 响应曲线 1

黄色为目标值，红色为实际值，可以看到，由于参数过小，速度调节过慢。将  $f_l$  增至 0.9，再次得到响应曲线如下：

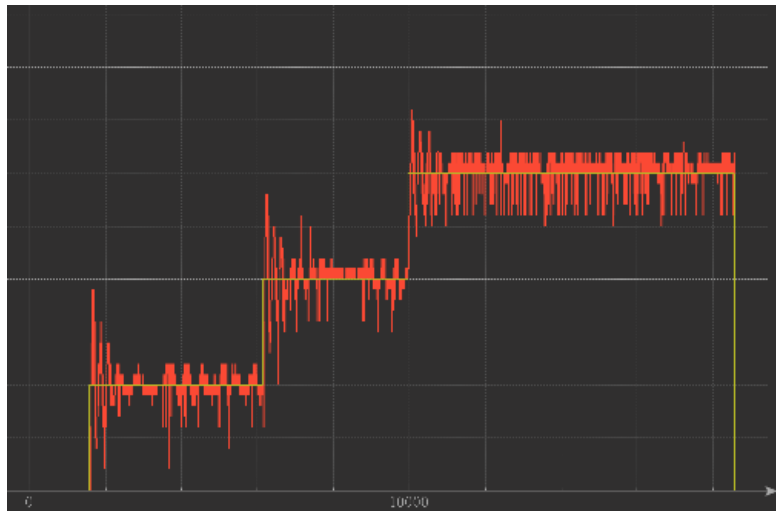


图 13 响应曲线 2

可以看到，虽然红线已经能够较好地跟随黄线，但在阶跃处仍有震荡，这说明 PID 控制的实时比例控制成分过小，故调整  $f_P$  至 10.0，得曲线如下：

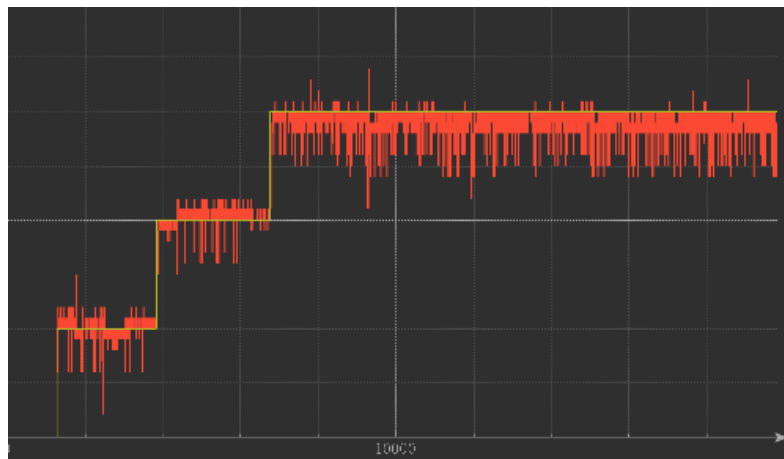


图 14 响应曲线 3

这时控制效果已经相当理想，为取得更好的效果，可以对参数进行微调，从而找到更好的参数。对于多个参数，调参过程与上述类似，均是通过拟合过程中出现的问题，对对应的参数进行调整或是微调。

## (2) 速度环和角度环参数的调节

我们在角度环控制中只使用了比例和微分两个控制量，没有使用积分的原因是：1，测量过程中的噪声会对累积误差造成干扰。2，摩擦力会自行消除稳定误差。这里用占空比来调节速度，由 PWM 得到的值控制，0 对应占空比 0%，999 对应占空比 100%。我们需要确定比例和微分项的系数，即  $f_P$  和  $f_D$ 。首先，我们这里需要负反馈，因此  $f_P$  应该是一个正值。我们从较小的值（比如 10）开始逐渐增大  $f_P$ ，在  $f_P$  到达 50 的时候发现小车在受到干扰后出现了大幅度的低频摆动，这代表  $f_P$  已经足够大了。接下来我们增加微分项系数  $f_D$ ，抑制低频摆动。首先，我们容易测试得知  $f_D$  也是一个正值，我们依旧从一个较小的值开始增加  $f_D$ 。在增加至 2.0 时，性能达到比较稳定的状态；如果增加到 3.0，小车则会开始高频剧烈抖动，这说明 2.0 附近是比较合适的值。此时小车已经可以实现自平衡，但是依然无法长时间直立，完整的功能需要在加入速度环后才能实现。

与角度环不同，速度环中我们需要用正反馈来控制——这很好理解，因为当小车处于非静止状态时，车轮需要加速去“追上”车身。我们仍然可以用简单的测试来确定。速度环控制中我们采用比例和积分控制，这是为了防止微分放大噪声，同时消除稳定误差。同样地，我们从较小值开始增加  $f_I$ ，发现增加到 0.15 的时候小车已经有了大幅度摆动，说明 0.15 过大，我们在这里取小一点的值，如 0.1。再来调试  $f_P$ ，增加到 20 的时候小车已经可以抵消  $f_P$  造成的来回摆动了，而且小车对干扰具有较强的抵抗力。如果我们继续增加  $f_P$ ，比如加到 30，这时虽然小车回复力度增加，但是稍微一点干扰就会使小车大幅度摆动，因此这个值太大，20 左右就比较合适。

至此，我们已经得到了一组可行的参数。我们可以继续实验和微调来获得能够使小车性能更优的参数。

# 四、 刚体机器人的小脑——STM32单片机在自平衡小车上的应用

## 1. 命名与用途

STM32，从字面上来理解，ST 是意法半导体，M 是 Microelectronics 的缩写，32 表示 32 位，STM32 就是指 ST 公司开发的 32 位微控制器。我们使用的 STM32 芯片全称是 STM32F103C8T6，具体的命名规则可以参看下表：

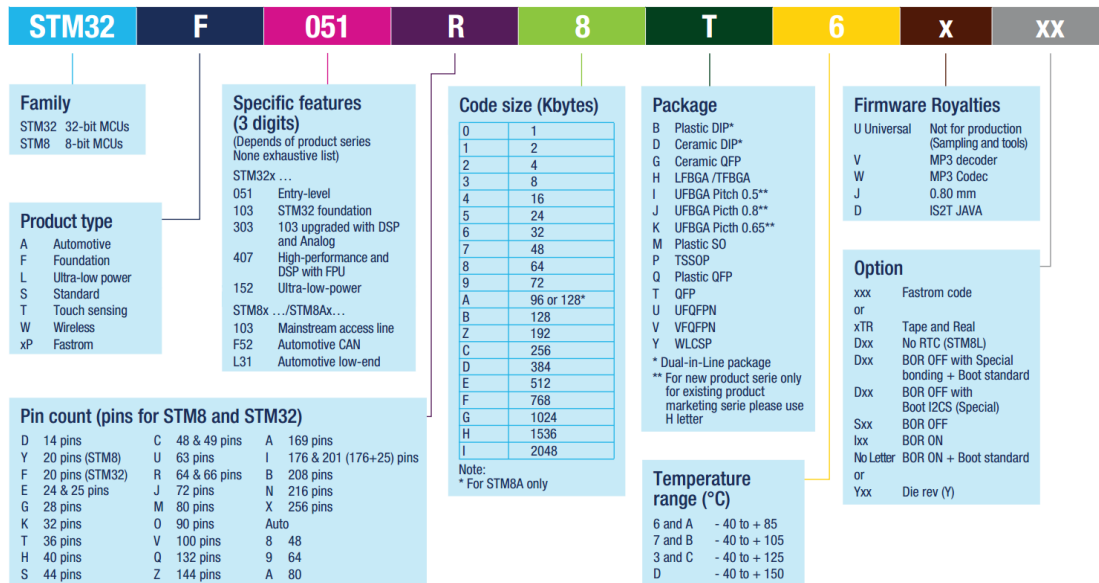


图 15 STM 芯片的命名规则

相当于芯片的功能,上面有很多的通信接口,这些通信接口可以接传感器,可以接设备,例如 LED 灯,从而实现对设备的控制。生活中最常见的应用 STM32 的就是无人机,扫地机器人等。

## 2. 如何使用STM32开发

需要两个软件, MDK-ARM 软件以及 STM32CubeMX 软件。

### (1) MDK-ARM 软件

MDK-ARM 是一个针对 ARM 芯片,集代码编辑,编译,链接和下载于一体的集成开发环境 (IDE)。界面如下:

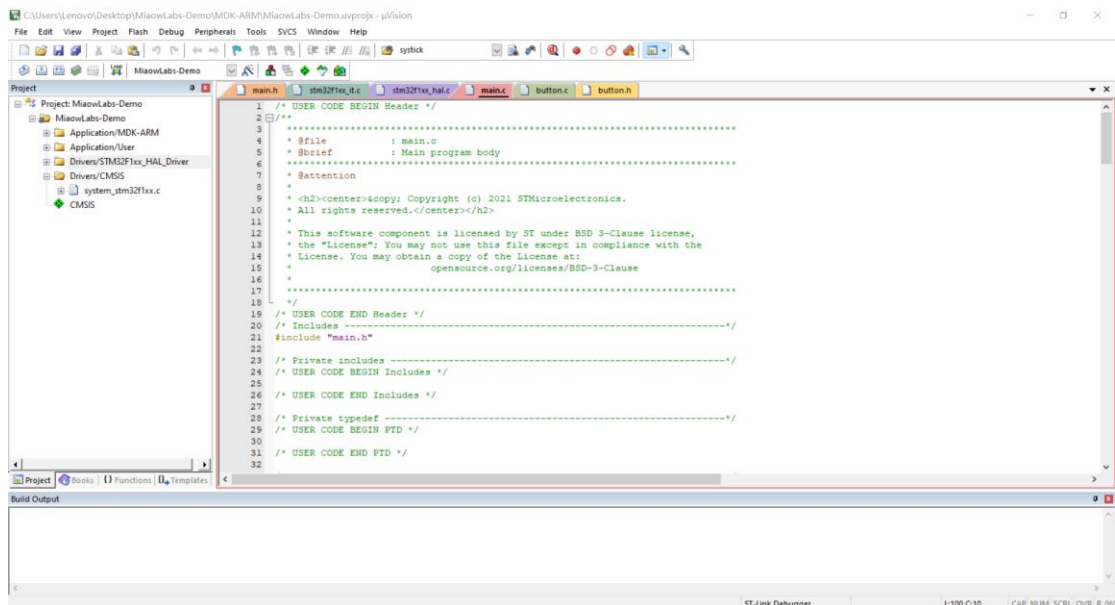


图 16 MDK-ARM 的操作界面

此软件可以通过 HAL 库修改及编译实现控制机器人的代码, 可以理解为就是写代码使

用的集成开发环境。

## (2) STM32CubeMX 软件

STM32CubeMX 软件是 STM32 ARM Cortex-M 微控制器的图形配置和底层代码生成工具，可以通过图形化向导可以快速、轻松配置 STM32 系列单片机的底层驱动，并生成相应的初始化 C 代码。界面如下：

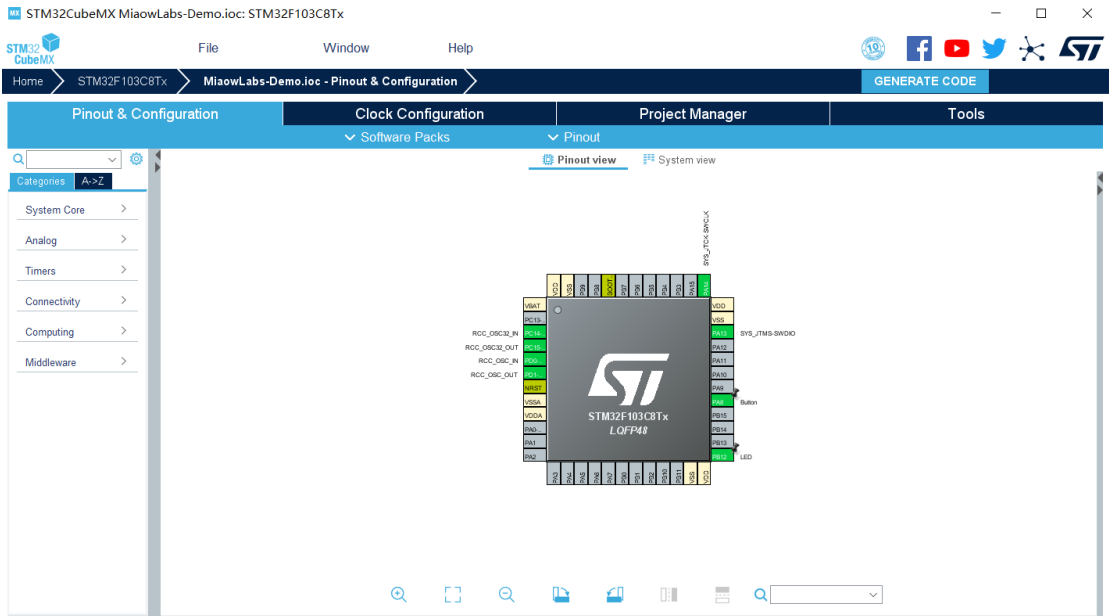


图 17 STM32CubeMX 的操作界面

我们的第一步是使用 STM32CubeMX 生成一个初始化的工程。直接打开 STM32CubeMX 软件，点击 New Project 一栏下的 ACCESS TO MCU SELECTOR，这时候 STM32CubeMX 就会自动从网上下载相关的代码包，接着搜索找到我们小车使用的 STM32C8 型号的芯片，双击选择之后就会自动跳出初始化界面。

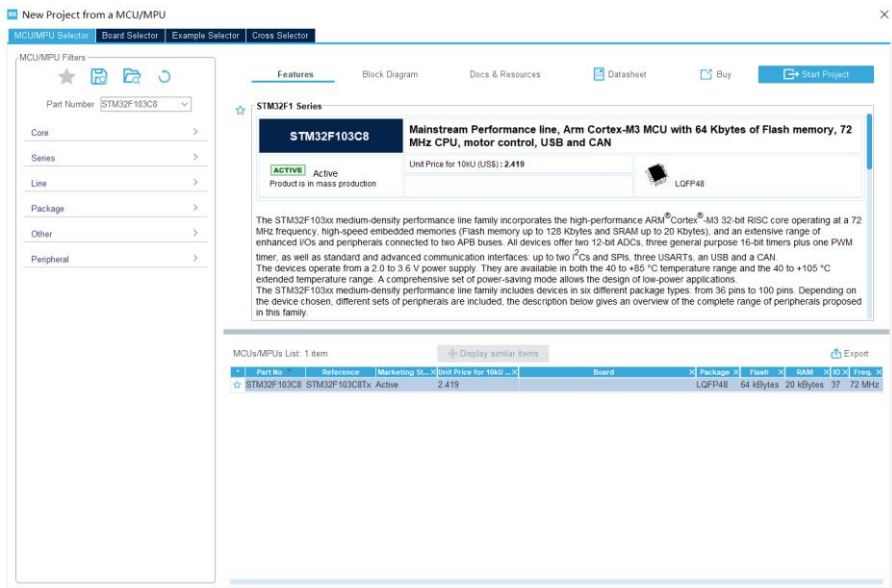


图 18 初始化界面

在 Project Manager 一栏完成设置之后点击右上角 GENERATE CODE 就会生成工程文件夹。

这是工程文件夹：

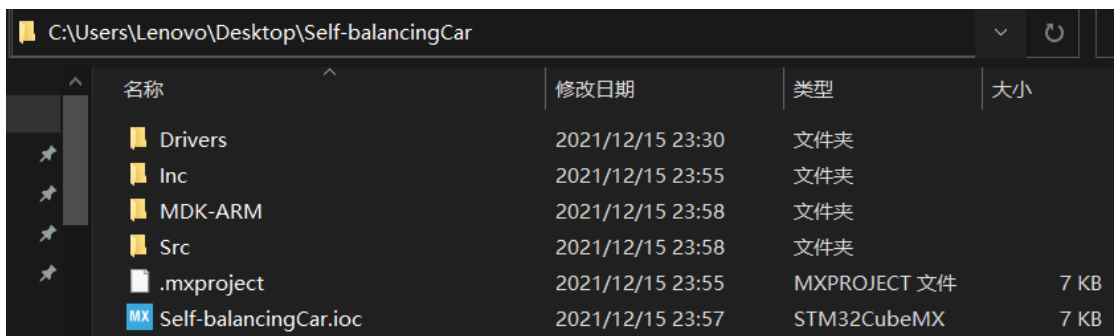


图 19 工程文件夹

点击.ioc 文件可以进入 STM32CubeMX 软件界面对芯片进行设置，Drive 文件夹主要包含 HAL 固态库相关文件，Src 文件夹存放所有的 C 语言代码文件 (xxx.c)，Inc 文件夹存放所有的 C 语言头文件 (xxx.h)。MDK-ARM 文件夹存放工程的相关编辑。进入 MDK-ARM 文件夹，点击.uvprojx 文件可以进入 MDK-ARM 软件界面进行代码编写。



图 20 MDK-ARM 文件夹

下面再认识一下 MDK-ARM 工程中的文件：

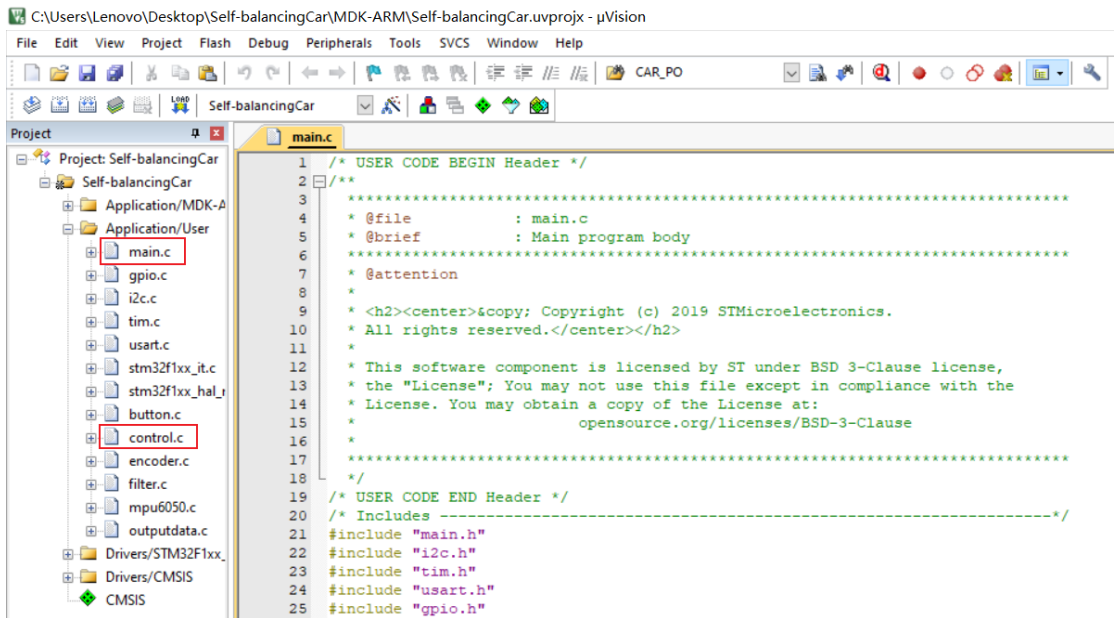


图 21 MDK-ARM 工程中的文件

main.c 是代码编译的文件，主循环在这个文件中，在 main.c 中调用函数即可实现相关功能。

control.c 是控制算法文件。

### 3. 小车控制代码

主要思路：

- 角度环控制：实时读取传感器相关参数并计算小车的倾角，根据每次得到的反馈运用 PD 对电机参数进行设置。
- 速度环控制：实时读取计算小车的速度，并根据得到每次得到的反馈运用 PI 对电机参数进行设置。

具体任务：

- 读取传感器相关参数并计算小车的倾角
- 读取并计算小车的速度
- 角度 PD 控制算法，速度 PI 控制算法
- 对电机参数进行设置更新
- 实时实现
- 调参

下面分别对每个任务展开讲解。

#### a) 读取传感器相关参数并计算小车的倾角（使用 MPU6050 加速度计、陀螺仪）

建立坐标系：

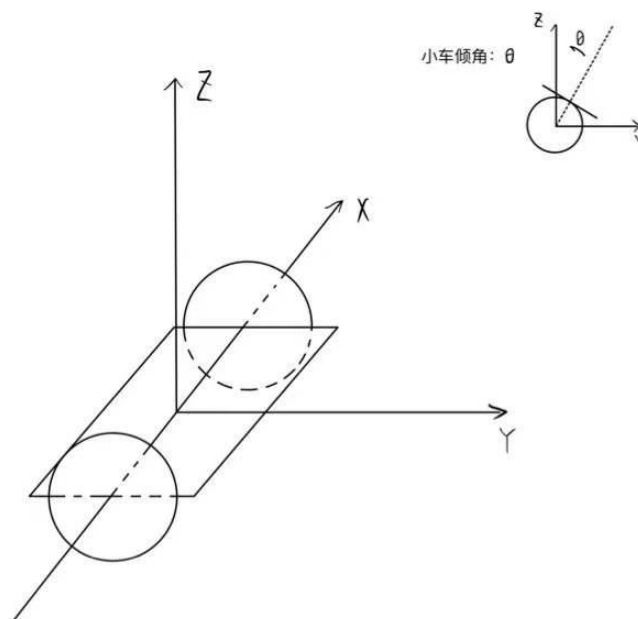


图 22 建立在小车上的坐标系

control.c 相关代码如下：

```

57
58 void GetMpuData(void) //读取MPU-6050数据
59 {
60     MPU_Get_Accelerometer(&x_nAcc,&y_nAcc,&z_nAcc); //获取MPU6050加速度数据
61     MPU_Get_Gyroscope(&x_nGyro,&y_nGyro,&z_nGyro); //获取MPU6050陀螺仪数据
62 }
63
64 void AngleCalculate(void) //角度计算
65 {
66     //-----加速度数据处理-----
67     //量程为±2g时，灵敏度：16384 LSB/g
68     x_fAcc = x_nAcc / 16384.0;
69     y_fAcc = y_nAcc / 16384.0;
70     z_fAcc = z_nAcc / 16384.0;
71
72     g_fAccAngle = atan2(y_fAcc,z_fAcc) * 180.0 / 3.14;
73
74     //-----陀螺仪数据处理-----
75     //范围为2000deg/s时，换算关系：16.4 LSB/(deg/s)
76     g_fGyroAngleSpeed = x_nGyro / 16.4; //计算角速度值
77
78     //-----互补滤波-----
79     g_fCarAngle = ComplementaryFilter(g_fAccAngle, g_fGyroAngleSpeed, dt);
80     g_fCarAngle = g_fCarAngle - CAR_ZERO_ANGLE; //减去机械零点偏移
81
82     //OutData[0]=g_fAccAngle; //发送加速度初步计算的角度
83     //OutData[1]=g_fGyroAngleSpeed; //发送陀螺仪角速度
84     //OutData[2]=g_fCarAngle; //发送数据融合得到的角度
85 }
86

```

图 23 读取相关传感器参数并计算倾角的代码

GetMpuData(void)是通过调用现成的 mpu6050.c 代码包直接读取 6 轴运动处理传感器 MPU6050 的相关参数。

加速度计得到的数据是 accx, accy, accz，除以 Sensitivity 得到各方向的加速度（这里主要是重力引起的加速度值）。然后计算 Z、Y 方向角度值得到小车加速度倾角 Acc。

陀螺仪得到的数据是 gyrox, gyroy, gyroz，减掉零偏置后除以 Sensitivity 得到绕各轴的角速度。这里我们只需要绕 x 轴的角速度 Gyro。

这两个器件的缺点在于，在一定情况下都会产生误差累积，解决办法就是利用互补滤波对得到的两组数据进行整合。

$$\text{Angle} = \alpha * (\text{Angle} + \text{Gyro} * \text{dt}) + (1 - \alpha) * \text{Acc}$$

计算互补滤波的文件为 filter.c，为 control.c 提供 ComplementaryFilter()函数

#### b) 读取并计算小车的速度（使用 TIM 读取左右电机脉冲）

control.c 相关代码如下：

```

87 void GetMotorPulse(void) //读取电机脉冲
88 {
89     g_nRightMotorPulse = (short) ( __HAL_TIM_GET_COUNTER(&htim4)); //获取计数器值
90     g_nRightMotorPulse = (-g_nRightMotorPulse);
91     __HAL_TIM_SET_COUNTER(&htim4,0); //TIM4计数器清零
92     g_nLeftMotorPulse = (short) ( __HAL_TIM_GET_COUNTER(&htim2)); //获取计数器值
93     __HAL_TIM_SET_COUNTER(&htim2,0); //TIM2计数器清零
94
95     g_lLeftMotorPulseSigma += g_nLeftMotorPulse; //速度外环使用的脉冲累积
96     g_lRightMotorPulseSigma += g_nRightMotorPulse; //速度外环使用的脉冲累积
97 }
98
99

```

图 24 计算小车速度的代码



主要使用 STM32 中的 TIM 模块, 只要提前在 STM32CubeMX 软件中配置好 TIM2、TIM4 接口, 将其改为编码器模式生成相关代码, 就可以在 MDK-ARM 工程中调用。这里采用了计算单位时间内累计脉冲的方法来替代速度的计算 (之后在速度外环控制中使用脉冲累计并且每 25ms 清零一次, 也就是说这里的脉冲频率是以 25ms 为单位时间的)。电机的转速与脉冲频率成正比。

### c) 角度 PD 控制算法, 速度 PI 控制算法

control.c 相关代码如下:

```
175 void AngleControl(void) //角度环控制函数
176 {
177     q_fAngleControlOut = (CAR_ANGLE_SET - q_fCarAngle) * q_fAngle_P + (CAR_ANGLE_SPEED_SET - q_fGyroAngleSpeed) * q_fAngle_D; //PD控制器
178 }
179
180
181 void SpeedControl(void) //速度外环控制函数
182 {
183     float fP, fI; //速度环pi参数.
184     float fDelta; //临时变量, 用于存储误差
185
186     q_fCarSpeed = (q_lLeftMotorPulseSigma + q_lRightMotorPulseSigma) / 2; //左轮和右轮的速度平均值等于小车速度
187     q_lLeftMotorPulseSigma = q_lRightMotorPulseSigma = 0; //全局变量, 注意及时清零
188
189     q_fCarSpeed = 0.7 * q_fCarSpeedPrev + 0.3 * q_fCarSpeed; //低通滤波, 使速度更平滑
190     q_fCarSpeedPrev = q_fCarSpeed; //保存前一次速度
191
192     fDelta = CAR_SPEED_SET;
193     fDelta -= q_fCarSpeed; //误差=目标速度-实际速度
194
195     fP = fDelta * q_fSpeed_P;
196     fI = fDelta * q_fSpeed_I;
197
198     q_fCarPosition += fI;
199
200     //设置积分上限设限
201     if((int)q_fCarPosition > CAR_POSITION_MAX) q_fCarPosition = CAR_POSITION_MAX;
202     if((int)q_fCarPosition < CAR_POSITION_MIN) q_fCarPosition = CAR_POSITION_MIN;
203
204     q_fSpeedControlOutOld = q_fSpeedControlOutNew; //保存上一次输出
205
206     q_fSpeedControlOutNew = fDelta * fP + q_fCarPosition * fI; //PI控制器, 输出=误差*p+误差积分*I
207
208 }
```

图 25 控制角度和速度的算法代码

➤ 角度 PD 控制:

$$u_k = k_P * e_k + k_D * e'_k$$

其中,  $e_k = \text{目标角度} - \text{实际角度}$ ,  $e'_k = \text{目标角速度} - \text{实际角速度}$ 。

选择理由: 在小车倾角相关参数的测量过程中不可避免会有误差, 积分容易让这些误差累积起来, 造成较大的偏误。

➤ 速度 PI 控制:

$$u_k = k_P * e_k + k_I * \sum e_k$$

其中,  $e_k = \text{目标速度} - \text{实际速度}$ ,  $\sum e_k = \text{小车位移}$

选择理由: 这里速度的控制是正反馈, 也就是说当小车存在朝一个方向运动的速度时电机必须提供相同的速度去抵消, 而且小车的速度越大, 电机提供的速度也要越大。微分项会让小车的速度反应过大产生高频振荡。

### d) 对电机参数进行设置更新

control.c 相关代码如下:

```

121 |
122 |
123 | d SetMotorVoltageAndDirection(int nLeftMotorPwm,int nRightMotorPwm)//设置电机电压和方向
124 | {
125 |     f(nRightMotorPwm < 0)//反转
126 |     {
127 |         HAL_GPIO_WritePin(AIN1_GPIO_Port, AIN1_Pin, GPIO_PIN_SET);
128 |         HAL_GPIO_WritePin(AIN2_GPIO_Port, AIN2_Pin, GPIO_PIN_RESET);
129 |         nRightMotorPwm = (-nRightMotorPwm);//如果计算值是负值,负值只是表示反转,先转负为正,因为PWM寄存器只能是正值
130 |         __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, nRightMotorPwm);
131 |     }else//正转
132 |     {
133 |         HAL_GPIO_WritePin(AIN1_GPIO_Port, AIN1_Pin, GPIO_PIN_RESET);
134 |         HAL_GPIO_WritePin(AIN2_GPIO_Port, AIN2_Pin, GPIO_PIN_SET);
135 |         __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, nRightMotorPwm );
136 |     }
137 |     f(nLeftMotorPwm < 0)//反转
138 |     {
139 |         HAL_GPIO_WritePin(BIN1_GPIO_Port, BIN1_Pin, GPIO_PIN_SET);
140 |         HAL_GPIO_WritePin(BIN2_GPIO_Port, BIN2_Pin, GPIO_PIN_RESET);
141 |         nLeftMotorPwm = (-nLeftMotorPwm);//如果计算值是负值,负值只是表示反转,先转负为正,因为PWM寄存器只能是正值
142 |         __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, nLeftMotorPwm);
143 |     }else//正转
144 |     {
145 |         HAL_GPIO_WritePin(BIN1_GPIO_Port, BIN1_Pin, GPIO_PIN_RESET);
146 |         HAL_GPIO_WritePin(BIN2_GPIO_Port, BIN2_Pin, GPIO_PIN_SET);
147 |         __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, nLeftMotorPwm);
148 |     }
149 | }
150 |
151 |
152 | void MotorOutput(void)//电机输出函数,将直立控制、速度控制、方向控制的输出量进行叠加,并加入死区常量,对输出饱和作出处理。
153 | {
154 |     //这里的电机输出等于角度环控制量 + 速度环外环,这里的 - g_fSpeedControlOut 是因为速度环的极性跟角度环不一样,角度环是负反馈,速度环是正反馈
155 |     g_fLeftMotorOut = g_fAngleControlOut - g_fSpeedControlOut;
156 |     g_fRightMotorOut = g_fAngleControlOut - g_fSpeedControlOut;
157 |     /*增加电机死区常数*/
158 |     if((int)g_fLeftMotorOut>0) g_fLeftMotorOut += MOTOR_OUT_DEAD_VAL;
159 |     else if((int)g_fLeftMotorOut<0) g_fLeftMotorOut -= MOTOR_OUT_DEAD_VAL;
160 |     if((int)g_fRightMotorOut>0) g_fRightMotorOut += MOTOR_OUT_DEAD_VAL;
161 |     else if((int)g_fRightMotorOut<0) g_fRightMotorOut -= MOTOR_OUT_DEAD_VAL;
162 |     /*输出饱和和处理,防止超出PWM范围*/
163 |     if((int)g_fLeftMotorOut > MOTOR_OUT_MAX) g_fLeftMotorOut = MOTOR_OUT_MAX;
164 |     if((int)g_fLeftMotorOut < MOTOR_OUT_MIN) g_fLeftMotorOut = MOTOR_OUT_MIN;
165 |     if((int)g_fRightMotorOut > MOTOR_OUT_MAX) g_fRightMotorOut = MOTOR_OUT_MAX;
166 |     if((int)g_fRightMotorOut < MOTOR_OUT_MIN) g_fRightMotorOut = MOTOR_OUT_MIN;
167 |     SetMotorVoltageAndDirection((int)g_fLeftMotorOut,(int)g_fRightMotorOut);
168 | }
169 |
170 |
171 |
172 |
173 |

```

图 26, 图 27 设置更新电机参数的相关代码

STM32 控制引脚分配:

- PB0 --> AIN1
- PB1 --> AIN2
- PA3 --> BIN1
- PA4 --> BIN2
- PA6 (TIM3\_CH1 ) --> PWMA
- PA7 (TIM3\_CH2) --> PWMB

左边是 STM32 引脚, 右边是标签, 其中 PWMA、AIN1、AIN2 控制右侧电机, PWMB、BIN1、BIN2 控制左侧电机。

以控制左侧电机为例。控制电机主要有两个参数, 一个是旋转速度, 一个是旋转方向, PWMB 控制旋转速度, BIN1、BIN2 控制旋转方向。

➤ 旋转速度控制:

可以直接在 STM32CubeMX 操作完成。通过控制占空比设置,

$$\text{占空比} = \frac{(\text{Pulse} + 1)}{(\text{TIM}_{\text{XARR}} + 1)}$$

占空比描述了电机通电的时间占总时间的比例, 例如设置为 50%即为半全速转动。

设置 TIM3\_CH2 (Channel 2) :

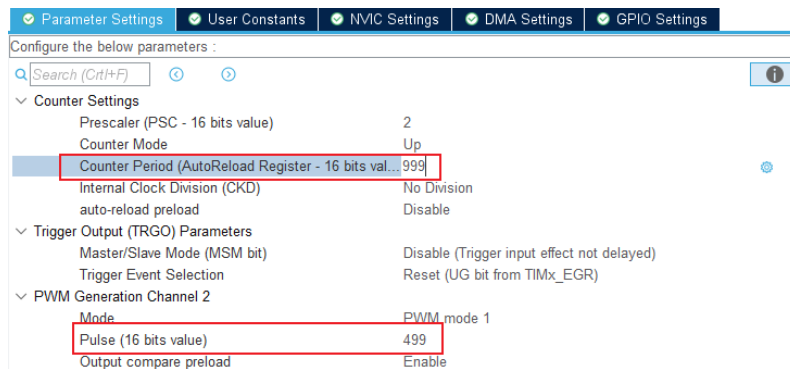


图 28 设置 TIM3\_CH2

上图中 Counter Period 设为 999，即计数周期（TIMx\_ARR）设为 999。

生成代码之后可以直接在 MDK-ARM 工程中调用相关函数进行更改，在 control.c 中调用了 HAL 固态库函数 `_HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, nLeftMotorPwm)` 更改了 Output compare preload 一项，实现了更改左电机输出。

#### ➤ 旋转方向控制：

首先在 STM32CubeMX 中设置 PB0、PB1、PA3、PA4，将引脚配置为 GPIO\_Output。

然后在 MDK-ARM 工程中调用函数 `HAL_GPIO_WritePin(BIN1_GPIO_Port, BIN1_Pin, GPIO_PIN_SET)` 更改电平设置。

完成 `SetMotorVoltageAndDirection()` 函数的编写之后再编写 `MotorOutput()` 函数将上一步 PID 控制量经过叠加处理之后赋值给电机设置。

### e) 实时实现（SysTick 函数）

```

183 void SysTick_Handler(void)
184 {
185     /* USER CODE BEGIN SysTick_IRQn 0 */
186
187     g_nMainEventCount++; //每进行一次中断，主事件函数自动加1
188
189     g_nSpeedControlPeriod++; //速度环控制周期计算量自动加1
190     SpeedControlOutput(); //速度环控制平滑输出处理，速度的pwm改变量如果在25ms时刻计算出后立即输出，会造成不平滑抖动等，这段代码就是把这25ms周期计算一次得3
191     if(g_nMainEventCount>=5) //SysTick是1ms一次，这里判断语句大于5就是5ms运行一次
192     {
193         g_nMainEventCount=0; //主事件循环每5ms循环一次，这里清零，重新计时。
194         GetMotorPulse();
195         else if(g_nMainEventCount==1) //这1ms时间片段获取数据和角度计算
196             GetPulseData(); //获取HPU-6050数据
197         AngleCalculate(); //进行角度计算
198         else if(g_nMainEventCount==2) {
199             AngleControl(); //这1ms时间片段进行角度控制
200         } else if(g_nMainEventCount==3) {
201             g_nSpeedControlCount++;
202             if(g_nSpeedControlCount >= 5)
203             {
204                 SpeedControl(); //速度控制，25ms进行一次
205                 g_nSpeedControlCount=0; //清零
206                 g_nSpeedControlPeriod=0; //清零
207             }
208
209         } else if(g_nMainEventCount==4) {
210             MotorOutput(); //电机输出函数，每5ms执行一次
211         }
212         ButtonScan();
213     /* USER CODE END SysTick_IRQn 0 */
214     HAL_IncTick();
215     /* USER CODE BEGIN SysTick_IRQn 1 */
216

```

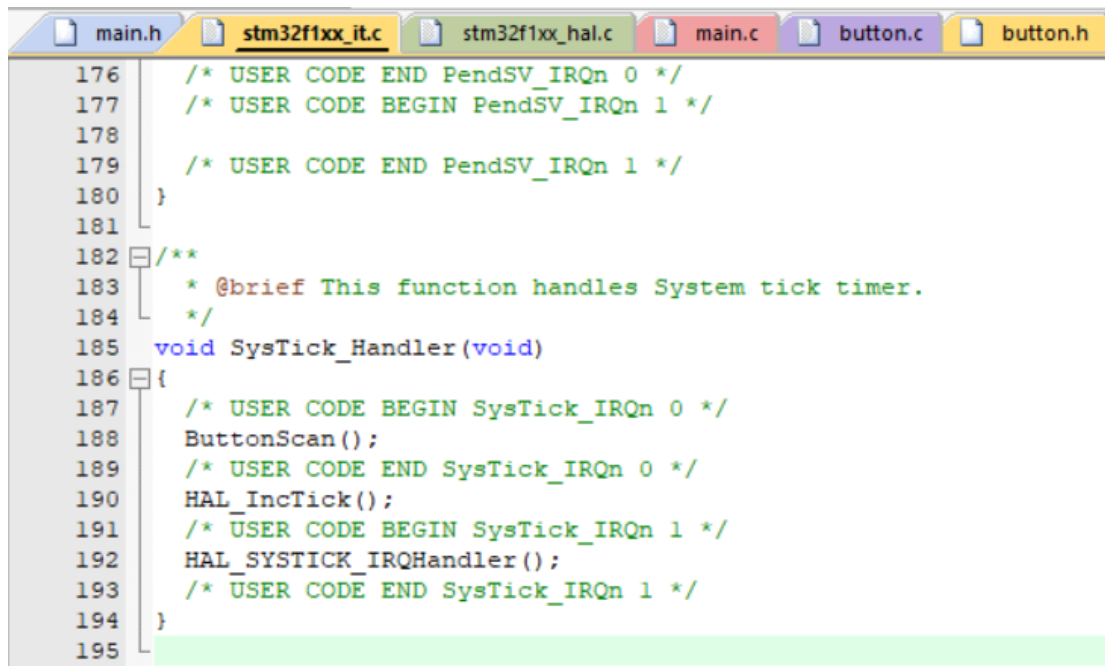
图 29 SysTick 函数

SysTick 是 Cortex 系统定时器，主要功能是定时中断，默认 1ms 中断一次。

使用：

首先在 STM32CubeMX 中配置相关设置，然后 generate code。

然后打开 MDK-ARM 工程在 Application/User 文件夹的 stm32f1xx\_it.c 文件中找到 SysTick 中断服务函数 SysTick\_Handler：



```
176  /* USER CODE END PendSV_IRQn 0 */
177  /* USER CODE BEGIN PendSV_IRQn 1 */
178
179  /* USER CODE END PendSV_IRQn 1 */
180  }
181
182  /**
183   * @brief This function handles System tick timer.
184   */
185  void SysTick_Handler(void)
186  {
187      /* USER CODE BEGIN SysTick_IRQn 0 */
188      ButtonScan();
189      /* USER CODE END SysTick_IRQn 0 */
190      HAL_IncTick();
191      /* USER CODE BEGIN SysTick_IRQn 1 */
192      HAL_SYSTICK_IRQHandler();
193      /* USER CODE END SysTick_IRQn 1 */
194  }
195
```

图 30 SysTick 中断服务函数

最初始的 SysTick\_Handler 函数中只包含 HAL\_IncTick(), 只要在这个中断函数中添加函数以及 HAL\_SYSTICK\_IRQHandler(), 就可以实现每 1ms 运行一次, 例如上图中的 ButtonScan(), 代表每 1ms 运行一次 ButtonScan()。

还有一个有用的函数 HAL\_Delay(): 直接加入主循环中就可以实现延时。例如 HAL\_Delay(5000)意思就是延时 5s。

回到代码, 我们的目标是: 每 25ms 运行一次速度控制, 每 5ms 运行一次角度控制。具体实现: 首先设置主事件计数 g\_nMainEventCount, 每 1ms 加 1, 每 5ms 清零 (if 语句, 当 g\_nMainEventCount==5 时归零), 然后每次在 g\_nMainEventCount==3 时让 g\_nSpeedControlCount 加一, 并判断 g\_nSpeedControlCount 是否为 5, 是就清零, 并运行 SpeedControl(), 这样就实现了每 25ms 运行一次进行一次速度控制。

更具体地说, 我们将 5ms 分成了五步, 每一步执行不同的函数:

- 第 1ms: 执行 GetMpuData()和 AngleCalculate()得到小车倾角;
- 第 2ms: 执行 AngleControl()进行角度控制;
- 第 3ms: 这里设置 g\_nSpeedControlCount 变量, 每次都让这个变量+1, 并判断这个变量是否为 5, 如果为 5 就执行 SpeedControl()进行速度控制, 效果也就是每 25ms 执行一次 SpeedControl()。
- 第 4ms: 执行 MotorOutput()将 PID 控制量赋值给电机。
- 第 5ms: 让 g\_nMainEventCount=0, 执行 GetMotorPulse()获得电机脉冲并且累加。

#### f) 调参

为了实现平衡, 我们将目标速度、目标角度、目标角速度设置为 0, 剩下关于 PID 算法共 4 个参数需要调节, 分别是角度环 fP 值, fD 值和速度外环 fP 值, fI 值。

首先调节角度环参数。

fP 值极性为正, 从 0 开始增大时小车的响应速度会逐步提高。

fD 值极性为正, 从 0 开始增大时小车的低频大幅度抖动将会逐渐减弱, 并且向高频抖动过渡。

其次调节速度环参数。

fl 值极性为正，从 0 开始增大时小车逐渐出现低频大幅度来回摆动。

fp 值极性为正，从 0 开始增大时小车摆动逐渐减小，并且小车对外力的响应逐渐变得灵敏。

## 五、 刚体机器人的大脑——利用树莓派拓展小车的功能

### 1. 上位机简介——以树莓派为例

通过上文的知识我们已经可以构造一个简单的具有一定控制能力的机器人啦，我们的自平衡小车也可以通过 STM32 芯片的控制达到一个比较稳定的平衡效果。然而，我们发现 STM32 的算力非常有限，这些通过单片机实现的简单的功能还与现在前沿的比较复杂的刚体机器人相差甚远。于是，我们了解到想要进一步实现更复杂的功能需要上位机的支持。

上位机指可以直接发送操作指令的计算机或单片机，一般提供用户操作交互界面并向用户展示反馈数据。如果说我们把 STM32 比作刚体机器人的小脑，负责对外界刺激进行简单的反馈，那么树莓派这一类上位机就是刚体机器人的大脑。我们利用大脑可以进行更加复杂的决策。更复杂的决策就是指一些需要高计算速度和内存的算法来实现的过程。比如说，处理 RGB 图像数据、深度数据、激光雷达数据等占用较大空间的数据结果，或者利用这些数据运行建图导航等算法。

常用的上位机一般有树莓派、NVIDIA Jetson TX2 等。其实这些大同小异，主要是一些算力上的区别。简单来说这些芯片就是一个电脑的主机，上面有 USB 接口，HDMI 接口等，我们把它连接上显示屏、鼠标键盘就完全可以把它当成电脑来用，上网、玩游戏这些都可以。我们把它配上 5V 的电源就可以利用它小巧的特点做很多有意思的小发明。这里选择树莓派作为我们的上位机，我们把它装在了我们的自平衡小车上，想要它完成以下的功能：利用它自带的摄像头，处理外界的图像信号，当检测到对应的二维码时，向 STM32 发送指令，让小车执行相应的运动。

### 2. 如何建立树莓派与stm32、PC间的通信

为了实现上述的功能，我们首先就要建立树莓派与 STM32、我的个人电脑的通信。

对于 STM32，我们采用 USB 串口和树莓派连接。我们在 STM32 的程序中，通过开启 USART 模块，在 usart.c 文件中对 printf 和 scanf 两个函数重定向就可以简单的利用我们原来学过的 C 语言知识实现串口信号的收发。在树莓派里我们新建一个 python 程序，我们利用 serial 包可以实现对串口的输入和接收。当然也可以在一开始调试的时候用 sccom 串口助手对串口收发功能进行测试。

对于我的笔记本电脑，我主要选择了两种通讯方式，一种是可视化的远程桌面的连接，第二种是利用 putty 直接进入树莓派的命令行。为了实现与 PC 的通讯，我们在烧录完树莓派的镜像后首先需要开启 ssh 协议。接着，我们把我们的树莓派和 PC 连接到一个网络下，比如我手机的热点。最后，我们在自己的笔记本上利用树莓派的 ip 地址就可以实现远程访问的功能。

我们在测试中发现，树莓派与小车的数据传输速度是很快的，但是有时候极少情况会出现数据的缺失等现象。我们调查发现，我们可能可以通过改变串口的协议来解决这个问题（比如 IIC 协议）。同时，树莓派与 PC 的传输速度较慢，树莓派上采集到的摄像头信息传递到 PC 会存在一段较长的延时。因此，我们在用树莓派传输数据时，可以先进行一个简单的

预处理（比如利用傅里叶变换滤除高频噪声等），这样可以有效提高传输速度。

### 3. 二维码图像处理功能展示

我们编写了一个简单的 python 程序, 利用 pyzbar 和 cv2 模块采集和展示我预设的二维码数据, 当树莓派第一次采集到二维码信息时, 会通过串口向 stm32 发送一个速度值信号, 改变小车的速度。在远程桌面中, 运行效果如下:

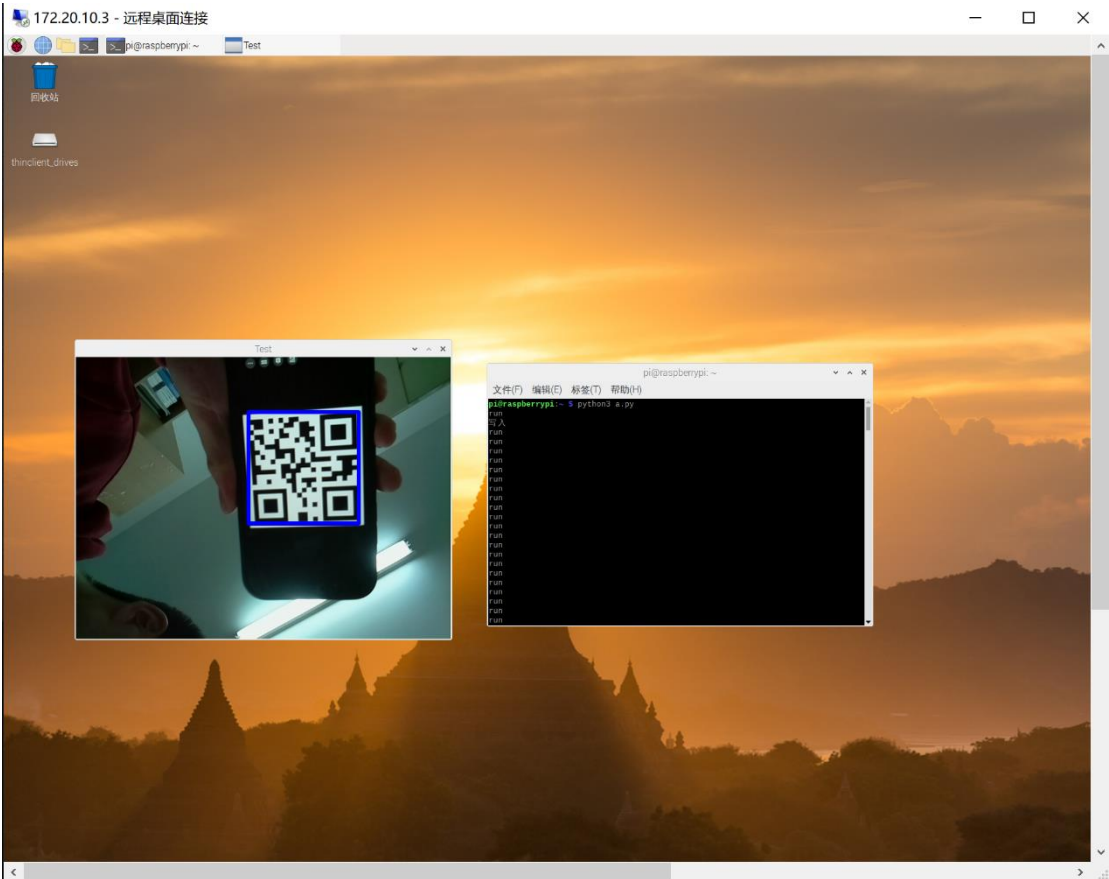


图 31 二维码图像处理功能展示

实验中, 树莓派很好的识别出了二维码, 小车也和我们预想的一样, 以我们设定的速度跑了起来。

## 六、 反思与总结

综上所述, 我们组本学期制作自平衡小车的大作业可以说是在了解了刚体机器人几块重要的方面的基础上做了一个非常简略的机器人模型。这样的制作的重要意义在于为我们开展进一步的研究打下了重要的基础和框架, 我们可以以它为基础做进一步的开发。在硬件和结构上, 我们可以重新设计小车的架构, 利用分压器合并 STM32 和树莓派的供电模块等; 在软件开发上, 我们可以进一步利用树莓派和 ROS 系统设计和应用更加复杂的算法, 比如 SLAM、运动感知、导航等。最后, 我们总结了设计刚体机器人的一个总体的思路就是先设计一个利于完成待定任务的结构, 然后利用单片机对相关的舵机和传感器进行简单的实时控制和数据采集发送给树莓派, 最后在树莓派或者 PC 上对单片机的数据进行处理给单片机发送控制信号, 完成复杂的任务。





图 32 设计刚体机器人的思路总结

### 【参考文献】

- 【1】 解文周、张子璇、台永鹏：《自平衡车串级与并行 PID 控制方法比较研究》，《电子世界》2021,(17),88-90 DOI:10.19353/j.cnki.dzsj.2021.17.038
- 【2】 梁华、李晓虹、杨光祥：《两轮自平衡机器人动力学模型分析及 PID 控制方法研究》，重庆师范大学学报(自然科学版) 2016,33(01),163-167
- 【3】 STM32 固件库使用手册的中文翻译版[EB/OL].[2007.10].  
<https://www.stmcu.com.cn/Designresource/list/STM32%20MPU/document>
- 【4】 STM32F10xxx 中文参考手册[EB/OL].[2010.1.10].  
<http://www.st.com/stonline/products/literature/rm/13902.pdf>
- 【5】 Joseph Yiu,宋岩.CM3 权威指南 CnR2[EB/OL].[2008.07.02]. [www.ouravr.com](http://www.ouravr.com)
- 【6】 野火.STM32 HAL 库开发实战指南—基于 F103 指南者[EB/OL].[2011.11.03].  
[http://doc.embedfire.com/products/link/zh/latest/tutorial/ebf\\_stm32\\_hal\\_tutorial.h](http://doc.embedfire.com/products/link/zh/latest/tutorial/ebf_stm32_hal_tutorial.html)  
[tml](http://doc.embedfire.com/products/link/zh/latest/tutorial/ebf_stm32_hal_tutorial.html)
- 【7】 自平衡小车的入门指南 <https://c.miaowlabs.com/>
- 【8】 树莓派新手指南(树莓派实验室). <https://shumeipai.nxez.com/>