# 轮 趣 科 技

# WHEELTEC B570 平衡小车 开发手册

推荐关注我们的公众号获取更新资料

版本说明：

| 版本 | 日期 | 内容说明 |
|---|---|---|
| V5.7 | 2021/8/16 | 第一次发布 |
| | | |

网址：www.wheeltec.net

# 序言

在工程实践中，应用最广泛的控制规律调节器为比例、微分、积分控制，简称 PID 控制。当被控对象的结构和参数不能完全掌握，或者得不到精确的数学模型时，应用 PID 算法最为方便快捷。近年来，双轮自平衡小车以其行走灵活、便利、节能等特点得到了很大的发展，在市场上也有相应的产品。平衡小车虽然结构简单，但是十分精妙，是学习和验证 PID 算法的优秀载体之一。

本文介绍了平衡小车的原理，分析了平衡小车的部分代码，有理论推导和实践过程，相信通过此文，读者能快速上手平衡小车，理解 PID 控制的思想和原理。

# 目录

# 1. 平衡原理

## 1.1 直立控制

我们玩过一个游戏，让直木棒在手掌上平衡。当直木棒向一边倾斜时，我们的手掌就往这边“追”这个木棒，使得它平衡在我们的手掌不倒。双轮平衡小车也是类似的原理。我们可以把小车简化为一个倒立摆（我们提供了详细的物理模型，但为了方便分析，使用简化的模型），而且这个倒立摆只有前后两个方向可以运动，建立如图 1-1 所示物理模型。

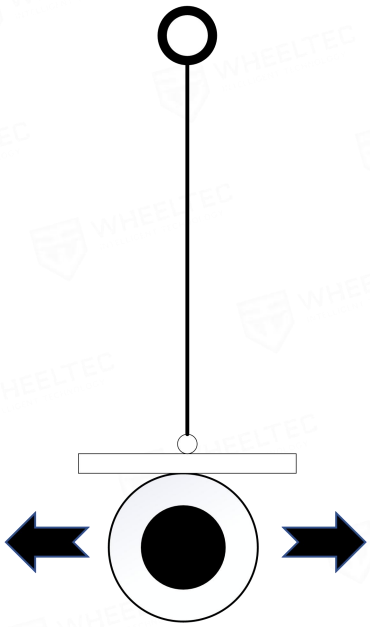

图 1-1 小车简化物理模型

当小车向一边倾斜时，我们分析它的受力情况。可以看到小车受到自身的重力作用，还有沿杆向上的支持力，如图 1-2 所示，把重力分解，可以得出合力为 $mgsin\theta$，与倾斜的方向相同，小车会加速倒下！

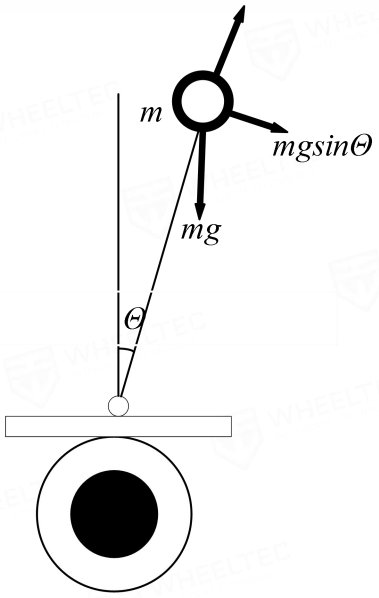


图 1-2 小车倾斜受力分析

那么如何才能让小车静止在中立位置呢？如果给小车一个外力，使得它与合力的方向相反，那么就可以让小车回复到平衡位置了。小车的动力来自与底部的两个车轮，这个外力只能由车轮给出，小车作变速运动时，车体自然会受到一个力的作用。

假设小车往前倾斜时，控制小车的车轮使小车作加速运动，加速度大小为 $a$。因为小车系统是一个非惯性系统(选取不同的参考系，物体的运动情况也不同。若整个小车以地面为参考系，那么小车符合牛顿运动定律，此时为惯性参考系，若简化为倒立摆的小车上方(指除去轮子后的部分)以小车轮子为参考系，如图 1-3 所示，此时小车为非惯性参考系)，小车上方的圆球会受到一个方向与加速度相反，大小与加速度成比例的惯性力，如图 1-3 所示，比例系数为质量$m$，即$\vec{F} = -m\vec{a}$。把这个水平向左的惯性力分解，可以得到垂直于杆的力$-macos\theta$，如图 1-3 所示。

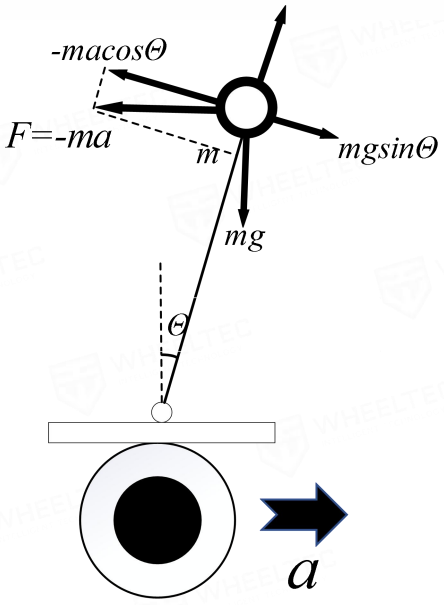


图 1-3 施加加速度后物理模型

可知小车受到的回复力为$F = mgsin\theta - macos\theta$，若$\theta$很小，那么可以认为$sin\theta = \theta$，$cos\theta = 1$，且假设小车的反馈系统中加速度$a$与小车的倾角成比例关系，比例系数为$k_1$，那么有$F = mgsin\theta - macos\theta = mg\theta - ma = mg\theta - mk_1\theta$。当$k_1 > g$时，回复力 $F$ 的方向就与倾斜方向相反了，此时小车会回复到平衡的位置。

若小车以上述分析加入加速度$a$(此加速度仅与角度有关)，如果小车不能准确地到达中立位并稳定下来，必然会使令小车在中立位置振荡，增大了平衡的时间。为了减少振荡，尽快稳定在平衡位置，还需要增加额外的阻尼力，增加的阻尼力大小与角速度成正比，方向相反，比例系数设为$k_2$。加上阻尼力后，小车的回复力为：$F = mg\theta - mk_1\theta - mk_2\theta' = mg\theta - m(k_1\theta + k_2\theta')$，$\theta'$是角度的微分，即角速度。把所有的外力整合，可得小车加速度控制算法：$a = k_1\theta + k_2\theta'$。即给小车一个加速度 $a$，只要保证$k_1 > g$，$k_2 > 0$ 就可以令小车保持在平衡位置。

如何控制小车的加速度呢？电机的转速是通过改变在其上的驱动电压大小来实现的，对应不同的电压，电机转速变化曲线如图 1-4 所示。

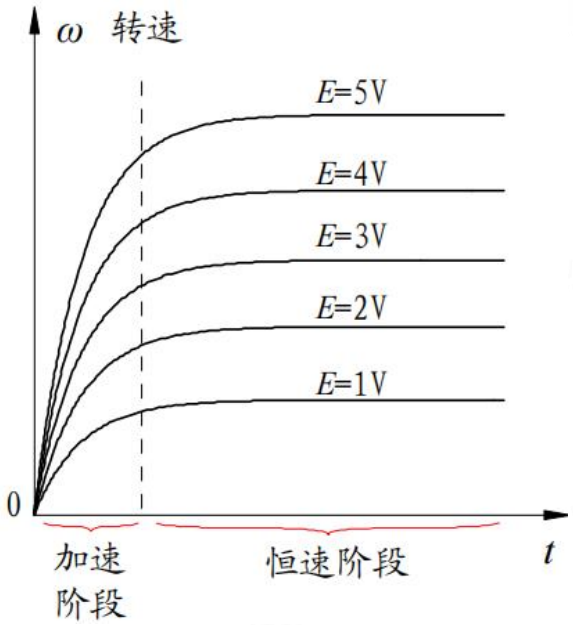


图 1-4 不同电压下电机转速

可以看到，电机运动分为加速阶段和恒速阶段，且在加速阶段中加速度近似和施加在电机上的电压成正比。控制平衡的时候，我们每次控制电机的时间都很短，可以看作电机一直处于加速阶段。所以控制小车的加速度就转变为了控制电机上的电压。

输出到电机的电压变化可以通过 PWM（脉冲宽度调制）来实现。PWM 调制技术通过改变每周期脉宽占比，从而使电机两端的平均电压得到改变。PWM 输出的平均电压公式为：

$$U_d = \frac{1}{T}\int_0^T U_s dt = DU_m$$

式中，T 为脉冲时钟周期，$U_s$为瞬时电压，$U_m$为电压幅度最大值，D 为 PWM 电压占空比(0<D<1)。可以看出平均电压与 PWM 电压的占空比成正比，改变单片机的 PWM 数值大小，就能成比例地改变电机的电压。

综合上面的分析，控制 PWM 大小就相当于控制小车的加速度大小。我们的加速度算法$a = k_1\theta + k_2\theta'$同样适用于控制电机的 PWM，这其实就是直立环的 PD 控制器。

## 1.2速度控制

如果小车出现倾角，在直立控制的作用下就会使小车在倾斜的方向加速，我们可以利用小车的这个特性来进行速度控制。控制速度实际上就变成了控制小车

的倾角。

获取车轮的速度可以通过读取编码器数值来获得。小车的速度控制对快速性要求并不高，但是对于准确性有一定的要求。PID 控制中微分(Didderential)控制主要的作用是减少振荡，加快稳定速度，积分(Intergral)控制主要的作用是减少静态误差，所以我们可以建立速度 PI 闭环控制，输出一个角度使小车达到目标速度，这其实就是串级 PID 控制，让速度控制的输出作为角度控制的输入，角度控制的输出直接作用于电机上。

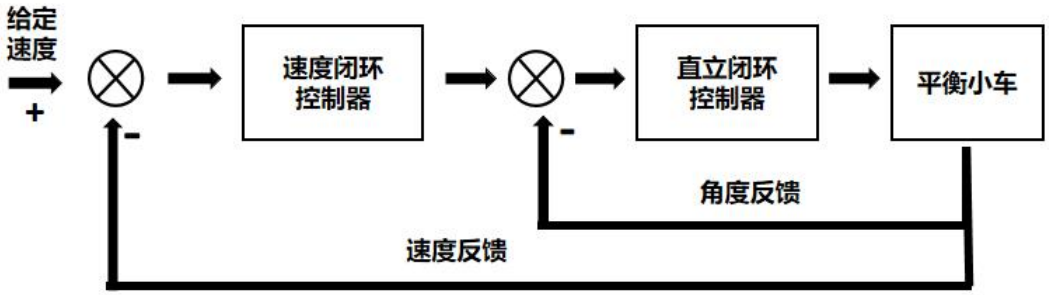


图 1-5 串级 PID 系统

假如速度环输出为$\theta_1$，作为目标角度输入于直立环，直立环的输出$a$直接作用于电机，使小车产生一个倾角，那么有如下关系：

$$a = kp * (\theta - \theta_1) + kd * \theta'$$

$$\theta_1 = kp_1 * e(k) + ki_1 * \sum e(k)$$

式中，$\theta$为当前小车的倾角，$\theta'$为倾角的微分，即角速度，$e(k)$为速度环中目标速度与当前速度的偏差，$\sum e(k)$为偏差的积分项。我们把这两个式子进行合并，可以得到下面公式：

$$a = kp * \theta + kd * \theta' - kp[kp_1 * e(k) + ki_1 \sum e(k)]$$

$a$ 为直接输出于小车的 PWM。观察式子可以知道，这个串级 PID 系统实际上是由一个 PD 控制器和一个 PI 控制器组成，我们可以分拆优化为两个控制环分别叠加到电机 PWM 上。

## 1.3转向控制

令两电机的转速不同就可以实现转向，所以我们需要分别加不同的电压在左右两个电机上，从而让电机转速产生差值。目标转速越大则两轮差速越大，则可以得到简单的比例控制，代码如下：

```
int Turn(float gyro)
{
     static float Turn_Target,turn,Turn_Amplitude=54;
     float Kp=Turn_Kp,Kd;  //修改转向速度，请修改 Turn_Amplitude 即可
    //==================遥控左右旋转部分================//
     if(1==Flag_Left)           Turn_Target=-Turn_Amplitude/Flag_velocity;
     else if(1==Flag_Right)   Turn_Target=Turn_Amplitude/Flag_velocity;
     else Turn_Target=0;
     if(1==Flag_front||1==Flag_back)  Kd=Turn_Kd;
     else Kd=0;   //转向的时候取消陀螺仪的纠正
  //==================转向 PD 控制器================//
     turn=Turn_Target*Kp/100+gyro*Kd/100;//结合 Z 轴陀螺仪进行 PD 控制
     return turn;     //转向环 PWM 右转为正，左转为负
}
```

由于小车转向时由较大的转动惯量，而且我们希望小车直行时能尽量的走直线，所以引入 Z 轴角速度反馈。

# 2. 角度与角速度测量

## 2.1 角度与角速度获取

直立控制是通过角度与角速度反馈来进行的，所以角度与角速度的测量至关重要。本系统使用 MPU6050 作为姿态传感器，集成一个加速度传感器和一个陀螺仪，可以输出三轴的加速度与角速度。角速度的获取可以通过陀螺仪来直接读取，角度的获取可以有两种方法来测量：一是通过加速度计的加速度分量来计算，二是通过陀螺仪输出的角速度进行积分获得。

MPU6050 的坐标系定义如图 2-1。

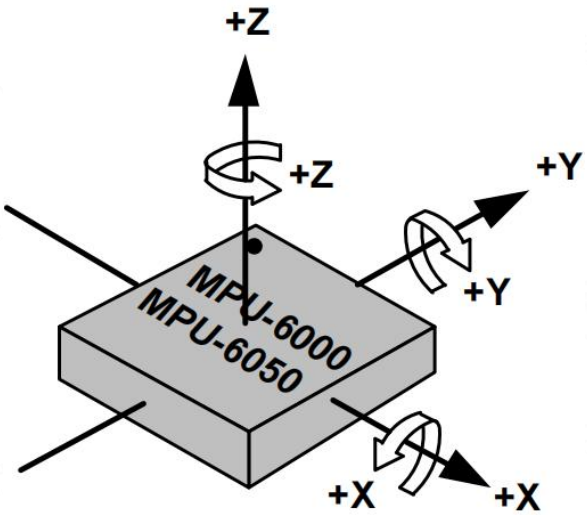


图 2-1 MPU6050 坐标系

当传感器的正方向 Z 轴垂直指向天空时，由于此时受到地球重力的作用，此时加速度计 Z 轴的读数应为正，而且理想情况下应为$g$。注意此时读取加速度计并不是重力加速度，而是物体自身的运动加速度，正因为自身的运动加速度与重力加速度大小相等方向相反，芯片才能保持静止。

当传感器静止不动时，我们仅绕 X 轴旋转一定的角度$\theta$，此时加速度方向一直与 X 轴垂直，X 轴并无加速度分量，忽略 X 轴，把加速度分解，如图 2-2 所示，可以很容易就算出传感器绕 X 轴的角度。

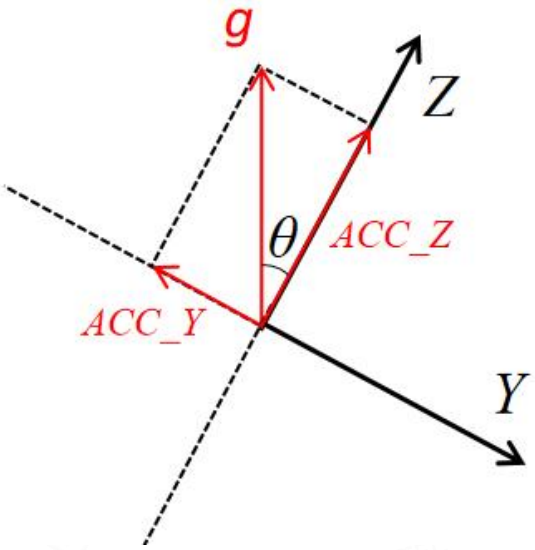


图 2-2 绕 X 轴加速度计

可知：

$$tan\theta = ACC\_Y/ACC\_Z$$

故

$$\theta = arctan(ACC\_Y/ACC\_Z)$$

当仅绕 Y 轴旋转时也是同样的原理。当绕 Z 轴旋转时，因为重力加速度固定为 Z 轴方向，故在 X 与 Y 轴无加速度分量，仅仅通过加速度计无法得出绕 Z 轴角度，若想得到绕 Z 轴角度，只能通过角速度积分获得，但因为有偏差，一段时间后将不再具有参考意义。

物体的旋转运动就是绕三轴旋转角度的叠加，我们读取加速度计的数据，根据公式进行处理就可以获取相应的姿态角。故小车绕 X 轴与 Y 轴的角度可用以下公式算出：

$$\begin{cases} \theta\_X = arctan(ACC\_Y/ACC\_Z) \\ \theta\_Y = arctan(ACC\_X/ACC\_Z) \end{cases}$$

在完全静止的情况下确实通过加速度计就可以获取到所需的角度，但是在实际应用中，小车因为车身摆动等情况会产生加速度，它叠加在测量信号上会无法准确地反映出车模的倾角，图 2-3 所示。

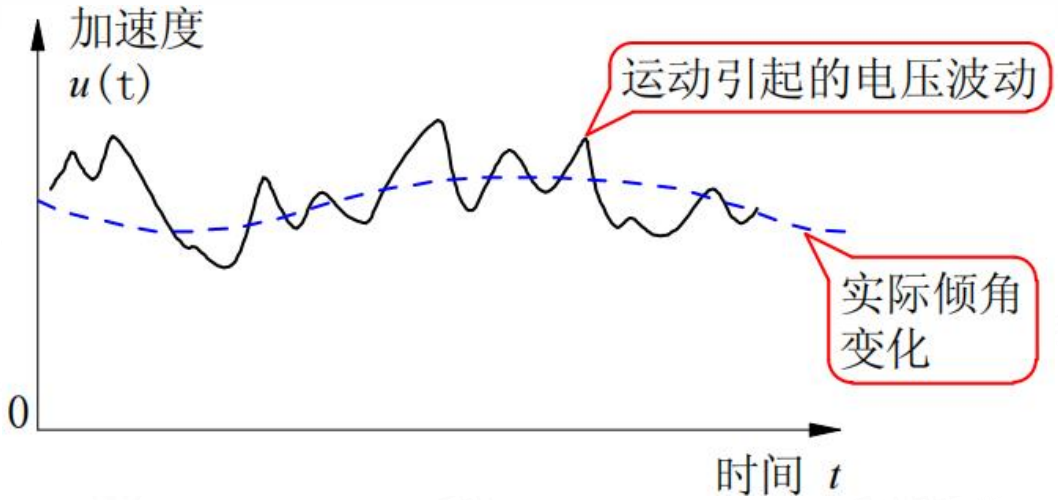


图 2-3 测量倾角与实际倾角对比

传感器安装的高度越低越能抑制因运动产生的加速度，但是还是无法切底消除这种影响。

那么通过陀螺仪的角速度进行积分得到角度呢？如果测量的角速度存在微小的误差或漂移，经过积分运算之后，形成积累误差，随着时间增加，角度信息将不再准确，这一个方法也是不太可行的。

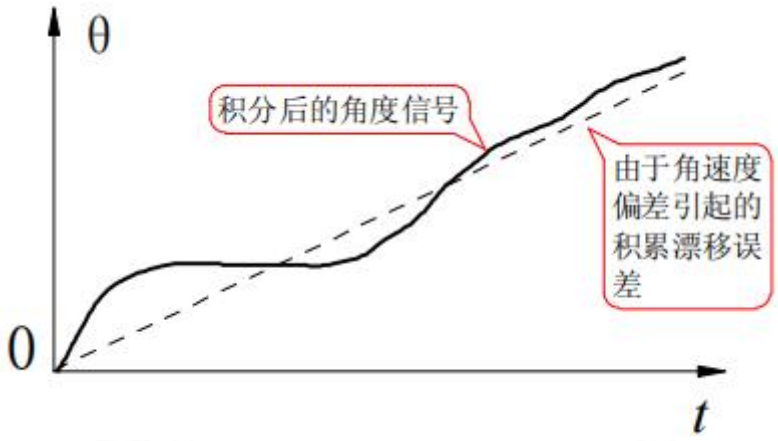


图 2-4 陀螺仪积累误差

我们可以综合加速度计和陀螺仪的角度，进行滤波和平滑处理得到准确的角度。程序提供了三种得到准确的角度的算法：1.DMP 算法 2.互补滤波算法 3.卡尔曼滤波算法。

DMP 算法是 MPU6050 自带的一种滤波方法，只要进行一些初始化，使用官方的库函数就可以读取出四元数，根据公式就可以计算出姿态角。

## 2.2互补滤波

通过加速度计和陀螺仪获取的角度都有一定的缺点，加速度计获取的角度长期来看比较准确，但是波动大，可以认为其掺杂了高频噪声；陀螺仪获取的角度短时间比较准确，但有积分误差，可以认为其掺杂了低频噪声。我们可以分别让他们通过一个低通滤波器和一个高通滤波器然后叠加在一起，这就是互补滤波算法。

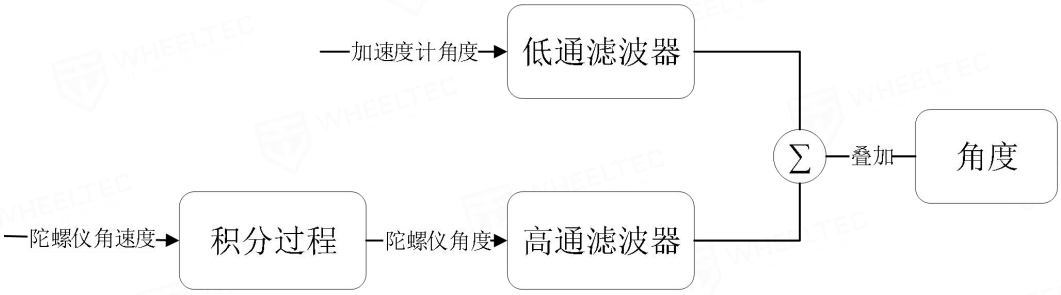


图 2-5 互补滤波过程

假设这个低通滤波器其系统函数为$G_1(s) = 1/(\tau s + 1)$，那么高通滤波器系统函数为$G_2(s) = \tau s/(\tau s + 1)$，$\tau$是滤波器的时间常数。可以注意到$G_1(s) + G_2(s) = 1$，必须要保证滤波后增益为 1，这就是所谓“互补”。

设通过加速度计转换到的角度的拉氏变换为$\theta_m(s)$，陀螺仪输出角速度拉氏变换为$\Omega(s)$，输出为$\theta(s)$，可以画出下面系统框图。

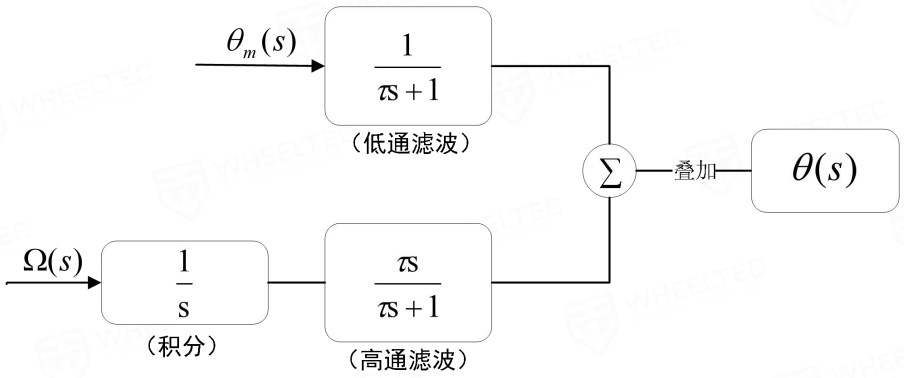


图 2-6 互补滤波系统框图

经过一些处理，可以把积分项去掉，系统简化为角速度乘以比例系数$\tau$再和加速度计的角度叠加，通过一个低通滤波器后输出。系统框图简化如下：

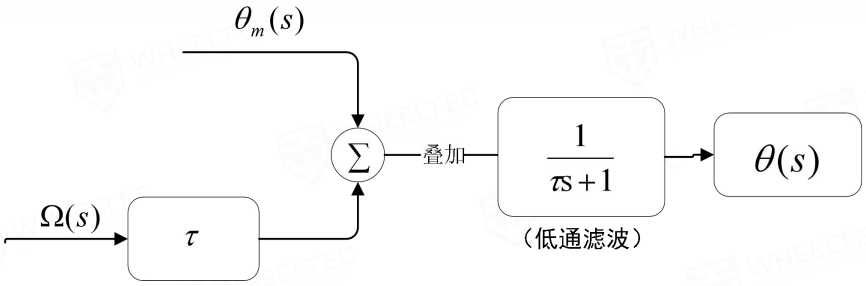


图 2-7 系统框图简化

可以看出，经过简化后少了纯积分环节，那么系统便不再具有积累误差。

我们来推导一下低通滤波器的时域表达式。设低通滤波器系统函数为

$$G_1(s) = 1/(\tau s + 1)$$

设输入为$x(s)$，输出为$y(s)$，那么有

$$y(s) = G_1(s)x(s) = [1/(\tau s + 1)]x(s)$$

作反变换可得时域表达式为

$$y'(t) = \frac{1}{\tau}x(t) - \frac{1}{\tau}y(t)$$

因便于计算机程序处理，我们作离散化处理并简化，可以得到离散时域表达式如下

$$y(n) = \frac{\Delta t}{\Delta t + \tau}x(n) + \frac{\tau}{\Delta t + \tau}y(n-1)$$

令$K = \frac{\Delta t}{\Delta t+\tau}$，那么表达式可以简化为

$$y(n) = Kx(n) + (1-K)y(n-1)$$

这就是低通滤波器的时域表达式，在我们的互补滤波和速度环里面都有应用。

同样的道理，互补滤波中，角速度乘以比例系数 τ 再和加速度计的角度叠加，经过低通滤波器后时域表达式为：

$$\theta(n) = \frac{\Delta t}{\Delta t + \tau}\theta_m(n) + \frac{\tau}{\Delta t + \tau}[y(n-1) + \Delta t\Omega(n)]$$

令$K = \frac{\Delta t}{\Delta t+\tau}$，那么可以简化为：

$$\theta(n) = K\theta_m(n) + (1-K)[y(n-1) + \Delta t\Omega(n)]$$

对应我们的代码为：

```
angle = K1 * angle_m+ (1-K1) * (angle + gyro_m * dt);
```

可以看出，互补滤波就是通过加速度计获取的角度对陀螺仪积分的角度进行校准，从而积分的角度逐步跟踪到加速度传感器所得到的角度。K1 与 1-K1 是对

这两个角度取不同的权重，可以表示我们对不同数据的信任程度。

## 2.3卡尔曼滤波

传感器测量的数据总是有很多的不确定性，比如有很多的噪声，而这些噪声大部分都符合高斯分布。对于我们的小车，输入是角速度，输出是角度，这是一个线性的系统。如果一个系统是线性的系统，而且这些不确定性是符合高斯分布的，那么我们就可以使用卡尔曼滤波算法进行最优估计。卡尔曼滤波的思想就是使用系统的状态方程预测当前的值，使用传感器测出来的观测值来修正这个预测值。与互补滤波一样，可以选择不同的权重来实现，但是这个权重是动态变化的。

我们知道一个系统的状态方程和传感器的测量方程如下：

状态方程：$X_k = AX_{k-1} + BU_{k-1} + W_{k-1}$

测量方程为：$Z_k = HX_k + V_k$

其中，$X_k$和$Z_k$分别表示系统的状态矩阵和系统的观测量矩阵，$U_{k-1}$是系统的输入量，A 和 B 是系统参数，一般为矩阵的形式。H 为观测矩阵，表示把实际量转换为观测量的一个参数。在实际过程中，会掺杂有噪声，$W_{k-1}$和$V_k$分别表示过程噪声和观测噪声，他们服从高斯分布，即 W~N(0,Q)，V~N(0,R)，Q 和 R 是一个协方差矩阵。

卡尔曼滤波可以分为两大步骤，分别叫做预测和更新。下面给出卡尔曼的公式。

预测部分：

$$\widehat{X}_k^- = A\widehat{X}_{k-1} + BU_{k-1} \quad (1)$$

$$P_k^- = AP_{k-1}A^T + Q_{k-1} \quad (2)$$

更新部分：

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \quad (3)$$

$$\widehat{X}_k = \widehat{X}_k^- + K_k(Z_k - H\widehat{X}_k^-) \quad (4)$$

$$P_k = (I - K_k H)P_k^- \quad (5)$$

上面公式中，上标带负号的表示根据状态方程计算出来的数值，是理想的输出，也叫先验估计，后面需要对这个值进行修正。上面带尖的是系统某时刻的最优估计，是滤波后的输出，也叫后验估计。Q 是系统过程噪声协方差，R 为测量噪声协方差，$K_k$是卡尔曼增益，用于修正先验估计，$P_k$表示 k 时刻后验估计协

方差矩阵，$P_k^-$表示先验估计协方差矩阵，$I$是单位矩阵。

分析一下具体的小车系统。当小车静止时，陀螺仪依旧会输出一个数值，这个数值就是静差，而且这个值是变化的。令小车 k 时刻的角度为$Angle_k$，k 时刻的角度为$Gyro_k$，小车陀螺仪 k 时刻的静差为$Gyro_bias_k$，可以列出下面方程：

$$Angle_k = Angle_{k-1} + (Gyro_{k-1} - Gyro\_bias_{k-1})dt + W_{k-1}$$

$$Gyro\_bias_k = Gyro\_bias_{k-1} + W_{k-1}$$

把它写成矩阵形式，即得到小车的状态方程：

$$X_k = \begin{pmatrix} Angle \\ Gyro\_bias \end{pmatrix}_k = \begin{pmatrix} 1 & -dt \\ 0 & 1 \end{pmatrix}\begin{pmatrix} Angle \\ Gyro\_bias \end{pmatrix}_{k-1} + \begin{pmatrix} dt \\ 0 \end{pmatrix}Gyro_{k-1} + \begin{pmatrix} W_{k-1} \\ W_{k-1} \end{pmatrix}$$

同理，令小车通过加速度计计算得到的角度为$Accel_angle_k$，则小车的测量方程可以写为：

$$Z_k = Accel\_angle_k = (1 \quad 0)\begin{pmatrix} Angle \\ Gyro\_bias \end{pmatrix}_k + R_k$$

所以我们就得到了小车的系统参数矩阵 A、B 和观测矩阵 H。接下来就可以开始计算迭代，进行最优估计。

首先进行先验估计，即对应公式的预测部分：

$$\widehat{X_k^-} = \begin{pmatrix} \widehat{Angle}^- \\ \widehat{Gyro\_bias}^- \end{pmatrix}_k = \begin{pmatrix} 1 & -dt \\ 0 & 1 \end{pmatrix}\begin{pmatrix} \widehat{Angle} \\ \widehat{Gyro\_bias} \end{pmatrix}_{k-1} + \begin{pmatrix} dt \\ 0 \end{pmatrix}Gyro_{k-1}$$

可以把它写为方程：

$$\begin{cases} \widehat{Angle}_k^- = \widehat{Angle}_{k-1} + (\widehat{Gyro}_{k-1} - \widehat{Gyro\_bias}_{k-1})dt \\ \widehat{Gyro\_bias}_k^- = \widehat{Gyro\_bias}_{k-1} \end{cases}$$

协方差矩阵先验估计公式为：$P_k^- = AP_{k-1}A^T + Q_{k-1}$，设协方差矩阵 $\mathrm{P} = \begin{pmatrix} \mathrm{P}_{00} & \mathrm{P}_{01} \\ \mathrm{P}_{10} & \mathrm{P}_{11} \end{pmatrix}$，$Q_{k-1}$为系统过程的协方差矩阵，即

$$Q_{k-1} = \begin{pmatrix} cov(Angle, Angle) & cov(Angle, Gyro\_bias) \\ cov(Gyro\_bias, Angle) & cov(Gyro\_bias, Gyro\_bias) \end{pmatrix}_{k-1} * dt$$

$$= \begin{pmatrix} Q\_angle & 0 \\ 0 & Q\_gyro \end{pmatrix}_{k-1} * dt$$

因为为$Angle$和$Gyro_bias$是相互独立的，故$cov(Angle, Gyro_bias) = cov(Gyro_bias, Angle) = 0$。可以看到我们的协方差矩阵最终是由加速度算出的角度的方差和陀螺仪的角速度方差组成，矩阵后面乘了一个$dt$，这是因为$Q_k$是与 k 时刻的时间相关的，当距离上次更新时间越长，那么这个方差也会越大，就比如陀螺仪积分出来的角度，时间越长，那么偏差累积也会越大。把参数代入，可

得协方差矩阵先验估计如下：

$$P_k^- = AP_{k-1}A^T + Q_{k-1} = A\begin{pmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{pmatrix}_{k-1}A^T + \begin{pmatrix} Q\_angle & 0 \\ 0 & Q\_gyro \end{pmatrix}_{k-1} * dt$$

$$= \begin{pmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{pmatrix}_{k-1}$$

$$+ \begin{pmatrix} -dtP_{10} - dtP_{01} + dt^2P_{11} + dt * Q\_angle & -dtP_{11} \\ -dtP_{11} & dt * Q\_gyro \end{pmatrix}_{k-1}$$

这个公式可以很容易在程序中实现。

我们已经完成了预测部分，那么就可以进行更新修正。先计算卡尔曼增益，公式为：$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}$，计算逆矩阵部分$(H_k P_k^- H_k^T + R_k)^{-1}$，因为 H 为一个一行两列的矩阵，故这部分算出来应该是一个数而不是矩阵。算出来结果为 $1/[(P_{00})_k^- + R_k]$，其中$(P_{00})_k^-$是$P_k^-$矩阵的第一行第一列的元素。最后算出的卡尔曼增益为：

$$K_k = \begin{pmatrix} K_0 \\ K_1 \end{pmatrix} = \begin{pmatrix} (P_{00})_k^- / [(P_{00})_k^- + R_k] \\ (P_{10})_k^- / [(P_{00})_k^- + R_k] \end{pmatrix}$$

有了卡尔曼增益，那么就可以修正系统的输出了，根据公式 4 和公式 5 就可以算出系统输出的最优估计和协方差矩阵的更新，需要使用到传感器的测量值 $Accel\_angle_k$。据协方差更新公式，可得协方差更新为：

$$P_k = (I - K_k H)P_k^- = \{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} K_0 \\ K_1 \end{pmatrix}(1 \quad 0)\}\begin{pmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{pmatrix}_k^-$$

$$= \begin{pmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{pmatrix}_k^- + \begin{pmatrix} -K_0P_{00} & -K_0P_{01} \\ -K_1P_{00} & -K_1P_{01} \end{pmatrix}_k^-$$

通过一次次的迭代，最终使输出的角度平滑准确。

# 3. 电路设计

## 3.1 整体电路设计

分析我们的需求，可以得到下面的主要的电路系统框图。

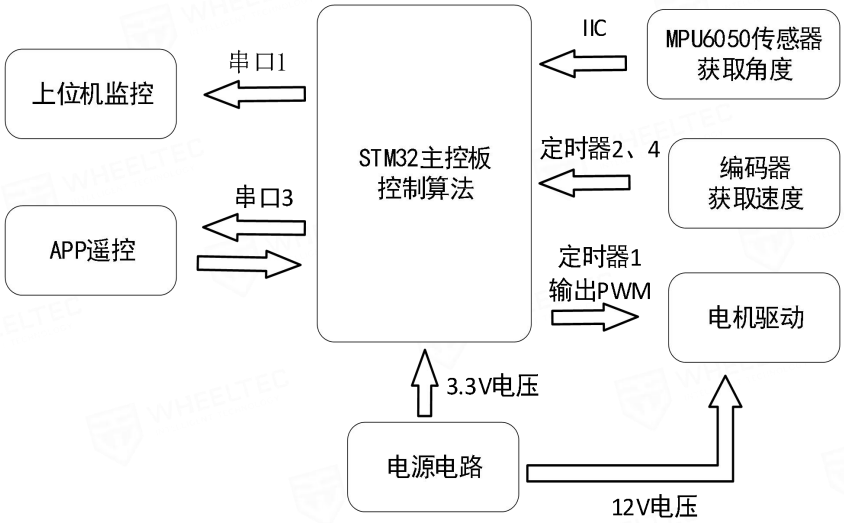


图 3-1 电路系统框图

根据框图，我们把电路主要划分为几个模块：

1.最小单片机系统：整合处理所有模块的数据，并进行控制，实现所需功能。

2.串口电路：与上位机进行通信。

3.电源系统：提供所有模块的供电电压。

4.电机驱动电路：输出电压驱动电机。

5.姿态检测电路：获取小车的角度与角速度信息。

## 3.2 32 底层最小系统介绍

小车使用的主控芯片为 32F103C8T6，具有丰富的片上外设，而且性能高，功耗低，非常适合用于平衡小车。主控板主要组成为电源电路、晶振电路、复位电路等，最小系统原理图如 3-2 所示

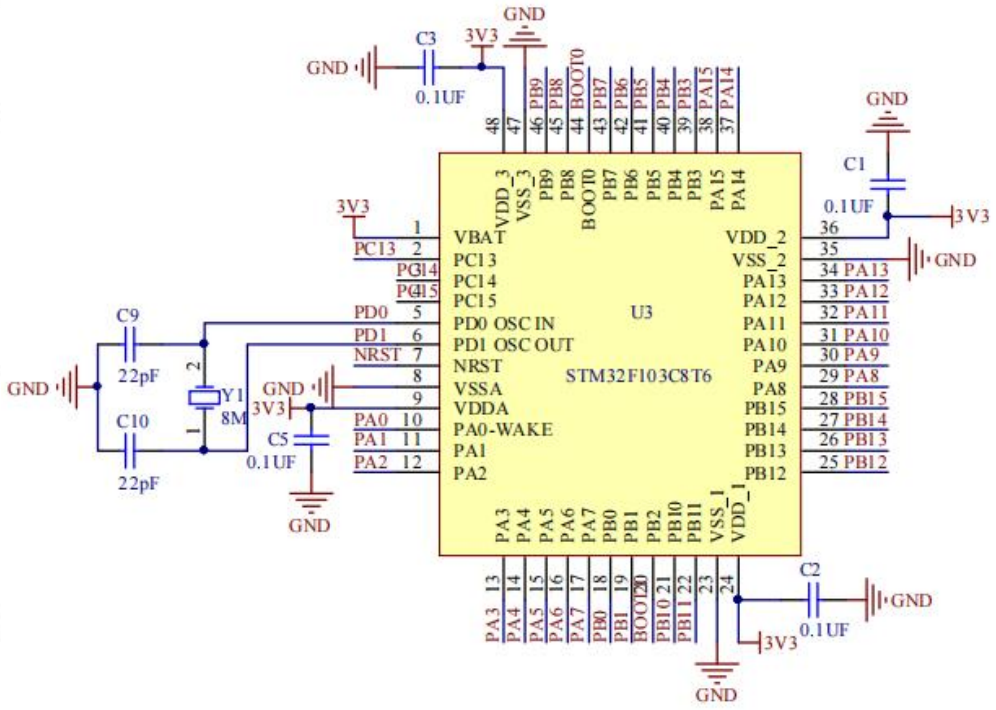


图 3-2 32 芯片最小系统

芯片采用 8MHz 无源晶振，可倍频到 72MHz。主控芯片主要是采集传感器的数据进行高速处理、实现控制算法的运行和与上位机进行通信等。

## 3.3 串口电路设计

32 芯片的串口可以用于一键下载程序和上位机数据查看调试等功能，仅使用手机 MicroUSB 数据就能实现功能，所以串口电路的设计很有必要。

CH340G 芯片是我们选用的 USB 转串口的芯片，电路原理图如图 3-3。

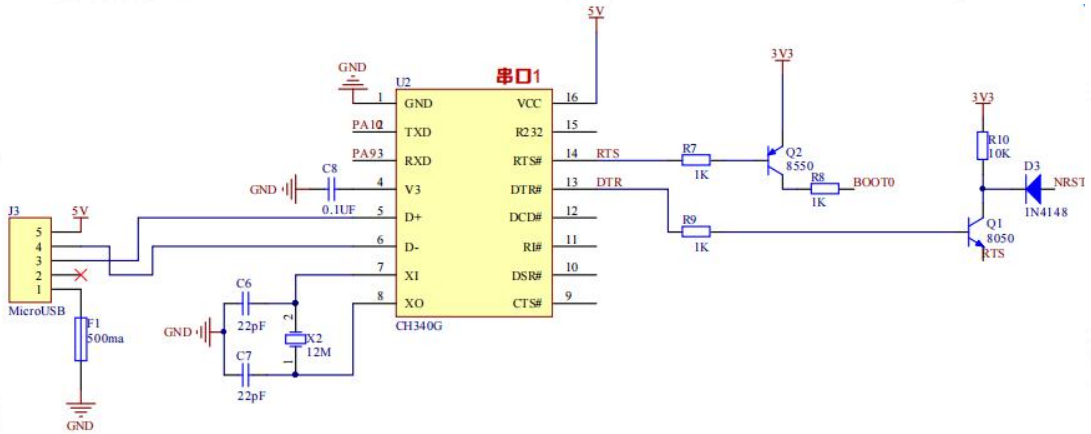


图 3-3 USB 转 TTL 电路

电脑与单片机之间不能通过 USB 直接进行通信，必须要使用 USB 转串口进

行通信。芯片通过 USB 转换成 TTL 串口，输出和输入电压是根据芯片供电电压进行自适应的，若使用 5V 供电，那么串口输出和采样都是 5V，3.3V 同理。5V 供电时芯片 V3 引脚需要接一个 0.1uF 的电容到地，3.3V 供电时直接将 V3 脚与 3.3V 电源引脚短接就可以了。

## 3.4电源电路设计

小车使用锂电池供电，输出电压为 12V，我们需要提供三种电压给系统使用。一是 12V 电压，直接输入电机驱动芯片，以驱动电机；二是 5V 电压，电机驱动芯片内部逻辑部分和编码器、超声波模块和串口芯片等的电源供给；最后是 3.3V 电压，供给单片机使用。

我们选用降压芯片 LMS2596，这是一个降压型电源管理单片集成电路，输出电流可达 3A，同时具有很好的线性与负载调节特性。固定输出版本有 3.3V、5V、12V，可调版本可以输出小于 37V 的各种电压。有了开关电压调节器，可以非常方便地调整供电电压。主控板降压芯片选用 AMS1173，可以把 5V 电压降压为稳定的直流 3.3V 供给芯片使用。电源电路原理图如图 3-4。

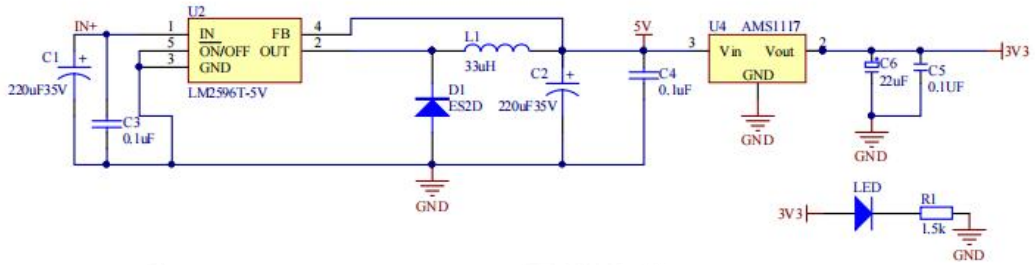


图 3-4 电源电路

因 32 芯片的 ADC 最大检测电压为 3.3V，故令电池电压通过分压后再送入单片机的 ADC 检测通道，分压后电压检测值为电池电压的 1/11，通过简单的分压公式可以计算出来。

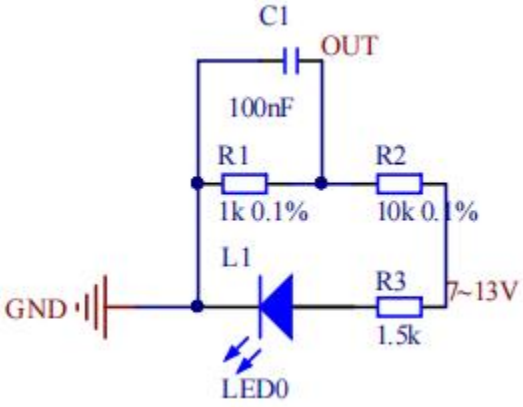


图 3-5 电池电压分压

## 3.5MPU6050 传感器电路设计

我们选用姿态传感器 MPU6050，集成了三轴加速度计和三轴陀螺仪，以及一个可扩展的数字运动处理器 DMP，可用 IIC 接口连接一个第三方的数字传感器，如磁力计等。此集成模块使得角速度和加速度传感器处于同样的环境中，对消除零点漂移有着积极的作用。MPU6050 内部自带 ADC，在获取姿态模拟信息后，可以把它直接转化为数字量，我们只需读取相应寄存器即可，十分方便。为了满足需要和测量的准确，传感器提供了不同的量程，陀螺仪范围可从±250°到±2000°，加速度计范围可从±2g 到±16g，满足大部分使用场景。我们程序中设置了陀螺仪的量程为±2000°，加速度计的量程为±2g。

MPU6050 传感器模块原理图如图 3-6 所示。模块使用 IIC 进行通信，其 SDA 和 SCL 接口需要使用上拉电阻上拉到 VCC，IIC 通信协议依靠软件模拟进行驱动。

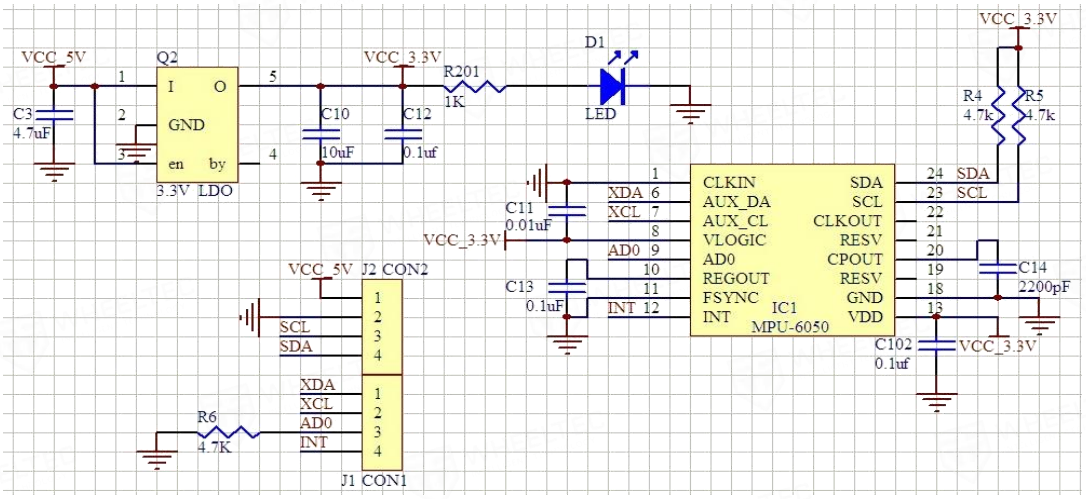


图 3-6 MPU6050 模块原理图

## 3.6电机驱动电路设计

由于电机有两个，故需要两组电机驱动电路桥电路。我们选用 TB6612FNG 电机驱动芯片，此芯片为双通道电路输出，可以驱动两个电机，其中的 A 和 B 分别为一组电机的输入端和输出端，原理图如下所示。

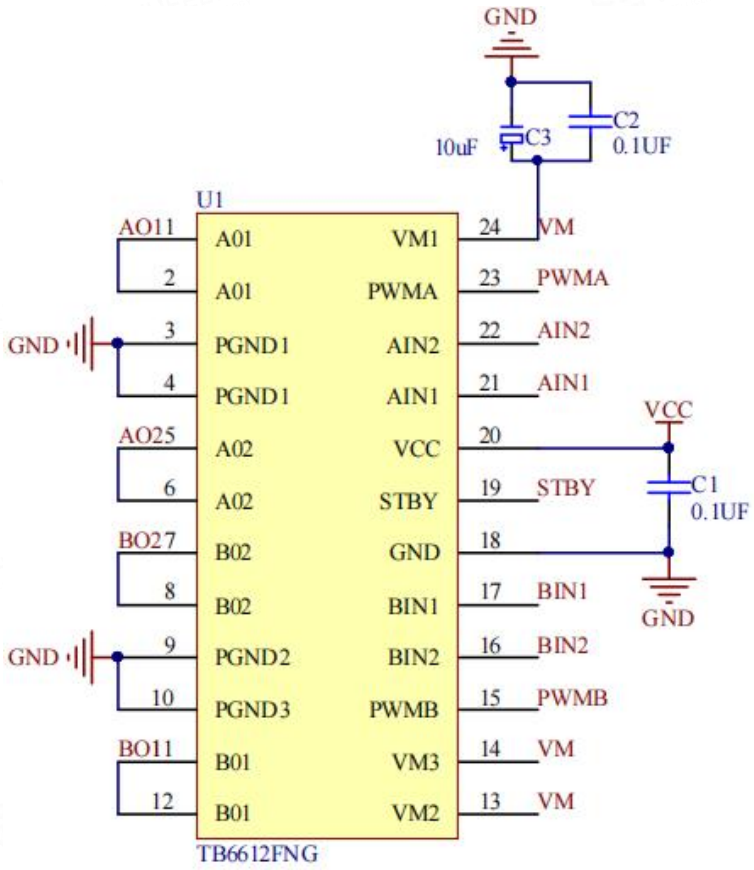


图 3-7 电机驱动电路

电机集成霍尔传感器，能够读出脉冲计数，从而得到电机速度。在程序中，我们使用 M 法测速，即在一定时间周期内，测量编码器输出的脉冲个数$M_1$来计算转速，用个数除以时间就可以得到编码器输出脉冲的频率。频率$f_1 = M_1/T_c$。因存在测量时间内首尾变个脉冲问题，可能会有两个脉冲的误差，但在高速下可以忽略不计。

假设电机转动一圈可以产生 Z 个脉冲，其中 Z=4*编码器线数*电机减速比，这里 4 代表四倍频，同时采集 A 和 B 相的上升沿和下降沿，用频率$f_1$除以一圈脉冲个数就可以得到单位时间内的转速(单位：转/分钟)：

$$N = f_1/Z * 60 = 60M_1/ZT_c \qquad (r/min)$$

# 4. 软件开发

双轮平衡车系统能够正常运转，需要硬件与软件之间的相互合作。双轮平衡小车的软件编写包含：主程序编写、电机驱动、直立控制功能算法实现、编码器信息获取、超声波模块功能实现、蓝牙接收和角度滤波算法等程序实现。

## 4.1主程序设计

本系统中，所有程序的初始化声明均在主程序进行，主程序决定了整个系统的框架，可以主程序为基础拓展其他的功能模块。

主控芯片采用 32F103C8T6 芯片，在程序中充分利用了改芯片的外设资源。所用到的外设资源如下：

ADC：采集电阻分压后的电池电压

TIM1：初始化为 PWM 输出，CH1，CH4 输出双路 10KHZ 的 PWM 控制电机

TIM2：初始化为正交编码器模式，硬件采集编码器 1 的数据

TIM3：CH3 初始化为超声波的回波采集接口

TIM4：初始化为正交编码器模式，硬件采集编码器 2 的数据

USART1：通过串口 1 把数据发送到串口调试助手

USART3：通过串口 3 接收蓝牙遥控的数据，接收方式为中断接收。并发送数据给 APP

IIC：利用 IO 模拟 IIC 取读取 MPU6050 的数据，原理图上 MPU6050 连接的是硬件 IIC 接口，但是因为 32 芯片的硬件 IIC 不稳定，采用模拟 IIC

SPI：利用 IO 模拟 SPI 去驱动 OLED 显示屏

GPIO：读取按键输入，控制 LED，控制电机使能和正反转

SWD：提供用于在线调试的 SWD 接口

EXTI：由 MPU6050 的 INT 引脚每 5ms 触发一次，作为控制时间基准

主函数主要做一些初始化工作，后面都有注释。然后进入一个死循环，主要控制功能在外部中断函数里面，5ms 定时中断由 MPU6050 的 INT 引脚触发。我们把比较耗时的 OLED 显示和 APP 发送数据等函数放在主函数里面，因为 5ms 定时中断由 MPU6050 的 INT 引脚触发，需严格保证采样和数据处理时间的同步。

# 4.2电机驱动程序设计

电机驱动程序设计主要是负责决定左右电机的运行方向和速度。我们知道，电机驱动电压为 12V，只靠芯片输出驱动电压是不可行的，所以需要使用电机驱动芯片 TB6612FBG，而单片机用于驱动电机芯片。

TB6612FBG 与 32 芯片接线表如下，PWMB、PWMA 分别用于控制左右电机的转速，BIN1、BIN2 用于控制用于控制左电机的正反转，AIN1、AIN2 用于控制右轮的正反转。

| TB6612FBG | PWMA | AIN1 | AIN2 | PWMB | BIN1 | BIN2 |
|---|---|---|---|---|---|---|
| 32 底层主控板 | PA8 | PB14 | PB15 | PA11 | PB13 | PB12 |

表 4-1 TB6612FBG 与 32 芯片接线

其中 PA8 是单片机的定时器 1 的通道 4，输出 PWM 用于驱动右轮电机；PA11 接定时器 1 的通道 1，输出 PWM 驱动左轮电机。以下以右轮电机为例说明 AIN1 与 AIN2 驱动电机方向的真值表。

| AIN1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|
| AIN2 | 0 | 0 | 1 | 1 |
| 功能 | 停止 | 正转 | 反转 | 停止 |

表 4-2 电机驱动真值表

注意，必须要有 PWM 的输入才有电机线 AO1 和 AO2 的输出，才能驱动电机转动。PWM 的大小对应电机转速大小。

电机初始化步骤如下：

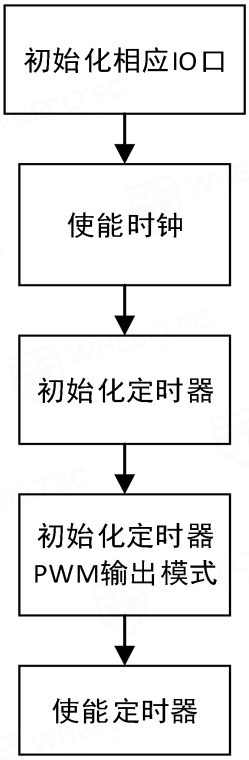


图 4-1 电机初始化步骤

具体代码可在 motor.c 里的 void MiniBalance_PWM_Init(u16 arr,u16 psc)函数里查看。

驱动电机正反转与转速控制的代码实现如下：

```
void Set_Pwm(int motor_left,int motor_right)
{
  if(motor_left>0)          BIN1=1,          BIN2=0; //前进
  else                      BIN1=0,          BIN2=1; //后退
  PWMB=myabs(motor_left);   //PWM 大小对应转速的大小
  if(motor_right>0)         AIN2=1,          AIN1=0; //前进
  else                      AIN2=0,          AIN1=1; //后退
  PWMA=myabs(motor_right);
}
```

PWMA 与 PWMB 是寄存器 TIM1_CCR1 与 TIM1_CCR4 的宏定义，此寄存器为输出比较的比较值，令定时器的计数与这个值比较，输出高或低电平，实现 PWM 方波输出。

## 4.3直立控制功能算法设计

直立控制采用经典的 PID 算法，其结构简单易实现，无需精确的模型便可实现较好的控制，是当下最为成熟的一种控制策略。PID 控制主要包含比例、积

分、微分这三项控制，是一个通过目标值与当前值进行比较，得到控制偏差，然后通过对偏差的比例、微分与积分进行控制，使偏差趋向于零的过程。PID 公式如下：

$$u(t) = k_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

式中，$e(t)$为当前时刻目标值与当前值的偏差，$u(t)$为输出的控制量。为方便计算机处理，作离散化处理：

$$u(k) = k_p e(k) + K_i \sum e(k) + K_d[e(k) - e(k-1)]$$

对于我们具体的系统，直立环控制使用 PD 控制，即比例-微分控制。通过角度与角速度（角度的微分）反馈，使小车达到平衡状态。反馈公式为：

$$PWM = K_p(Middle\_angle - \theta) + K_d(0 - \theta')$$

式中，Middle_angle 是我们直立环的目标角度，我们设定其为零，θ是当前的角度。通过反馈输出 PWM 驱动电机进行平衡。

代码实现如下：

```
int Balance(float Angle,float Gyro)
{
    float Angle_bias,Gyro_bias;
    int balance;
    Angle_bias=Middle_angle-Angle; //求出平衡的角度中值 和机械相关
    Gyro_bias=0-Gyro;
    balance=-Balance_Kp/100*Angle_bias-Gyro_bias*Balance_Kd/100;  //计算平衡控制的电机 PWM  PD 控制   kp 是 P 系数 kd 是 D 系数
   return balance;
}
```

为方便 APP 调参并且提高调试精度，我们已对所有 PID 参数放大了 100 倍，故在这里我们除以 100 获得真实的 PID 参数。

最基本的平衡不仅要求小车能保持不倒，还要求小车能尽量停在原地，即小车速度为 0，这时候需要加入速度环。故最基本的平衡需要两个环：直立环+速度环。

速度环使用 PI 控制器，我们尽量要求速度变化缓慢和平稳，减缓速度差值以减少对平衡环的影响，所以在速度环中，我们应用了一阶低通滤波，使速度缓慢变化。一阶低通滤波器时域表达式为：$y(n) = Kx(n) + (1-K)y(n-1)$，其中

K 为滤波系数，系数越小，滤波越平稳，曲线更平滑，但灵敏度低；系数越大，灵敏度越高，但结果不稳定。在程序中我们取滤波系数 K=0.2。速度环反馈公式为：

$$PWM = K_p e(k) + K_i \sum e(k)$$

$e(k)$是目标速度与当前速度的偏差，$\sum e(k)$是速度偏差的积分。代码实现如下，已去掉遥控控制等功能。

```
int Velocity(int encoder_left,int encoder_right)
{
    static float velocity,Encoder_Least,Encoder_bias;
     static float Encoder_Integral;
   //============速度 PI 控制器=====================//
  Encoder_Least =0-(encoder_left+encoder_right); //获取最新速度偏差=目标速度（此处为零）-测量速度（左右编码器之和）
    Encoder_bias *= 0.8;                    //一阶低通滤波器
    Encoder_bias+= Encoder_Least*0.2;         //一阶低通滤波器
  //相当于上次偏差的 0.8 + 本次偏差的 0.2，减缓速度差值，减少对直立的干扰
    Encoder_Integral +=Encoder_bias;   //积分出位移 积分时间：5ms
    if(Encoder_Integral>10000)  Encoder_Integral=10000; //积分限幅
    if(Encoder_Integral<-10000) Encoder_Integral=-10000; //积分限幅
    velocity=-Encoder_bias*Velocity_Kp-Encoder_Integral*Velocity_Ki;//速度控制
    if(Turn_Off(Angle_Balance)==1||Flag_Stop==1)   Encoder_Integral=0;
//电机关闭后清除积分
     return velocity;
}
```

在程序中作了积分限幅处理，以免输出的 PWM 超出执行器的能力范围。

## 4.4 编码器程序设计

电机自带霍尔编码器测速模块，能输出左右电机的实时速度。当电机旋转时，编码器会输出若干个脉冲方波。为了方便判断转向，一般会输出 AB 两路相位相差 90 度的脉冲信号。我们的编码器精度为 13 线，电机减速比为 30，故转一圈可以输出两路 390 个脉冲。如果使用 32 芯片中定时器的编码器接口模式 3，可以进行上升沿和下降沿计数，那么电机转一圈就可以计算 1560 个数，电机正转或反转，编码器计数方向可向上或向下。

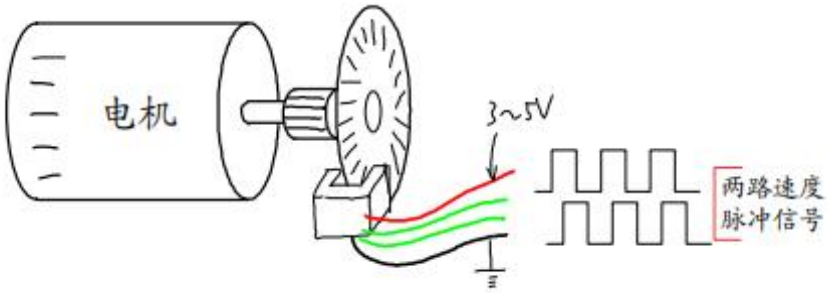


图 4-2 编码器输出脉冲

我们左轮的编码器 A 相和 B 相使用定时器 2 的通道 1 和通道 2，右轮编码器 A 相和 B 相使用定时器 4 的通道 2 和通道 1。编码器接线表如下：

| 左编码器 | A 相 | B 相 |
|---|---|---|
| 32 芯片 | PA0 | PA1 |

| 右编码器 | A 相 | B 相 |
|---|---|---|
| 32 芯片 | PB7 | PB6 |

表 4-3 编码器与 32 芯片接线

编码器初始化流程如下：

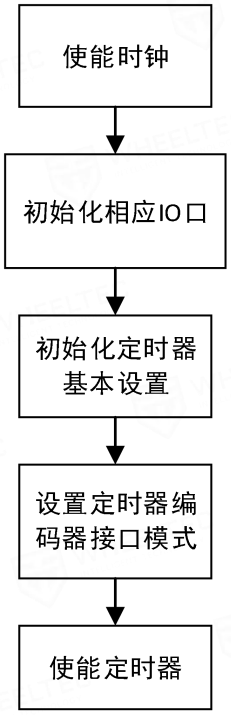


图 4-3 编码器初始化

具体代码可以在 encoder.c 中的 void Encoder_Init_TIM2(void)和 void Encoder_Init_TIM4(void)函数里面查看。

我们读取编码器函数如下：

```
int Read_Encoder(u8 TIMX)
{
   int Encoder_TIM;
   switch(TIMX)
     {
        case 2:  Encoder_TIM= (short)TIM2 -> CNT;  TIM2 -> CNT=0;break;
        case 3:  Encoder_TIM= (short)TIM3 -> CNT;  TIM3 -> CNT=0;break;
        case 4:  Encoder_TIM= (short)TIM4 -> CNT;  TIM4 -> CNT=0;break;
        default: Encoder_TIM=0;
     }
    return Encoder_TIM;
}
```

读取 TIMx_CNT 可以知道当前的计数值，读取完之后把寄存器清 0。因为初始化了编码器模式，当车轮后退的时候是从 0 开始往下计数，直接读取寄存器将会读到一个很大的数，而不是计数量，故需要对读取到的数据作一些处理，用强制类型转换可以解决这个问题。程序中把读取到的数值强制转换为 short 类型，我们知道，在计算机系统中，数值一律用补码来表示和储存，一个有符号变量最高位是他的符号位，1 表示负数，0 表示正数。定时器寄存器储存的是 16 位数据，把这个数据转换为有符号的 short 类型时，因为储存的是补码，那么，我们从 0 开始向下计数时，读出寄存器数据就得到了计数量的补码，自然得到了一个负数的计数量。

## 4.5超声波模块功能实现

超声波模块主要负责获取障碍物与小车之间的距离，以实现避障和跟随的功能。模块型号为 HC-SR04，工作原理为：给触发引脚一个至少 10us 的脉冲信号，此时模块会发出超声波，回波引脚会由低电平变为高电平，若模块接受到弹回来的超声波，回波口就会从高电平跳变到低电平，我们通过测量高电平的持续时间，通过计算公式就可以算出距离。时序图如图 4-4 所示。

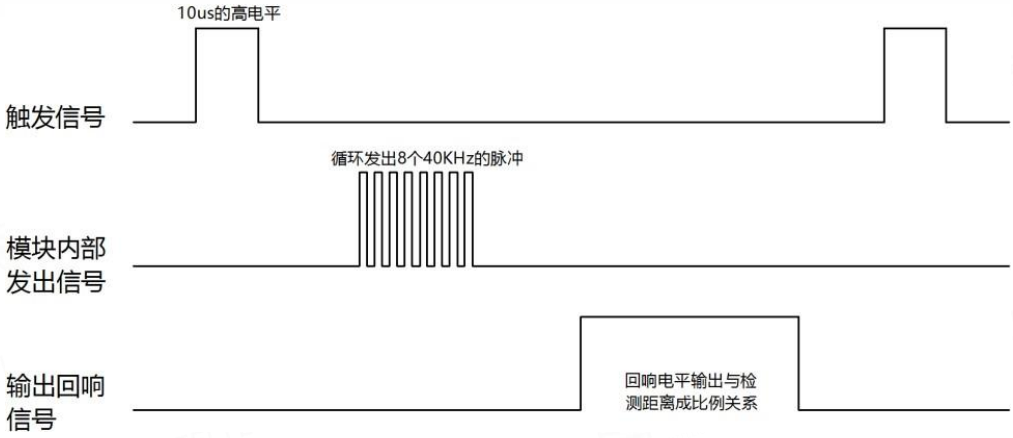


图 4-4 超声波模块时序图

计算公式为：测试距离=(高电平时间*声速 (340M/S))/2;

除去电源引脚后，模块与 32 芯片接线如表

| 超声波模块 | 触发引脚：Trig | 回波引脚：Echo |
|---|---|---|
| 32 底层主控板 | PB1 | PB0 |

表 4-4 接线表

程序使用到定时器 3 通道 3 的输入捕获功能。我们定义了一个 16 位的变量 TIM3CH3_CAPTURE_STA，当完整地捕获到一次高电平时，我们把第 7 位置 1，当我们捕获到上升沿时把第 6 位置 1，若定时器发生溢出，则变量自增。因为当定时器溢出一次，变量就自增一次，如果知道每次溢出的时间或每个计数的时间，就可以通过这个变量求得高电平持续时间。通过位操作来判断定时器是否成功获得高电平脉宽，并且计算高电平持续时间。初始化步骤如下：

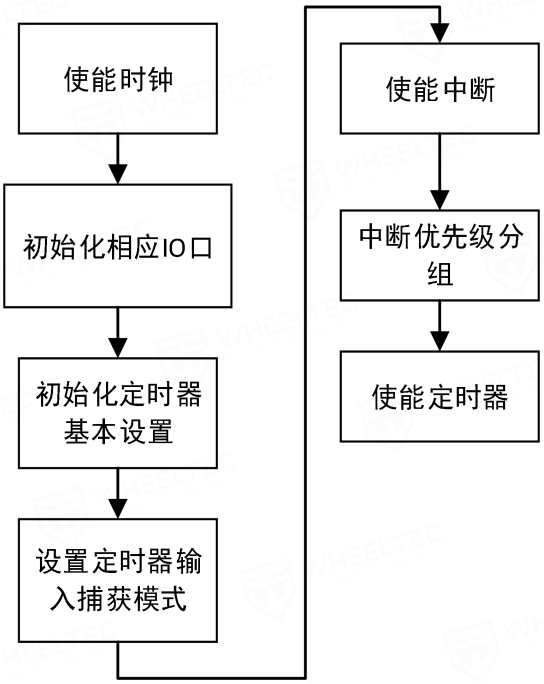


图 4-5 超声波模块初始化

读取距离的函数放在中断控制函数里面，每进入两次中断函数就执行一次，即 10ms 读取一次。读取的时候先发出触发信号，即发出 15us 高电平脉冲，进行下一次距离测量。代码如下：

```
void Read_Distane(void)
{
     PBout(1)=1;
     delay_us(15);
     PBout(1)=0;
     if(TIM3CH3_CAPTURE_STA&0X80)//成功捕获到了一次高电平
     {
         Distance=TIM3CH3_CAPTURE_STA&0X3F; //取出后 6 位
         Distance*=65536;                   //溢出时间总和
         Distance+=TIM3CH3_CAPTURE_VAL; //得到总的高电平时间
         Distance=Distance*170/1000;    //时间*声速/2(来回) 一个计数 0.001ms
         TIM3CH3_CAPTURE_STA=0;          //开启下一次捕获
     }
}
```

在初始化里，我们设置了定时器自动重装值为 0XFFFF，即一个周期计数为 65536 个，设置了预分频器为 71，根据公式：溢出时间=(重装值-1)*(预分频系数-1)/频率，我们的 CPU 频率为 72MHz，可以算出一个计数的时间为 0.001ms。把变量 TIM3CH3_CAPTURE_STA 的后 6 位取出来后可以得到溢出的次数，乘以

65536 并加上最后一次的计数即可以得到总的计数是多少，那么就可以得到高电平持续时间了，再根据公式就可以得出距离。

当检测到回波引脚的跳变时，会进入定时器 3 中断函数，定时器 3 中断函数代码如下：

```
    void TIM3_IRQHandler(void)
    {
        u16 tsr;
        tsr=TIM3->SR;
        if((TIM3CH3_CAPTURE_STA&0X80)==0)//判断第 7 位是否为 1，若不是则说明还未
成功捕获
        {
            if(tsr&0X01)//定时器溢出
            {
                 if(TIM3CH3_CAPTURE_STA&0X40)//已经捕获到高电平了
                 {
                     if((TIM3CH3_CAPTURE_STA&0X3F)==0X3F)//高电平太长了
                     {
                          TIM3CH3_CAPTURE_STA|=0X80;       //标记成功捕获了一次
                            TIM3CH3_CAPTURE_VAL=0XFFFF;
                     }else TIM3CH3_CAPTURE_STA++;
                 }
            }
            if(tsr&0x08)//捕获 3 发生捕获事件
            {
                if(TIM3CH3_CAPTURE_STA&0X40)      //捕获到一个下降沿
                {
                    TIM3CH3_CAPTURE_STA|=0X80;   //标记成功捕获到一次高电平脉宽
                    TIM3CH3_CAPTURE_VAL=TIM3->CCR3; //获取当前的捕获值.
                    TIM3->CCER&=~(1<<9);      //CC1P=0 设置为上升沿捕获
                }
                else
                {
                     TIM3CH3_CAPTURE_STA=0;  //清空
                     TIM3CH3_CAPTURE_VAL=0;
                     TIM3CH3_CAPTURE_STA|=0X40;      //标记捕获到了上升沿
                     TIM3->CNT=0;                    //计数器清空
                     TIM3->CCER|=1<<9;               //CC1P=1 设置为下降沿捕获
                }
            }
        }
        TIM3->SR=0;//清除中断标志位
    }
```

# 4.6蓝牙功能程序设计

蓝牙模块用于 APP 与小车之间的通信，能实现控制小车的运动转向与查看数据、PID 调参等功能。蓝牙模块与手机的通信通过无线蓝牙进行，与单片机的通信使用串口 3，相当于一个桥梁的作用。

串口 3 初始化步骤如下：

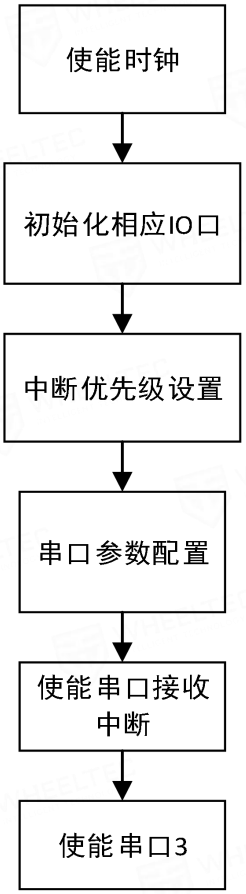


图 4-6 串口 3 初始化

APP 界面的每一个操作是向平衡小车发送不同的命令，小车收到命令后作出相应的处理。表 4-5 为详细的 APP 界面每个操作发送的信息。

| APP 摇杆 | ↑ | ↗ | → | ↘ | ↓ | ↙ | ← | ↖ |
|---|---|---|---|---|---|---|---|---|
| 小车接收到的数据 | 0x41 | 0x42 | 0x43 | 0x44 | 0x45 | 0x46 | 0x47 | 0x48 |
| 小车实现的效果 | 前进 | 右转 | 右转 | 右转 | 后退 | 左转 | 左转 | 左转 |
| | | | | | | | | |

| 按键 | 加速 | 减速 | 推摇杆后松手 | | | |
|---|---|---|---|---|---|---|
| 小车接收到的数据 | 0x58 | 0x59 | 0x5A | | | |
| 小车实现的效果 | 加速 | 减速 | 刹车 | | | |

表 4-5 APP 发送信息

程序如何实现方向控制呢？在串口 3 接受中断里面，当收到数据后就进行判断，操作相应的前进后退、左转右转标志位，然后在 5ms 的定时中断函数里判断这些标志位从而作出动作。

蓝牙还用于 32 芯片发送数据给手机 APP，void APP_Show(void)函数是主控板发送给手机 APP 的数据，用于显示编码器、电池电压、倾角和 PID 参数等信息。

## 4.7 电池电压检测

平衡车使用的是 12V 的锂电池供电，当电压低于一定程度时，需要自动关停小车防止电池过放（电池同时也自带过放保护，程序与硬件双层保护），所以电池电压是我们密切关注的变量之一。电池电压通过 ADC 进行测量，但是 32 芯片最大能转换 3.3V 的模拟电压，故将电池电压通过电阻分压，再送到 32 芯片进行测量。分压后测量的值为原电压的 1/11，最终测得的电压需乘以 11。

主控 32F103C8T6 芯片的 ADC 可以达到 12 位分辨率，可同时检测多个通道的信号。本程序采用单通道、单次转换软件触发模式。初始化步骤如下：

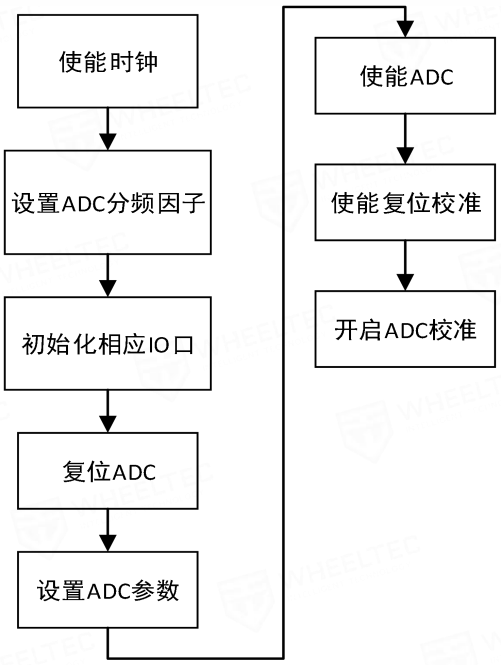


图 5-7 电压检测初始化

读取 ADC 转换结果代码如下

```
    u16 Get_Adc(u8 ch)
    {
        //设置指定 ADC 的规则组通道，一个序列，采样时间
        ADC_RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_239Cycles5 );
//ADC1,ADC 通道,采样时间为 239.5 周期
        ADC_SoftwareStartConvCmd(ADC1, ENABLE); //使能指定的 ADC1 的软件转换启动
功能
        while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));//等待转换结束
        return ADC_GetConversionValue(ADC1);//返回最近一次 ADC1 规则组的转换结果
    }
```

读取转换结果前要设定采样时间，采样时间越长，准确度就越高。缺点是会降低 ADC 转换速率。ADC 转换时间公式为：

总转换时间=采样时间+12.5 个周期

代码运行结束后会返回一个 12 位的数值，得到这个数值，我们就可以转换成电压了。在 int Get_battery_volt(void)函数里面转换成电压代码如下

```
    Volt=Get_Adc(Battery_Ch)*3.3*11*100/4096;    //电阻分压,具体根据原理图简单分
析可以得到
```

12 位数值最大可表示为十进制数 4096，32 芯片的 ADC 最大可测量电压为 3.3V，通过简单的比例就可以转换成电池电压。为了处理方便，把电压数据放大 100 倍变为 int 整形。

# 4.8角度滤波算法

① DMP 算法

我们想要得到姿态角数据，即航向角、横滚角进而俯仰角数据，需要利用原始的数据进行姿态解算，而姿态解算比较复杂，难度较大，此时可以利用 MPU6050 官方自带的数字运动处理器，即 DMP。

使用官方提供的函数 int dmp_read_fifo(short *gyro, short *accel, long *quat, unsigned long *timestamp, short *sensors, unsigned char *more)就可以读取出四元数，注意次四元数是放大了 2^30 倍的，需要变换为原来的数据。读出四元数后根据公式就可以计算出姿态角。代码如下

```
    void Read_DMP(void)
    {
          unsigned long sensor_timestamp;
            unsigned char more;
            long quat[4];
            dmp_read_fifo(gyro, accel, quat, &sensor_timestamp, &sensors, &more);
        //读取 DMP 数据
                if (sensors & INV_WXYZ_QUAT )
                {
                  q0=quat[0] / q30;
                  q1=quat[1] / q30;
                  q2=quat[2] / q30;
                  q3=quat[3] / q30;          //四元数
                  Roll = asin(-2 * q1 * q3 + 2 * q0* q2)* 57.3; //计算出横滚角
                  Pitch = atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1 - 2 * q2*
q2 + 1)* 57.3; // 计算出俯仰角
                  Yaw = atan2(2*(q1*q2 + q0*q3),q0*q0+q1*q1-q2*q2-q3*q3) * 57.3;
     //计算出偏航角
                }
    }
```

② 互补滤波算法

进行滤波前需要先读取原始的角速度，并且读出加速度计数据用于计算角度。MPU6050 的数据读取和写入均通过其内部的寄存器来实现，这些寄存器都是 1 个字节，即 8 位。角速度和加速度的数据分为高 8 位与低 8 位，分别用 2 个寄存器储存，读取数据后应把他们整合在一起，如读取 X 轴角速度数据，代码如下：

```
    Gyro_X=(I2C_ReadOneByte(devAddr,MPU6050_RA_GYRO_XOUT_H)<<8)+I2C_ReadOneByte
```

```
(devAddr,MPU6050_RA_GYRO_XOUT_L);     //读取 X 轴陀螺仪，高 8 位与低 8 位整合
```

MPU6050 采用 16 位有符号数作为陀螺仪测量数据输出，所以最小的数为-32767，最大的数为 32767。我们初始化了陀螺仪的最大量程为±2000°/s，所以数字 32767 对应 2000°/s，同理负数也是如此。用 32767 除以 2000 即可以得到灵敏度为 16.4，即每度多少个读数，官方手册也给出了各个量程的灵敏度。把陀螺仪读出的数字除以灵敏度就可以转换为角速度(°/s)。在此之前，我们需要作一些处理，把读数转换成有符号类型的数，就像读取编码器一样，代码如下：

```
if(Gyro_X>32768)  Gyro_X-=65536; //数据类型转换  也可通过 short 强制类型转换
```

读数转换为角速度(°/s)，以 X 轴为例，代码如下

```
Gyro_X=Gyro_X/16.4;                    //陀螺仪量程转换，灵敏度 16.4
```

已经得到了角速度，那么使用加速度计的数据计算出角度就可以使用滤波算法了。在前面已经说明了角度计算公式。X 轴与 Y 轴角度计算代码如下：

```
Accel_Angle_x=atan2(Accel_Y,Accel_Z)*180/PI;   //计算倾角并转换单位
Accel_Angle_y=atan2(Accel_X,Accel_Z)*180/PI;   //计算倾角并转换单位
```

用公式计算出来的角度为弧度制，需要把角度与角速度的单位统一起来，一律使用度作单位，所以最后作了单位转换。

互补滤波程序如下，入口参数为角度与角速度。

```
float First_order_filter_x(float angle_m, float gyro_m)
{
    static float angle;
    float K1 =0.02;
    angle = K1 * angle_m+ (1-K1) * (angle + gyro_m * dt);
    return angle;
}
```

**③ 卡尔曼滤波算法**

卡尔曼滤波原理上面已经说明，主要是预测与更新两个步骤。程序主要是一些计算工作，用于实现所说的两个步骤。程序代码如下：

```
float Kalman_Filter_x(float Accel,float Gyro)
{
    static float angle_dot;
    static float angle;
    float Q_angle=0.001;    // 过程噪声的协方差
    float Q_gyro=0.003;     //0.003 过程噪声的协方差 过程噪声的协方差为一个一行两列矩阵
    float R_angle=0.5;      // 测量噪声的协方差 既测量偏差
    char  C_0 = 1;
```

```
static float Q_bias, Angle_err;
static float PCt_0, PCt_1, E;
static float K_0, K_1, t_0, t_1;
static float Pdot[4] ={0,0,0,0};
static float PP[2][2] = { { 1, 0 },{ 0, 1 } };
```

我们定义的过程噪声协方差和测量噪声协方差都是常数，所以按照公式步骤一步步算出来即可。

```
angle+=(Gyro - Q_bias) * dt; //角度先验估计
```

以下部分代码对应于上面推导的协方差矩阵先验估计公式：

$$P_k^- = AP_{k-1}A^T + Q_{k-1} = A\begin{pmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{pmatrix}_{k-1}A^T + \begin{pmatrix} Q\_angle & 0 \\ 0 & Q\_gyro \end{pmatrix}_{k-1} * dt$$

$$= \begin{pmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{pmatrix}_{k-1}$$

$$+ \begin{pmatrix} -dtP_{10} - dtP_{01} + dt^2P_{11} + dt * Q\_angle & -dtP_{11} \\ -dtP_{11} & dt * Q\_gyro \end{pmatrix}_{k-1}$$

这里数组 PP 表示为协方差矩阵$P_k^-$的先验估计值，注意，因为$dt^2P_{11}$这部分很小，可以忽略不计，所以在程序中把它去掉。

```
Pdot[0]=Q_angle - PP[0][1] - PP[1][0]; // Pk-先验估计误差协方差的微分
Pdot[1]=-PP[1][1];
Pdot[2]=-PP[1][1];
Pdot[3]=Q_gyro;
PP[0][0] += Pdot[0] * dt;   // Pk-先验估计误差协方差微分的积分
PP[0][1] += Pdot[1] * dt;   // 先验估计误差协方差
PP[1][0] += Pdot[2] * dt;
PP[1][1] += Pdot[3] * dt;
```

以下代码求卡尔曼增益，卡尔曼增益为一个两行一列的矩阵，对应我们推导的公式为：$K_k = \begin{pmatrix} K_0 \\ K_1 \end{pmatrix} = \begin{pmatrix} (P_{00})_k^- / [(P_{00})_k^- + R_k] \\ (P_{10})_k^- / [(P_{00})_k^- + R_k] \end{pmatrix}$，E 是分母部分。

```
PCt_0 = C_0 * PP[0][0];
PCt_1 = C_0 * PP[1][0];
E = R_angle + C_0 * PCt_0;  //求卡尔曼增益分母
K_0 = PCt_0 / E;
K_1 = PCt_1 / E;            //计算卡尔曼增益
```

这里更新协方差矩阵，即协方差矩阵后验估计，对应推导的公式：

$$P_k = (I - K_kH)P_k^- = \{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} K_0 \\ K_1 \end{pmatrix}(1 \quad 0)\}\begin{pmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{pmatrix}_k^-$$

$$= \begin{pmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{pmatrix}_k^- + \begin{pmatrix} -K_0P_{00} & -K_0P_{01} \\ -K_1P_{00} & -K_1P_{01} \end{pmatrix}_k^-$$

```
t_0 = PCt_0;
t_1 = C_0 * PP[0][1];
PP[0][0] -= K_0 * t_0;       //后验估计误差协方差
```

```
        PP[0][1] -= K_0 * t_1;
        PP[1][0] -= K_1 * t_0;
        PP[1][1] -= K_1 * t_1;
```

这里根据卡尔曼增益进行最优估计，即后验估计：

```
        angle   += K_0 * Angle_err;    //后验估计
        Q_bias  += K_1 * Angle_err;   //后验估计
        angle_dot   = Gyro - Q_bias;  //输出值(后验估计)的微分=角速度
        return angle;
    }
```

# 5. 控制流程图

## 5.1 小车外设关系图

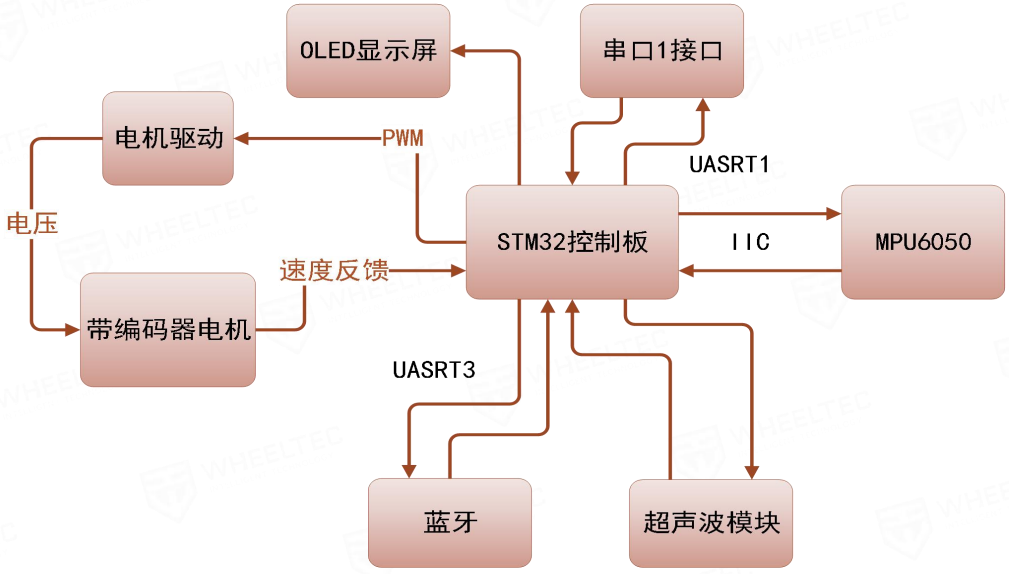


图 5-1 小车外设关系图

在平衡小车中，核心控制器是 32 开发板，用到了很多外设，包括：MPU6050 六轴传感器、蓝牙模块。超声波模块、OLED 显示屏和电机等。外设与控制器之间的连接如图 5-1 所示。

## 5.2 电机控制流程图

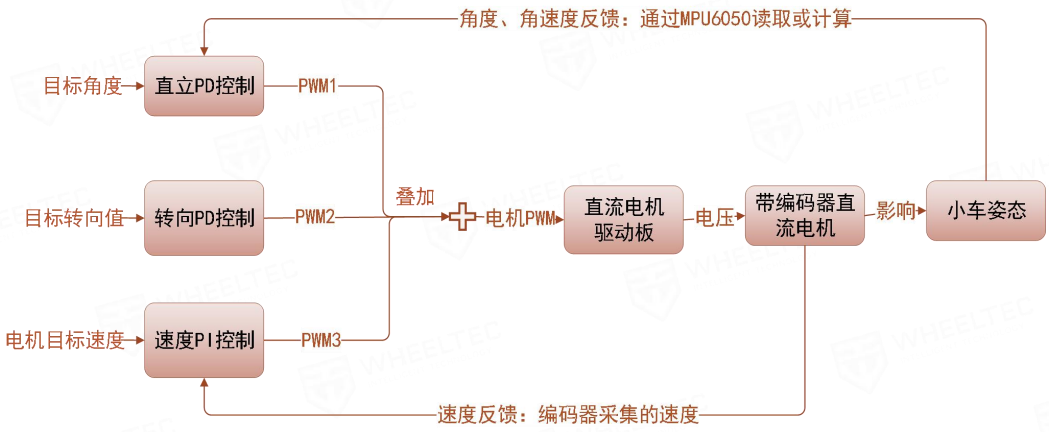


图 5-2 电机控制流程图

小车的直立与运动通过控制电机来实现，电机通过给定小车当前的角度与速度来进行反馈闭环控制。需要注意一下转向环，因为转向的实现是通过两个电机

的差速来实现的，故左右轮转向部分叠加到电机的 PWM 的极性应该相反。

## 5.3程序结构框图

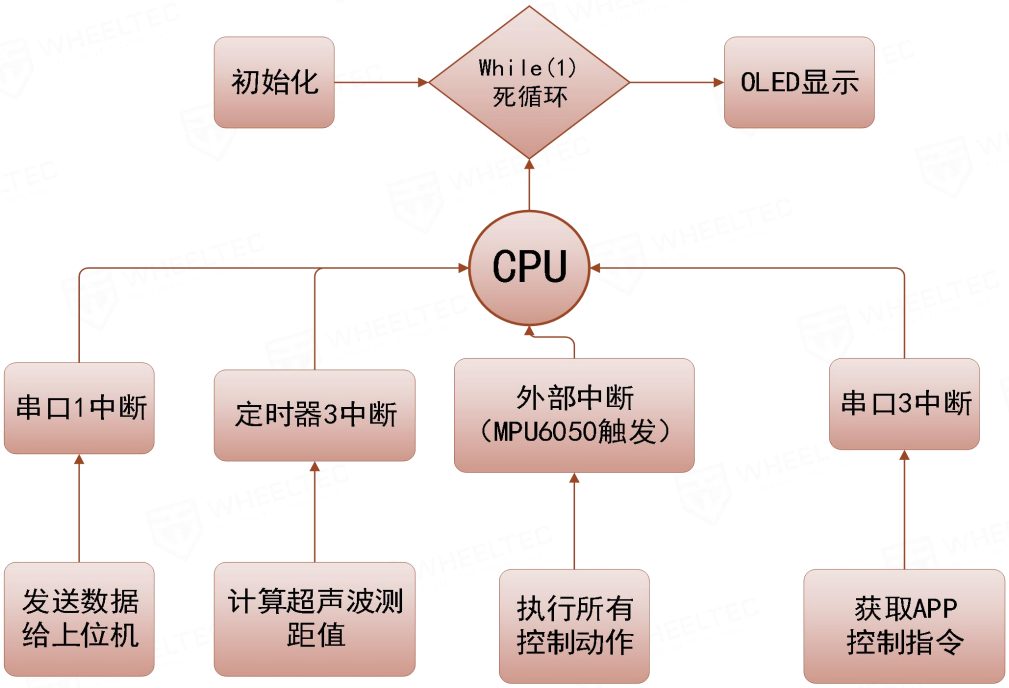


图 5-3 程序结构框图

程序结构如图 5-3 所示。程序经过一系列的初始化，进入到 while 死循环里，其中 OLED 显示函数放在里面。我们小车的全部控制动作都在由 MPU6050 触发的中断里面。串口 3 中断用于接收 APP 的指令，串口 1 中断为发送数据给上位机，定时器 3 中断为捕获中断，可用于计算超声波测距值。

# 6. PID 参数调节

小车能否稳定地运行或者快速地调整需要精心地调试 PID 参数，我们需要调试平衡小车的直立环、速度环和转向环。我们的手机 APP 能够进行 PID 参数调节，减少了多次烧录程序的步骤，十分方便。

## 6.1 确定小车的机械中值

把平衡小车放在地面上，绕电机轴旋转平衡小车，记录能让小车接近平衡的角度，一般都是 0 度附近。

## 6.2 PID 参数极性调节

### ① 直立环极性调节

直立环使用 PD 控制器，需要调节 Balance_Kp 和 Balance_Kd 这两个参数。

我们先估算一下 Balance_Kp 和 Balance_Kd 的大小。我们的 PWM 设定 7200 代表占空比为 100%，假如我们设定 Balance_Kp 值为 720，那么平衡小车在±10°的时候就会满转，这显然太大了，所以我们可以知道 Balance_Kp 在 0-720 之间。我们得到的 MPU6050 输出陀螺仪的原始数据，通过观察发现最大值不会超过 4 位数，而且根据 7200 代表占空比 100%，那么 Balance_Kd 的值大概在 0-2 之间。

先确定 Balance_Kp 的极性。若小车倾斜时也向倾斜的方向加速，那么极性就是对的，反之，即为错的。尝试调节 Balance_Kp 大小为 20000(放大了 100 倍，实际为 200，这个数值为范围内的任意数值)，屏蔽转向环和速度环，并且设置 Balance_Kd 值为 0，可以看到小车倾斜时加速倒下，并不能朝倾斜方向加速，故 Balance_Kp 的极性应为负，我们在 Balance_Kp 前面加上负号，方便后续调试。如下所示：

```
    balance=-Balance_Kp/100*Angle_bias-Gyro_bias*Balance_Kd/100;
//计算平衡控制的电机 PWM  PD 控制   kp 是 P 系数 kd 是 D 系数
```

接下来确定确定 Balance_Kd 的极性。若向前倾斜小车，小车也向前前进，那么极性就是对的，反之为错误的。把 Balance_Kp 值设置为 0，设置 Balance_Kd 为-100(实际为-1)，然后把小车放在角度中值，向前倾斜小车，可以发现小车也向前行进，说明此时极性是对的。

② 速度环极性调节

速度环使用 PI 控制，需要调节 Velocity_Kp 和 Velocity_Ki 这两个参数。积分项由偏差的积分得到，所以积分控制与比例控制的极性相同。根据工程经验，在不同的系统中，PID 参数相互直接有一定的比例关系，在本系统中，我们可以把 Velocity_Ki 设置为 Velocity_Kp/200，这样只要确定比例控制的大小与极性，就可以知道积分控制项的大小与极性了。

先估算 Velocity_Kp 的大小。我们每 5m 读取一次编码器，左右编码器相加最大值大概在 84 左右。假设速度偏差达到最大速度的 50%的时候，系统输出电机最快速度即 PWM 输出 7200。我们可以大概估算：最大值=7200/(84*50%)=170。另外，速度环是正反馈过程，当倾斜加速时，想要速度慢下来，速度环 PWM 就必须输出更大，让小车快速摆正。如果使用负反馈，当小车以一定的速度运行时，我们通过减速让小车慢下来，小车会因为惯性向前倒下。

先把直立环和转向环屏蔽，单独调试速度环。当转动轮子时，轮子速度越来越快，那么速度环极性就是对的。设定 Velocity_Kp 为 5000(实际为 50)，Velocity_Ki 为 25(实际为 0.25)，极性取负时，转动轮子，可以发现轮子一直加速，这是我们想要的正反馈效果。所以可以确定 Velocity_Kp 和 Velocity_Ki 的极性为负。在代码中加上负号如下

```
    velocity=-Encoder_bias*Velocity_Kp/100-Encoder_Integral*Velocity_Ki/100;
//速度控制
```

③ 转向环极性调节

我们要求小车前进的时候能尽量的走直线，所以通过 Z 轴角速度反馈来纠正小车的偏航。转向环为 PD 控制，我们主要调试 Kd 参数。Kp 参数为遥控控制小车的部分，此参数的极性关系到遥控转向时的方向，若遥控时方向相反，可把 Kp 极性取反，或者把叠加在电机 PWM 部分的 Turn_Pwm 极性取反，如下代码

```
    Motor_Left=Balance_Pwm+Velocity_Pwm+Turn_Pwm;    //计算左轮电机最终 PWM
    Motor_Right=Balance_Pwm+Velocity_Pwm-Turn_Pwm;   //计算右轮电机最终 PWM
```

Kp 值的大小不太重要，可固定为一个数值，若修改转向速度，修改代码里的 Turn_Amplitude 参数即可。

首先我们估计 Kd 参数的大小，观察 Z 轴角速度原始数据可以知道最大不超过 4 位数，而且占空比 100%时 PWM 为 7200，故 Kd 大小参数应该在 0-2 之间。我们屏蔽直立环和转向环，单独调试转向环。当我们用手摁着小车在地上旋转时，

若出现电机对抗我们的旋转的现象，那就说明极性是对的，若很容易旋转，说明转向环帮助了我们旋转，这个时候极性是错的。取 Kd 参数为 0.6，用手摁着小车在地上旋转，可以发现，此时电机出现了对抗我们的旋转的现象，说明此时极性是对的，Kd 参数取正。

## 6.3PID 参数大小调节

参数的大小调节使用我们的手机 APP 进行，十分方便。把程序里屏蔽掉的直立环与速度环打开，烧录到我们的小车上，然后用 APP 连接小车的蓝牙。打开手机的调试界面，点击获取设备参数，就可以把小车的 PID 参数发送到手机 APP 了，如图 7-1 所示。滑动参数的滑块就可以把参数发送给小车。


图 6-1 APP [调试] 界面

① 直立环参数大小调节

先把所有参数调节为 0，然后就可以开始调节直立环参数大小了。点击对应的参数，可以修改最大最小值，如图 7-2 所示。为了防止手滑，调节的时候应该关闭电机。首先调节 Balance_Kp，每次增大 Balance_Kp 直至小车出现低频抖动，保留此时的值。增大 Balance_Kd 直至出现高频抖动，此时记录直立环的参数值，乘以 0.6 作为直立环最终参数。


图 6-2 参数最值设置

② 速度环参数大小调节

把上述所得的直立环最终参数保留，即乘以 0.6 的参数。然后开始调节速度环。每次增大速度环的 Velocity_Kp 值，其中 Velocity_Ki 值也要设定为 Velocity_Kp 的 1/200，观察小车是否能立在原地。若增大到小车出现抖动，而且用手推动小车会发生大幅回摆，说明此参数不可取，应减少到适合的数值。

③ 转向环参数大小调节

保留调试好的直立环和速度环，开始调节转向环 Kd 参数的大小。逐渐加大转向环 Kd 参数，令小车前进，观察小车走直线的效果。若小车出现了高频抖动的现象，说明此参数不可取，取前面的相对理想的结果。我们调节 PID 参数并不一定要十分完美，只需要在此过程中学习 PID 的思想即可。