平衡小车调试指南

前面已经学习了一些平衡小车的基础知识，接下来将和大家一起以工程的思想去完成一个平衡小车的调试，包括平衡小车的直立环、速度环、转向环，一般我们是先调试直立环，再调试速度环，最后调试转向环。另外需要说明的是，因为我们使用的电机性能非常好，对 PID 参数不敏感，也就是说每个参数的取值范围都很广，这将对我们接下来的调试有很大的帮助。

目录

1.1 平衡小车直立控制调试

平衡小车直立环使用 PD（比例微分）控制器，其实一般的控制系统单纯的 P 控制或者 PI 控制就可以了，但是那些对干扰要做出迅速响应的控制过程需要 D（微分）控制。

下面是直立 PD 控制的代码：

```
int balance(float Angle,float Gyro)
{
    float Bias,kp=300,kd=1;
    int balance;
    Bias=Angle-0;                  //计算直立偏差
    balance=kp*Bias+Gyro*kd;   //计算直立 PWM
    return balance;                //返回直立 PWM
}
```

入口参数是平衡小车倾角和 Y 轴陀螺仪（这个取决于 MPU6050 的安装），我们的小车前进方向是 MPU6050 的 X 轴的正方向，电机轴与 Y 轴平行。前面两行是相关变量的定义，第三行是计算角度偏差，第四行通过 PD 控制器计算直立 PWM，最后一行是返回。

调试过程包括确定平衡小车的机械中值、确定 kp 值的极性（也就是正负号）和大小、kd 值的极性和大小等步骤。

在调试直立环的时候，我们要在定时中断服务函数里面屏蔽速度环和转向环，如下图所示：

```
   Balance_Pwm =balance(Angle_Balance,Gyro_Balance);
 //Velocity_Pwm=velocity(Encoder_Left,Encoder_Right);
 //Turn_Pwm    =turn(Encoder_Left,Encoder_Right,Gyro_Turn);
```

1.1.1 确定平衡小车的机械中值

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，一般都在 0° 附近的。我们调试的小车正好是 0 度，所以就是 Bias=Angle-0;

1.1.2 确定 kp 值的极性（令 kd=0）

首先我们估计 kp 的取值范围。我们的 PWM 设置的是 7200 代表占空比 100%，

假如我们设定 kp 值为 720，那么平衡小车在±10°的时候就会满转。根据我们的感性认识，这显然太大了，那我们就可以估计 kp 值在 0~720 之间，首先大概我们给一个值 kp=-200,我们可以观察到，小车往哪边倒，电机会往那边加速让小车到下，就是一个我们不愿看到的正反馈的效果。说明 kp 值的极性反了，接下来我们设定 kp=200,这个时候可以看到平衡小车有直立的趋势，虽然响应太慢，但是，我们可以确定 kp 值极性是正的。具体的数据接下来再仔细调试。

1.1.3 确定 kp 值的大小（令 kd=0，请结合本小节开头的直立控制函数理解）

确定参数的原则是：kp 一直增加，直到出现大幅度的低频抖动。

设定 kp=200,这个时候我们可以看到，小车虽然有平衡的趋势，但是显然响应太慢了。

设定 kp=350,这个时候我们可以看到，小车虽然有平衡的趋势，而且响应有所加快，但是响应还是不足以让小车保持平衡。

设定 kp=500,这个时候我们可以看到，小车的响应明显加快，而且来回推动小车的时候，会有大幅度的低频抖动。说明这个时候 kp 值已经足够大了，需要增加微分控制削弱 p 控制，抑制低频抖动。

1.1.4 确定 kd 值的极性（令 kp=0）

我们得到的 MPU6050 输出的陀螺仪的原始数据，通过观察数据，我们发现最大值不会超过 4 位数（正常应用在平衡小车上的时候），再根据 7200 代表占空比 100%，所以我们估算 kd 值应该在 0~2 之间，我们先设定 kd=-0.5，当我们拿起小车旋转的时候，车轮会反向转动，并没有能够实现跟随效果。这说明了 kd 的极性反了。接下来，我们设定 kd=0.5,这个时候我们可以看到，当我们旋转小车的时候，车轮会同向以相同的速度跟随转动，这说明我们实现了角速度闭环，至此，我们可以确定 kd 的极性是正的。具体的数据接下来再仔细调试。

1.1.5 确定 kd 值的大小（令 kp=500，请结合本小节开头的直立控制函数理解）

确定参数的原则是：kd 一直增加，直到出现高频抖动。

设定 kd=0.5,这个时候我们可以看到，低频大幅度频抖动已经基本消除。

设定 kd=1,这个时候我们可以看到，整体性能已经非常棒。

设定 kd=1.7,这个时候我们可以看到，小车开始出现高频剧烈抖动(调试过程遇到这种情况请马上关闭小车，长时间高频抖动会导致驱动被烧坏的)

至此，我们可以确定得到 kp=500,kd=1.7 是 P、D 参数的最大值。然后我们进行最关键的一步，对每个系数乘以 0.6,取整得到 kp=300,kd=1，这就是最终我们需要的参数，这样做的原因是，我们之前得到的参数是 kp、kd 最大值，理想值是根据我们的工程经验，对每个数据乘以 0.6 得到。这个时候我们可以看到，小车没有任何的抖动，非常平稳，但是依然无法保持长时间的直立，直立很短一段时间后会往一个方向加速倒下。这个等我们下面加上速度环才能得到更好的性能。只有直立环是很难让小车达到很好的直立效果的。至此，直立调试部分就告一段落了。

1.2 平衡小车速度控制调试

平衡小车速度环使用 PI（比例积分）控制器，这也是速度控制最常使用的控制器。PI 控制器是一种线性控制器，它根据给定值与实际输出值构成控制偏差，将偏差的比例（P）和积分（I）通过线性组合构成控制量，对被控对象进行控制。

下面是速度 PI 控制的代码（不包括遥控部分，遥控部分后面再单独讲解）：

```
int velocity(int encoder_left,int encoder_right)
{
    static float Velocity,Encoder_Least,Encoder;
    static float Encoder_Integral;
    float kp=80,ki=0.4;
    Encoder_Least =(Encoder_Left+Encoder_Right)-0;
    Encoder *= 0.7;
    Encoder += Encoder_Least*0.3;
    Encoder_Integral +=Encoder;
    Velocity=Encoder*kp+Encoder_Integral*ki;
    return Velocity;
```

}

入口参数为左右轮编码器。前面 3 行是相关变量的定义，第四行是获取最新的速度偏差，第 5 和第 6 行是对速度偏差进行低通滤波，第 7 行是对偏差积分得到位移。第 8 行是使用速度 PI 控制器计算速度控制 PWM，第 9 行是返回。

以上代码实现的效果是：使小车在保持平衡的同时速度为零。

调试过程包括计算速度偏差、确定 kp 和 ki 值的极性（也就是正负号）与大小。

1.2.1 计算速度偏差

根据公式

偏差=测量值-目标值

测量值我们使用左右编码器之和表示，我们没有必要纠结于是否要除以 2，因为这样就引入舍去误差，我们需要的其实是一个可以表示速度变化的变量。另外，我们的目标速度设置为零。所以，可以得到

Encoder_Least =(Encoder_Left+Encoder_Right)-0;

然后，我们对速度值进行低通滤波，具体的系数由工程经验得到。这样做的目的是为了减缓速度值的变化，防止速度控制对直立造成干扰，因为平衡小车系统里面，直立控制是主要的，其他控制对于直立来说都是一种干扰。具体实现代码如下：

```
Encoder *= 0.7;
Encoder += Encoder_Least*0.3;
```

1.2.2 确定 kp 与 ki 值的极性

为了调试方便，接下来我们先关闭之前已经调试好的直立控制部分，如下图所示：

```
//Balance_Pwm =balance(Angle_Balance,Gyro_Balance);
  Velocity_Pwm=velocity(Encoder_Left,Encoder_Right);
//Turn_Pwm    =turn(Encoder_Left,Encoder_Right,Gyro_Turn);
```

积分项由偏差的积分得到，所以积分控制和比例控制的极性是相同的，而根据工程经验，在不同的系统中，PID 参数相互之间会有一定的比例关系。在我们

的平衡小车速度控制系统里面，一般我们可以把 ki 值设置为 ki=kp/200;这样，只要我们可以得到 kp 值的大小和极性，就可以完成速度控制部分的参数整定了。显然，这样大大缩短了 PID 参数整定的时间。

我们通过 STM32 定时器的编码器接口模式对编码器进行四倍频，并使用 M 法测速（每 10ms 的脉冲数）得到小车的速度信息，通过观察数据，我们发现两路编码器相加最大值在 160 左右，而考虑到系统的反应时间，我们假定当速度偏差达到最大速度 50%的时候，系统输出电机的最快速度，再根据 7200 代表占空比 100%，我们可以大概估算

kp 最大值=7200/(160*50%)=90

另外要说明的是，虽然这里的 PI 控制器是速度控制常用的一种控制器，但是和普通的调速系统不一样，这里的速度控制是正反馈的，当小车以一定的速度运行的时候，我们要让小车停下来，小车需要行驶更快的速度去“追”，小车运行的速度越快，去“追”的速度也就越快，所以这是一个正反馈的效果。如果使用常规的速度负反馈，当小车以一定的速度运行的时候，我们通过减速让小车慢下来，小车会因为惯性向前倒下。

下面介绍一种确定速度控制是正反馈还是负反馈的方法。根据之前的估计，先设定 kp=-50，ki=kp/200,当我们拿起小车，旋转其中一个小车轮胎的时候，根据我们设定的速度偏差

Encoder_Least =(Encoder_Left+Encoder_Right)-0;

另外一个车轮会反向转动，让偏差趋向于零。这就是常规的速度控制里面的负反馈，不是我们需要的效果。接下来设定 kp=50，ki=kp/200,此时，当我们旋转其中一个小车轮胎的时候，两个轮胎会往相同的方向加速，直至电机的最大速度，这是典型的正反馈效果，也是我们期望看到的。至此，我们可以确定 kp,ki 的符号应该是正的。

1.2.3 确定 kp 与 ki 的大小（开启直立控制）

下面我们进行平衡小车速度控制 kp 与 ki 值的整定，此时需要打开直立环，因为我们需要结合直立环观察速度环对直立环的影响，如下图所示：

```
Balance_Pwm =balance(Angle_Balance,Gyro_Balance);
Velocity_Pwm=velocity(Encoder_Left,Encoder_Right);
//Turn_Pwm    =turn(Encoder_Left,Encoder_Right,Gyro_Turn);
```

在平衡小车速度控制系统里面，一般我们可以把 ki 值设置为 ki=kp/200,所以我们只需要对 kp 值进行整定即可。在调试的过程中设定速度控制的目标为零，所以，调试的理想结果应该是：小车保持平衡的同时，速度接近于零。实际上，因为小车存在比较大的转动惯量和惯性，并且齿轮减速器存在死区，很难调试到让小车完全保持静止的，我们调试平衡小车只是为了学习 PID 控制算法，所以，没有必要花太多的时间去调参数，让小车完全静止，只要能够大概实现我们需要的功能，并在这个过程对 PID 有进一步的了解即可。

首先，设定 kp=40,ki=kp/200 这个时候我们可以看到，小车的速度控制比较弱，很难让速度恒定。

设定 kp=60,ki=kp/200 这个时候我们可以看到，小车的速度控制的响应有所加快，但是来回摆动还是有点大，还是不足以让小车保持接近于静止的状态。

设定 kp=80,ki=kp/200 这个时候我们可以看到，小车已经性能很完美了，我们接下来尝试加大 kp 值看一下效果。

设定 kp=100,ki=kp/200 这个时候我们可以看到，小车虽然回正力度增大了，而且响应更加快了，但是稍微加入一点的干扰都会让小车大幅度摆动，抗干扰能力明显不足，所以这组参数不可取。

至此，我们可以确定得到 kp=80,kd=0.4 是速度控制 P、I 参数的理想值。

我们再来体检一下速度控制负反馈在平衡小车里面的效果，设定 kp=-80,ki=kp/200 这个时候我们可以看到，小车会迅速往一个方向倒下。也就是说常规的速度负反馈在我们这边是“帮倒忙”了！

至此，速度控制调试部分就告一段落了，如果要加入遥控前进后退功能的话，速度 PI 控制函数应该改成如下所示(其中加粗部分为是实现遥控功能的代码)：

```
int velocity(int encoder_left,int encoder_right)
{
    static float Velocity,Encoder_Least,Encoder,Movement;
    static float Encoder_Integral;
    float kp=80,ki=0.4;
    if(1==Flag_Qian)      Movement=-90;    //===如果前进标志位置 1  位移为负
    else if(1==Flag_Hou)  Movement=90;     //===如果后退标志位置 1  位移为正
```

```
    else    Movement=0;
    Encoder_Least =(Encoder_Left+Encoder_Right)-0;   //获取最新速度偏差==测量速度（左右编码器之和）-目标速度（此处为零）
    Encoder *= 0.7;                                  //===一阶低通滤波器
    Encoder += Encoder_Least*0.3;                    //===一阶低通滤波器
    Encoder_Integral +=Encoder;                      //===积分出位移 积分时间：10ms
    Encoder_Integral=Encoder_Integral-Movement;   //接收遥控器数据，控制前进后退
    if(Encoder_Integral>10000)      Encoder_Integral=10000;     //积分限幅
    if(Encoder_Integral<-10000)     Encoder_Integral=-10000;   //限制遥控最大速度
    Velocity=Encoder*kp+Encoder_Integral*ki; //===速度控制
    if(Turn_Off(Angle_Balance,Voltage)==1)     Encoder_Integral=0;   //电机关闭后清除积分
    return Velocity;
}
```

关于这个包括了遥控前进后退的速度控制函数，做如下解析：

1.在 usart3.c 中的串口 3 接收中断函数，改变 Flag_Qian 和 Flag_Hou，进而遥控小车。

2.Encoder_Integral=Encoder_Integral-Movement; 遥控的速度通过积分融入速度控制器，减缓了速度突变对直立控制的影响。

3.积分限幅是增加了遥控之后必不可少的，如果没有积分限幅，就无法限制小车的最大前进速度。这样在遥控的过程中，小车很容易倒下。换句话说积分的最大赋值决定了小车的最大前进速度，而 Movement 值决定了小车的给定速度。

1.3 平衡小车转向控制调试

平衡小车转向环使用 P（比例）控制器或者 P（比例）D（微分）控制器，我们前面说过，一般的控制系统单纯的 P 控制或者 PI 控制就可以了，转向环就是这种“一般的控制系统”，对响应要求不高，所以我们只使用 P 控制即可。

其实转向信息可以通过编码器和陀螺仪获取，所以转向环有多种控制方式，总结如下：

1.使用左右轮编码器数据之差的积分值作为偏差，以 Z 轴陀螺仪作为微分控制的输入进行 PD 控制，目标是保持转向角为设定值。优点是算法比较科学，引入微分控制可以增大比例控制系数以提高系统的响应。缺点是较复杂，积分项影响用户体验、编码器对车轮滑动无法检测、陀螺仪存在漂移。

2.使用 Z 轴陀螺仪的数据积分得到转向角作为偏差，以 Z 轴陀螺仪作为微分

控制的输入进行 PD 控制，目标是保持转向角为设定值。优点是避免了编码器对车轮滑动无法检测现象，引入微分控制可以增大比例控制系数以提高系统的响应。缺点是陀螺仪的漂移长时间积分导致系统误差增大。

3.使用左右轮编码器数据之差作为转向速度偏差进行 P 控制，目标是保持转向速度为设定值。优点是简单，缺点是编码器对车轮滑动无法检测，对编码器精度有较高要求。

4.使用 Z 轴陀螺仪的数据作为转向速度偏差进行 P 控制，目标是保持转向速度为设定值。优点是算法简单、避免了编码器对车轮滑动无法检测现象、陀螺仪漂移等问题，缺点是陀螺仪对高频信号采样失真。

我们本次调试使用了简单可靠的第 4 套方案。在平衡小车里面，相比于直立环和速度环，转向环是最不重要的，如果缺少了直立环和速度环，小车无法长时间保持直立。转向环的作用是使小车行驶的过程中，跟随我们给定的 Z 轴角速度，具体来说，如果我们设定的 Z 轴目标角速度为零，那么小车应该走一个直线，这也是我们本次实验需要完成的目标。

下面是转向 P 控制的代码（不包括遥控部分）：

```
int turn(int encoder_left,int encoder_right,float gyro)
{
    float Turn,Kp=1,Bias;
    Bias=gyro-0；
    Turn=-Bias*Kp;
    return Turn;
}
```

入口参数为左右轮编码器和 Z 轴陀螺仪，其中左右轮编码器在遥控部分才使用到。第 1 行是相关变量的定义，第 2 行是获取偏差值，在这里我们的目标角速度是 0，所以 Bias=gyro-0。第 3 行是使用 P 控制器计算转向控制 PWM，最后一行是返回。

以上代码实现的效果是：配合直立环和速度环，使小车保持直线行驶。其实在没有外部传感器的情况下，让小车走直线是一个世界级的难题，所以，这里我们只能让小车短距离行驶的轨迹接近于直线。

调试过程包括确定 kp 极性和大小。

1.3.1. 确定 kp 的极性

为了方便本小节的实验，我们先关闭之前调试好的直立环和速度环。如下图所示：

```
//Balance_Pwm =balance(Angle_Balance,Gyro_Balance);
//Velocity_Pwm=velocity(Encoder_Left,Encoder_Right);
  Turn Pwm    =turn(Encoder Left,Encoder Right,Gyro
```

我们得到的 MPU6050 输出的陀螺仪的原始数据，通过观察数据，我们发现最大值不会超过 4 位数（正常应用在平衡小车上的时候），再根据 7200 代表占空比 100%，所以我们估算 kp 值应该在 0~2 之间。

先设定 kp=-0.6,我们可以看到，当我们把小车摁在地上旋转的时候，我们可以很轻易的转动小车，说明目前小车没有通过负反馈把目标角速度控制在零附近，而是通过正反馈帮助我们旋转小车，说明了这个时候小车的转向系统是正反馈的。然后我们设定 kp=0.6，这个时候我们把小车摁在地上旋转会发现使用很大的力也难以转动小车，小车会反抗我们，并通过电机保持角速度为零，这就是典型的角速度负反馈效果，也是我们需要看到的效果。

1.3.2. 确定 kp 的大小

下面我们进行平衡小车转向控制 kp 值的整定，此时需要打开直立环和速度环，如下图所示：

```
Balance_Pwm =balance(Angle_Balance,Gyro_Balance);
Velocity_Pwm=velocity(Encoder_Left,Encoder_Right);
Turn_Pwm    =turn(Encoder_Left,Encoder_Right,Gyro_Turn);
```

首先，设定 kp=0.2，这个时候我们可以看到，小车的转向控制比较弱，走直线的偏差非常大。

设定 kp=0.6，这个时候我们可以看到，小车的角速度控制的响应有所加快，但是走直线还不是特别理想。

设定 kp=1，这个时候我们可以看到，小车走直线的效果已经很不错了，我们接下来尝试加大 kp 值看一下效果。

设定 kp=1.6 这个时候我们可以看到，小车虽然走直线的效果更好了，但是小车在急停的时候有剧烈的抖动。所以这组参数不可取。我们可以确定得到 kp=1 是转向控制 P 参数的理想值。

至此，转向控制调试部分就告一段落了，如果要加入遥控转功能的话，转向控制函数应该改成如下所示(其中加粗部分为是实现遥控功能的代码)：

```
int turn(int encoder_left,int encoder_right,float gyro)
{
    float Turn,Kp=1,Bias;
    if(1==Flag_Left)      Turn_Amplitude=1100;
    else if(1==Flag_Right)      Turn_Amplitude=-1100;
    else    Turn_Amplitude=0;
    Bias=gyro-0；
    Turn=-Bias*Kp;
    Turn+=Amplitude;
    return Turn;
}
```

关于这个包括了遥控左右转向的转向控制函数，做如下解析：

1.在 usart3.c 中的串口 3 接收中断函数，改变 Flag_Left 和 Flag_Right，进而遥控小车。

2.Turn+=Amplitude;　转向遥控叠加在转向控制里面。