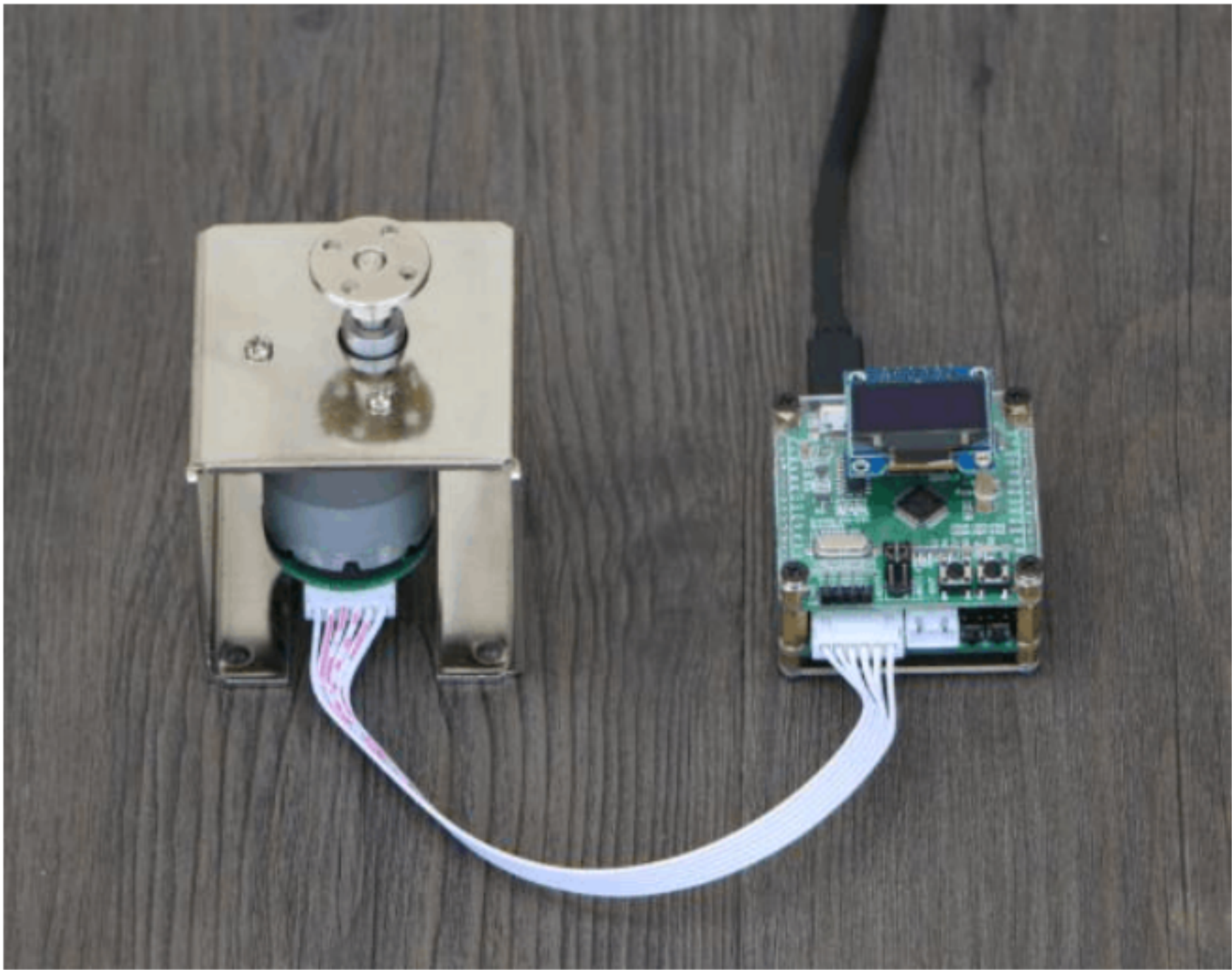


直流电机 PID 开发指南

(基于【平衡小车之家】直流电机 PID 学习套件 1.0)



目录

1. 位置闭环控制	2
1.1 理论分析	2
1.2 控制原理图	2
1.3 C语言实现	3
1.4 参数整定	4
2. 速度闭环控制	9
2.1 理论分析	9
2.2 控制原理图	9
2.3 C语言实现	10

PID 调节器出现于上世纪 30 年代。所谓 PID 控制，就是对偏差进行比例、积分和微分的控制。PID 由 3 个单元组成，分别是比例（P）单元、积分（I）单元、微分（D）单元。在工程实践中，一般 P 是必须的，所以衍生出许多组合的 PID 控制器，如 PD、PI、PID 等。

在我们的微处理器里面，因为控制器是通过软件实现其控制算法的，所以必须对模拟调节器进行离散化处理，这样它只需根据采样时刻的偏差值计算控制量。因此，我们需要使用离散的差分方程代替连续的微分方程。

假定采样时间很短时（比如 10ms），可做如下处理：

1) 用一阶差分代替一阶微分；

2) 用累加代替积分。

1 位置闭环控制

旋转编码器：电机转一圈编码器输出一系列的脉冲
根据计数器累加得到的总脉冲数除以一圈的脉冲数就得到电机转了几圈
就能得到滑块的位置信息
使用程序多测试几次就能知道电机到达哪个位置的时候计数值是多少
从而根据计数值来得到电机带动的滑块的位置

位置闭环控制就是根据编码器的脉冲累加测量电机的位置信息，并与目标值进行比较，得到控制偏差，然后通过对偏差的比例、积分、微分进行控制，使偏差趋向于零的过程。

编码器的线数，就是一圈输出多少个脉冲

1.1 理论分析

根据位置式离散 PID 公式

$$Pwm = K_p * e(k) + K_i * \sum e(k) + K_d [e(k) - e(k-1)]$$

$e(k)$ ：本次偏差

$e(k-1)$ ：上一次的偏差

$\sum e(k)$ ： $e(k)$ 以及之前的偏差的累积和；其中 k 为 1, 2, ..., k ;

$out = out + pid$
 $pid = p + i + d$
 $p = kp * (sv - pv) = kp * e0$, 系数 * 本次误差值
 $i = ki * (e0 + e1 + e2)$, 系数 * 所有的误差值之和
 $d = kd * (e1 - e0)$, 系数 * 最近 2 次的误差之差

Pwm 代表输出

1.2 控制原理图

图 1 为位置控制原理图。其中需要说明的是，我们这边是通过微机实现 PID 控制的，所以下面的【位置 PID 控制器】是一个软件实现的过程，比如在我们的代码里面就是一个我们定义的函数。

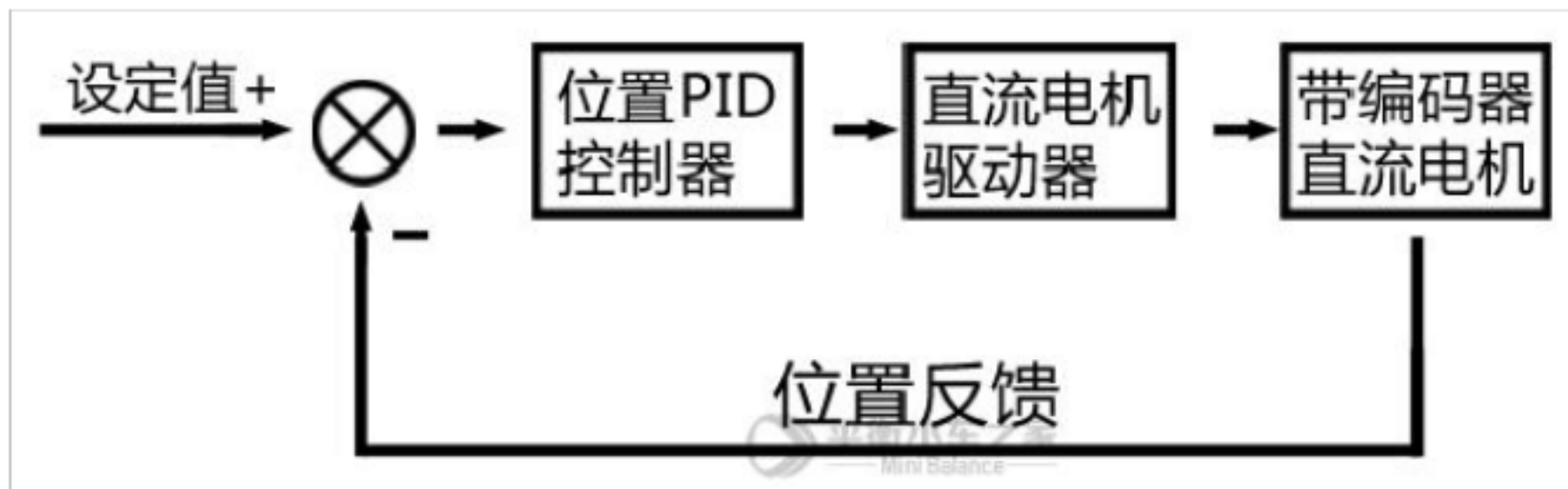


图 1

1.3 C 语言实现

如何把我们以上的理论分析和控制原理图使用 C 语言写出来呢，这是一个有趣且实用的过程。位置式 PID 具体通过 C 语言实现的代码如下：

```
int Position_PID (int Encoder,int Target) 过程值
pv: process value
设定值
sv: setpoint value
{
    static float Bias,Pwm,Integral_bias,Last_Bias;

    Bias=Encoder-Target; 误差=设定值-过程值(现在值) // 计算偏差

    Integral_bias+=Bias; 对所有的误差累加 // 求出偏差的积分

    Pwm=Position_KP*Bias+Position_KI*Integral_bias+Position_KD*(Bias-Last_Bias);

    Last_Bias=Bias; // 保存上一次偏差

    return Pwm; P: 系数乘以误差 I: 系数乘以所有误差累加和 D: 系数乘以最近2次误差之差 // 增量输出
}

```

入口参数为编码器的位置测量值和位置控制的目标值，返回值为电机控制 PWM 现在再看一下上面的控制原理图是不是更加容易明白了)。

第一行是相关内部变量的定义。

第二行是求出速度偏差，由测量值减去目标值。

第三行通过累加求出偏差的积分。

第四行使用位置式 PID 控制器求出电机 PWM

第五行保存上一次偏差，便于下次调用。

最后一行是返回。

然后，在定时中断服务函数里面调用该函数实现我们的控制目标：

Moto=Position_PID(Encoder,Target_Position);

Set_Pwm(Moto); //=== 赋值给 PWM寄存器

1.4 参数整定

首先我们需要明确我们的控制目标，也就是满足控制系统的 3 个要求：

- ① 稳定性
- ② 快速性
- ③ 准确性

具体的评估指标有最大超调量、上升时间、静差等。

最大超调量是响应曲线的最大峰值与稳态值的差，是评估系统稳定性的一个重要指标；上升时间是指响应曲线从原始工作状态出发，第一次到达输出稳态值所需的时间，是评估系统快速性的一个重要指标；静差是被控量的稳定值与给定值之差，一般用于衡量系统的准确性，具体可以参考图 2 的解析。

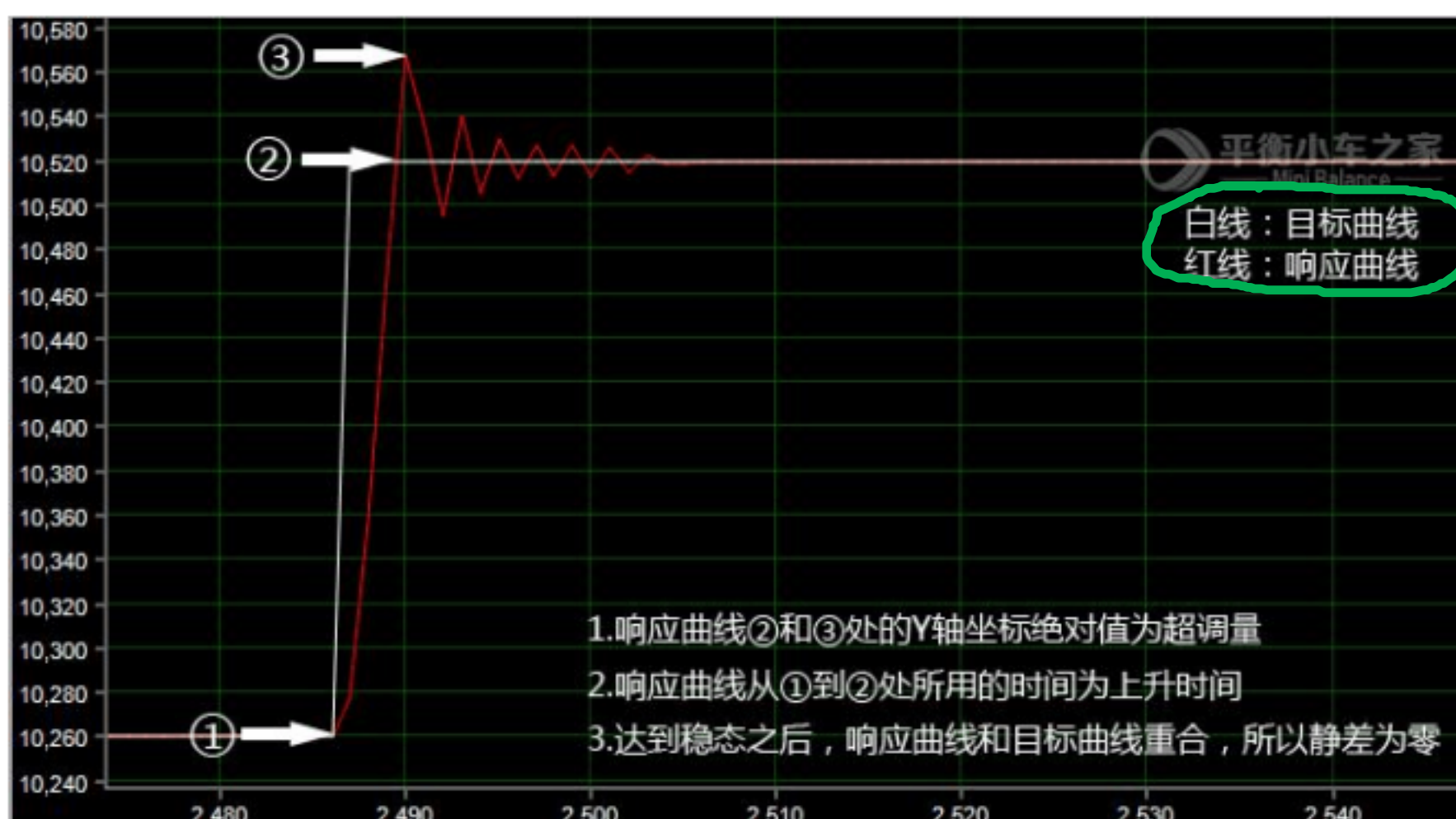


图 2

下面我们使用【平衡小车之家】直流电机 PID 学习套件 1.0 进行 PID 参数整定实验，使用套件的控制板上面的按键调节 PID 参数，然后通过观察上位机响应曲线，并评估控制效果，给出 PID 调节的心得。关于 P、I、D 三个参数的主要作用，可以大致又不完全地概况为：P 用于提高响应速度、I 用于减小静差、D 用于抑制震荡。

下面我们把控制目标从 10000 上升至 10260 时，观察响应曲线的变化。一般我们进行 PID 参数整定的时候，首先设 I 和 D 值为零，然后把 P 值从 0 逐渐增大，

直到系统震荡。

$KP=500, KI=0, KD=0$ 响应曲线如图 3：

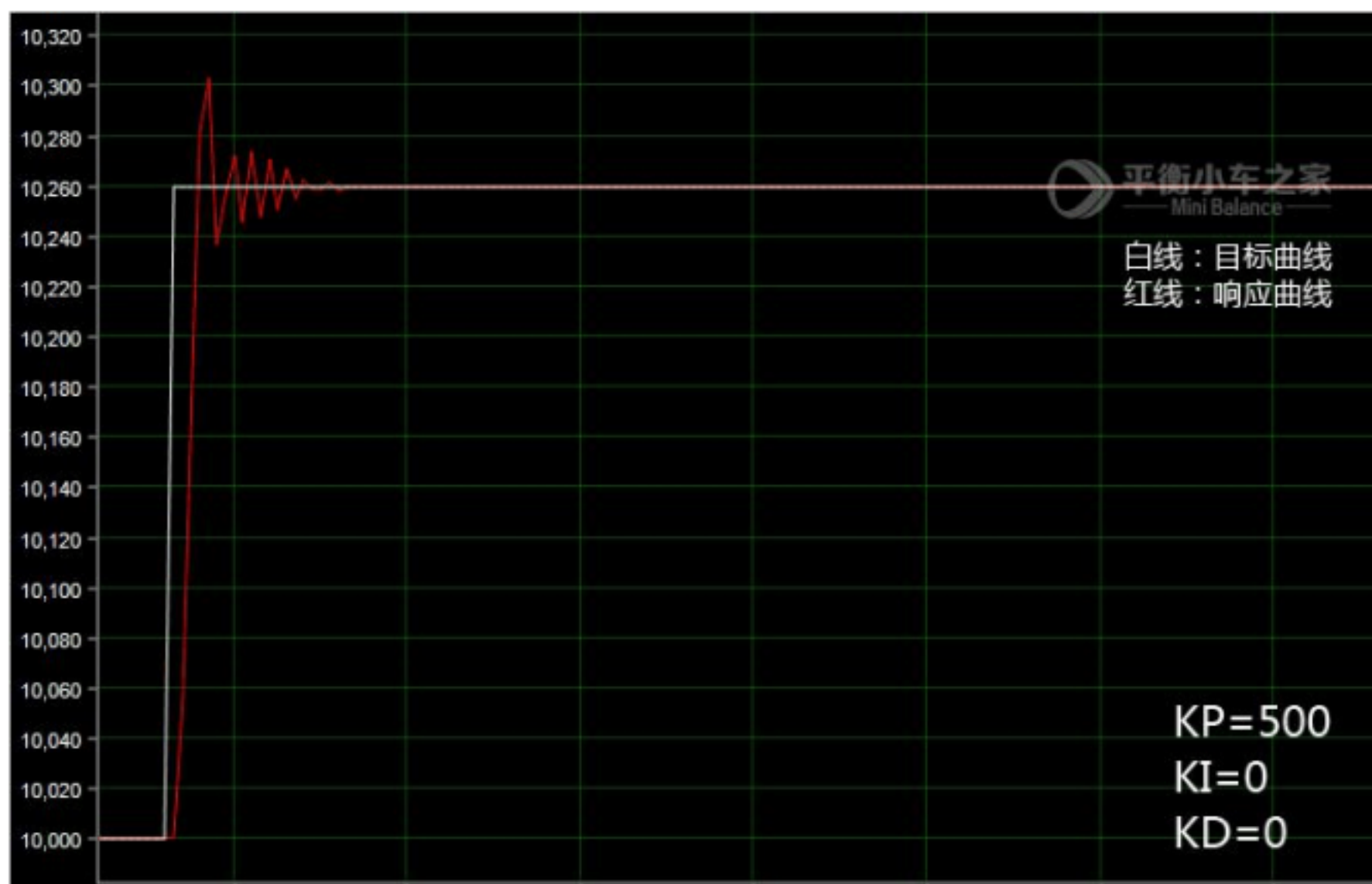


图 3

这个时候因为 P 值比较大，出现了震荡。可能大家会疑惑，为什么 I 值为零，但是没有静差呢？因为这个时候的 P 值已经很大了，静差一般是在 P 值较小而 I 值为零的时候出现的。为了验证我们的想法，我们对 PID 参数进行调整。

$KP=50, KI=0, KD=0$ 响应曲线如图 4：



图 4

据图分析，如我们所设想的，在 P 值较小的时候出现了静差，响应速度也明显降低。所以增大 P 值可以一定程度上消除静差，提高响应速度，但是会导致系统震荡，而加入微分控制可以有效抑制震荡。下面我们尝试一组新的 PID 参数。

$KP=500$, $KI=0$, $KD=400$ 响应曲线如图 5：



图 5

据图分析，加入微分控制之后，图 5 与图 3 相比，系统的震荡得到了抑制，震荡次数减少。事物都有两面性，微分控制也是弊端的。可以看到，系统的响应明显变慢了，因为引入微分控制相当于增大了系统的阻尼。这个时候我们需要结合 P 值和 I 值进行进一步的优化。

在实践生产工程中，不同的控制系统对控制器效果的要求不一样。比如平衡车、倒立摆对系统的快速性要求很高，响应太慢会导致系统失控。智能家居里面的门窗自动开合系统，对快速性要求就不高，但是对稳定性和准确性的要求就很高，所以需要严格控制系统的超调量和静差。所以 PID 参数在不同的控制系统中是不一样的。只要我们理解了每个 PID 参数的作用，我们就可以应对工程中的各种项目的 PID 参数整定了。

位置控制的调节经验可以总结为：先只使用 P 控制，增大 P 系数至系统震荡之后加入微分控制以增大阻尼，消除震荡之后再根据系统对响应和静差等的具体要求，调节 P 和 I 参数。

一般而言，一个控制系统的控制难度，一般取决于系统的转动惯量和对响应

速度的要求等。转动惯量越小、对响应速度要求越低，PID 参数就越不敏感。

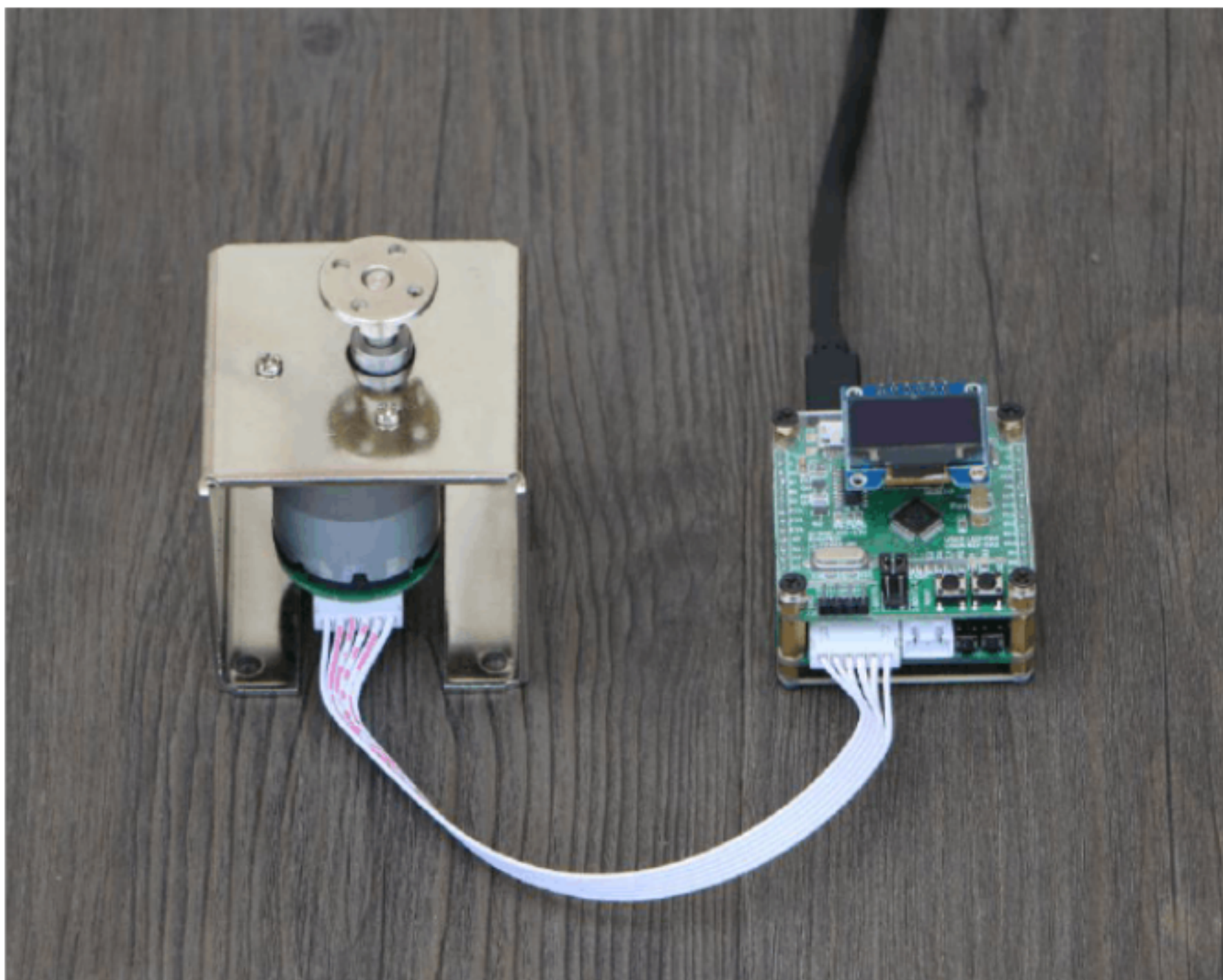


图 6

比如现在我们接到一个任务，使用如图 6 所示的【平衡小车之家】直流电机 PID 学习套件控制电机转 90° ，需要严格控制超调量、和静差。但是对响应速度无要求。

因为电机处于轻载的情况下，转动惯量很小，这是一个很容易完成的工作。根据上面的理论分析和实践，因为响应速度无要求，一般 P 应该给小一点，然后加大系统的阻尼防止超调，也就是 D 参数尽量大，另外因为 P 值较小，应该加入 I 控制减小静差。根据我们的经验和简单的参数整定，最终得到一组 PID 参数 $K_P=120$, $K_I=0.1$, $K_D=500$ 响应曲线如图 7 所示。

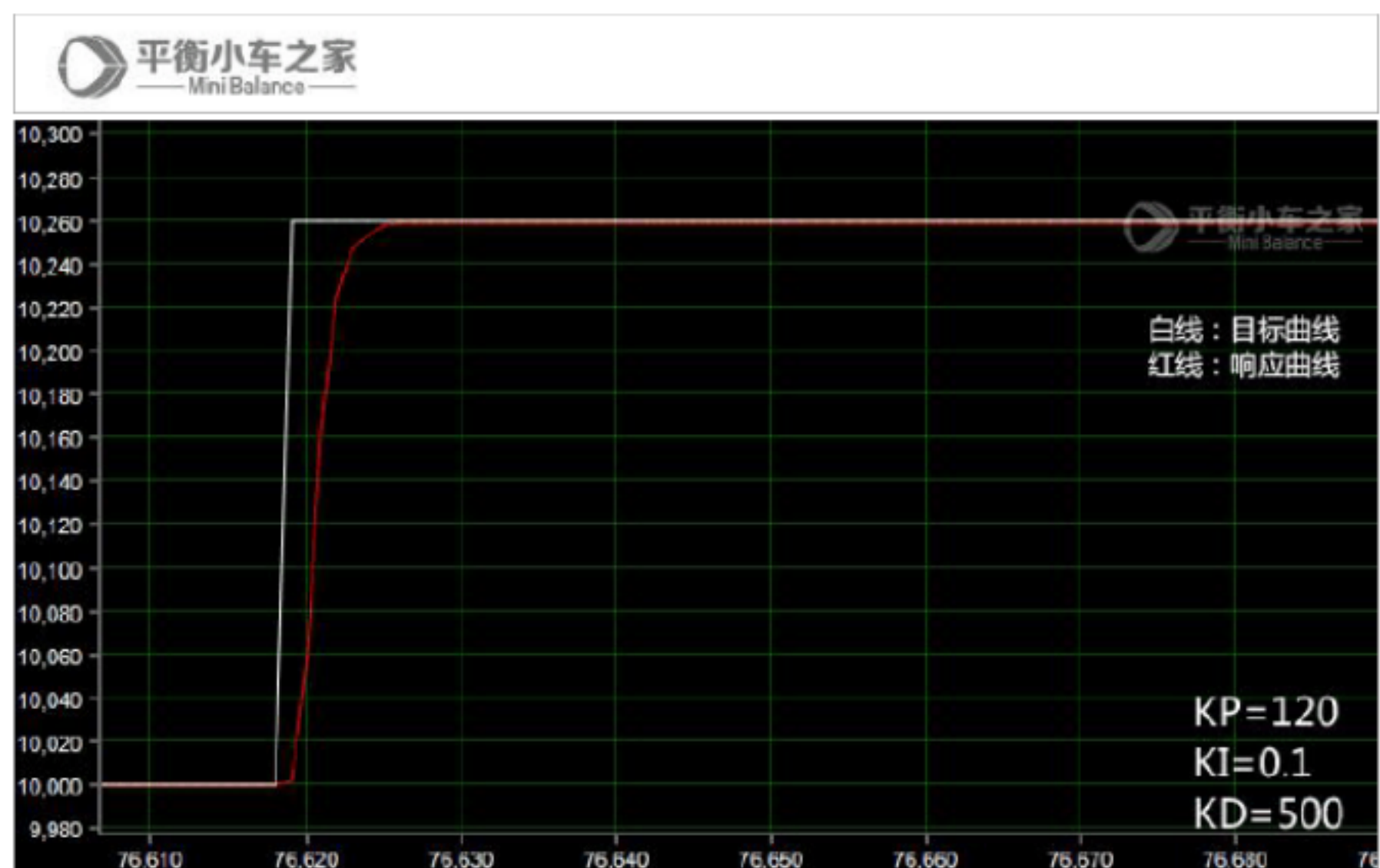


图 7

2. 速度闭环控制

速度闭环控制就是根据单位时间获取的脉冲数（这里使用了 M 法测速）测量电机的速度信息，并与目标值进行比较，得到控制偏差，然后通过对偏差的比例、积分、微分进行控制，使偏差趋向于零的过程。

一些 PID 的要点在位置控制中已经有讲解，这里不再赘叙。

需要说明的是，这里速度控制 20ms 一次，一般建议 10ms 或者 5ms，因为在这里电机是使用 USB 供电，速度比较慢，20ms 可以延长获取速度的单位时间，提高编码器的采值。

2.1 理论分析

根据增量式离散 PID 公式

$$Pwm = Kp[e(k) - e(k-1)] + Ki * e(k) + Kd[e(k) - 2e(k-1) + e(k-2)]$$

$e(k)$ ：本次偏差

$e(k-1)$ ：上一次的偏差

$e(k-2)$ ：上上次的偏差

Pwm 代表增量输出

在我们的速度控制闭环系统里面只使用 PI 控制，因此对 PID 控制器可简化为以下公式：

$$Pwm = Kp[e(k) - e(k-1)] + Ki * e(k)$$

2.2 控制原理图

图 8 为速度控制原理图。其中需要说明的是，我们这边是通过微机实现 PID 控制的，所以下面的【速度 PI 控制器】是一个软件实现的过程，比如在我们的代码里面就是一个我们定义的函数。

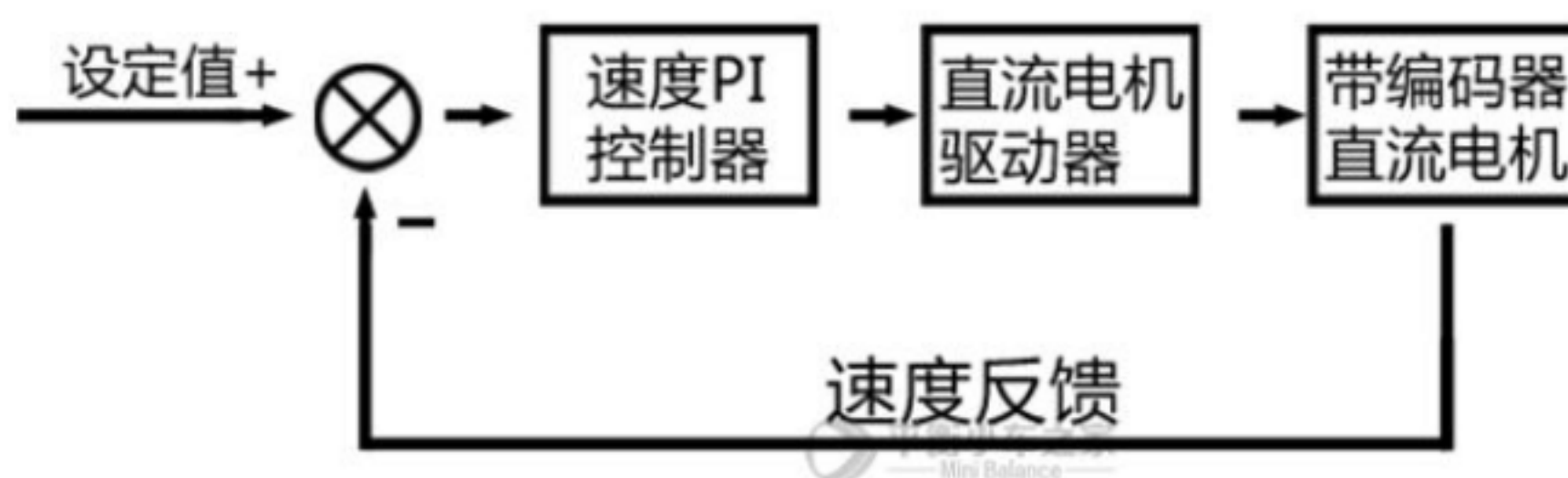


图 8

2.3 C 语言实现

增量式 PI 控制器具体通过 C 语言实现的代码如下：

```
int Incremental_PI (int Encoder,int 300Target)
{
    static float Bias,Pwm,Last_bias;
    Bias=Encoder-Target; 偏差=设定值-过程值 // 计算偏差
    Pwm+=Velocity_KP*(Bias-Last_bias)+Velocity_KI*Bias; // 增量式 PI 控制器
    Last_bias=Bias; // 保存上一次偏差
    return Pwm; // 增量输出
}
```

入口参数为编码器的速度测量值和速度控制的目标值，返回值为电机控制

PWM

第一行是相关内部变量的定义。

第二行是求出速度偏差，由测量值减去目标值。

第三行使用增量 PI 控制器求出电机 PWM

第四行保存上一次偏差，便于下次调用。

最后一行是返回。

然后，在定时中断服务函数里面调用该函数实现我们的控制目标：

```
Moto=Incremental_PI(Encoder,Target_Velocity);
```

```
Set_Pwm(Moto); //== 赋值给对应 MCU 的 PWM 寄存器
```

这里我们的参数 Velocity_KP=20,Velocity_KI=30 ,PI 参数在不同的系统中不一样，我们的代码中的 PID 参数，仅针对【平衡小车之家】直流电机 PID 学习套件 1.0 调试得到，大家可以使用控制板的按键调节 PI 参数，然后通过观察上位机波形评估控制效果。

以上的知识请结合完整代码理解，我们的代码基于 STM32F103C 控制器，但是把基于 C 语言的 PID 控制器部分剥离，并放在 control.c 里面，故对 STM32 不熟悉的同学依然可以使用记事本打开这个文件查看。

平衡小车之家版权所有，部分资料在 EEPV 有发表，如果需要转载或者引用，请联系我们。

直流电机 PID 学习套件详情地址：

<https://item.taobao.com/item.htm?spm=a1z10.1-c.w4004-9258381245.1.xfmqC1&id=532496279494>