功率控制

author：郭嘉豪

> - 完成了稳定功率控制
> - 稳定剩余能量控制
> - 被动电容

数据说明

对于机器人底盘，功率相关的数据来源有裁判系统、超级电容功率控制板量测和电机电调反馈，能够读取以下信息：

数据名	符号（单位）	数据源	频率（Hz）	描述
底盘功率上限	$P_m(\mathrm{W})$	裁判系统	10	当前等级下该兵种的底盘功率上限（max），超限则首先扣除缓冲能量
底盘缓冲能量	$Z(\mathrm{J})$	裁判系统	10	缓冲能量余量，随裁判系统功率量测动态变化，若耗尽将导致超功率惩罚。目前（2023赛季）无增益上限为 60 J，触发飞坡增益后增加至 250 J 一次
底盘功率	$P_r(\mathrm{W})$	裁判系统	10	裁判系统（referee）测得的底盘总功率
底盘功率	$P_c(\mathrm{W})$	超电主控	1000	电流计测得的底盘总功率
超级电容的能量	$C^e(J)$	超电主控	1000	$C^e(J)=0.5\times V^2\times C$，已知电容电压和电容值可以求得能量
电机转速	$\Omega(\mathrm{rpm})$	电调	1000	电机转速反馈
电机转矩电流	$i(\mathrm{A})$	电调	1000	电机转矩电流反馈，手册未指明量纲，经测试与输入一致：C620 电调原始值范围 $[-16384,16384]$，线性映射到 $[-20,20]\mathrm{A}$.

人为指定或计算得到以下数据，频率为 1KHz：

数据名	符号（单位）	描述

数据名	符号（单位）	描述
底盘功率模型计算值	$P_{model}(\mathrm{W})$	根据功率模型计算得到的底盘功率(可以和电流计当前功率做对比来观察拟合效果)
底盘功率模型预测值	$P_{pre}(\mathrm{W})$	根据电机功率模型计算得到，预测（predict）下一控制周期的底盘功率
动态底盘功率上限	$P_{set}(\mathrm{W})$	计算得到的动态变化、用于功率限制的底盘功率目标上限
底盘缓冲能量目标值	$Z_{set}(\mathrm{J})$	缓冲能量目标值，用户给定
超级电容的目标能量	$C^e_{set}(J)$	电容剩余能量目标值，用户给定
电机转速目标值	$\Omega_{set}(\mathrm{rpm})$	电机转速目标值，用户给定
电机转矩电流目标值	$i_{set}(\mathrm{A})$	电机转矩电流目标值，基于转速目标值由控制器计算得出。C620 电调原始值范围 $[-16384, 16384]$，线性映射到 $[-20, 20]\mathrm{A}$.

功率公式

功率模型

根据现有能够获取的数据，一个较为准确的电机功率模型为：

$$P = \Omega \cdot M + P_{loss}$$

其中 M 表示输出转矩，P_{loss} 表示除机械输出以外的功率损耗，$P_{loss} = I^2R + P_0'$ 表示热损耗和静息损耗

考虑电磁转矩和转矩电流 i_q 近似成正比（C620 电调搭配 M3508 电机速度闭环控制的电机性能曲线），定义转矩系数 K_0，有

$$M = K_0 \cdot i_q$$

> 根据电调手册，转矩常数为 $0.3\ \mathrm{N \cdot m/A}$，可作为 K_0的参考值

综上所述，对于单个电机的功率可由下式计算

$$P = K_0 \Omega I + R_0 I^2 + P_0'$$

对于安装有四个电机的整个底盘，功率模型如下

$$P = K_0 \sum_i^4 \Omega I + R_0 \sum_i^4 I^2 + P_0 \tag{1}$$

当我们已知每个电机的转速$\Omega_i(rpm)$和电流$I_i(A)$时带入上式即可求出当前底盘四个电机的功率(W)。

拟合功率系数

对于上一节的模型推导，我们发现只需要知道三个常数K_0、R_0、P_0即可求出功率，我们将通过采集数据拟合的方式来拟合得到三个常数准确值。

拟合模型

观察我们需要拟合的功率模型（1）可以化简成包含三个常数的如下数学模型

$$f(z) = ax + by + c$$

其中可知：$x = \sum_i^4 \Omega I$，$y = \sum_i^4 I^2$

常数：$a = K_0$，$b = R_0$，$c = P_0$

为了拟合常数，x、y已经可以求得，这里$f(z)$应该是裁判系统的功率当前值P_r，不过我们使用了自己安装的电流计该测量了电池的电压和电流，计算出电池功率P_c作为$f(z)$，该功率值可以等效于裁判系统P_r，实际观察中两个功率只差了一个常数，大约为3~4w，如下图

xxxxxxxxxx

因此，拟合常数时使用P_c并不会太影响拟合效果，反而由于P_c的频率高，与x、y的计算频率相当，所以拟合效果更佳。

综上：

$$f(z) = ax + by + c$$

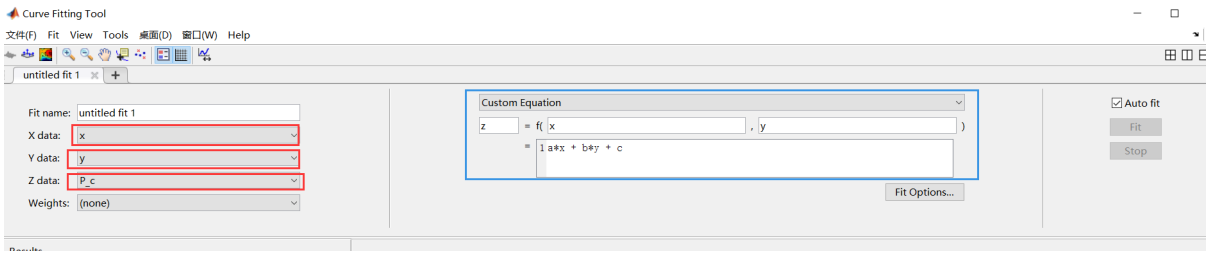
$$f(z) = P_c \qquad x = \sum_i^4 \Omega I \qquad y = \sum_i^4 I^2$$

采集数据利用matlab拟合

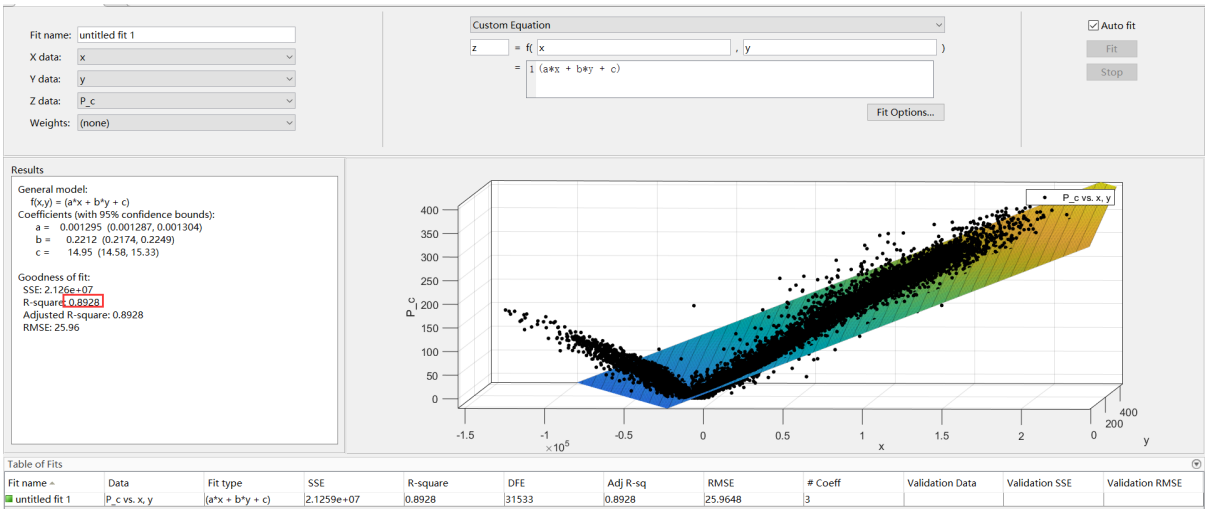
在代码中以1KHZ频率计算好x、y，并将x、y、P_c的值保存到表格中"power.csv"

```
date=xlsread('power.csv');
x=date(:,1);%取date的第一列数据
y=date(:,2)
P_c=date(:,3)
```

在命令行输入cftool打开拟合工具箱，设置自定义的公式如下图：



拟合效果：

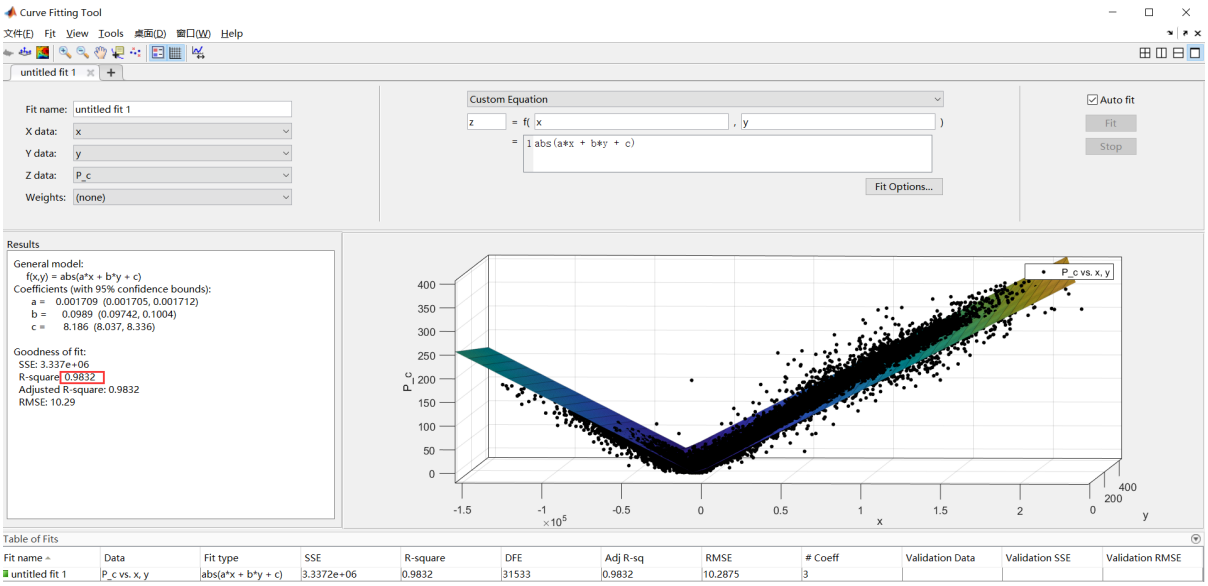


可以发现x负数的部分拟合不到，这是由于P_c和裁判系统P_r只有正值，没有负功率的预测，发现功率P_c大致关于x=0对称，因此我们修正一下拟合的公式，添加绝对值

综上：

$$f(z) = abs(ax + by + c)$$

拟合效果如下：



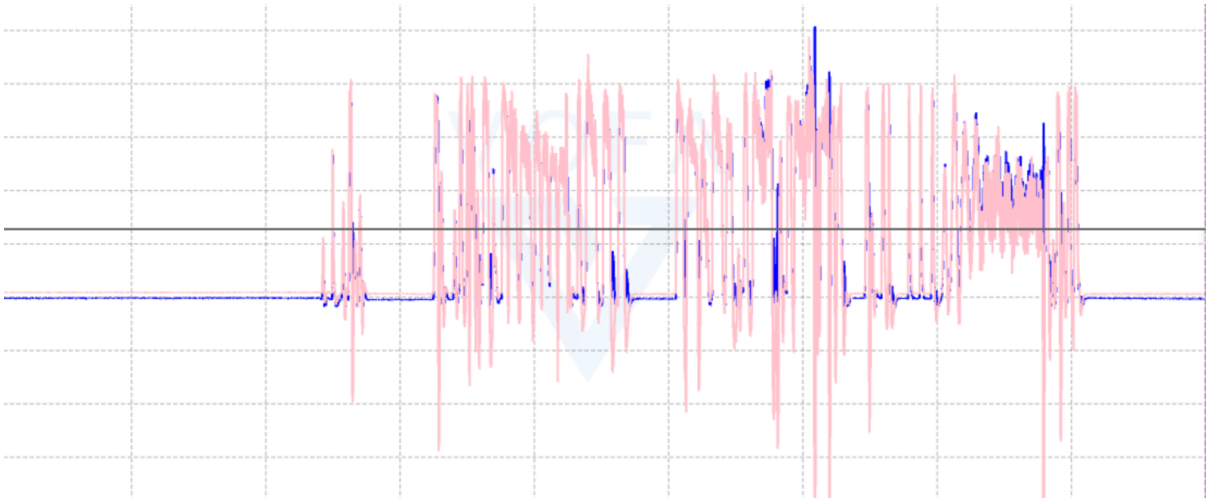
拟合的效果R^2也从0.89提升到了0.98，基本上说明该模型能完美拟合功率计算

$$K_0 = 0.001709，\ R_0 = 0.0989，\ P_0 = 8.186$$

至此我们即可通过转速和电流直接求得底盘整车在该状态下的功率P_{model}

$$P_{model} = K_0 \sum_{i}^{4} \Omega_{cur} i_{cur} + R_0 \sum_{i}^{4} i_{cur}^2 + P_0$$

将P_{model}和P_c绘制对比如下，可以看到基本完全重合：



功率控制

有了上述的功率模型，因此可以使用该模型来预测下一时间段的功率，即当我们通过can发送电流控制值给电机时，我们可以将即将要发送的电流Ω_{set}、i_{set}带入上式，因此可以计算出P_{pre}。

$$P_{pre} = K_0 \sum_i^4 \Omega_{set} i_{set} + R_0 \sum_i^4 i_{set}^2 + P_0$$

> Ω_{set}、i_{set}：一般会通过获取遥控器或者键盘的输入通过运动分解得到四个轮子的目标转速即Ω_{set}，然后通过PID控制器来求出i_{set}，而且由于电调的性能较好，我们假定只要通过can发送i_{set}，电机的电流会立刻变化到目标值，而转速变化具有一定滞后性，1ms的控制周期内变化非常小，因此Ω_{set}不一定是下一周期的转速实际值，假如轮子此时转速为5000rpm，转速目标设置为-4000rpm，那么此时用$\Omega_{set} = -4000$作为预测则非常不准确，使用Ω_{cur}则误差更小。

则修正后的功率预测式：

$$P_{pre} = K_0 \sum_i^4 \Omega_{cur} i_{set} + R_0 \sum_i^4 i_{set}^2 + P_0$$

由于每个轮子我们都使用了PI控制器来控制轮子的转速，PI控制器是线性控制器，即控制器可以简化为

$$i_{set} = K \times \Omega_{set} + M$$

其中K和M是已知的常数。（后续会推到K、M的计算公式，现在当作已知常数即可）

因为其他的参数都是已知的常量，因此预测功率可以表达成：

$$P_{pre} = f(\Omega_{set})$$

为了将功率控制不超过P_{set}，则当$P_{pre} > P_{set}$，时我们需要减小Ω_{set}或者i_{set}来达到削减下一周期功率的目的，这里我们通过削减当前目标转速从而不超过设置的功率，则设削减转速的方式为比例衰减：
$\Omega'_{set} = k_c \cdot \Omega_{set}$，使得$P'_{pre} = f(\Omega'_{set}) \leq P_{set}$，以下取等来解出最优$k_c$：

$$f(k_c \cdot \Omega_{set}) = P_{set} \tag{2}$$

将公式（2）整理后可以将其表示为 $ak_c^2 + bk_c + c = 0$ 的形式：

$$R_0 \sum_i^4 (K^2 \Omega_{set}^2) \cdot k_c^2 + \sum_i^4 (2R_0 KM\Omega_{set} + K_0 K \Omega_{cur} \Omega_{set}) \cdot k_c + \sum_i^4 (R_0 M^2 + K_0 M \Omega_{cur}) + P_0 - P_{set} = 0$$

最后，将系数a、b、c为：

$a = R_0 \sum_i^4 (K^2 \Omega_{set}^2)$

$b = \sum_i^4 \Omega_{set} (2R_0 KM + K_0 K \Omega_{cur})$

$c = \sum_i^4 (R_0 M^2 + K_0 M \Omega_{cur}) + P_0 - P_{set}$

由于a>0，则是一个开口向上的一元二次函数：

- 若解存在（$b^2 - 4ac \geq 0$），解得：

$$k_c = \frac{-b \pm \sqrt{b^2 - 4ac}}{2\alpha}, \text{取解} \in (0, 1)$$

- 若出现解不存在的情况，求使得 (4) 式左边结果最小的 k_l 值，以尽可能地限制功率，即二次函数的对称轴：

$$k_l = -\frac{b}{2a}$$

若某个电机 $K(k_c \Omega_{set}^j) + M$ 超出电机可输出的转矩电流上限，则应按照该上限额外计算一个转速限制 k_s，使得

$$K(k_c k_s^j \Omega_{set}^j) + M \triangleq i_{max}$$

并按照所有超限轮组中的 $k_s = \min\{k_s^j\}$，作用于底盘的全部轮组。经削减的轮组目标转速表示为：

$$\Omega'_{set} = k_s \cdot k_c \cdot \Omega_{set}$$

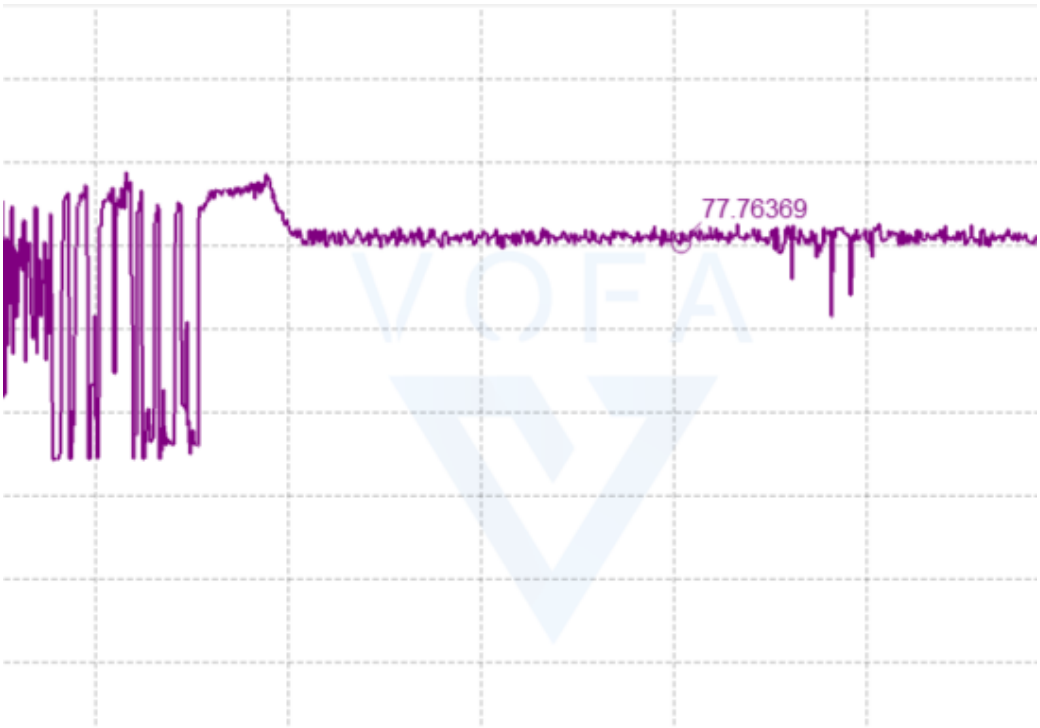
最后将功率控制模块削减后的Ω'_{set}分别经过四个PI控制器得到i'_{set}发送给电机即可，可以发现车子在运行过程中（需要的功率大于P_{set}时）P_r能很好的保持在P_{set}

前述的整个控制求解过程代码对应如下：

```
// 发送期望转速之后产生的功率预测
a=0,b=0,c=0;
for(int i=0;i<4;i++){
    Pid_Typedef *P = &(power_limit->pidChassisWheelSpeed[i]);
    a += RM3508_R * P->SetPoint * P->SetPoint * (power_limit->K[i]) * (power_limit-
>K[i]);
    b += P->SetPoint * (2*RM3508_R*power_limit->K[i]*power_limit->M[i] +
RM3508_K*power_limit->K[i]*power_limit->Speed_Now[i]);
    c += (RM3508_R*power_limit->M[i]*power_limit->M[i] + RM3508_K*power_limit-
>M[i]*power_limit->Speed_Now[i]);
}
c += RM3508_P0;
//这里相当于k_c=1时，预测功率
power_limit->predict_send_power = LIMIT_MAX_MIN(a + b + c,1000,-1000);//过滤转速预测异常数
据 W
//power_limit->predict_send_power = a + b + c;

// 超功率衰减
if(power_limit->predict_send_power > power_limit->set_power)
{
    // 模型衰减计算
    if(b*b < 4*(c - power_limit->set_power)*a){//   b方小于4ac：没有解
        power_limit->k_c = LIMIT_MAX_MIN(-b/(2*a) , 1.0,0.0);//   -b/(2a)
    }else{
        float32_t sqrt_result;
        arm_sqrt_f32(b * b - 4*(c - power_limit->set_power)*a,&sqrt_result);
        power_limit->k_c = LIMIT_MAX_MIN((-b + sqrt_result) / 2 / a,1.0 ,0.0);
    }

    //衰减
```

```
    for(int i=0;i<4;i++){
        power_limit->pidChassisWheelSpeed[i].SetPoint *= power_limit->k_c;//衰减
    }

}

//PI控制器计算目标电流
for(int i=0;i<4;i++){
    power_limit->Current_set[i] = PID_Calc(&(power_limit-
>pidChassisWheelSpeed[i]),power_limit->Speed_Now[i]);
}
```

最终控制的效果如下（P_{set} = 80W），非常稳定的控制效果，稳定少了3W是由于使用P_c作为拟合的目标产生的固定误差，上面拟合数据一节已经有介绍，可以补偿：



PI控制器求解$i_{set} = K \cdot k_c \cdot \Omega_{set} + M$

我们的PI控制算法如下：

```
float PID_Calc(Pid_Typedef *P, float ActualValue)
{
        P->PreError = P->SetPoint - ActualValue;
        P->SetPointLast = P->SetPoint;

        if(P->PreError > -P->ErrorMax && P->PreError < P->ErrorMax)
        {
            P->SumError += (P->PreError + P->LastError)/2;    //梯形积分提高精度
        }

        P->LastError = P->PreError;

        if(P->SumError >= P->IMax)
            P->SumError = P->IMax;
        else if(P->SumError <= -P->IMax)
```

```
            P->SumError = -P->IMax;

        P->POut = P->P * P->PreError;
        P->IOut = P->I * P->SumError;

        return LIMIT_MAX_MIN(P->POut+P->IOut,P->OutMax,-P->OutMax);
}
```

可以分为两种情况

- 积分饱和后或者$ABS(P->PreError) > ErrorMax$则不会再进行积分，积分项$K_i \cdot Sumerror$就是一个常值
- 积分项未饱和和误差小于最大误差，积分项正常按照梯形公式进行积分

两种情况下K、M的值不同推导如下：

1.可以正常积分时：

$$i_{set} = K_p \times (k_c \cdot \Omega_{set} - \Omega_{cur}) + K_I \times \left(\frac{k_c \cdot \Omega_{set} - \Omega_{cur} + \Delta\Omega_{last}}{2} + SumError_{last} \right)$$

$$i_{set} = (K_p + \frac{K_I}{2}) \cdot k_c \cdot \Omega_{set} + K_I \times \left(\frac{-\Omega_{cur} + \Delta\Omega_{last}}{2} + SumError_{last} \right)$$

则$K = K_p + \frac{K_I}{2} \quad M = K_I \times \left(\frac{-\Omega_{cur}+\Delta\Omega_{last}}{2} + SumError_{last} \right)$

2.不能正常积分时：

$$i_{set} = K_p \cdot k_c \cdot \Omega_{set} - K_p \cdot \Omega_{cur} + K_I \times SumError_{last}$$

则$K = K_p \quad M = -K_p \cdot \Omega_{cur} + K_I \cdot SumError_{last}$

实际代码中求K、M的代码，并且这里还求解了 k_s

```
/*
*calfromPID
*从pid控制算法中反解转速和电流映射关系常数K，M
*/
void calfromPID(Power_Limit_type *power_limit){
    power_limit->k_s_min = 1.0;
    for(int i=0;i<4;i++){
        Pid_Typedef *P = &(power_limit->pidChassisWheelSpeed[i]);
        short speed_now = power_limit->Speed_Now[i];

        float PreError =  P->SetPoint - speed_now;
        float SumError_last = P->SumError;
        float SumError = SumError_last;
        if(PreError > -P->ErrorMax && PreError < P->ErrorMax)
        {
            SumError += (P->PreError + P->LastError)/2;    //梯形积分提高精度
        }

        if(ABS(SumError) >= P->IMax || ABS(PreError) > P->ErrorMax)//不积分的情况
        {
```

```
            power_limit->M[i] = (-P->P*speed_now + P-
>I*SumError_last)*RM3508_CURRENT_RATIO;
            power_limit->K[i] = P->P*RM3508_CURRENT_RATIO;
        }
        else
        {
            power_limit->M[i] = (-P->P*speed_now + P->I*(SumError_last+0.5*(-
speed_now+P->LastError)))*RM3508_CURRENT_RATIO;
            power_limit->K[i] = (P->P + P->I/2)*RM3508_CURRENT_RATIO;
        }


        double k_s = 1.0;
        double K_W = power_limit->K[i]* P->SetPoint;
        double current = K_W*k_s + power_limit->M[i];
        if(ABS(current) > power_limit-
>pidChassisWheelSpeed[i].OutMax*RM3508_CURRENT_RATIO){
            double l = current>0?1:-1;
            k_s = LIMIT_MAX_MIN((l*power_limit-
>pidChassisWheelSpeed[i].OutMax*RM3508_CURRENT_RATIO - power_limit->M[i])/K_W,1.0,0.0);
            if(k_s < power_limit->k_s_min){
                power_limit->k_s_min = k_s;
            }
        }
    }
    //power_limit->k_s_min = 1.0;
    for(int i=0;i<4;i++){
        power_limit->pidChassisWheelSpeed[i].SetPoint *= power_limit->k_s_min;//最大值越
界衰减
    }

}
```

RM3508_CURRENT_RATIO是将电流值映射为到-20~20A

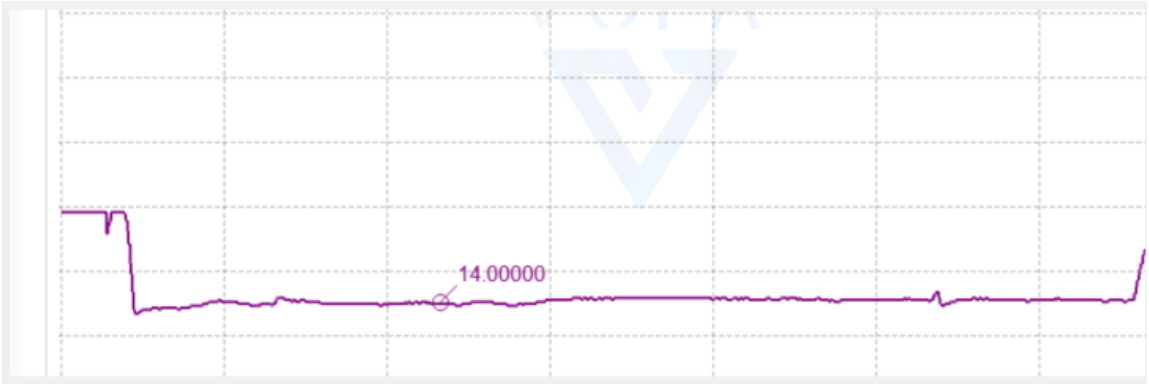
能量控制

当我们的上述控制能将当前功率很好的保持在P_{set}附近，则可以接着在功率控制之上增加别的控制

- 电池作为能量供应时，控制裁判系统剩余能量

  $P_{set} = P_{r_{max}} + 1.5 \times (Z_{cur} - Z_{set})$

  此时相当于对剩余能量做了一个P控制，1.5时系数P，实际控制效果如下：

  剩余能量很好的保持在14j，很平稳。

- 电容作为能量供应时，控制电容剩余能量

$$P_{set} = P_{r_{max}} + 1.5 \times (C^e_{cur} - C^e_{set})$$

$P_{r_{max}}$是当前裁判系统返回的最大功率

超电充电

我们简单的直接使用下式作为超级电容充电功率

$$P_{charge} = P_{set} - P_{pre}$$

$$I_{charge} = \frac{P_{charge}}{22.0V}$$

22.0为电池电压，将I_{charge}作为充电电流

被动电容

被动电容首先解决的问题是电容开关频率，这里我是将电容和电池切换的函数指定f（HZ）调用，调用时检测$Power_{set}$，$Power_{set} = CAP$或者BAT分别切换成电容供电或者电池供电，严格限制了电容切换频率

被动电容则是指定最低裁判系统剩余能量Z_{set}和最低电容剩余能量C^e_{set}后，将整车功率控制在P_{half}附近。一般的，$P_{half} > P_{r_{max}}$，为了保持在较大的P_{half}附近，会消耗剩余能量和电容能量，当两种能量接近最低阈值时切换供电模式即可，通过设置合理的P_{half}使得电容和电池交替达到更大功率更长续航

对应代码如下：

```
if(powerstate_cur == CAP){
    if (power_limit->capEnergy < power_limit->Min_capEnergy*1.2){   //最小阈值之前都是恒功
率
        power_limit->set_power = power_limit->referee_max_power +
                                 power_limit->P_capEngry*(power_limit->capEnergy -
power_limit->Min_capEnergy);
        power_limit->PowerState_Set = BAT;//切换成电池模式
    }else{
        power_limit->set_power = power_limit->HalfCAP_Power;
    }
}else{
    if (power_limit->remainEnergy < power_limit->Min_remainEnergy*1.2){ //最小阈值之前都
是恒功率
        power_limit->set_power = power_limit->referee_max_power +
                                 power_limit->P_remainEngry*(power_limit->remainEnergy
- power_limit->Min_remainEnergy);
        power_limit->PowerState_Set = CAP;//切换成电容模式
    }else{
        power_limit->set_power = power_limit->HalfCAP_Power;
    }
}
```