keil基于stc 教程

蜂鸣器使用

```
初始化 调用底库中的函数（gpio.h）
gpio_mode(P7_7,GPO_PP);
设置高低电平
P77 = 1 注意P77 是在STC6Fxx.h 中定义的 P77 直接调用就行
```

延时函数

```
延时函数初始化
delay_init();
调用函数 1000ms = 1s
delay_ms(1000);
```

液晶

```
液晶初始化
ips114_init();
液晶调用函数
ips114_clear(RED);
```

舵机

```
pwm_init(管脚号, 频率,中值);
pwm_duty(管脚号, 设定的值);
```

电机

```
初始化
pwm_init(PWMA_CH1P_P60, 10000, 0);     //初始化PWM5  使用引脚P2.5  输出PWM频率
10000HZ   占空比为百分之 pwm_duty /
pwm_init(PWMA_CH2P_P62, 10000, 0);
pwm_init(PWMA_CH3P_P64, 10000, 0);     //初始化PWM5  使用引脚P2.5  输出PWM频率
10000HZ   占空比为百分之
pwm_init(PWMA_CH4P_P66, 10000, 0);
执行函数 左轮正传
pwm_duty(PWMA_CH1P_P60, 0);
pwm_duty(PWMA_CH2P_P62, 3000);
执行函数 右轮正传
pwm_duty(PWMA_CH3P_P64, 0);
pwm_duty(PWMA_CH4P_P66, 3000);
```

电机差速算法

```
V_left=V*(1+B*tanα/2L);V_right=V*(1-B*tanα/2L);
```

```
tanα 根据舵机值转换
chasu = (val-mid) * 0.46457
chasu = (int)(chasu * 100) / 100 只保留两位小数
```

C车现在没有机械差速了，只能自己通过建模找到一个合适的建模公式来进行电子差速，我简单的查了一下资料，有个叫ackermann的模型，虽然是理想的，但简单的用在C车，效果还是很不错的，希望大家一起讨论，让今年的C车模绽放光彩~

假设B为后轮两轮的轮距，L为前后轮的间距，假如最优路径为左前方与车正前方夹角为α的直线（α有正有负），就是下图：

拐弯半径在左边和后轮齐平，转弯半径为R，那么后两个轮子的角速度相等，也就是后轮的w_left=w_right，那么
V_left/R_in=V_right/R_out=V/R（假设后轮中间的速度为V），而且R=L/tanα，R_in=R-B/2,R_out=R+B/2,
所以可以推出：

V_left=V*(1+B*tanα/2L);V_right=V*(1-B*tanα/2L);

这样就可以在程序里面把差速写上，V为根据黑线判断的速度，然后后两轮差速，如果觉得差速有点大，就可以乘上一个系数，觉得在弯道差速不够提前，就可以把括号里的1加大点。

昨天又想了一下觉得可以定下一个轮子的速度，然后只让另一个差速，就是每一次只变化一个轮子的速度，右拐时右轮不减速而是固定速度，左轮加速，这样拐弯可以不用减速，但是试了一下好像不太好。

目前我就是先用这个，开环，可以跑到2.3左右，效果比不加或者线性的加要好多了。希望大家集思广益，看能不能加上别的新东西，好让速度更快!

好晚了，睡了~祝大家的C车都能早日上3m!

```
使用keil 编译器生成自己的头文件的时候 切记 要将 头文件的形式 和 其他 头文件的形式 保持一致
```

电机调试 借鉴 三轮

```
int encoder = (nowspeed_L + nowspeed_r) / 2;
将 encoder 作为 左右轮的 实际速度 传入
montor1 += PID(encoder,setspeed_L);
montor2 += PID(encoder,setspeed_R);
进行限幅 很重要 ！！！！
montor1 = limit_protect(moter1,-900,900);
montor2 = limit_protect(moter2,-900,900);

传入pwm 进行运算
```

```
主销后倾、主销内倾、车轮外倾和前束
```

2.3.4 转向控制处理 智能车能够在高速情况下，根据各类规划路径及时动作，主要取决于对智能车转向的控制。在转弯时，合适的转向角度可以使智能车具有更好的路径，同时，也可以提高智能车运行时的稳定性，缩短时间。“经验公式”是一种很好的舵机控制算法，即“偏差的平方*系数+基础值”，在实际操作中，可以达到入弯迅速打角，直道快速回正的效果。刚做车时的“过弯甩尾”现象，原因是过弯后舵机PD控制环中的“P”值偏大，在其不应该振荡时，打角过大不能回中。而“经验公式”动态的可以动态调节P值，在直道时“P”值很小，减小震荡，快速回中，以此提速。值得注意的是，经验公式中的系数与基础值需要根据实际情况，调整合适。

对于车模速度，我们使用编码器通过增量式 PID 闭环控制。对于转弯进行差速处理，通过方向控制计算出的舵机角度来计算转弯的差速（如图 5.2 所示）。

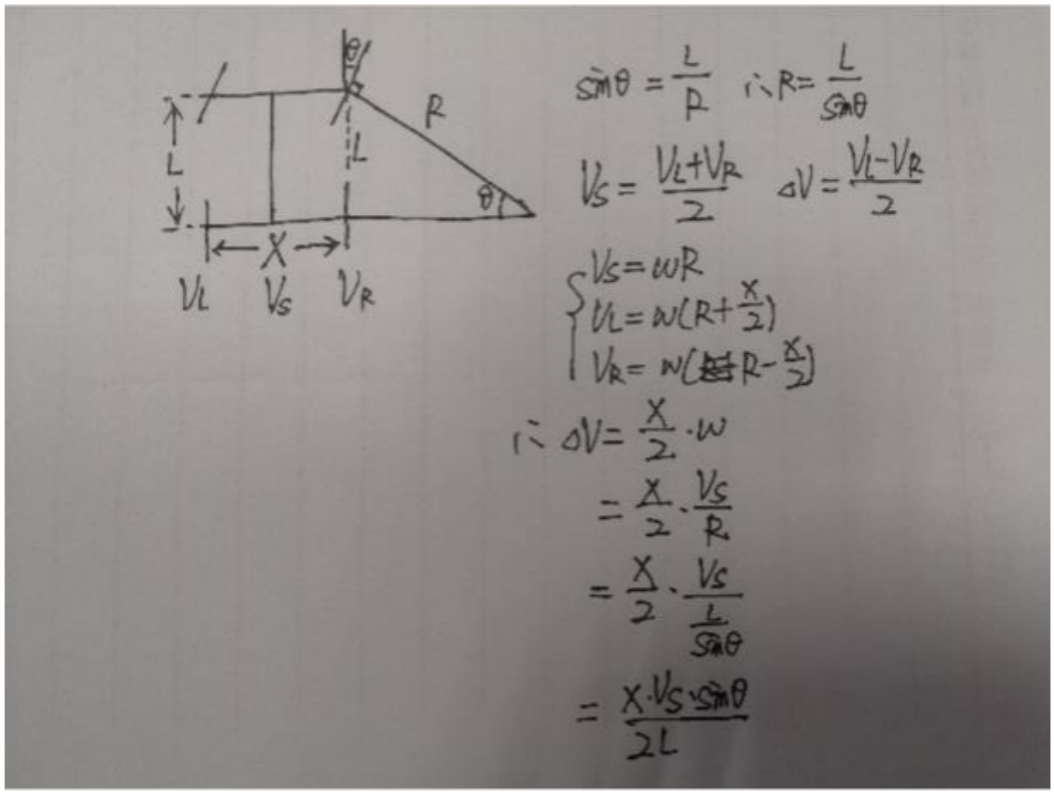


图 5.3 差速算法原理

对于差速的计算，我们通过建立小车转向的模型，对小车的状态分析，

5.3 车模方向控制

通过中间和两边的三个水平电感进行差比和计算偏差，然后对偏差进行位置式 PID 计算从而控制舵机的角度。

```
posError_H = ((adc_value[M0] - adc_value[L2]) /(adc_value[M0] + adc_value[L2])) - ((adc_value[M0] - adc_value[R2]) / (adc_value[M0] + adc_value[R2]));
```

图 5.4 差比和算法实现代码

通过设定中间电感的阈值来判断丢线进行强制打角转向。

设误差为err，则其公式为：

```
err=(L-R)*LIMIT/(L+R)   ;                    （公式1）
```

对公式1加以改良，引入中间电感，使其对弯道敏感度增加

```
err=(A*(L-R)+B*(LM-RM))*LIMIT/(A*(L+R)+B*(LM+RM));
```

（公式2）

由公式1和公式2中可以得出，无论信号源或赛道整体发生什么变化，输出误差值仅与两侧电感相对值有关，且值限制在[0,LIMIT]之内，对于不同的信号源与赛道有着很高的适应能力。

第三次推翻电磁加权算法后，复制粘贴之后由于失误，有一处更改被忽略了，导致产生bug后的公式为：

```
err=(A*(L-R)+B*(LM-RM))*LIMIT/(A*(L+R)+B*(LM-RM));  （公式3）
```

沿用了原先保存的P、D值后，发现小车逆时针跑赛道时，能完成全部元素并且能够容许的速度很高，测试了1.8m/s，2.2m/s，接近2.5m/s，大于2.5m/s的速度，直道与弯道都能很顺利的跑完，S弯也能较为顺利的完成（初步判断是因为给了较大的P值，补偿算法P值同样较大）。发现了小车过弯时过度切内缘行驶后，保守起见调低了P值，发现S弯有一半无法完成，十字环开始切外缘行驶。经过试错、排查，未能得出原因。又在过程中尝试了从另一侧发车，结果发现小车对于赛道的拟合性极差，开始怀疑代码对称性问题，迅速发现了bug。基于单侧道路的优秀表现，并没有直接将bug修复，而是抱着尝试的心理，将公式改为：

```
err=(A*(L-R)+B*(LM-RM))*LIMIT/(A*(L+R)+B*abs(LM-RM));
                                                    （公式4）
```

使用新的公式后，经实践得新的公式比原有公式所需P值更小（约原4/5），对于弯道的敏感度更高，能适应更快的速度，之后进行了数学方面的证明，得到的结论也确实如此（详细证明见附件B）。

经过了更多的调试，结合数学论证，得出了其相对于与其类似公式：

```
err=(L-R)*LIMIT/(L+R) +p*(LM-RM);                    （公式5）
```

的优势，同时也得出可以对中间电感的比重进行调节，进一步增加其弯道拟合效果，经改良，最终式如下：

```
err=(A*(L-R)+B*(LM-RM))*LIMIT/(A*(L+R)+C*abs(LM-RM));
                                                    （公式6）
```

由于其不再是单纯的差比和算法，根据其独特数学结构，将其命名为差比和差算法。其优势在于可以根据左、右、电感的值对左中、右中电感相对值产生的输出误差进行动态调节，但在这优势建立的同时，每次更换赛道，都要重新调节比重参数，否则将难以达到最高速度；考虑到电磁入环判断对于赛道的要求也同样比较苛刻，采用了调节信号源来适应小车的方案。

实现代码：

差比和差算法配合补偿算法，将进一步提升小车的负反馈强度，使小车紧切内环行驶。

（4） 基于电感的P、D方向控制

算法基本与电机PID原理相同。由于I值的作用一定程度上被小车惯性替代，故I值并不是很重要，不加以使用。方向PD的P值同样有着使舵机（伺服器）方向趋于目标方向的作用，而D值则是拟合方向改变的趋势，使舵机打角平滑。若P或D值过大，则会产生频繁过调导致舵机抖动严重；若P值不够，则会产生行驶路径切外环的现象；若D值不够，则会容易出现调节迟钝，具体体现在入弯切外（经典算法常见问题，实际上，差比和差算法使得这一现象难以出现），出环时更容易撞入环路肩，过十字时容易因为车位不正导致短暂失控撞路肩等。

其基本实现代码如下（采用了十个中断周期的误差输出均值作为前次误差以保证输出平滑）：

```
out=err*P+(err-lerr_ave)*D;
lerr[9]=lerr[8];
…
lerr[1]=lerr[0];
lerr_ave=(lerr[0]+…+lerr[9])/10;
```

5.5.5转向舵机的 PID 控制算法

对于舵机的闭环控制，我们采用了位置式 PID 控制算法，根据往届的技术资料和实际测试，经过反复测试，我们选择的 PID 调节策略是：

（1）将积分项系数置零，我们发现相比稳定性和精确性，舵机在这种随动系统中对动态响应性能的要求更高。更重要的是，在 KI 置零的情况下，我们通过合理调节 Kp，发现车能够在直线高速行驶时仍能保持车身非常稳定，没有振荡，基本没有必要使用 KI 参数；

（2）微分项系数 KD 使用定值，原因是舵机在一般赛道中都需要较好的动态响应能力；

（3）对Kp，我们使用了在程序中具体代码如下：

```
K_p=(|(e^(-|error| )- 1)/(e^(-|error| )+ 1)|/2+ 0.5)*bas_kp
5
```

其中，error是中心位置与中心值的偏差,bas_kp为基准Kp

图5.21 中点偏差和动态Kp值的函数曲线

经不断调试，最终我们选择了一组PD参数，得到了较为理想的转向控制效果。

(3) 对 Kp，我们使用了在程序中具体代码如下：↵

↵

$$K_p = \left(\left|\frac{e^{-|error|}-1}{e^{-|error|}+1}\right|/2 + 0.5\right) * bas\_kp$$

↵

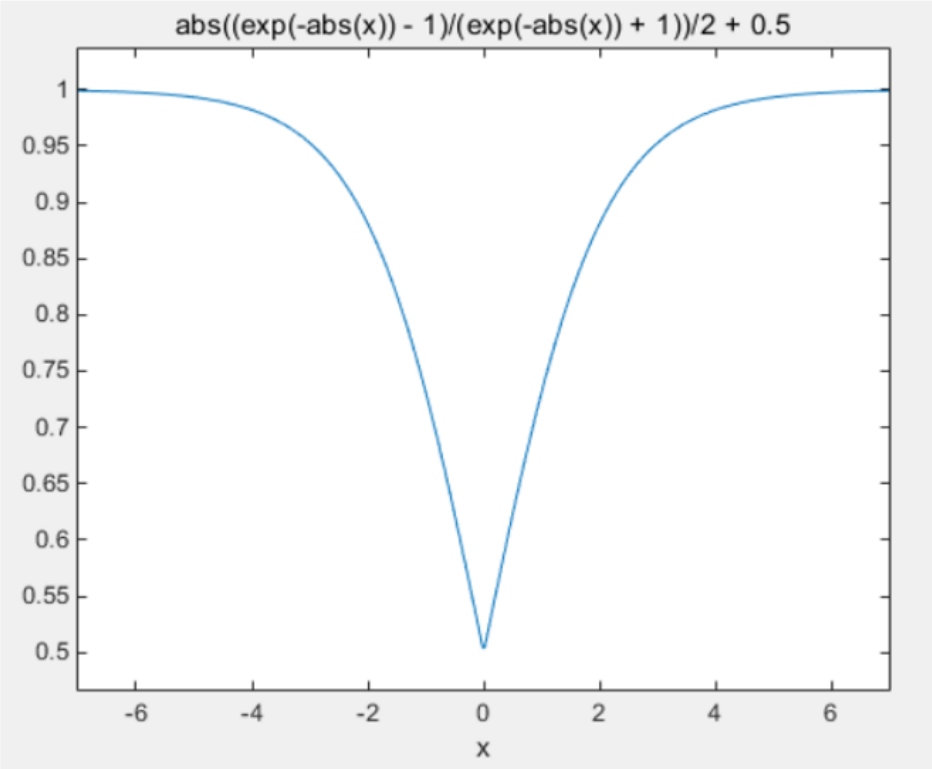
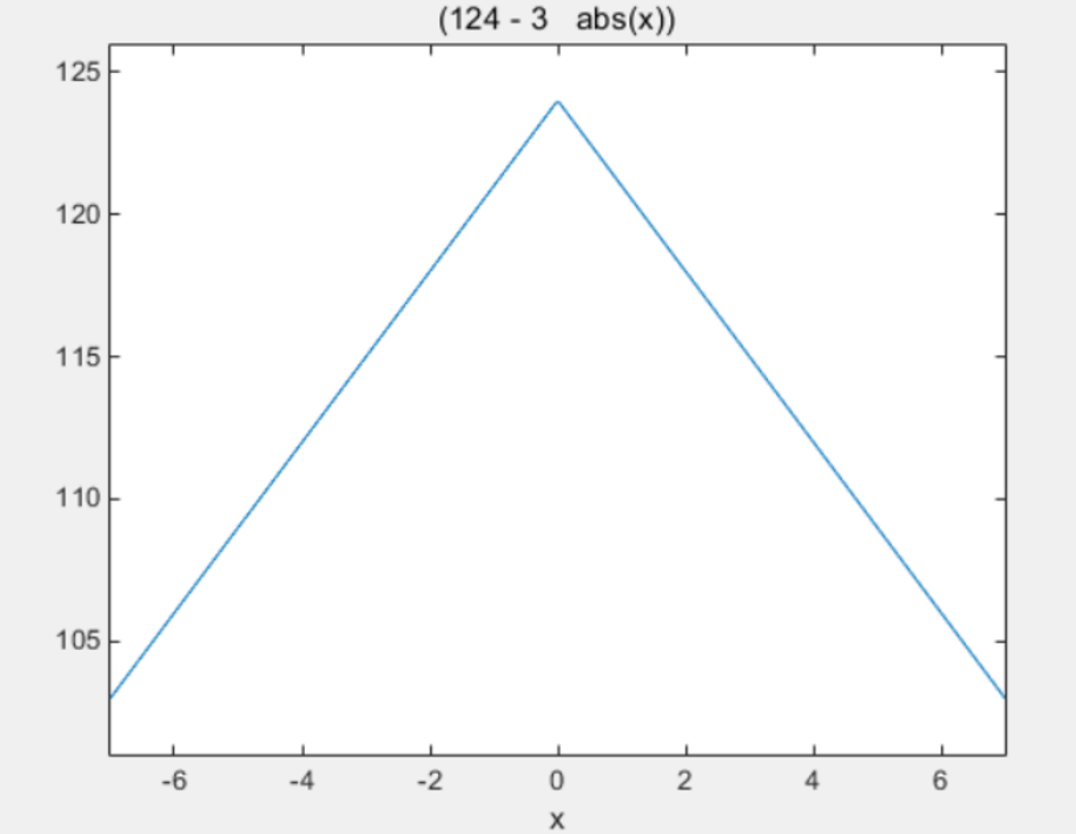


图 5.21 中点偏差和动态 Kp 值的函数曲线↵

↵

4.5 驱动电机的PI控制算法

对于速度控制，我们采用了增量式PI控制算法，基本思想是直道加速，弯道减速。经过反复调试，将每场图像得到的黑线位置与速度PI参考速度值构成二次曲线关系。在实际测试中，我们发现小车直道和弯道相互过渡时加减速比较灵敏，与舵机转向控制配合得较好。



在程序中具体代码如下：

SetSpeed = (124 - 3 * error) * bas_speed (公式4.5)

但是，该方法存在一定的局限。一方面是车在从弯道入直道时加速和从直道入弯道时减速达不到最好的控制效果，直道入弯道减速不够快速，弯道出直道加速的时机不够及时。因此我们做了进一步的改进，加速的曲线与减速的曲线分开形成滞回的效果，结果表明，控制效果更好。另一方面是没有考虑到实际比赛中长直道急速冲刺的情况，赛前在程序中人为设定直线速度不够灵活不够合理，所以我们在程序中根据图像识别长直线提高了直线速度HighestSpeed，使车能够在长直道上充分发挥潜能。

其中AD1~AD4分别是左边的一字电感、左边的竖直电感、右边的竖直电感和右边的一字电感的数据，K和Q分别是一字电感和竖直电感的比例系数，后续可以更改这两个系数的值使其适应不同的赛道元素。

$$error = \frac{K*(\sqrt{AD1}-\sqrt{AD4})+Q*(\sqrt{AD2}-\sqrt{AD3})}{AD1+AD2+AD3+AD4}$$

适应自己车的 阿卡曼

v = v + v(avg) * 0.006 * (duty_ - 650) * E （差速系数）(0 - 1)之间

```
a=0.0072*DuoJi_duty;
Speed_L=SetSpeed*(1+(0.32*tan(a)));
Speed_R=SetSpeed*(1-(0.32*tan(a)));
```