PID离散化及实现

2025年3月12日 16:30

【1】PID离散化

- 连续形式PID：

$$out(t) = K_p * error(t) + K_i * \int_0^t error(t)dt + K_d * \frac{derror(t)}{dt}$$

- 离散形式PID：

$$out(k) = K_p * error(k) + K_i * T \sum_{j=0}^{k} error(j) + K_d * \frac{error(k) - error(k-1)}{T}$$

- 若将T并入K_i和K_d，则：

$$out(k) = K_p * error(k) + K_i * \sum_{j=0}^{k} error(j) + K_d * (error(k) - error(k-1))$$

位置式PID与增量式PID公式

- 位置式PID：

$$out(k) = K_p * error(k) + K_i * \sum_{j=0}^{k} error(j) + K_d * (error(k) - error(k-1))$$

- 当$k = k-1$时：

$$out(k-1) = K_p * error(k-1) + K_i * \sum_{j=0}^{k-1} error(j) + K_d * (error(k-1) - error(k-2))$$

- 两式相减，得到增量式PID：

$$\begin{aligned}\Delta out(k) = {} & K_p * (error(k) - error(k-1)) \\ & + K_i * error(k) \\ & + K_d * (error(k) - 2error(k-1) + error(k-2))\end{aligned}$$

而位置式PID，输出限幅和积分限幅得分开进行

【2】PID在函数中的实现

PID程序实现

- 确定一个调控周期T，每隔时间T，程序执行一次PID调控

- $out(k) = K_p * error(k) + K_i * \sum_{j=0}^{k} error(j) + K_d * (error(k) - error(k-1))$

```
#include "stm32f10x.h"                  // Device header
#include "Delay.h"
#include "Timer.h"

/*定义变量*/
float Target, Actual, Out;              //目标值，实际值，输出值
float Kp = 值, Ki = 值, Kd = 值;         //比例项，积分项，微分项的权重
float Error0, Error1, ErrorInt;         //本次误差，上次误差，误差积分

int main(void)
{
    Timer_Init();

    while (1)
    {
```

```
void TIM2_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM2, TIM_IT_Update) == SET)
    {
        /*每隔时间T，程序执行到这里一次*/

        /**********执行PID调控**********/
        /*获取实际值*/
        Actual = 读取传感器();

        /*获取本次误差和上次误差*/
        Error1 = Error0;
        Error0 = Target - Actual;

        /*误差积分（累加）*/
```

- $out(k) = K_p * error(k) + K_i * \sum_{j=0}^{k} error(j) + K_d * (error(k) - error(k-1))$

```c
#include "stm32f10x.h"                  // Device header
#include "Delay.h"
#include "Timer.h"

/*定义变量*/
float Target, Actual, Out;          //目标值，实际值，输出值
float Kp = 值, Ki = 值, Kd = 值;     //比例项，积分项，微分项的权重
float Error0, Error1, ErrorInt;     //本次误差，上次误差，误差积分

int main(void)
{
    Timer_Init();

    while (1)
    {
        /*用户在此处根据需求写入PID控制器的目标值*/
        Target = 用户指定的一个值;
    }
}
```

```c
void TIM2_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM2, TIM_IT_Update) == SET)
    {
        /*每隔时间T，程序执行到这里一次*/

        /**********执行PID调控**********/
        /*获取实际值*/
        Actual = 读取传感器();

        /*获取本次误差和上次误差*/
        Error1 = Error0;
        Error0 = Target - Actual;

        /*误差积分（累加）*/
        ErrorInt += Error0;

        /*PID计算*/
        Out = Kp * Error0 + Ki * ErrorInt + Kd * (Error0 - Error1);

        /*输出限幅*/
        if (Out > 上限) {Out = 上限;}
        if (Out < 下限) {Out = 下限;}

        /*执行控制*/
        输出至被控对象(Out);
        /******************************/

        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    }
}
```

- $\Delta out(k) = K_p * (error(k) - error(k-1)) + K_i * error(k) + K_d * (error(k) - 2error(k-1) + error(k-2))$

```c
#include "stm32f10x.h"                  // Device header
#include "Delay.h"
#include "Timer.h"

/*定义变量*/
float Target, Actual, Out;          //目标值，实际值，输出值
float Kp = 值, Ki = 值, Kd = 值;     //比例项，积分项，微分项的权重
float Error0, Error1, Error2;       //本次误差，上次误差，上上次误差

int main(void)
{
    Timer_Init();

    while (1)
    {
        /*用户在此处根据需求写入PID控制器的目标值*/
        Target = 用户指定的一个值;
    }
}
```

```c
void TIM2_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM2, TIM_IT_Update) == SET)
    {
        /*每隔时间T，程序执行到这里一次*/

        /**********执行PID调控**********/
        /*获取实际值*/
        Actual = 读取传感器();

        /*获取本次误差、上次误差和上上次误差*/
        Error2 = Error1;
        Error1 = Error0;
        Error0 = Target - Actual;

        /*PID计算*/
        Out += Kp * (Error0 - Error1) + Ki * Error0
             + Kd * (Error0 - 2 * Error1 + Error2);

        /*输出限幅*/
        if (Out > 上限) {Out = 上限;}
        if (Out < 下限) {Out = 下限;}

        /*执行控制*/
        输出至被控对象(Out);
        /******************************/

        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    }
}
```

【3】平衡小车PID调参实例

```c
#include "pid.h"

float Vertical_Kp,Vertical_Kd;
float Velocity_Kp,Velocity_Ki;
uint8_t stop;

//直立环PD控制器
//输入：期望角度、真实角度、角速度
int Vertical(float Med,float Angle,float gyro_Y)
{
  int temp;
  temp=Vertical_Kp*(Angle-Med)+Vertical_Kd*gyro_Y;
  return temp;
}

//速度环PI控制器
//输入：期望速度、左编码器、右编码器
int Velocity(int Target,int encoder_L,int encoder_R)
{
  static int Err_LowOut_last,Encoder_S;
  static float a=0.7;
  int Err,Err_LowOut,temp;
  //1、计算偏差值
  Err=(encoder_L+encoder_R)-Target;
  //2、低通滤波
  Err_LowOut=(1-a)*Err+a*Err_LowOut_last;
  Err_LowOut_last=Err_LowOut;
  //3、积分
  Encoder_S+=Err_LowOut;
  //4、积分限幅(-20000~20000)
  Encoder_S=Encoder_S>20000?20000:(Encoder_S<(-20000)?(-20000):Encoder_S);
  if(stop==1)Encoder_S=0,stop=0;
  //5、速度环计算
  temp=Velocity_Kp*Err_LowOut+Velocity_Ki*Encoder_S;
  return temp;
}

//转向环PI控制器
//输入：期望速度、左编码器、右编码器
```

```c
//转向环PD控制器
//输入：角速度、角度值
int Turn(float gyro_Z,int Target_turn)
{
  int temp;
  temp=Turn_Kp*Target_turn+Turn_Kd*gyro_Z;
  return temp;
}
```

PID调参

```c
//闭环控制中间变量
int Vertical_out,Velocity_out,Turn_out,Target_Speed,Target_turn,MOTO1,MOTO2;
float Med_Angle=0;//平衡时角度值偏移量（机械中值）
//参数
float Vertical_Kp,Vertical_Kd;     //直立环 数量级（Kp：0~1000、Kd：0~10）
float Velocity_Kp,Velocity_Ki;     //速度环
float Turn_Kp,Turn_Kd;             //转向环
```

超调之后系数乘0.6

```c
//闭环控制中间变量
int Vertical_out,Velocity_out,Turn_out,Target_Speed,Target_turn,MOTO1,MOTO2;
float Med_Angle=7.6;//平衡时角度值偏移量（机械中值）
//参数
float Vertical_Kp=300,Vertical_Kd=3.4;     //直立环 数量级（Kp：0~1000、Kd：0~10）
float Velocity_Kp=0,Velocity_Ki=0;         //速度环 数量级（Kp：0~1）
float Turn_Kp,Turn_Kd;                     //转向环

uint8_t stop;

extern TIM_HandleTypeDef htim4;
```

所以我们接下来要给它乘一个系数0.6

（直立环）kp：大幅机械机械振荡；kd：小幅高频机械振荡

（速度环）主要是看平衡的时候，不要让电机产生太大的反向力矩

（转向环）kp不重要，主要是控制的小车转向的速度，kd是比较重要，控制小车不转向的时候走直线