基于stm32四轮小车简易PID控制

发布于2022-09-01 11:37:40 阅读 130

大家好，又见面了，我是你们的朋友全栈君。

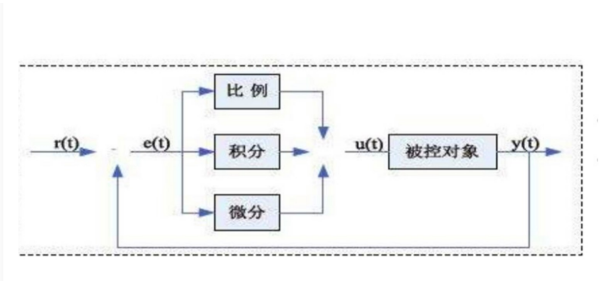
看前需知：作者本人使用的是四个普通的TT电机加编码器+增量式PID,适合PID初学者，但是需要对PID和增量式PID有一定的认知，本篇未有详细介绍，以代码应用为主，大佬勿喷。

文章目录

云中何曾落羽，踏遍三岛寻声

一、粗谈PID

PID在生活中很常见，举个例子。例如生活中，一个加热器需要对某个物体进行恒温控制，但是由于某种原因导致温度过高或者过低，这时候传感器会返回相应的数据，告诉控制器应当作出相应的调整，降温或者是加热，这就完成了一个简单的PID的闭环控制。PID就是修正公式里面的三个系数简称。
增量式PID控制将当前时刻的控制量和上一时刻的控制量做差，以差值为新的控制量，是一种递推式的算法。



二、使用的硬件设备

1、stm32f103rct6
2、某宝TT电机（小黄电机）+霍尔编码器
3、两块l2980驱动模块
4、四个18650电池供电

三、软件设计

1、四个电机分别使用IO口：PB8-PB9、PB10-PB11 、
PB12-PB13 、PB14-PB15。
2、PWM使用高级定时器TIM8的CH1-CH4，所用到的IO口：PC6 PC7 PC8 PC9。
3、四个编码器对应四个定时器TIM2 TIM3 TIM4 TIM5。开启自带的编码器模式，需同时开启对应的CH1和CH2。（对应的引脚可查看数据手册）
4、用TIM1进行中断计时。
注：TIM2需要完全重映射，因为未重映射时与TIM5的CH1，CH2引脚相同。
GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable , ENABLE); 这个禁用也不可忘了！

四、关键代码

1.TIM1定时器：

```
void Timer_Init(void)
{


TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);

 TIM_TimeBaseStructure.TIM_Period = 9999;//自动重新装载寄存器周期的值澹ㄛ计数值澹）
 TIM_TimeBaseStructure.TIM_Prescaler = 719;//时钟分频系数
 TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;//对外部时钟进行采样的时钟分频
 TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;//向上计数
 TIM_TimeBaseStructure.TIM_RepetitionCounter=0; //高级定时器1是用定时器功能配置这个才可以是正常的计数频率一开始的72mhz 值得注意的地方
    TIM_TimeBaseInit(TIM1,&TIM_TimeBaseStructure);//参数初始化

    TIM_ClearFlag(TIM1, TIM_FLAG_Update);
    TIM_ITConfig(TIM1, TIM_IT_Update, ENABLE);

    TIM_Cmd(TIM1, ENABLE);//启动定时器

}
```

2.TIM2编码器模式示例：

```
void Encoder_Init_TIM2(void)
{

 TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
  TIM_ICInitTypeDef TIM_ICInitStructure;
  GPIO_InitTypeDef GPIO_InitStructure;
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);//使能定时器2的时钟
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO, ENABLE);

  GPIO_PinRemapConfig(GPIO_FullRemap_TIM2, ENABLE);
  GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable , ENABLE);

```

作者介绍

全栈程序员站长
CTO
关注 专栏

文章	阅读量	获赞	作者排名
54.9K	8M	119.3K	1

目录

```
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;  //端口配置
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING; //浮空输入
  GPIO_Init(GPIOA, &GPIO_InitStructure);                          //根据设定参数初始化GPIOA


  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3; //端口配置
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING; //浮空输入
  GPIO_Init(GPIOB, &GPIO_InitStructure);                          //根据设定参数初始化GPIOA



  TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
  TIM_TimeBaseStructure.TIM_Prescaler = 0x0; // 预分频器
  TIM_TimeBaseStructure.TIM_Period = ENCODER_TIM_PERIOD; //设定计数器自动重装值
  TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;//选择时钟分频:不分频
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;TIM向上计数
  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
  TIM_EncoderInterfaceConfig(TIM2, TIM_EncoderMode_TI12, TIM_ICPolarity_Rising, TIM_ICPolarity_Rising);//使用编码器模式3
  TIM_ICStructInit(&TIM_ICInitStructure);
  TIM_ICInitStructure.TIM_ICFilter = 10;
  TIM_ICInit(TIM2, &TIM_ICInitStructure);
  TIM_ClearFlag(TIM2, TIM_FLAG_Update);//清除TIM的更新标志位
  TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
  //Reset counter
  TIM_SetCounter(TIM2,0);
  TIM_Cmd(TIM2, ENABLE);
}
```

3.电机初始化：

```
void dji_Init(void)
{


 GPIO_InitTypeDef  GPIO_InitStructure;
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);    //使能PB端口时钟
GPIO_InitStructure.GPIO_Pin =GPIO_Pin_11|GPIO_Pin_10|GPIO_Pin_9|GPIO_Pin_8;
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;         //推挽输出
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;        //IO口速度为50MHz
 GPIO_Init(GPIOB, &GPIO_InitStructure);                   //根据设定参数初始化GPIOB

}
```

4.TIM8PWM输出：

```
void PWM_Init(u16 arr,u16 psc)

{

GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    TIM_OCInitTypeDef  TIM_OCInitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM8, ENABLE);//
     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC , ENABLE);  //使能GPIO外设时钟使能

// GPIO_PinRemapConfig(GPIO_FullRemap_TIM3, ENABLE); //全映射 PC6-9

   //设置该引脚为复用输出功能,输出TIM8 CH1的PWM脉冲波形
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7|GPIO_Pin_8|GPIO_Pin_9; //TIM_CH1
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;  //复用推挽输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);


    TIM_TimeBaseStructure.TIM_Period = arr; //设置在下一个更新事件装入活动的自动重装载寄存器周期的值
    TIM_TimeBaseStructure.TIM_Prescaler =psc; //设置用来作为TIMx时钟频率除数的预分频值 不分频
    TIM_TimeBaseStructure.TIM_ClockDivision = 0; //设置时钟分割:TDTS = Tck_tim
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  //TIM向上计数模式
    TIM_TimeBaseInit(TIM8, &TIM_TimeBaseStructure); //根据TIM_TimeBaseInitStruct中指定的参数初始化TIMx的时间基数单位


    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1; //选择定时器模式:TIM脉冲宽度调制模式2
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //比较输出使能
    TIM_OCInitStructure.TIM_Pulse = 0; //设置待装入捕获比较寄存器的脉冲值
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High; //输出极性:TIM输出比较极性高


TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Reset ;

TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Disable;

//TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCNIdleState_Reset;

    TIM_OC1Init(TIM8, &TIM_OCInitStructure);  //根据TIM_OCInitStruct中指定的参数初始化外设TIMx
    TIM_OC2Init(TIM8, &TIM_OCInitStructure);
    TIM_OC3Init(TIM8, &TIM_OCInitStructure);
      TIM_OC4Init(TIM8, &TIM_OCInitStructure);
    TIM_CtrlPWMOutputs(TIM8,ENABLE);    //MOE 主输出使能

    TIM_OC1PreloadConfig(TIM8, TIM_OCPreload_Enable);  //CH1预装载使能
    TIM_OC2PreloadConfig(TIM8, TIM_OCPreload_Enable);
        TIM_OC3PreloadConfig(TIM8, TIM_OCPreload_Enable);
        TIM_OC4PreloadConfig(TIM8, TIM_OCPreload_Enable);
    TIM_ARRPreloadConfig(TIM8, ENABLE); //使能TIMx在ARR上的预装载寄存器

    TIM_Cmd(TIM8, ENABLE);  //使能TIM8

}
```

5.PID：

```
static double   Proportion=0.45;                                //比例常数 Proportional Const
static double   Integral=0.1;                                  //积分常数 Integral Const
static double   Derivative=0;                                  //b不采用微分

/*******************增量式PID控制设计**********************************/
//NowPoint当前输出值
//SetPoint设定值
int PID_Calc1(int NowPoint,int SetPoint)
{

                            //微分常数 Derivative Const
    static int      LastError1;                                 //Error[-1]
```

```
  static int      PrevError1;                             //Error[-2]
  int iError,Outpid;                                      //当前误差

  iError=SetPoint-NowPoint;                               //增量计算
  Outpid=(Proportion * iError)                          //E[k]项
            -(Integral * LastError1)      //E[k-1]项
            +(Derivative * PrevError1);   //E[k-2]项

  PrevError1=LastError1;                     //存储误差，用于下次计算
  LastError1=iError;
  return(Outpid);                                         //返回增量值
}

int PID_Calc2(int NowPoint,int SetPoint)
{

                          //微分常数 Derivative Const
    static int      LastError2;                           //Error[-1]
    static int      PrevError2;                           //Error[-2]
  int iError,Outpid;                                      //当前误差

  iError=SetPoint-NowPoint;                               //增量计算
  Outpid=(Proportion * iError)                          //E[k]项
            -(Integral * LastError2)      //E[k-1]项
            +(Derivative * PrevError2);   //E[k-2]项

  PrevError2=LastError2;                     //存储误差，用于下次计算
  LastError2=iError;
  return(Outpid);                                         //返回增量值
}


int PID_Calc3(int NowPoint,int SetPoint)
{

                          //微分常数 Derivative Const
    static int      LastError3;                           //Error[-1]
    static int      PrevError3;                           //Error[-2]
  int iError,Outpid;                                      //当前误差

  iError=SetPoint-NowPoint;                               //增量计算
  Outpid=(Proportion * iError)                          //E[k]项
            -(Integral * LastError3)      //E[k-1]项
            +(Derivative * PrevError3);   //E[k-2]项

  PrevError3=LastError3;                     //存储误差，用于下次计算
  LastError3=iError;
  return(Outpid);                                         //返回增量值
}


int PID_Calc4(int NowPoint,int SetPoint)
{

                          //微分常数 Derivative Const
    static int      LastError4;                           //Error[-1]
    static int      PrevError4;                           //Error[-2]
  int iError,Outpid;                                      //当前误差

  iError=SetPoint-NowPoint;                               //增量计算
  Outpid=(Proportion * iError)                          //E[k]项
            -(Integral * LastError4)      //E[k-1]项
            +(Derivative * PrevError4);   //E[k-2]项

  PrevError4=LastError4;                     //存储误差，用于下次计算
  LastError4=iError;
  return(Outpid);                                         //返回增量值
}
```

6.中断服务函数：

```
void dj1_Init(void)
{


int Encoder_Front_Left,Encoder_Front_Right,Encoder_Back_Right,Encoder_Back_Left;
int Left_t,Right_t,Encoder_R,Encoder_L;
int Moto_Front_Left,Moto_Front_Right,Moto_Back_Left,Moto_Back_Right;
int para1,para2,para3,para4;//增量
int SetPoint1=30;//设置目标值单位RPM
int SetPoint2=30;
//使用减速比是1:120的减速箱
#define SetPoint_back SetPoint1*6240/600//换算成编码器速度，因为最终pid控制的是编码器的脉冲数量
#define SetPoint_front SetPoint2*6240/600//换算成编码器速度，因为最终pid控制的是编码器的脉冲数量

//Time1定时器1中断服务函数
//200ms定时
void TIM1_UP_IRQHandler(void)
{

    if(TIM_GetFlagStatus(TIM1, TIM_IT_Update) != RESET)   //时间到了
    {

        TIM_ClearITPendingBit(TIM1, TIM_FLAG_Update);//清中断

     Encoder_Front_Left=myabs(Read_Encoder(2));  //读取编码器
     Encoder_Front_Right=myabs(Read_Encoder(3));
     Encoder_Back_Left=myabs(Read_Encoder(4));
     Encoder_Back_Right=myabs(Read_Encoder(5));

      para1=PID_Calc1(Encoder_Front_Left,SetPoint_back);  //左电机，计数得到增量式PID的增量数值
       para2=PID_Calc2(Encoder_Front_Right,SetPoint_back);
       para3=PID_Calc3(Encoder_Back_Left,SetPoint_front);
       para4=PID_Calc4(Encoder_Back_Right,SetPoint_front);




           if((para1<-3)||(para1>3)) // 不做 PID 调整，避免误差较小时频繁调节引起震荡。
           {

      Moto_Front_Left +=para1;
           }
                 if(Moto_Front_Left>3500) Moto_Front_Left=3500;//限幅
               TIM8->CCR1=Moto_Front_Left;//更新pwm

           if((para2<-3)||(para2>3)) // 不做 PID 调整，避免误差较小时频繁调节引起震荡。
           {

      Moto_Front_Right +=para2;
           }
                 if(Moto_Front_Right>3500) Moto_Front_Right=3500;//限幅
               TIM8->CCR2=Moto_Front_Right;


    ///
```

```
            if((para3<-3)||(para3>3)) // 不做 PID 调整，避免误差较小时频繁调节引起震荡。
            {

        Moto_Back_Left +=para3;
            }
                    if(Moto_Back_Left>3500) Moto_Back_Left=3500;//限幅
                TIM8->CCR3=Moto_Back_Left;//更新pwm

            if((para4<-3)||(para4>3)) // 不做 PID 调整，避免误差较小时频繁调节引起震荡。
            {

        Moto_Back_Right +=para4;
            }
                    if(Moto_Back_Right>3500) Moto_Back_Right=3500;//限幅
                TIM8->CCR4=Moto_Back_Right;//更新pwm
            delay_ms(2);

         }


}

int myabs(int a)
{

 int temp;
   if(a<0)
    temp=-a;
   else
    temp=a;
   return temp;
}
```

总结

由于本人为新手，代码可能显得冗长，本文关于速度的计算设置就不写了，可以结合其他大佬博主的文章尝试理解，需要代码的可以评论区留言+一键三连（嘿嘿嘿）。书写不易，感谢支持，大家一起进步！

https HTTP Java 网络安全

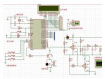
相关文章

基于stm32的智能小车(远程控制、避障、循迹)

学完stm32，总是想做点东西"大显身手"一下，智能小车就成了首选项目，其核心只是就是PWM输出，I/O口引脚电平判断。

全栈程序员站长

多个完整设计提供参考包括单片机、8086、STM32制作教程和资料-转…



在网上收集了接近上千个完整设计的单片机、8086、STM32制作教程和资料-转发分享（涵盖了大部分的毕设课设题目），学习单片机的最好教程，也可以作为帮助大家在做...

嵌入式互联网

直立智能车（平衡车）毕业论文[通俗易懂]

随着微电子控制技术的发展和人们对出行工具的日益增长的需求，一款简单易操作、容易携带、清洁无污染的两轮自平衡车开始走进大众的视野，但这种小型代步工具仍可能存在一定...

全栈程序员站长

做一台STM32小车需要学些什么?

我觉得这个问题挺有意思的， 都是拉电话线的专业，都是在学习STM32，都对小车感兴趣，那就让我这个通信老狗来谈谈自己的见解~

Mculover666

基于51单片机智能小车的设计与实现转弯避障_基于单片机的智能小车设计

学习智能小车系统，有助于提高搭建系统的能力和对自动控制技术的理解。智能小车是一个较为完整的智能化系统，而智能化的研究已成为我国追赶世界科技水平的重要任务。智能小...

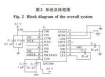
全栈程序员站长

智能小车资料源码大全下载_清翔智能小车资料

今天给大家分享一下智能小车的资料，包括制作流程、原理图设计和源码等，不下于60辆智能小车的制作经验。其中历届智能小车的开发资料就有90个文件了。

全栈程序员站长

c语言智能车跑道检测程序,基于金属检测的智能循迹小车设计



摘 要： 为解决当前循迹小车存在性能稳定性差的问题，提出一种基于金属检测的智能循迹小车设计方法。采用LDC1000设计一种金属循迹智能小车，介绍系统总体设计框架...

全栈程序员站长

树莓派小车C语言循迹,自动循迹小车_单片机/STM32/树莓派/Arduino/开发板创意项目-聚丰…

金属探测LDC1314是根据电磁感应原理制成的，将一金属置于变化的磁场当中时，根据电磁感应原理就会在金属内部产生涡流，涡流产生的磁场反过来又影响原磁场，这种变化...

全栈程序员站长

倒立摆起摆控制_旋转倒立摆原理

近期在学习简易旋转倒立摆装置，倒立摆其实是一个十分经典的自动控制模型，不过开始学习了解结构和原理还是花了很多时间，在思路以及调试过程中遇到了很多困难。 我认为...

全栈程序员站长

图文详解PID调参

在工程中，如果我们要用单片机做一个温控系统，其系统组成一般如下：一个采集温度的ADC，一个输出温度的加热头以及一个用于运行控制算法的单片机，如果我们要维持温...

全栈程序员站长

计算机控制技术期中测试素材2020版

小车安装有测距传感器，与障碍物距离的数值通过四位LED七段数码管显示，具备基本测距显示和避障行驶功能：

zhangrelay

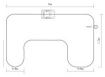
地心一号-超迷你自平衡小车（完）

先说arduino的来源吧。他是意大利的一名教师和一名晶体工程师发明的一个灵活性非常高的电子平台，主要是因为当时的学生跟他抱怨找不到便宜好用的微控制器。

MCU起航

2016年四川省TI杯电子设计竞赛B题

B题：自动循迹小车 1．任务 设计制作一个自动循迹小车。小车采用一片 TI公司LDC1314或LDC1000电感数字转换器作为循迹传感器，在规定的平面跑...



全栈程序员站长

基于keil5新建STM32F10x寄存器版本工程

前面文章分享了很多关于STM32F103系列知识点、物联网相关的小项目，工程都采用的是寄存器方式编写；很多小伙伴接触STM32开始都采用库函数编程，不清楚如何使...

DS小龙哥

stm32电机控制之控制两路直流电机！看完你会了吗

小车使用的电机是12v供电的直流电机，带编码器反馈，这样就可以采用闭环速度控制，这里电机使用PWM驱动，速度控制框图如下：

用户6754675

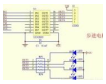
基于STM32设计的遥控小车(手机APP+GPS+温湿度+ESP8266)

手机APP: 采用QT设计，程序支持跨平台编译运行(Android、IOS、Windows、Linux都可以编译运行，对应平台上QT的环境搭建，之前博客已经发...

DS小龙哥

动手智能小车记(5)-坦克底盘硬件模块大杂烩

关于小车，之前也写了好几期了，先给大家放一段视频，后面打算将机械臂集成到坦克上，目前还在调试中，夹具已经能够控制夹紧和松开了：



杨源鑫

简易旋转倒立摆及控制系统实现方案_旋转倒立摆与pid控制

完整文档和源码：https://github.com/Kevincoooool/inverted-pendulum + 2017年成都信息工程大学 第六届...

全栈程序员站长

更多文章 ›