自动控制

2023/4/24

电子系统导论教学团队

实验目的

- 了解反馈控制的基本原理
- 掌握PID控制的基本方法

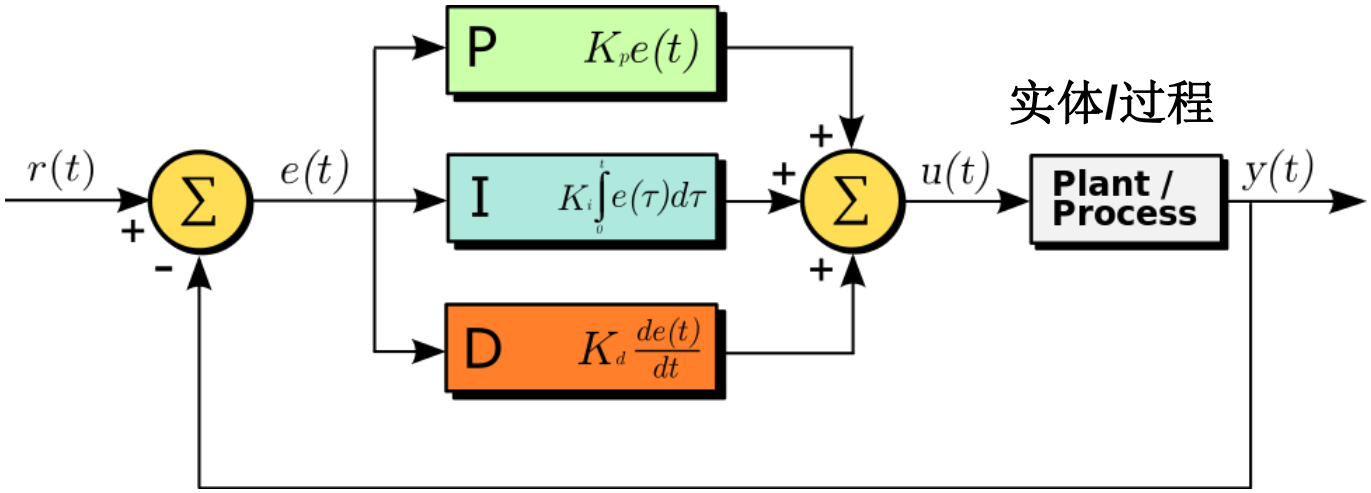
自动（反馈）控制

- 自动控制（automatic control）是指在没有人直接参与的情况下，利用外加的设备或装置，使机器、设备或生产过程的某个工作状态或参数自动地按照预定的规律运行。当今的闭环自动控制技术都是基于反馈的概念以减少不确定性。反馈理论的要素包括三个部分：测量、比较和执行。测量关键的是被控变量的实际值，与期望值相比较，用这个偏差来纠正系统的响应，执行调节控制。
- 自动化控制技术的广泛应用开始于欧洲的工业革命时期。瓦特在发明蒸汽机的同时，应用反馈原理，于1788年发明了离心式调速器。当负载或蒸汽量供给发生变化时，离心式调速器能够自动调节进气阀的开度，从而控制蒸汽机的转速。
- 常见例子：空调温控，车速控制

PID控制

- 在工程实际中，应用最为广泛的调节器控制规律为比例(proportion)、积分(integral)、微分(derivative)控制，简称PID控制，又称PID调节。

PID控制原理



- *r(t)*是所需的设定值；
  *y(t)*是测量的过程输出值；
  *e(t)*即*偏差是设定值与过程值的差（r(t)-y(t)）；*
  *u(t)是*经过计算得到的控制变量，输入到控制器中。
- P : proportion（非负比例系数），就是偏差乘以一个常数。
  I : integral（非负积分系数），就是对偏差进行积分运算。
  D : derivative（非负微分系数），对偏差进行微分运算。

PID控制原理

- 以控制小车速度为例
  - 例如将小车速度设定值$r(t)$为3m/s，由码盘测得速度为3.2m/s，即过程输出值$y(t)$；偏差$e(t)$为-0.2m/s。
  - P：将-0.2m/s乘以一个系数（正）输入到控制器中，以减小输出的占空比，则车轮转速将降低，向设定值靠近。K_P越大则调节的灵敏度越大，但过大可能会使实际速度低于3m/s（超调）。

PID控制原理

- 以控制小车速度为例
  - I：只经过比例调节的小车，可能稳定后的速度为3.1m/s，存在稳态误差-0.1m/s；虽然误差很小，但是因为积分项也会随着时间的增加而加大，它推动控制器的输出增大，从而使稳态误差进一步减小，直到等于0。

PID控制原理

- 以控制小车速度为例
  - D：小车中有些组件存在较大惯性或者滞后性，其变化总是落后于误差的变化。假设经比例调节后实际速度为3.1m/s，则设定速度与实际速度的差值由-0.2m/s变为-0.1m/s，$e(t)$的差分为0.1m/s^2，将此差分乘以系数（正）加到控制变量中，相比只有比例环节减缓了速度降低的趋势（减小超调量）。

PID数学表达

- 整体控制功能可表达为

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(t')\,dt' + K_\mathrm{d} \frac{de(t)}{dt}$$

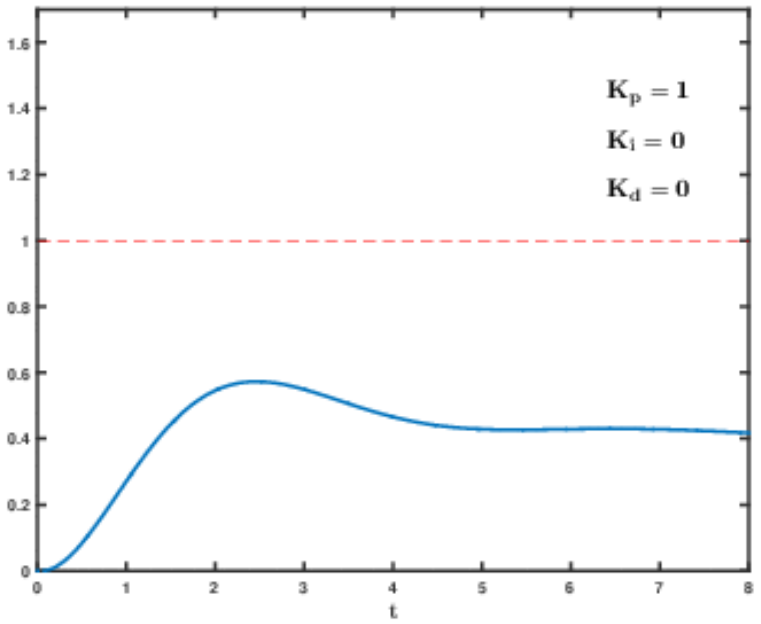
  - K_p、K_i、K_d都是非负的，分别表示比例项，积分项和微分项的系数

PID参数整定

- PID控制的难点不是编程，而是控制器的参数整定，以下只介绍手动调参
  - 独立增加参数的影响

调整方式	上升时间	超调量	安定时间	稳态误差	稳定性
↑ K_p	减少 ↓	增加 ↑	小幅增加 ↗	减少 ↓	变差 ↓
↑ K_i	小幅减少 ↘	增加↑	增加 ↑	大幅减少↓↓	变差↓
↑ K_d	小幅减少 ↘	减少↓	减少↓	变动不大→	变好 ↑

  - 输入*r(t)*为阶跃响应，不同参数下的输出*y(t)*响应



PID参数整定

- 调参秘诀：
  - 先调比例系数，再调微分系数，有需要再加积分系数。

- K_p、K_i、K_d对PID系统输出影响
  - 增大比例系数使系统反应灵敏，调节速度加快，并且可以减小稳态误差。但是比例系数过大会使超调量增大，振荡次数增加，调节时间加长，动态性能变坏，比例系数太大甚至会使闭环系统不稳定；
  - 增大微分系数可以减小超调量和稳定时间。
  - 增大积分系数会减小稳态误差，但会增大超调量和稳定时间；

离散型PID

- 在计算机上进行PID调节时只能用离散型PID。
- 假设采样间隔为T，则在第kT时刻：
  - 偏差 $e(k)=r(k)-y(k)$;
  - 积分环节用加和的形式表示，即 $e(k)+e(k\text{-}1)+\cdots$
  - 差分环节用斜率的形式表示，即 $e(k)$ - $e(k\text{-}1)$
  - 从而有位置式PID ：

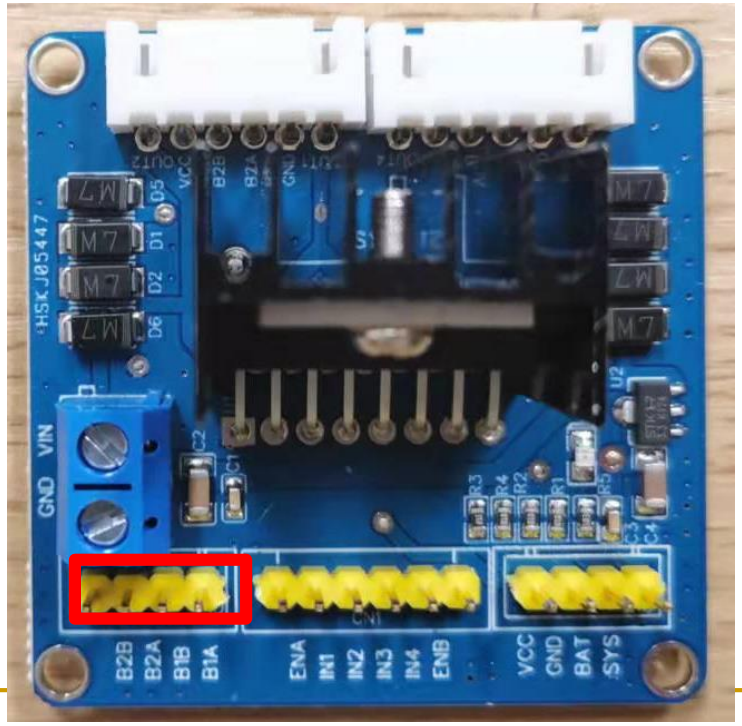
$$u(k)=K_P e(k)+K_I\sum_{j=1}^{k}e(j)+K_D(e(k)-e(k-1))$$

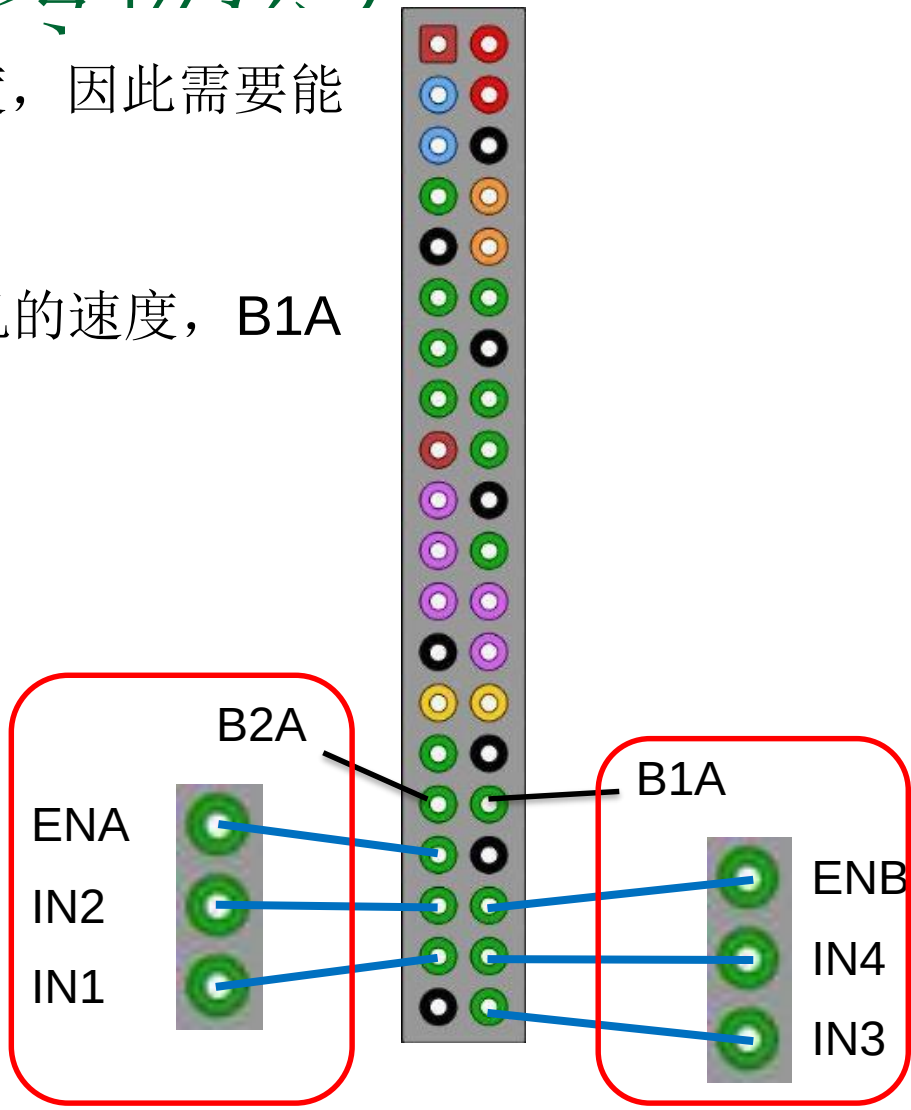
  - 由两次的$u(k)$相减可以得到增量式PID:

$$\Delta u(k)=u(k)-u(k-1)=K_P(e(k)-e(k-1))+K_I e(k)+K_D(e(k)-2e(k-1)+e(k-2))$$

电机连接（一种参考接法）

- 本实验需要霍尔编码器测电机速度，因此需要能够控制电机。
- 电机驱动板的连接与第9节相同。
- GPIO连接效果如图，B2A测A电机的速度，B1A测B电机的速度





PID控制双轮小车直行(Python实现)

```
class PID:
    """PID Controller
    """

    def __init__(self, P=2.0, I=0.0, D=1.0, speed=2.8, duty=30):

        self.Kp = P
        self.Ki = I
        self.Kd = D
        self.err_pre = 0
        self.err_last = 0
        self.u = 0
        self.integral = 0
        self.ideal_speed = speed
        self.last_duty = duty
        self.pre_duty = duty
```

- 定义一个PID类，其中实例变量Kp、Ki、Kd分别为PID三个参数，err_pre、err_last分别为当前和上一次理想转速和实际转速的差值，u为PID的输出，integral为累计偏差量，ideal_speed为理想转速，last_duty和pre_duty分别为上一次和当前占空比(%)。
- 通过__init__方法初始化实例变量，创建PID类实例时可以传入各实例变量的参数。

PID控制双轮小车直行(Python实现)

```
def update(self,feedback_value):
    self.err_pre = self.ideal_speed - feedback_value
    self.integral+= self.err_pre
    self.u = self.Kp*self.err_pre + self.Ki*self.integral + self.Kd*(self.err_pre-self.err_last)
    self.err_last = self.err_pre
    self.pre_duty = self.last_duty + self.u
    if self.pre_duty > 100:
        self.pre_duty = 100
    elif self.pre_duty < 0:
        self.pre_duty = 0
    self.last_duty = self.pre_duty
    return self.pre_duty
```

- feedback_value为实际测得的转速，由此更新err_pre以及integral。
- 根据离散形式PID公式（位置型），计算每一次调节的输出u。
- 更新err_last 、pre_duty和last_duty，要注意占空比范围是0-100(%)。

PID控制双轮小车直行(Python实现)

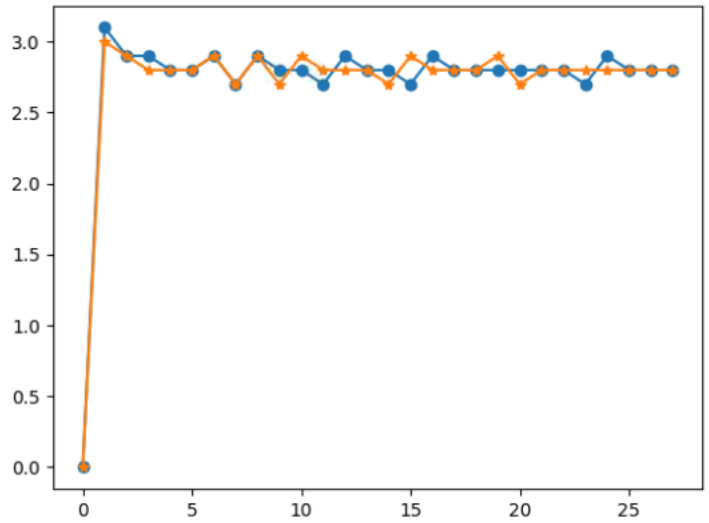
```
speed = 1.9
l_origin_duty = 5
r_origin_duty = 5
pwma.start(l_origin_duty)
pwmb.start(r_origin_duty)
L_control = PID(30,0.06,20,speed,l_origin_duty)
R_control = PID(40,0.01,23,speed,r_origin_duty)

try:
    while True:
        pwma.ChangeDutyCycle(L_control.update(lspeed))
        pwmb.ChangeDutyCycle(R_control.update(rspeed))
        x.append([i])
        y1.append(lspeed)
        y2.append(rspeed)
        time.sleep(0.1)
        i+=  0.1
        print ('left: %f  right: %f lduty: %f rduty: %f'%(lspeed,rspeed,L_control.pre_duty,R_control.pre_duty))

except KeyboardInterrupt:
    pass
```

- speed为设定的理想转速，l_origin_duty和r_origin_duty分别为左右轮初始占空比。
- L_control和R_control分别为控制左右轮的PID类实例，可以传入PID参数。
- 时间间隔设置为0.1s。

  (详细代码见pid_control.py)

PID控制双轮小车直行(Python实现)

- 绘制转速(1/s)-时间(s)图像



- 横轴是时间轴（间隔一秒）纵轴是转速（$r(t)$），蓝色圆点代表左轮，黄色星代表右轮。根据绘制图像以及前述PID参数整定方法调整参数。
- 上图输入为阶跃信号，输出为阶跃响应

附：增量式PID控制的Python实现示例

```
#   e(k) = target - feedback(k)
#   U(k) = Kp*e(k) + Ki*Integral(0, k, e(k)) + kd*(e(k)-e(k-1))
#   delta_U(k) = U(k) - U(k-1)
#                        = Kp*(e(k)-e(k-1)) + Ki*e(k) + Kd*(e(k)-2*e(k-1)+e(k-2))
class PID(object):
    def __init__(self, pid_params, target, init_u, u_range):
        self.__Kp = pid_params[0]
        self.__Ki = pid_params[1]
        self.__Kd = pid_params[2]
        self.__target = target
        self.__u = init_u
        self.__u_min = u_range[0]
        self.__u_max = u_range[1]

        self.__e = self.__target
        self.__e_1 = self.__target
        self.__e_2 = self.__target

    def update(self, feedback):
        self.__e_2 = self.__e_1
        self.__e_1 = self.__e
        self.__e = self.__target - feedback
        du = self.__Kp*(self.__e-self.__e_1)+self.__Ki*self.__e+self.__Kd*(self.__e-2*self.__e_1+self.__e_2)
        self.__u += du
        if self.__u > self.__u_max:
            self.__u = self.__u_max
        if self.__u < self.__u_min:
            self.__u = self.__u_min
        return self.__u
```

(详细代码见inc_pid.py)

实验内容

- 通过PID控制，使得小车可以自主直线行走。

实验报告中需要回答的问题

1. P、I、D的各自作用是什么？
2. 如果发现小车走不直，如何确定问题和优化？

致谢

- 本课件由以下同学协助编写
  - 郝晓昱（14307130017）
  - 卢文龙（16307130105）