6 有刷直流电机速度/位置 PID 闭环控制

6.1 PID 控制原理

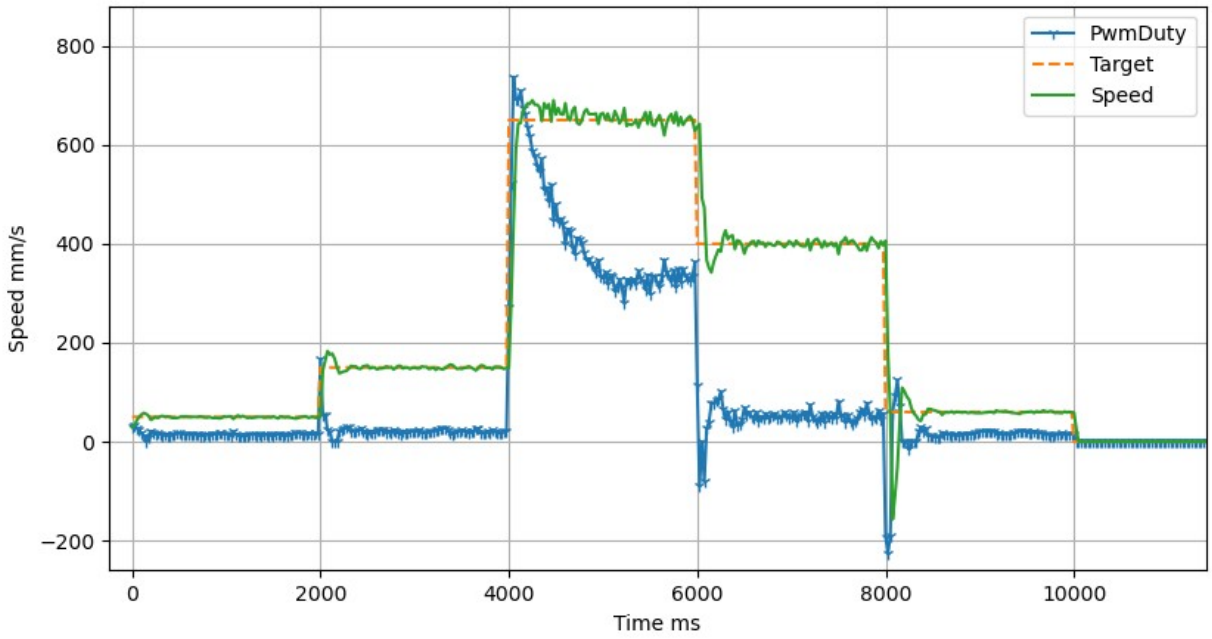
6.2 直流电机 PID 速度控制

6.3 PID 位置控制

直流电机电流/速度/位置 PID 控制总结

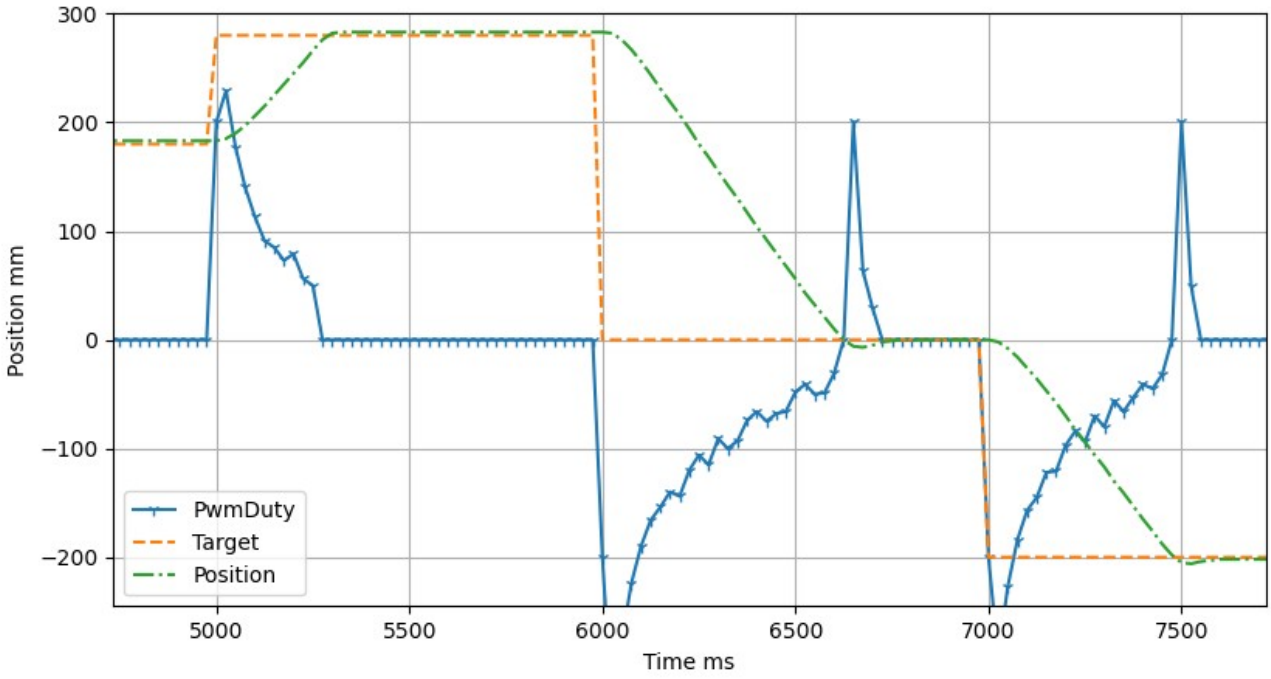
这节我们看一下直流电机速度和位置闭环 PID 控制的实际效果。

先看一下文档里的数据曲线，这幅图是速度 PID 控制曲线，横坐标是时间，单位豪秒，纵坐标是速度或者 PWM 占空比，速度单位是 mm/S，就是按 52mm 直径车轮折算出的小车速度。占空比数值 0－1000 对应 0－100%。



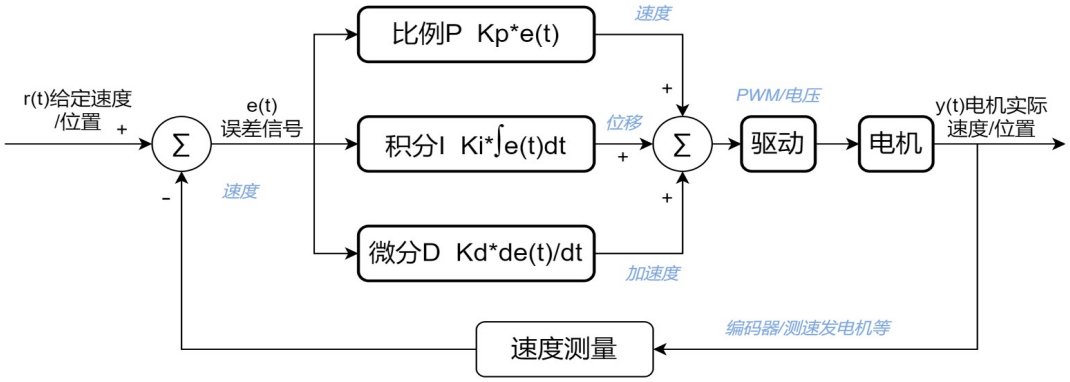
数据曲线有三条，蓝色是 PWM 占空比，橙色是目标速度值，绿色是编码器测出的实际电机速度。这里是车轮悬空状态的电机测试数据，最高速度 650mm/s，就是 0.65 米每秒。整体来看这里速度控制的效果还算可以。

再看一下位置控制的数据， 横坐标还是时间，橙色是目标位置值。在第 5 秒时目标位置从 180 变为 280，蓝色的 PWM 占空比立即增加，这时电机也开始转动，绿色的位置曲线开始向目标值靠拢。从这里 3 处到达目标值的数据看，位置有大约 3－5mm 的超调。



关于 PID 控制的原理和 microPython 程序代码，我们在下一节讲解。谢谢观看。

这节我们看一下 PID 控制的原理，被控对象建模，并总结 PID 优缺点。



被控系统模型建立，理论分析 或 实测辨识

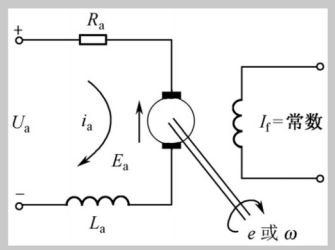


图 2-6　电枢控制的直流电动机系统

【例 2-4】 试列写如图 2-6 所示的电枢控制的直流电动机的微分方程。

解（1）确定输入量是电枢电压 U_a，输出量是电动机角速度 ω，负载转矩 M_L 是扰动输入。

（2）忽略电枢反应、磁滞等影响，激磁电流 I_f 为常数，则激磁磁通视为不变，变量关系为线性关系。

（3）根据基尔霍夫定律写出电枢回路方程式为

$$L_a\frac{di_a}{dt}+R_a i_a+E_a=U_a \tag{2-20}$$

式中：L_a 为电枢回路总电感；R_a 为电枢回路总电阻。

（4）列写中间变量辅助方程

由于激磁磁通不变，电枢反电势 E_a 与转速成正比，即

$$E_a=k_e\omega \tag{2-21}$$

式中：k_e 为电势系数（伏/弧度/秒），由电动机结构参数确定。

电机轴上机械运动方程为

$$M_D-M_L=J\frac{d\omega}{dt} \tag{2-22}$$

式中：$J=\frac{GD^2}{4g}$ 为转动惯量（计算到电动机轴上，单位为千克·米·秒^2），GD^2 为飞轮转矩（千克·米^2），M_L 为负载转矩（千克·米），M_D 为电动机转矩（千克·米）。

电磁转矩方程可写为

$$M_D=k_m i_a \tag{2-23}$$

式中：k_m 是转矩系数，由电动机结构参数确定。

（5）将式（2-20）～式（2-23）联立求解，得

$$\frac{L_aJ}{k_mk_e}\frac{d^2\omega}{dt^2}+\frac{R_aJ}{k_ek_m}\frac{d\omega}{dt}+\omega=\frac{1}{k_e}U_a-\frac{R_a}{k_ek_m}M_L-\frac{L_a}{k_ek_m}\frac{dM_L}{dt} \tag{2-24}$$

若不考虑电动机的负载转矩，即设 $M_L=0$，则式（2-24）可简化为

$$\frac{L_aJ}{k_ek_m}\cdot\frac{d^2\omega}{dt^2}+\frac{R_aJ}{k_ek_m}\cdot\frac{d\omega}{dt}+\omega=\frac{1}{k_e}U_a \tag{2-25}$$

令 $T_a=\frac{L_a}{R_a}$（单位为秒）为电磁时间常数，$T_m=\frac{JR_a}{k_ek_m}$（单位为秒）为电动机的机电时间常数，则式（2-25）可写为

$$T_aT_m\frac{d^2\omega}{dt^2}+T_m\frac{d\omega}{dt}+\omega=\frac{1}{k_e}U_a$$

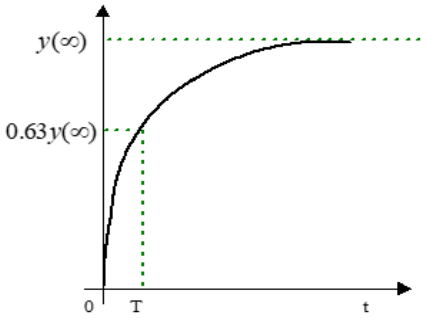
理论分析结果说明直流电机速度控制系统是二阶系统。

但是，我们这里的小功率空心杯直流电机，通过实测给定的 PWM 脉宽值和电机速度，系统实际近似为一阶惯性系统。

例：若预先确定系统结构为一阶惯性环节

$$G(s)=\frac{K}{Ts+1}$$

通过阶跃响应求取其中具体的参数。



解：$\frac{Y(s)}{U(s)}=\frac{K}{Ts+1}$

$$Ty'(t)+y(t)=Ku(t)=K$$

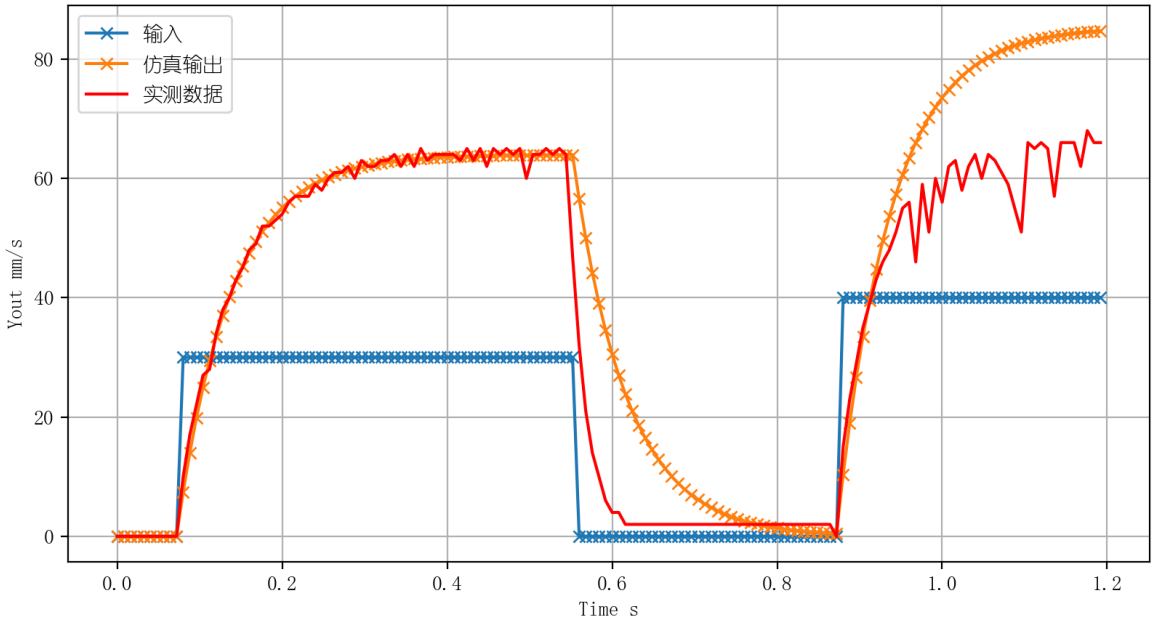
$$y'(t)+\frac{1}{T}y(t)=\frac{K}{T} \qquad \text{所以} \quad y(t)=K(1-e^{-\frac{t}{T}})$$

$$t\to\infty \quad K=y(\infty) \qquad t=T \quad y(T)=y(\infty)(1-e^{-1})=0.63y(\infty)$$

常用方法：近似法、半对数法、切线法、两点法；面积法。

低阶 无噪声　　阶数较高 无噪声或噪声较小

下图给出了在相同输入（PWM 脉宽）下，被控直流电机辨识模型理论仿真输出与实际直流电机的输出速度曲线。能够看到在 0—0.5 秒，理论输出与实测速度非常一致，0.5 秒之后数据相差很大。这说明实际的控制对象具有很强的非线性，很难甚至是不可能建立被控对象的精确动态数学模型。



PID 实际应用需要考虑的各种因素：

输入输出参数的测量误差与限制条件；

是否允许给定信号测试；

是否允许超调，输出量是否需要限幅；

是否限制系统输出变化率；

是否有测量等参数滞后等。

MicroPython PID 直流电机控制代码简单讲解：

（注： 此处代码还不完善，也未整理，仅作为讲解演示）

```
class PID:
    posPrev = 0

    def __init__(self, M, kp=1, kd=0, ki=0, umaxIn=800, eprev=0,
                       max_run_ms=2500, tolerance=10,
                       loop_maxms = 500, loop_minms=25,
                       max_interItem=800, dbg_info=0, ditem_toler = 5):
        self.kp = kp
        self.kd = kd
        self.ki = ki
        self.eintegral = 0.0
        self.I_Item = 0.0
        self.P_Item = 0.0
        self.D_Item = 0.0
        self.maxI_Item = max_interItem
        self.tolerance = tolerance   #
        self.ditem_toler = ditem_toler
        self.M = M                  # Motor object
        self.umaxIn  =umaxIn
        self.eprev = eprev
        self.max_pid_run_ms = max_run_ms
        self.ms_stamp = ticks_ms()
        self.out_time_flags = 0
        self.loop_stamp_ms = ticks_ms()
        self.loop_T_maxms = loop_maxms
        self.loop_T_minms = loop_minms
        self.loop_pre_pos = 0  #self.M.pos
        self.target = 0
        self.realvalue = 0
        self.pid_start_ms_stamp = ticks_ms()
        self.print_cnt = 0
        self.ctl_cnt = 0
        self.pid_type = 'init'
        self.dbg_info = dbg_info

    def release_pid_motor(self):
        self.M.release_motor()

    def deinit_pid_motor_pwm(self):
        self.M.deinit_motor_pwm_pin()
```

```
def evalu(self,value, target, deltaT):     # 计算 PID 控制量
    e = target-value       # 误差信号

    #if abs(e) <= abs(self.tolerance):    #
    #    return 0

    # Derivative
    dedt = (e-self.eprev)/(deltaT)    # 计算微分值

    self.P_Item = self.kp*e     #  P  比例项

    # Integral
    self.ctl_cnt = self.ctl_cnt + 1

    self.eintegral = self.eintegral + e*deltaT
    #print('eintegral',self.eintegral)
    self.I_Item = self.ki * self.eintegral  #  I  积分项
    #print('I_Item',self.I_Item)
    if self.I_Item > self.maxI_Item:       #  限制积分项幅值，以减小超调
        self.I_Item = self.maxI_Item
    elif self.I_Item < -self.maxI_Item:
        self.I_Item = -self.maxI_Item

    if abs(dedt) <= abs(self.ditem_toler):
        dedt = 0
    self.D_Item = self.kd*dedt    #  I  微分项
    # Control signal
    u = self.P_Item + self.I_Item + self.D_Item

    # Direction and power of  the control signal
    if u > 0:
        if u > self.umaxIn:
            u = self.umaxIn
    else:
        if u < -self.umaxIn:
            u = -self.umaxIn

    self.eprev = e
    return u

def setSpeedForPID(self, speed_mmps=0):
    self.pid_start_ms_stamp = ticks_ms()
    self.out_time_flags = 0
    self.target = speed_mmps
```

```
        if self.pid_type != 'speed':
            self.loop_stamp_ms = ticks_ms()
            self.pid_type = 'speed'

    def setPositionForPID(self, posi_mm=0):
        self.pid_start_ms_stamp = ticks_ms()
        self.out_time_flags = 0
        self.target = posi_mm
        if self.pid_type != 'position':
            self.loop_stamp_ms = ticks_ms()
            self.pid_type = 'position'

    def speedPIDloop(self):
        if self.pid_type != 'speed':
            return

        loop_ms = ticks_ms()
        # 采样周期 时间单位 ms
        if loop_ms - self.loop_stamp_ms < self.loop_T_minms:
            return

        # 电机堵转造成超时
        if loop_ms - self.pid_start_ms_stamp > self.max_pid_run_ms:
            if self.out_time_flags <= 1:
                self.M.breakStop()     # out of time, stop motor
                #print('pid run out of time.')
                self.out_time_flags = self.out_time_flags + 1
            return
        if self.out_time_flags > 0:
            return

        deltaT_ms = loop_ms - self.loop_stamp_ms
        self.loop_stamp_ms = loop_ms

        state = disable_irq()
        enc_pos = self.M.pos
        per_us = self.M.enc_period_us
        enc_pre_us = self.M.stamp_us
        enable_irq(state)

        if self.target == 0:
            if per_us > 100:
                self.realvalue = self.M.direction*self.M.k_mmps/per_us
            if enc_pos == self.loop_pre_pos:
```

```
                self.realvalue = 0
            self.loop_pre_pos = enc_pos
            self.print_pid_info()
            self.M.breakStop()

        elif (self.loop_pre_pos != enc_pos)and(per_us>100):
            #self.realvalue = 1000000/per_us/self.M.encoder_circle_pulse   # speed in rps
            # 编码器脉冲时间宽度测量法(T 法)计算电机速度
            self.realvalue = self.M.direction*self.M.k_mmps/per_us    # speed in mmps
            self.loop_pre_pos = enc_pos
            x = int(self.evalu(self.realvalue, self.target, 1))
            self.M.setPWMduty(x)
            self.print_pid_info()

        # 速度为 0 时，没有编码器脉冲，按 PID 控制量启动电机
        elif (loop_ms-self.M.interrupt_stamp_ms) >= self.loop_T_maxms:
            self.realvalue = 0    # speed in mmps
            self.loop_pre_pos = enc_pos
            x = int(self.evalu(self.realvalue, self.target, 1))
            self.M.setPWMduty(x)
            self.print_pid_info()
        return


def print_pid_info(self):
    if self.dbg_info == 1:
        print('U:', self.M.curpwm, 'T:',int(self.target), 'V:', int(self.realvalue))


def positionPIDloop(self):
    loop_ms = ticks_ms()
    if loop_ms - self.loop_stamp_ms < self.loop_T_minms:
        return

    if self.pid_type != 'position':
        return

    if loop_ms - self.pid_start_ms_stamp > self.max_pid_run_ms:
        if self.out_time_flags <= 1:
            self.M.setPWMduty(0)     # out of time, stop motor
            #print('pid run out of time.')
            self.out_time_flags = self.out_time_flags + 1
        return
    if self.out_time_flags > 0:
```

```
        return

deltaT_ms = loop_ms - self.loop_stamp_ms
self.loop_stamp_ms = loop_ms

state = disable_irq()
enc_pos = self.M.pos
per_us = self.M.enc_period_us
enc_pre_us = self.M.stamp_us
enable_irq(state)

if (self.loop_pre_pos != enc_pos)and(per_us>100):
        self.realvalue = enc_pos*self.M.k_pos_mm
        self.loop_pre_pos = enc_pos

        if (self.target - self.realvalue) > abs(self.tolerance):
                # Call for control signal
                if self.eintegral < 0:
                        self.eintegral = 0
                if self.I_Item < 0:
                        self.I_Item = 0
                x = int(self.evalu(self.realvalue, self.target, 1))
                self.M.setPWMduty(x)
                self.print_pid_info()
        elif (self.realvalue - self.target) > abs(self.tolerance):
                if self.eintegral > 0:
                        self.eintegral = 0
                if self.I_Item > 0:
                        self.I_Item = 0
                x = int(self.evalu(self.realvalue, self.target, 1))
                self.M.setPWMduty(x)
                self.print_pid_info()
        else:
                #
                x = 0
                self.eintegral = 0
                self.I_Item = 0
                self.M.breakStop()
                self.print_pid_info()
else:
        self.realvalue = enc_pos*self.M.k_pos_mm
        self.loop_pre_pos = enc_pos
        if abs(self.realvalue-self.target) > abs(self.tolerance):
```

```
                x = int(self.evalu(self.realvalue, self.target, 1))
                self.M.setPWMduty(x)
                self.print_pid_info()
            else:
                # 小于位置允许误差后清 0 积分项，开启刹车停止电机
                x = 0
                self.eintegral = 0
                self.I_Item = 0
                self.M.breakStop()
                self.print_pid_info()
        return
```

PID 控制算法优：点：

1、不需要准确的被控对象参数模型，应用广泛；

2、算法易于理解，编程实现简单；

3、调试参数较少，易达到满足要求的控制效果；

缺点：

1、不合适的控制参数会造成系统不稳定；控制系统首先必要需求就是稳定性，满足稳定性后才能考虑准确度和快速响应等性能权衡取舍。

2、最优控制效果难以评定；

3、控制参数相互影响，调试过程比较繁锁，需要一定的调试经验；