

Part1 PID_control算法

参考

1.Datawhale
ai-hardware-robotics/Q2-机器人基础和控制、手眼协调/PID_Control.md at main · datawhalechina/ai-hardware-robotics

2. b站
通俗易懂的 PID 控制算法讲解 哔哩哔哩 bilibili

目录

A. 5个维度梳理PID概念

B. 简单情景模拟

C. 后续真实场景实现可以用到的参考

A. 5个维度：

机器学习数据分析算法考虑的5个维度：**机制 | 特征、模型 | 参数、阈值**（这里我感觉有一定的关联，就“不太严谨的”套用一下这个框架）

1. 机制：

开环与闭环（Open loop vs Closed loop）

1. Open loop（开环控制）
- “过程视角”：仅依据预设指令进行控制，不关注输出或环境变化。
 - 实现简单，但在面对扰动或不确定性时，系统鲁棒性较差。
2. Closed loop（闭环控制 / 反馈控制）
- “状态视角”**：系统将输出结果或传感器感知的反馈作为input的一部分，用以调整行为。
 - 输出反过来影响输入，使系统具备**适应环境变化**的能力，因此具有更强的鲁棒性。
 - 特别是在一些**变量难以建模或无法精确量化的场景**中，使用输出（或误差）作为输入进行反馈控制，是一种极为有效的策略。
 - 由于闭环系统依赖实时反馈，感知器在系统中的作用尤为关键，直接影响控制效果。
3. Open loop + Closed loop
- 结合开环与闭环控制策略，兼顾响应速度与适应能力，是复杂系统中常见的做法。

PD——如何控制使误差减小且可以达到稳态；I——内模原理

- 见**建模**：高亮部分

2. 模型：

PID控制器通过以下三个基本组件来计算控制输入：

- 比例（P）：直接对误差进行比例放大，可以迅速减少误差，但可能导致系统不稳定。
- 积分（I）：对误差进行积分，可以消除系统的静态误差，但可能导致系统响应变慢。
- 微分（D）：对误差的导数进行预测，可以预测误差的变化趋势，从而提前进行调整，提高系统的响应速度和稳定性。

- e(t) 是当前误差，即期望输出与实际输出之间的差值。
- K_p 是比例系数。
- K_i 是积分系数。
- K_d 是微分系数。

- 实现步骤
-


计算误差：计算当前输出与期望输出之间的误差。 比例项：根据比例系数和误差计算比例项。 积分项：对误差进行积分，并根据积分系数计算积分项。 微分项：计算误差的导数，并根据微分系数计算微分项。 计算控制输入：将比例项、积分项和微分项相加，得到控制输入。 应用控制输入：将控制输入应用于被控系统。

$$u(t) = K_p \cdot e(t) + K_i \int_0^t e(\tau) d\tau + K_d \cdot \frac{de(t)}{dt}$$

整体公式：

3. 参数：

- 比例（P）：直接对**误差**进行比例放大，可以**迅速减少误差**，但可能导致**系统不稳定**。
- 微分（D）：对**误差的导数**进行预测，可以**预测误差的变化趋势**，从而**提前进行调整**，提高系统的**响应速度和稳定性**。
- 积分（I）：对误差进行**积分**，可以**消除系统的静态误差**，但可能导致**系统响应变慢**。

 PID交互式-可视化-参数的影响

<https://rossning92.github.io/pid-simulation/>

B. 情形模拟

简单情景模拟：

在重力条件下，允许空气阻力扰动，如何通过控制升力，使无人机停留在指定高度（简化：仅考虑高度）

"特征/物理量分析"

- 我们能控制的只有升力，而不是位置
- 但位置信息可以采集
- 可以发现所控制的量（升力）应该是误差量的（导数关系）——>所以如果用**力学模型控制**，那么误差量即**传感器接收的量**要找线位移或角位移？

建模：

由动力学模型：

升力 = 重力 + 空气阻力 + 惯性力（期望值，即期望最终加速度 * m）

因此，我们能控制的升力的值=重力补偿+阻力补偿+PD控制器（+I调整静态误差）

$F_L = mg + k\dot{z} + m(K_p e + K_d \dot{e}) (+I)$

==其中：第一项是重力补偿—保证能悬停不掉下来；第二项是阻力补偿—对抗空气阻力扰动；第三项是P—使升力朝误差减小的方向变化；第四项是D（e的导数）—抑制速度，避免达到稳态时速度没刹住，即避免来回震荡==
==由此，**可以通过调整升力，使误差减小且可以达到稳态***==
==但可能稳态时，位置误差将趋于一个**稳定的非零值**。由此，加入积分项 I可以迫使误差堆积，从而迫使升力变化。不断使误差由**稳定的非零值**趋于**零**。==

I的引入就是**内模原理**的体现：积分项在频域里给系统引入了一个原点极点 => 变为I型系统 => 可以跟踪阶跃输入（高度设定）无稳态误差。

C 实战准备

[ai-hardware-robotics/02-机器人基础和控制、手眼协调/PID_Control.md at main · datawhalechina/ai-hardware-robotics](#)

Baseline(位置PID):

```
class PID:
    # pid的初始化赋值
    def __init__(self, Kp, Ki, Kd, setpoint=0, sample_time=0.01):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.setpoint = setpoint
        self.sample_time = sample_time

        self.prev_error = 0
        self.integral = 0

    # pid的cal_process
    def update(self, measured_value):
        error = self.setpoint - measured_value # 计算误差
        self.integral += error * self.sample_time # 积分
        derivative = (error - self.prev_error) / self.sample_time # 微分

        output = self.Kp * error + self.Ki * self.integral + self.Kd * derivative # 计算控制输入
        self.prev_error = error # 保存误差

    return output

pid = PID(Kp=1.0, Ki=0.1, Kd=0.01, setpoint=100)
measured_value = 90 # 假设的当前测量值
control_input = pid.update(measured_value)

print(f"Control Input: {control_input}")
```

Datawhale经验分享

- 一般情况下，我们是通过波形图的稳定以及响应来判定PID算法的效果的。
- 一般情况下，我们可以先设置一个较大的比例系数，然后逐渐减小比例系数，直到系统的响应变得平滑。
- 积分项的作用是消除系统的静态误差，但是积分项的引入会使系统的响应变慢。
- 微分项的作用是预测误差的变化趋势，从而提前进行调整，提高系统的响应速度和稳定性。

调参：

参数整定找最佳，	从小到大顺序查。
先是比例后积分，	最后再把微分加。
曲线振荡很频繁，	比例度盘要放大。
曲线漂浮绕大弯，	比例度盘往小扳。
曲线偏离回复慢，	积分时间往下降。
曲线波动周期长，	积分时间再加长。
曲线振荡频率快，	先把微分降下来。

动差大来波动慢，微分时间应加长。
理想曲线两个波，前高后低四比一。
一看二调多分析，调节质量不会低。