PID ????????? MATLAB ??

PID ????????? MATLAB ??

PID 控制器的核心目標只有一個：「讓系統的『現在數值』，盡可能完美地追上你設定的『目標數值』」。

它計算輸出訊號 $u(t)$ 的數學方程式如下：$$ u(t) = K_p e(t) + K_i \int_{0}^{t} e(\tau) d\tau + K_d \frac{de(t)}{dt} $$

公式裡面的 $e(t)$ 代表誤差 **(Error)**，也就是「目標值 - 現在值」。而 K_p、K_i、K_d 就是我們要調整的三個魔法參數。

為了秒懂這個公式，我們把 PID 想像成 **「開車想要精準停在紅綠燈的停止線上」**：

---

1. ???? P?I?D ????

? P (Proportional???) —— ?????

- 原理：誤差多大，我踩油門/煞車的力道就多大。
- 開車情境：離停止線還有 100 公尺，我油門踩到底（誤差大，出力大）；離停止線剩 10 公尺，我油門放輕一點（誤差小，出力小）。
- 優點：反應最直接、最快。
- 缺點：會產生 **「穩態誤差 (Steady-state error)」** 。如果車子很重（有阻力），當你離停止線只剩 1 公分時，P 算出來的力道太小了，根本推不動車子，導致你永遠停不到那完美的 0 誤差線上。

? I (Integral???) —— ?????

- 原理：把過去到現在的所有誤差「累積（加總）」起來。只要誤差一直存在，我的出力就會越來越大。
- 開車情境：你卡在離停止線 1 公分的地方停住了（穩態誤差）。I 兄弟在旁邊計時：「卡住 1 秒、卡住 2 秒、卡住 3 秒 ...」，隨著時間累積，他會幫你慢慢把油門踩深，直到車子終於越過那 1 公分。
- 優點：完美消除「穩態誤差」，保證最終一定能碰到目標。

- 缺點：因為力量會一直累積，容易衝過頭，導致 **「超調 (Overshoot)」** （也就是車子衝過停止線，然後還要倒車回來）。

? D (Derivative???) —— ?????

- 原理：計算誤差的「變化率」。它感覺到誤差正在快速縮小，就會提早踩煞車。
- 開車情境：雖然離停止線還有 20 公尺，但你看時速表發現車速高達 100 km/h，你知道如果再不踩煞車絕對會衝過頭！所以你提早收油門踩煞車。這就是 D 的作用：阻尼（煞車）效果。
- 優點：抑制 P 和 I 造成的超調（衝過頭）現象，讓系統更快穩定下來。
- 缺點：對「雜訊 (Noise)」極度敏感。如果感測器數值跳動很大，D 會以為誤差劇烈變化，導致系統跟著發瘋狂抖。

---

2. ????? PID ???? (Tuning Guide)

工程師在現場調 PID 參數時，絕對不是亂猜，通常會遵循這個順序：

1. 先調 **K_p (P 參數)**：把 K_i 和 K_d 都設為 0。慢慢把 K_p 調大，直到系統反應夠快，但開始出現一點點震盪為止。
2. 再加 **K_i (I 參數)**：把 K_i 慢慢調大，目的是消除最後那一點點停不下來的穩態誤差。這時系統可能會衝過頭 (Overshoot)。
3. 最後加 **K_d (D 參數)**：把 K_d 稍微調大，用來壓制剛剛 K_i 造成的衝過頭現象，讓曲線變得平滑完美。

---

3. MATLAB ?? PID ????

在 MATLAB 中，你完全不需要自己刻那個複雜的微積分方程式，官方已經幫你寫好超級方便的 `pid` 函數了！

```
% 1. 設定你的 PID 參數
Kp = 1.5;
Ki = 0.5;
Kd = 0.1;

% 2. 建立一個 PID 控制器物件
C = pid(Kp, Ki, Kd);

% 3. 假設你有一個馬達的數學模型 (轉移函數 Transfer Function)
```

```
% 這裡假設一個簡單的系統 G(s) = 1 / (s^2 + 2s + 1)
numerator = 1;
denominator = [1, 2, 1];
G = tf(numerator, denominator);

% 4. 將 PID 控制器與馬達系統組合成「閉迴路 (Closed-loop)」
sys_cl = feedback(C * G, 1);

% 5. 畫出系統的「步階響應 (Step Response)」看看控制效果！
figure;
step(sys_cl);
title('PID 控制系統響應');
grid on;
```

4. ???? PID ????? (Step Response)

在畫出來的曲線上，橫軸是「時間 (秒)」，縱軸是「系統的當前數值」。完美的情況是線條瞬間從 0 垂直升到 1 然後死死貼在 1 上面，但現實世界的物理系統做不到，一定會有一個過渡期。

我們最需要盯著看的，就是以下兩個指標：

? 1. ??? (Overshoot, %OS) —— ????????

- 圖上怎麼看：觀察曲線第一次超過目標值 **(1)** 的最高峰 **(Peak)**。
- 定義：那個最高峰比起目標值多出了百分之多少。
- 實例：目標是 1，但系統一開始衝太快，最高點來到 1.25 才掉下來，那你的 Overshoot 就是 **25%**。
- 工程意義：* 如果 Overshoot 太大，代表系統太「激動」了。
  - 在某些場合這是致命的！例如：電梯如果 Overshoot 很大，乘客會撞到天花板；機械手臂如果 Overshoot 很大，會把原本要夾的零件撞飛。
- 怎麼解決：通常是 P 或 I 給太大了，可以嘗試增加 **D (**微分**)** 參數來增加阻尼（煞車），把那個山峰壓下來。

?? 2. ???? (Settling Time, T_s) —— ??????????

- 圖上怎麼看：觀察曲線上下震盪到最後，什麼時候開始「永遠」停留在目標值附近不再亂跑。
- 定義：系統進入並一直保持在「目標值的 $\pm 2%$ (或 $\pm 5%$) 誤差範圍內」所需的時間。
- 實例：系統雖然很快就衝到 1，但在 0.9 到 1.1 之間上下震盪了很久，直到第 5 秒鐘，它才穩定在 0.98 到 1.02 這個超窄的管子裡不出來，那 Settling Time 就是 **5** 秒。

- 工程意義：決定了你的機器「反應有多乾脆」。Settling Time 越短，代表機器越快準備好進行下一個動作。

---

5. MATLAB ????????????? `stepinfo`

你完全不需要自己拿尺在螢幕上量最高點或算時間！MATLAB 提供了一個超級強大的函數 `stepinfo`，只要一行程式碼，它就會直接算出所有的詳細數據。

```
% 假設你已經建立好了閉迴路系統 sys_cl
% sys_cl = feedback(C * G, 1);

% 1. 使用 stepinfo 自動計算所有特性
info = stepinfo(sys_cl);

% 2. 在指令視窗印出最重要的兩個數據
disp(['超調量 (Overshoot): ', num2str(info.Overshoot), ' %']);
disp(['穩定時間 (Settling Time): ', num2str(info.SettlingTime), ' 秒']);
```

? ??? UI ??????????

如果你不想打程式碼，當你用 `step(sys_cl)` 把圖畫出來之後：

1. 在圖片的白色背景處按滑鼠右鍵。
2. 選擇 **Characteristics (**特徵**)**。
3. 勾選 **Peak Response (**峰值響應 **/** 看 **Overshoot)** 和 **Settling Time (**穩定時間**)**。
4. 圖上就會出現兩個小黑點，滑鼠點一下黑點，就會直接浮現精確的數值標籤！

Revision #2
Created 2026-04-01 02:06:37 UTC by TaipeiTechRacing
Updated 2026-04-06 16:39:15 UTC by TaipeiTechRacing