PID Controllers

Contents map

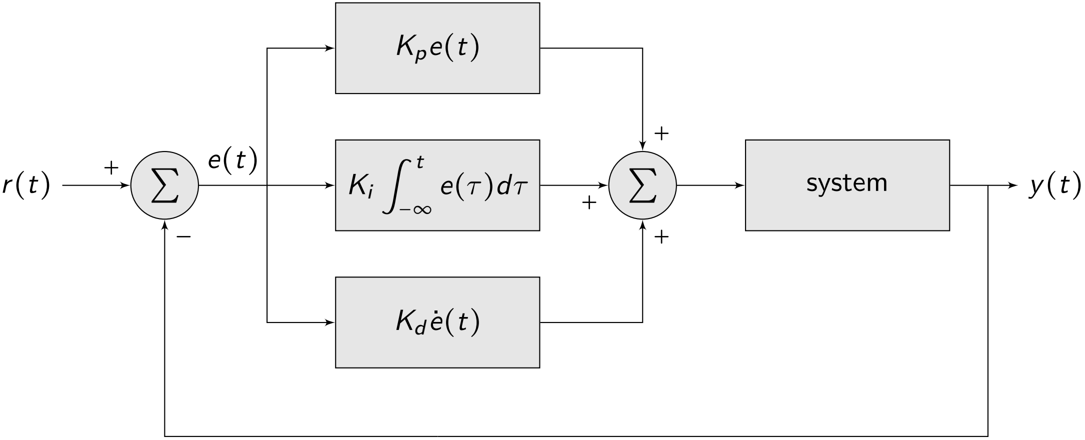
developed content units	**taxonomy levels**
empirical tuning of PID	u2, e3
pole placement with PID	u2, e3

prerequisite content units	**taxonomy levels**
transfer function	u1, e2
PID controller	u1, e2

Main ILO of sub-module "PID Controllers"

Design a PID controller to place the closed-loop poles at desired locations

Crash-slide on PIDs



implicit assumption: we can measure $y(t)$*!* (see also `https://www.youtube.com/watch?v=UR0hOmjaHp0`!)

How does changing the PID gains impact the Closed-Loop response?

K_P

$\uparrow \implies$ faster response, but may cause overshoot/oscillations

$\downarrow \implies$ slower response, reduced overshoot (but higher steady-state error)

How does changing the PID gains impact the Closed-Loop response?

K_P

$\uparrow \implies$ faster response, but may cause overshoot/oscillations

$\downarrow \implies$ slower response, reduced overshoot (but higher steady-state error)

K_I

$\uparrow \implies$ eliminates steady-state error faster, but risks instability/windup

$\downarrow \implies$ reduces oscillations but may leave residual error

How does changing the PID gains impact the Closed-Loop response?

K_P

$\uparrow \implies$ faster response, but may cause overshoot/oscillations

$\downarrow \implies$ slower response, reduced overshoot (but higher steady-state error)

K_I

$\uparrow \implies$ eliminates steady-state error faster, but risks instability/windup

$\downarrow \implies$ reduces oscillations but may leave residual error

K_D

$\uparrow \implies$ dampens oscillations, improves stability (but amplifies noise)

$\downarrow \implies$ smoother control, but slower rejection of disturbances

model free tuning

Manual Tuning (Trial and Error)

this approach works only for already stable systems!

Algorithm:

- start with all gains at zero ($K_P = 0,\ K_I = 0,\ K_D = 0$)
- increase K_P until the system oscillates
- add K_D to dampen oscillations
- introduce K_I to eliminate steady-state error
- iteratively fine-tune for desired performance

Ziegler-Nichols (Open-Loop) Method

(A step-response based tuning)

Algorithm:

- Apply a step input, and measure:
  - dead time (L), i.e., if there is a delay before the response
  - time constant (T)
- Use the Z-N table:

$$K_P = 1.2T/L \qquad T_I = 2L \qquad T_D = 0.5L$$

- Connect in closed-loop, test, and refine

Ziegler-Nichols (Closed-Loop) Method

(. . . for a more aggressive tuning)

Algorithm:

- Set $K_I = 0$, $K_D = 0$
- Increase K_P until the output shows sustained oscillations (K_u)
- Measure the oscillation period (P_u)
- Use the alternative Z-N table

$$K_P = 0.6K_u \qquad T_I = P_u/2 \qquad T_d = P_u/8$$

- Test, and refine

Other Empirical PID Tuning Methods

When no plant model is available

- **Relay Tuning (Åström-Hägglund)**: set on-off switching to estimate K_u and P_u
- **Cohen-Coon**: optimized for disturbance rejection (open-loop)
- **Tyreus-Luyben**: conservative Z-N modification for robustness
- **Software Auto-Tuning**: automated gain calculation via test signals

When Matlab definitely rules

https://www.mathworks.com/help/slcontrol/cat_scd_pid_autotuning.html

When shall I use model-free PID tuning?

When, simultaneously:

- the plant dynamics are simple
- there are no big safety risks
- a rough tuning suffices
- you need quick deployment

When shall I avoid model-free PID tuning?

If at least one of the following happens:

- the system is unstable/high-order
- doing testing means risking damaging something
- precision is critical
- you know that strong nonlinearities will be present

model based tuning (via poles placement)

Example with a first-order plant

Given: $G(s) = \dfrac{1}{s+1}$ (first-order system)

Goal: have a closed-loop pole at $s = -4$

Try: use a proportional controller: $C(s) = K_P$

Find the closed-loop TF: $\dfrac{K_P G(s)}{1 + K_P G(s)} = \dfrac{K_P}{s + 1 + K_P}$

Set the parameter accordingly: $s + (1 + K_P) = s + 4 \implies K_P = 3$

Example with a second-order plant

Given: $G(s) = \dfrac{1}{s(s+1)}$

Goal: have two closed-loop poles at $s = -2 \pm j2$ (and thus $s^2 + 4s + 8$)

Try: use a PID controller: $C(s) = K_P + \dfrac{K_I}{s} + K_D s$

Find the closed-loop TF: i.e., find the ddenominator of $1 + C(s)G(s)$ and set it so to contain the wished roots

Summarizing, poles placement $=$

- pick the desired poles based on time response specifics
- derive desired characteristic polynomial
- write the closed-loop transfer function with the PID parameters
- match the polynomials & solve for K_P, K_I, K_D

Will you always be able to place all the poles where you want?

NO!

Most important python code for this sub-module

Python Enables Symbolic Matching of PID Coefficients

`sympy`

Self-assessment material

Question 1

What is the first step in designing a PID controller using pole placement?

Potential answers:

I: Tune K_P using trial-and-error
II: Write the plant transfer function in state-space
III: Choose desired closed-loop poles based on time-domain specs
IV: Set the integral gain to zero initially

Question 2

What is the main goal of pole placement when designing a controller?

Potential answers:

I: To cancel all poles and zeros of the system
II: To achieve desired time-domain behavior such as settling time and overshoot
III: To make the transfer function purely algebraic
IV: To eliminate the need for feedback
V: I do not know

Question 3

How does the derivative term (K_D) in a PID controller primarily affect the pole placement of a system?

Potential answers:

I: It shifts the system poles toward the imaginary axis
II: It always eliminates steady-state error
III: It has no influence on the pole placement
IV: It influences the damping and stability by modifying the characteristic equation
V: I do not know

Question 4

What is the key mathematical operation used to design PID gains through pole placement?

Potential answers:

I: Taking the inverse Laplace transform of the plant
II: Eliminating zeros from the open-loop transfer function
III: Matching the closed-loop characteristic polynomial to a desired one
IV: Factorizing the numerator of the open-loop transfer function
V: I do not know

Question 5

In a first-order system controlled by a proportional gain K_P, what is the effect of increasing K_P?

Potential answers:

I: The pole moves further left on the real axis, increasing system speed

II: The pole becomes complex and causes oscillations

III: The system gain decreases and response slows down

IV: The zero of the system moves into the right-half plane

V: I do not know

Question 6

Which of the following best describes the correct order of steps for PID pole placement design?

Potential answers:

I: Compute the system output first, then choose PID gains, then set desired poles

II: Start with experimental PID gains, simulate, and refine based on intuition

III: Choose desired poles, derive the corresponding characteristic polynomial, and match it with the actual closed-loop polynomial to solve for gains

IV: Eliminate the need for poles by transforming to frequency domain

V: I do not know

Recap of sub-module "PID Controllers"

- Pole placement allows us to achieve desired dynamics
- PID gains shift the closed-loop poles
- Match desired characteristic polynomial with actual one
- Use symbolic or numerical tools to solve for K_P, K_I, K_D

?