Table of Contents I

notes

- this is the table of contents of this document; each section corresponds to a specific part of the course

PID Controllers

notes

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

notes

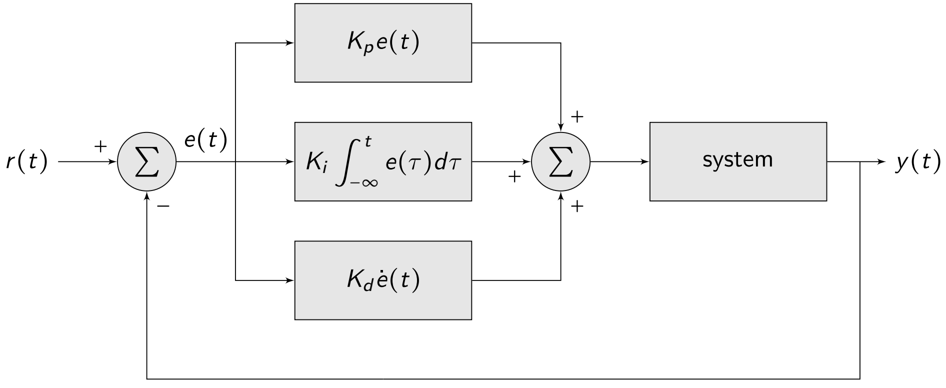
Main ILO of sub-module "PID Controllers"

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

notes

- By the end of this module you shall be able to design a PID controller that achieves desired dynamics via pole placement.

Crash-slide on PIDs



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

notes

- you will see this controller in big details in the next courses, for now let's only get some intuitions
- important thing: we need sensors and processing units, to be able to implement this. This means that we need to allocate money for buying and installing this piece of hardware - may be more expensive than open loop control

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

notes

- Trade-off: aggressive tuning ($\uparrow K_P, K_I, K_D$) speeds up response but risks instability
- For your systems, always balance performance with robustness!

model free tuning

notes

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

notes

- This requires no prior knowledge of the plant
- This is also simple but time-consuming; and you can do it only if you have stable systems

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

notes

- It seems like magic, but actually it has some theoretical foundations
- Note that this works well for first-order + delay systems
- Note also that this typically gives a conservative starting point, and in practice one may need to do some refinement

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

notes

- Risk of instability during tuning
- Best for systems where testing oscillations is safe

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

notes

- Choice depends on system safety, noise, and performance requirements

When Matlab definitely rules

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

notes

- this method implements some autotuning algorithms, essentially sending some probing signals and performing the tuning automatically. Very powerful and works well very often

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

notes

- you won't make a bad choice if:
- your plant dynamics are slow, stable, low-order systems like temperature control
- performing oscillations or step tests are not an issue
- there are no strict performance requirements
- you have no time for doing system identification
- thus, model-free methods trade off robustness for simplicity
- always validate tuned parameters in a safe environment!

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

notes

- you are going to get into troubles if:
- you have unstable or complex dynamics, as inverted pendulums, aerospace systems etc
- you are dealing with high-speed motors, chemical processes, or potentially dangerous stuff (things that achieve any form of high energy states)
- you spent a lot of money on the plant, and you want it to perform – e.g., medical devices, robotics
- if you have some hysteresis, dead zones, and strong nonlinearities in general, then you shall not do model free PIDs

model based tuning (via poles placement)

notes

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: $s + (1 + K_P) = s + 4 \implies K_P = 3$

notes

- This shows how to match closed-loop poles with desired dynamics

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

notes

- Start from desired poles get desired polynomial match with real system

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

notes

- You should now be able to apply this pseudo-algorithm to tune a PID by pole placement
- essentially you shall solve for K_P, K_I, K_D so that closed-loop denominator matches what you want

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

NO!

notes

- Note that you may not be able to place the poles where you want (i.e., the systems above to do not have solutions)
- this is an indication that the controller structure that you chose has not enough complexity to enable you to do the 'poles picking' operation

Most important python code for this sub-module

notes

Python Enables Symbolic Matching of PID Coefficients

`sympy`

notes

- Symbolically compute closed-loop polynomial and match to desired one

Self-assessment material

notes

Question 1

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

Potential answers:

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

Solution 1:

The first step is to decide where you want the poles to bethis determines the desired system behavior.

notes

- see the associated solution(s), if compiled with that ones :)

Question 2

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

Potential answers:

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

Solution 1:

Pole placement is used to ensure the closed-loop poles correspond to desired system dynamics, influencing speed, damping, and stability.

notes

- see the associated solution(s), if compiled with that ones :)

Question 3

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

Potential answers:

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

Solution 1:

The derivative term modifies the system's dynamics, particularly by increasing damping and thus influencing the position of the closed-loop poles.

notes

- see the associated solution(s), if compiled with that ones :)

Question 4

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

Potential answers:

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

Solution 1:

Pole placement design requires expressing the closed-loop characteristic equation and matching its coefficients with those of a desired polynomial to solve for the PID gains.

notes

- see the associated solution(s), if compiled with that ones :)

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: **(correct)** The pole moves further left on the real axis, increasing system speed
II: **(wrong)** The pole becomes complex and causes oscillations
III: **(wrong)** The system gain decreases and response slows down
IV: **(wrong)** The zero of the system moves into the right-half plane
V: **(wrong)** I do not know

Solution 1:

For first-order systems, increasing K_P moves the closed-loop pole leftward (more negative real part), which speeds up the response.

notes

- see the associated solution(s), if compiled with that ones :)

Question 6

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

Potential answers:

I: **(wrong)** Compute the system output first, then choose PID gains, then set desired poles
II: **(wrong)** Start with experimental PID gains, simulate, and refine based on intuition
III: **(correct)** Choose desired poles, derive the corresponding characteristic polynomial, and match it with the actual closed-loop polynomial to solve for gains
IV: **(wrong)** Eliminate the need for poles by transforming to frequency domain
V: **(wrong)** I do not know

Solution 1:

Correct PID pole placement design involves selecting desired dynamics, deriving

notes

- see the associated solution(s), if compiled with that ones :)

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

notes

- The most important remarks from this sub-module are these ones