Systems Laboratory, Spring 2025

Damiano Varagnolo – CC-BY-4.0

notes

- welcome to the course!
- on this side of this document you will find notes that accompany the text typically visualized in class
- these notes are meant to convey the messages that are not displayed in the text on the side, and basically constitute what the teacher intends to say in class

Table of Contents I

notes

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

what is control

notes

Contents map

developed content units	taxonomy levels
feedforward	u1, e1
feedback	u1, e1
model based control	u1, e1
model free control	u1, e1

prerequisite content units	taxonomy levels
ODE	u1, e1

notes

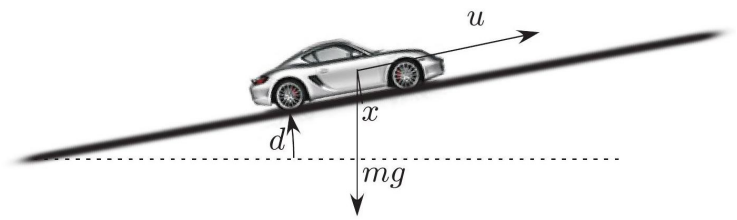
Main ILO of sub-module "what is control"

Interpret automatic control as an opportune operation on the dynamics of a system

notes

- by the end of this module you shall be able to do this

Example: speed control



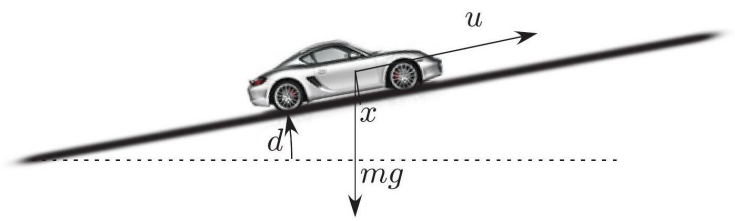
main parameters:

- m mass of the car
- b friction coefficient
- g gravity coefficient

notes

- in this example we shall make the car keep a target speed independently of the inclination of the road

A sufficiently accurate model for the purpose



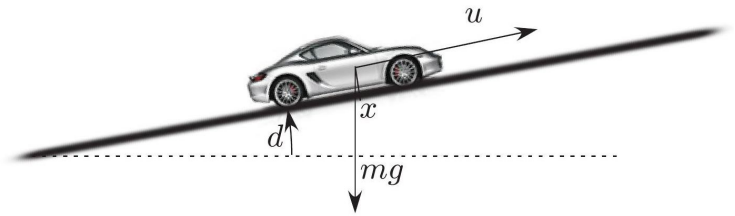
from Newton laws: $m\ddot{x} = -b\dot{x} + u - mg\sin(d)$

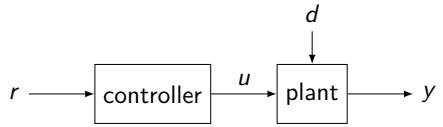
control objective: minimize $|y(t) - r(t)|$ with $r(t) =$ wished speed

notes

- and (trust me) this is a good enough model for the purpose of designing a control law

Feedforward control: "I think I know which $d(t)$ will happen, and I compensate for that"





$u(t)$ = something that compensates that $\widehat{d}(t)$

notes

- assuming we have an estimate / forecast $\widehat{d}(t)$, knowing the model we know how big we need u to compensate that disturbance

Open loop / feedforward control: "I think I know which $d(t)$ will happen, and I compensate for that"

$$u(t) = \text{something that compensates that } \widehat{d}(t)$$

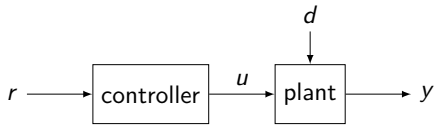
problem: if $\widehat{d} - d$ is big, then we expect $y - r$ to be big too, and we won't be able to note this!

notes

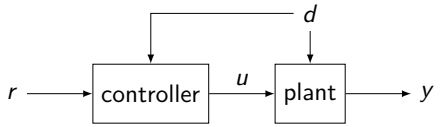
- of course pure feedforward / open loop control may be problematic, especially for the fact that there won't be ways to detect if there are big errors and react to them

Note that open loop ≠ feedforward

Open loop:



Feedforward:



notes

- in summary, open-loop control works without feedback or adjustment based on the system's state, while feedforward control adjusts proactively based on expected disturbances or changes. Both are reactive in the sense that they dont correct in real-time based on the output, but feedforward aims to address known disturbances before they affect the system
- the differences will be more clear as we go on with the course

Question 1

Open loop / feedforward control is so simple and naïve that no system in the world uses it

Potential answers:

I: **(wrong)** true
II: **(correct)** false
III: **(wrong)** I do not know

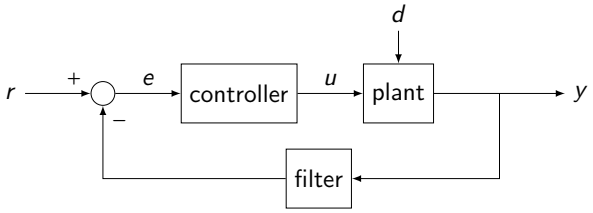
Solution 1:

Absolutely false! Open-loop controllers are simple, cost-effective, and require no feedback, making them easy to design and implement. They are fast since they dont need to process sensor data, making them suitable for time-sensitive applications. However, they are less accurate and cannot correct for disturbances or system variations. Examples include a washing machine running a fixed cycle, a microwave heating for a set time, a traffic light operating on a fixed schedule, an irrigation system with a timer, and an electric kettle that shuts off based on time rather than temperature.

notes

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

Feedback control: "I measure something, and depending on what I measure I take a decision"



*(= designing a controller, i.e., in this case designing a function that maps the signal **e** into the signal **u**)*

notes

- note that this is again the block scheme of feedback control, that is NOT the unique way of doing control (there is also 'feedforward', for example)

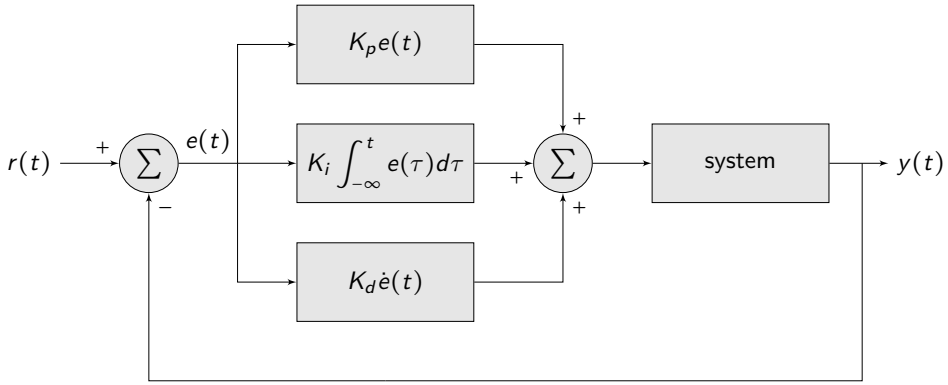
Main dichotomy on how to build a feedback controller

- model free *(e.g., PIDs)*
- model based *(e.g., MPCs)*

notes

- actually there are more details here you will discover while studying control systems. For now consider though just this dicothomy

Crash-slide on PIDs

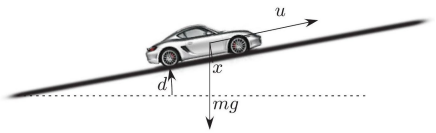


implicit assumption: we can measure $y(t)$*!* (see also `https://www.youtube.com/watch?v=UR0hOmjaHp0`!)

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

PID for the speed control task



$u(t) =$ K_p error right now
$+K_i$ sum of all past errors
$+K_d$ current tendency of the error

notes

- so from an intuitive perspective it is good to keep in mind that each component has a certain meaning
- tuning PIDs is an art. A nice book on this is this: `https://link.springer.com/book/10.1007/1-84628-586-0`

Question 2

A PID is guaranteed to work well

Potential answers:

I: **(wrong)** yes, always
II: **(wrong)** no, never
III: **(correct)** no, it depends on how well tuned it is
IV: **(wrong)** I do not know

Solution 1:

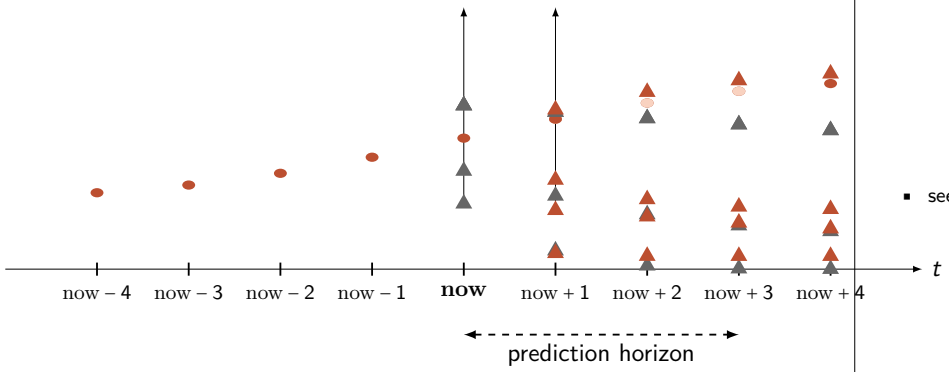
A PID controller is not guaranteed to work well in all cases; its performance depends on proper tuning. Poorly tuned PID controllers can lead to instability, slow response, or excessive oscillations. The three gains (proportional (K_p), integral (K_i), and derivative (K_d)) must be adjusted based on the system dynamics. For example, if K_p is too high, the system may oscillate or become unstable. If K_i is too high, the system may suffer from overshoot and integral windup. If K_d is too high, the system may become too sensitive to noise.

notes

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

Crash-slide on MPCs

signals measured past y wished future y potential u, and corresponding forecasted y best u = "best" forecasted y execute $u(\text{now})$



- what is control 15

notes

- see also https://www.youtube.com/watch?v=UR0hOmjaHp0!

Question 3

A MPC is guaranteed to work well

Potential answers:

I: **(wrong)** yes, always
II: **(wrong)** no, never
III: **(correct)** no, it depends on how good the model is
IV: **(wrong)** I do not know

Solution 1:

Model Predictive Control (MPC) is not guaranteed to work well in all cases; its performance depends on how accurately the model represents the real system. Since MPC relies on predicting future system behavior using a model, inaccuracies in the model can lead to poor performance or instability.

Key factors affecting MPC performance:

- **Model Accuracy:** If the model does not capture the system dynamics well, predictions will be incorrect, leading to suboptimal control actions.

- what is control 16

notes

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

Question 4

Is MPC guaranteed to work better than PID?

Potential answers:

I: **(wrong)** yes, always
II: **(wrong)** no, never
III: **(correct)** no, it actually depends on the situation
IV: **(wrong)** I do not know

Solution 1:

MPC is not always guaranteed to work better than PID; its effectiveness depends on the specific system and control objectives. While MPC offers advantages such as constraint handling, predictive capabilities, and optimization-based control, it also has drawbacks compared to PID.

Key factors influencing the choice between MPC and PID:

- **System Complexity:** PID works well for simple, well-modeled systems, while MPC is better suited for multivariable or highly constrained systems.
- **Computational Resources:** PID is computationally inexpensive and easy to implement, whereas MPC requires solving an optimization problem at each step, making it more demanding.

notes

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

Question 5

Is closed loop control guaranteed to work better than open loop control?

Potential answers:

I: **(wrong)** yes, always
II: **(wrong)** no, never
III: **(correct)** no, it actually depends on the situation
IV: **(wrong)** I do not know

Solution 1:

Closed-loop control is not always guaranteed to work better than open-loop control; the effectiveness of each approach depends on the specific application and system characteristics. While closed-loop control provides feedback and can correct errors, open-loop control can be sufficient or even preferable in certain scenarios.

Key factors influencing the choice between open-loop and closed-loop control:

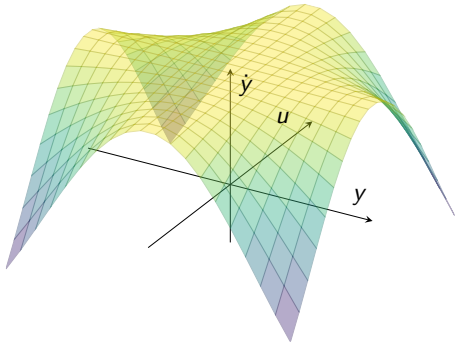
- **System Variability:** Closed-loop control is beneficial when the system is

notes

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

But eventually, what is control?

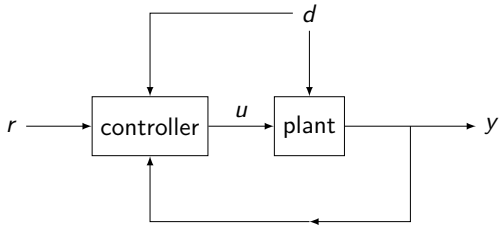
an algorithm to compute $u(t)$ starting from the available information



notes

- and there are many different algorithms to do so. A goal that you may put to yourself in these years is to arrive at knowing a sufficient number of them, and in details enough to have a feeling of the pros and cons, so to choose the right approach depending on the needs

A final note: in practice it is a good choice to combine both feedback and feedforward actions



notes

- this block scheme shows that there is both a feedback and a feedforward action

Summarizing

Interpret automatic control as an opportune operation on the dynamics of a system

- think at what feedforward and feedback mean
- think at the fact that essentially they are ways of computing u, and that that u enters the dynamics of the system

notes

- you should now be able to do this, following the pseudo-algorithm in the itemized list

Most important python code for this sub-module

notes

Note: going through everything here would take months - just be aware of their existence and start playing with them

- `https://python-control.readthedocs.io/en/0.10.1/`
- `https://pypi.org/project/simple-pid/`
- `https://www.do-mpc.com/en/latest/`

notes

- these libraries all relate to what we saw in this module; watch out though that

Self-assessment material

notes

Question 6

PID control requires a model of the system to function correctly.

Potential answers:

I: **(wrong)** yes, always
II: **(correct)** no, it works without a model
III: **(wrong)** I do not know

Solution 1:

A PID controller works without requiring a model of the system. Instead, it uses feedback from the systems output to adjust the control input. The three parameters proportional (K_p), integral (K_i), and derivative (K_d) are tuned based on system behavior, but no explicit system model is necessary. This makes PID controllers simple and widely applicable, even in systems where modeling is difficult or not feasible.

notes

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

Question 7

Model Predictive Control (MPC) can only be applied when the model is perfect.

Potential answers:

I: **(wrong)** yes, the model must be perfect
II: **(correct)** no, it works with approximate models
III: **(wrong)** I do not know

Solution 1:

MPC does not require a perfect model, though its performance depends on how accurately the model represents the system. If the model is approximate, the controller may still work well, but the performance may degrade if the model is too far from reality. In practice, methods like robust or adaptive MPC are used to handle model inaccuracies and disturbances.

notes

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

Question 8

Feedforward control is generally better than feedback control for handling disturbances.

Potential answers:

I: **(wrong)** yes, feedforward is always better
II: **(correct)** no, feedback control is better for disturbances
III: **(wrong)** I do not know

Solution 1:

Feedforward control can be effective when disturbances are predictable, as it compensates for them proactively. However, feedback control is generally better for handling unexpected disturbances or system changes because it can adjust in real-time based on the system's output. Feedback ensures that the system can respond to unmeasured or unforeseen variations, making it more robust in dynamic environments.

notes

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

Question 9

Open-loop control is more reliable than closed-loop control in all situations.

Potential answers:

I: **(wrong)** yes, open-loop is always more reliable
II: **(correct)** no, it depends on the system and application
III: **(wrong)** I do not know

Solution 1:

Open-loop control is simpler and can be more reliable in cases where the system is predictable and not subject to disturbances. However, closed-loop control is more reliable when disturbances or system variations are present, as it adjusts based on feedback. The choice between open-loop and closed-loop control depends on the specific system dynamics, complexity, and performance requirements.

notes

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

Question 10

PID controllers are always preferable to MPC in terms of performance.

Potential answers:

I: **(wrong)** yes, PID always outperforms MPC
II: **(correct)** no, it depends on the system and objectives
III: **(wrong)** I do not know

Solution 1:

PID controllers are well-suited for simple systems with few inputs and outputs, especially when the system is well-understood and not subject to significant disturbances. However, MPC can be more powerful in handling complex, multivariable systems with constraints. MPC is capable of optimizing system behavior over a time horizon, making it suitable for systems where PID controllers might not be able to effectively manage multiple interacting variables or constraints. Therefore, the choice depends on the system's complexity and control objectives.

notes

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

Recap of sub-module "what is control"

- designing a controller means designing an algorithm that transforms information into decision
- there are several types of controllers, each with pros and cons
- taking decisions (i.e., actuating u) means modifying the dynamics of the system

notes

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