Table of Contents I

notes

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

Visualizing systems with block schemes

notes

Contents map

developed content units	taxonomy levels
block scheme	u1, e1

prerequisite content units	taxonomy levels
ODEs	u1, e1

notes

Main ILO of sub-module "Visualizing systems with block schemes"

Explain the purpose of block diagrams in control systems by comparing their industrial and analytical applications

Construct block diagram representations of first-order differential equations by identifying and connecting appropriate functional blocks

Distinguish between static and dynamic blocks by analyzing their mathematical representations and memory requirements

Simplify complex block diagrams to single equivalent blocks by applying series, parallel, and feedback reduction rules

Interpret feedback loops in block diagrams by relating their presence to system equations and dynamic behavior

notes

- By the end of this module you shall be able to: translate equations to diagrams, analyze block properties, perform diagram reductions, and explain feedback implications
- Focus on conceptual understanding rather than rote procedures
- Apply these skills to both analysis and design contexts

Roadmap

- the most common block schemes
- first order systems as block schemes

notes

- this module is quite lightweight, and discusses only graphical representations

Block diagrams - why?

- used very often in companies
- aid visualization *(until a certain complexity is reached. . .)*
- enable "drag & drop" way of programming

- in this course, primarily used for interpretations

notes

- When you'll work in industry, you'll see these diagrams everywhere - they're the lingua franca of control engineers
- The visualization helps tremendously, though I must warn you - when systems get really complex, the diagrams can become messy
- The drag & drop approach makes programming controllers much more intuitive than writing pure code
- For our course, pay special attention to how we'll use them to interpret system behavior

Block diagrams - why? Part 2

Convenient operation in control systems analysis

- step 1: identify single-input single-output subsets (blocks)
- step 2: represent the overall system as an interconnection of such subsets

notes

- Think of this like building with Lego blocks - we first identify simple components, then connect them
- This modular approach is powerful because complex systems become manageable when broken down
- In exams, I'll often ask you to follow exactly this two-step process when analyzing systems
- Remember: even the most complex control system in a rocket is just many SISO blocks connected together

Static block (a.k.a. memoryless block)

= representation of a static (i.e., instantaneous) relationship between input and output

$$y(t) = f\big(u(t)\big)$$

$$u(t) \longrightarrow \boxed{f(\cdot)} \longrightarrow y(t)$$

they can be linear or nonlinear, depending on $f(\cdot)$

notes

- This is the simplest type of block - what goes in immediately affects what comes out
- Imagine this like a simple math function - put in a number, get out a result, no memory involved
- The linear case is like y=ax, while nonlinear could be y=xš - we'll see both are important
- Practical example: a resistor is a static block (V=RI), while a capacitor is dynamic

Dynamic block (a.k.a. block with memory)

= representation of a dynamic relationship between input and output (i.e., such that the output $y(t)$ at time t does not depend only on the input $u(t)$ at the same time t, but also on its behavior at different times – potentially not only past)

$$y(t) = f\big(u, y\big)$$

$$u(t) \longrightarrow \boxed{f(\cdot)} \longrightarrow y(t)$$

they can be linear or nonlinear, depending on $f(\cdot)$

notes

- Now we're dealing with systems that have memory - like your brain remembering past inputs
- This is where things get interesting! The output depends on history, not just the present
- The equation shows y depends on both u and itself - this recursive nature is crucial
- Most physical systems are dynamic - think of a car's suspension or a heating system
- The linear case leads us to transfer functions, which we'll study in depth

Example of dynamic block: transfer function

$$y(t) = y_\ell(t) + u * h(t)$$

$$u(t) \longrightarrow \boxed{H(s)} \longrightarrow y(t)$$

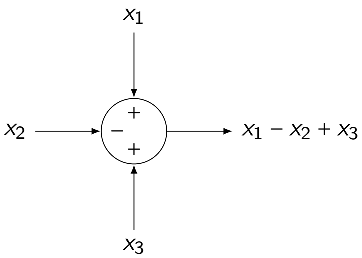
notes

- Here we see the power of Laplace transforms - differential equations become algebraic!
- The transfer function H(s) encapsulates all the system dynamics in one neat package
- The convolution (that * symbol) shows how past inputs affect current output
- We'll spend weeks working with these - they're that important in control theory
- The y(t) term represents initial conditions - never forget these in your exams!

Operations with the block schemes

notes

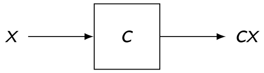
Most common block diagrams - sum of n signals



notes

- This circle with many inputs is our workhorse for combining signals
- Notice the + and - signs - they're crucial for feedback control
- In practice, these represent physical summing junctions in analog systems
- When you see this, think "algebraic sum" - all inputs added according to their signs
- Pro tip: Always double-check the signs during exams!

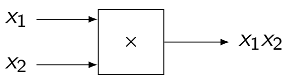
Most common block diagrams - multiplication for a constant



notes

- This simple block represents amplification or attenuation
- The K can be a gain in an amplifier or a conversion constant
- In physical systems, think of this as a lever or a gear ratio
- Surprisingly powerful - most linear controllers are just fancy versions of this!
- Remember: The arrow direction matters - Kx is different from xK in matrix cases

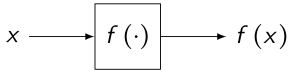
Most common block diagrams - multiplication of two signals



notes

- Now we're entering nonlinear territory - exciting!
- This represents physical phenomena like mixing or modulation
- Notice there's no Œ symbol - we use adjacency to denote multiplication
- Common in real systems: think of volume control (signal Œ gain)
- Warning: Analyzing systems with these gets complex fast!

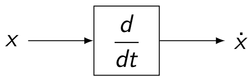
Most common block diagrams - generic functions



notes

- This is our "anything goes" block - f can be any mathematical function
- We use this when standard blocks don't capture the relationship
- Examples: sensors with nonlinear responses, lookup tables, AI components
- The challenge: How to analyze systems with these? We'll learn methods
- On your drawings, always label f clearly - is it sin(), exp(), or something else?

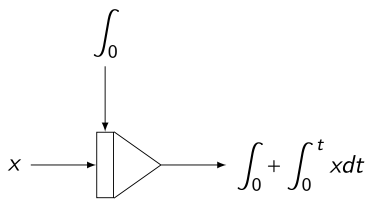
Most common block diagrams - derivatives



notes

- The s operator represents differentiation - a powerful concept
- Physically, think of this as a dashpot or any rate-sensitive device
- In control, derivatives help anticipate future behavior (D-control)
- Warning: Derivatives amplify noise - be careful in real implementations
- We'll see how to handle these properly when we study PID controllers

Most common block diagrams - integrals



notes

- The 1/s operator is integration - the opposite of differentiation
- Physically, this represents accumulation - like water in a tank
- In control, integrals eliminate steady-state error (I-control)
- Crucial point: Never forget initial conditions with integrators!
- Practical issue: Integrators can wind up - we'll discuss anti-windup techniques
- Remember: In discrete systems, integration becomes summation

Discussion: how do we represent a first order differential equation with a block scheme?

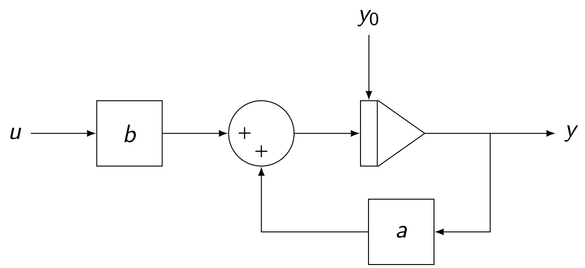
$\dot{y} = ay + bu$

notes

- Let's think together about how to translate this equation into a block diagram - it's like building with Legos!
- First, identify the components: we have a derivative, a multiplication, and a sum
- The video will show the step-by-step construction, but try to sketch it yourself first
- Remember: $\dot{y}$ means we'll need an integrator block to get y from $\dot{y}$
- This simple example contains all the key concepts we'll use for more complex systems

The solution

$\dot{y} = ay + bu$



Discussion: do you note the presence of a feedback loop?

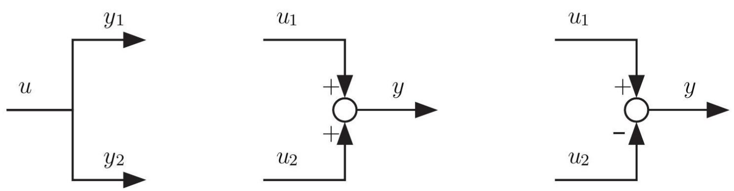
notes

- Here's the complete picture - notice how we've translated each mathematical operation into a block
- The integrator (1/s block) is crucial - it's what makes this system dynamic
- The gain blocks a and b represent the equation coefficients
- That feedback loop is the heart of the system's behavior! It's what makes the solution exponential
- This is our first example of feedback - something we'll see everywhere in control systems
- The negative sign in the feedback is important - it creates stability in many cases
- Think about physical examples: RC circuits, thermal systems - they all have this structure

Interconnections

notes

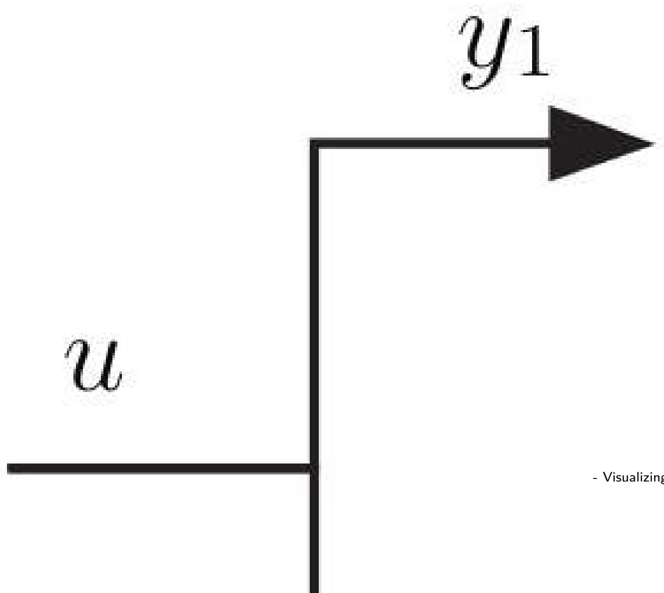
Branching points and summing junctions



notes

- These are the "plumbing" components of our block diagrams - they route signals
- The branching point is like a T-joint in pipes - it duplicates a signal without changing it
- Summing junctions are where signals combine - crucial for feedback and feedforward
- In real systems, these represent physical connections or mathematical operations
- Always check your connections twice - it's easy to misplace a + or - sign!

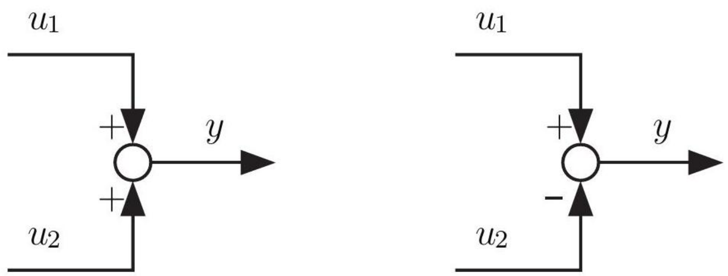
Branching points



notes

- This is the simplest "splitter" - one input, two identical outputs
- Physically, think of a wire splitting into two, or a signal being measured while also being used
- No computation happens here - it's just signal distribution
- In digital systems, this represents data being sent to multiple processes
- Important: The branches don't affect each other - no loading effects in ideal diagrams

Junctions



= blocks with two inputs $u_1(t)$, $u_2(t)$ and one output $y(t)$, in which

$$y(t) = u_1(t) + u_2(t)$$

or

$$y(t) = u_1(t) - u_2(t)$$

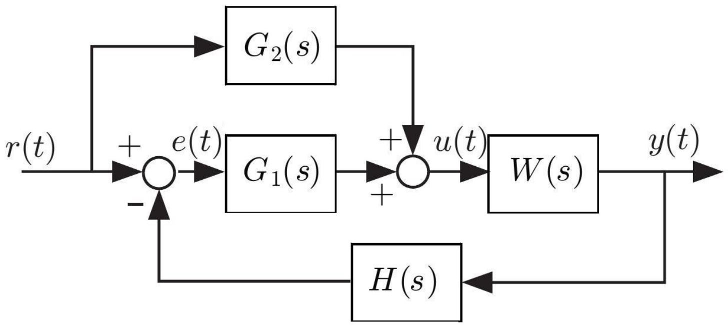
notes

- These circles are the workhorses of control systems - where signals combine
- The signs are crucial! A misplaced minus can turn negative feedback into positive
- In analog circuits, these represent operational amplifier configurations
- The output is instantaneous - no memory or dynamics in the junction itself
- Notice how compact the notation is - a whole computation in one symbol!

Reduction of block schemes to a single block

notes

Example



complicated $\Longrightarrow$ can we rewrite it in a simpler way? Algebraic relations:

$$Y(s) = W(s)U(s)$$
$$U(s) = G_1(s)E(s) + G_2(s)R(s)$$
$$E(s) = R(s) - H(s)Y(s)$$

notes

- This looks complex, but we can tame it step by step - like solving a puzzle
- First approach: Write all equations, then substitute - the "brute force" method
- Second approach: Look for patterns we can simplify piece by piece
- The goal is to find Y(s)/R(s) - the overall transfer function
- Notice the feedback loop through H(s) - that will be key to our solution

Example, part 2

$$Y(s) = W(s)U(s)$$
$$U(s) = G_1(s)E(s) + G_2(s)R(s)$$
$$E(s) = R(s) - H(s)Y(s)$$

implies

$$\begin{aligned} Y(s) &= W(s)\left[G_1(s)E(s) + G_2(s)R(s)\right] \\ &= W(s)\left[G_1(s)(R(s) - H(s)Y(s)) + G_2(s)R(s)\right] \end{aligned}$$

multiplying and grouping terms in $Y(s)$ and in $R(s)$ separately we obtain

$$(1 + W(s)G_1(s)H(s))\,Y(s) = W(s)\,(G_1(s) + G_2(s))\,R(s)$$

therefore

$$\frac{Y(s)}{R(s)} = \frac{W(s)\,(G_1(s) + G_2(s))}{1 + W(s)G_1(s)H(s)}$$

and so the relationship between the input $r(t)$ and the output $y(t)$ can be described by a single block with transfer function

$$\frac{W(s)\,(G_1(s) + G_2(s))}{1 + W(s)G_1(s)H(s)}$$

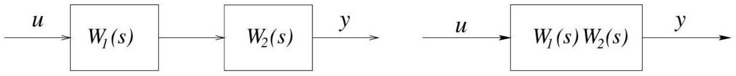
and thus the relationship between the input $r(t)$ and the output $y(t)$ can be

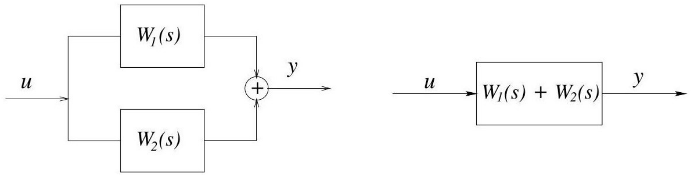
notes

- See how the algebra systematically simplifies the diagram? That's the power of transfer functions!
- The denominator 1+WGH is characteristic of feedback systems - remember this form
- The numerator combines the parallel paths (G+G) through W
- This result tells us the entire system's behavior in one compact expression
- You'll use this exact procedure many times in control design and analysis

Reduction of block diagrams to a single block

Specific case: series or parallel blocks



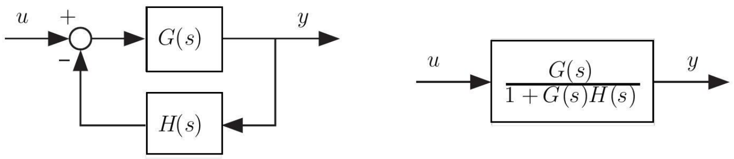


notes

- Series connection: Multiply the transfer functions - like a chain of systems
- Parallel connection: Add the transfer functions - like multiple paths
- These are the block diagram equivalents of basic arithmetic operations
- Important: The order matters in series connection (GG GG in general)
- These rules let us collapse complex diagrams piece by piece

Reduction of block diagrams to a single block

Specific case: feedback

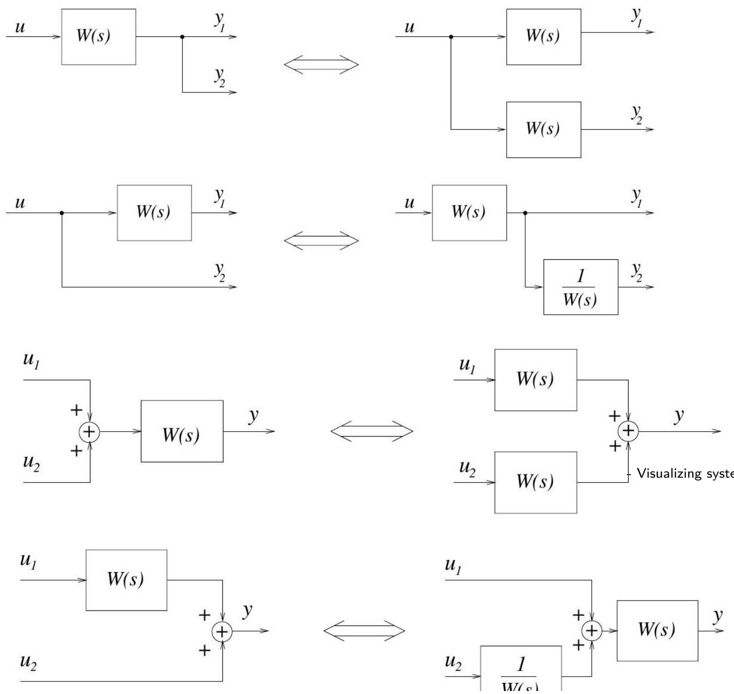


notes

- This is the most important reduction rule - the feedback formula
- Notice the characteristic form: G/(1+GH) for negative feedback
- The denominator 1+GH determines stability - we'll study this deeply later
- Positive feedback would have 1-GH in the denominator
- This single formula explains how control systems can regulate themselves

Reduction of block diagrams to a single block

Specific case: moving blocks around



notes

- These are like "algebraic identities" for block diagrams
- Moving blocks past summing junctions requires adjusting the paths
- These rules help rearrange diagrams into standard forms we can reduce
- Very useful when dealing with complex interconnections
- Remember: These transformations must preserve the overall input-output relationship

Suggested videos introducing Simulink

- `https://www.youtube.com/watch?v=pFICO_syIIs` (10 minutes)
- `https://www.youtube.com/watch?v=QIAxyLchf4k` (50 minutes)

notes

- a lot of companies use Simulink, a Matlab-based graphical programming environment for modeling, simulating and analyzing multidomain dynamical systems
- it can be integrated with the rest of the Matlab environment (i.e., either drive or be scripted from it)
- so maybe better to take a look at the first video, if you never used it before
- we are not going to use it in this course

Summarizing

Explain the purpose of block diagrams in control systems by comparing their industrial and analytical applications

Construct block diagram representations of first-order differential equations by identifying and connecting appropriate functional blocks

Distinguish between static and dynamic blocks by analyzing their mathematical representations and memory requirements

Simplify complex block diagrams to single equivalent blocks by applying series, parallel, and feedback reduction rules

Interpret feedback loops in block diagrams by relating their presence to system equations and dynamic behavior

notes

- you should now be able to do this, following the information given in the slides above

Most important python code for this sub-module

notes

The `control` package

Example:

- # Series connection
  series = control.series(G1, G2)
- # Parallel connection
  parallel = control.parallel(G1, G2)
- # Feedback connection feedback = control.feedback(G1, G2)

notes

- this python library is definitely useful for control engineers!

Self-assessment material

notes

Question 1

In a block diagram representation of a first-order differential equation $\dot{y} = ay + bu$, why does the feedback path emerge?

Potential answers:

I: **(wrong)** Because we need to implement a controller
II: **(correct)** Because the output y affects its own rate of change $\dot{y}$
III: **(wrong)** Because all dynamic systems require feedback
IV: **(wrong)** Because it represents the input signal $u(t)$
V: **(wrong)** I do not know

Solution 1:

The feedback path emerges naturally from the mathematical structure of the differential equation itself. The term ay means the current state y influences its own derivative $\dot{y}$, creating an inherent feedback relationship. This isn't added by design but is fundamental to the system's dynamics.

notes

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

Question 2

What is the fundamental difference between a branching point and a summing junction in block diagrams?

Potential answers:

I: **(wrong)** Branching points perform calculations while summing junctions don't
II: **(correct)** Branching points duplicate signals while summing junctions combine them
III: **(wrong)** Summing junctions can only handle two inputs while branching points can have many outputs
IV: **(wrong)** Branching points require memory while summing junctions are memoryless
V: **(wrong)** I do not know

Solution 1:

Branching points and summing junctions serve fundamentally different purposes.

notes

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

Question 3

When reducing a complex block diagram to a single equivalent block, what does the denominator of the resulting transfer function typically represent?

Potential answers:

I: **(wrong)** The gain of the input signal
II: **(wrong)** The time delay of the system
III: **(correct)** The feedback characteristics of the system
IV: **(wrong)** The nonlinearities in the system
V: **(wrong)** I do not know

Solution 1:

The denominator of the reduced transfer function (typically in the form $1 + GH$) captures the system's feedback characteristics. This term determines crucial properties like stability and dynamic response. The numerator represents the forward path characteristics. This distinction is fundamental to understanding closed-loop system behavior.

notes

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

Question 4

Why does a dynamic block require memory while a static block doesn't?

Potential answers:

I: **(wrong)** Because dynamic blocks are always digital implementations
II: **(correct)** Because dynamic blocks depend on past values of input/output
III: **(wrong)** Because static blocks can only represent linear relationships
IV: **(wrong)** Because dynamic blocks operate at higher frequencies
V: **(wrong)** I do not know

Solution 1:

The key distinction lies in time-dependence. Static blocks represent instantaneous relationships (output depends only on current input), while dynamic blocks represent relationships where the output depends on the history of inputs/outputs (through derivatives, integrals, or delays). This historical dependence is what we mean by "memory" in systems.

notes

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

Question 5

What is the conceptual reason why series-connected blocks can be reduced by multiplying their transfer functions?

Potential answers:

I: **(wrong)** Because multiplication is commutative
II: **(wrong)** Because it's an arbitrary convention
III: **(correct)** Because each block's output becomes the next block's input
IV: **(wrong)** Because the Laplace transform requires it
V: **(wrong)** I do not know

Solution 1:

The multiplication rule emerges naturally from the series connection structure. The first block's output $G_1(s)X(s)$ becomes the second block's input, which then outputs $G_2(s)[G_1(s)X(s)] = [G_2(s)G_1(s)]X(s)$. This chaining of operations mathematically leads to multiplication of transfer functions. The commutative property is a consequence, not the reason.

notes

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

Recap of module "Visualizing systems with block schemes"

- block representations are alternative representations
- they enable graphical coding, that is used quite a lot in big companies

notes

- what we learned in this module is this