

Faithfulness in Model-Based Development

1. Model-based development in computer science and in control and how to make them converge
 2. The case of sampling
 3. A topological approach
-

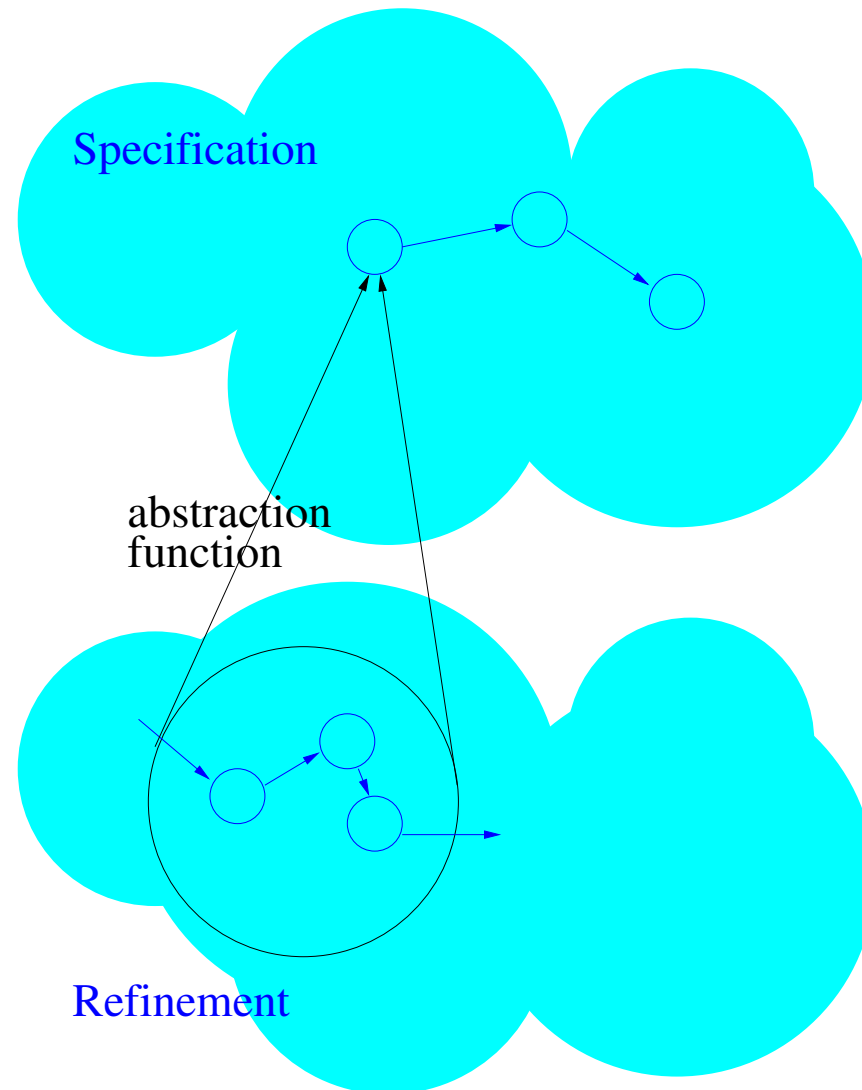
Model-based Design in Computer Science

- Starts from a non deterministic **specification**
- Based on successive property-preserving **refinements**
- Until an **implementation** is reached

Remarks:

- This is an idealised scheme, seldom fulfilled
 - Yet has a paradigmatic value
 - Some real-world impressive achievements in control!!
 - **B method (Abrial)**: Paris, Barcelona, New York subways
-

Model-based design in computer science



Model-based design in computer science _____

Further steps:

- Add implementation details:
 - Separate controllers from environment !!!
 - Generate control programs
-

Model-based design control

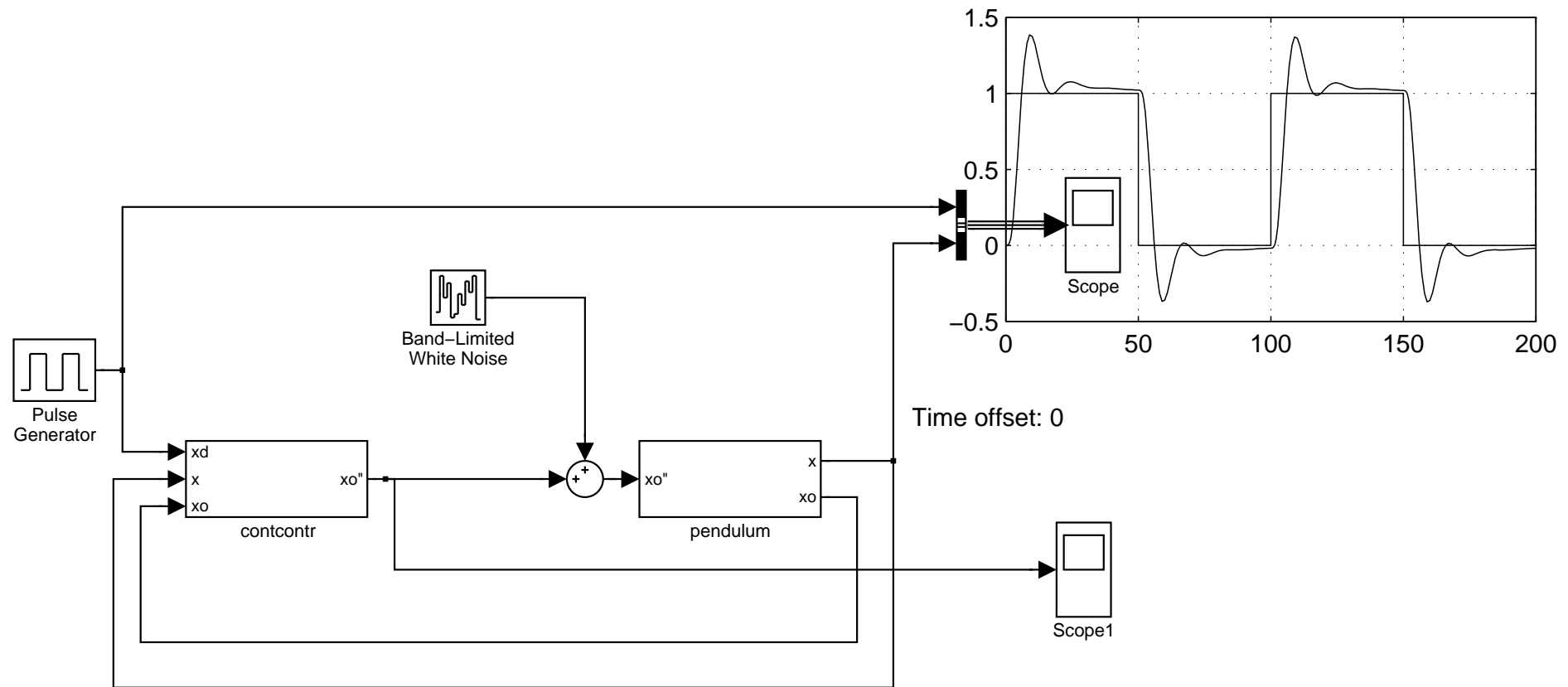
- Start from a perfect model
- Design a robust controller
- Add perturbations and implementation details and checks for robustness

Remarks:

- This is also an idealised scheme
-

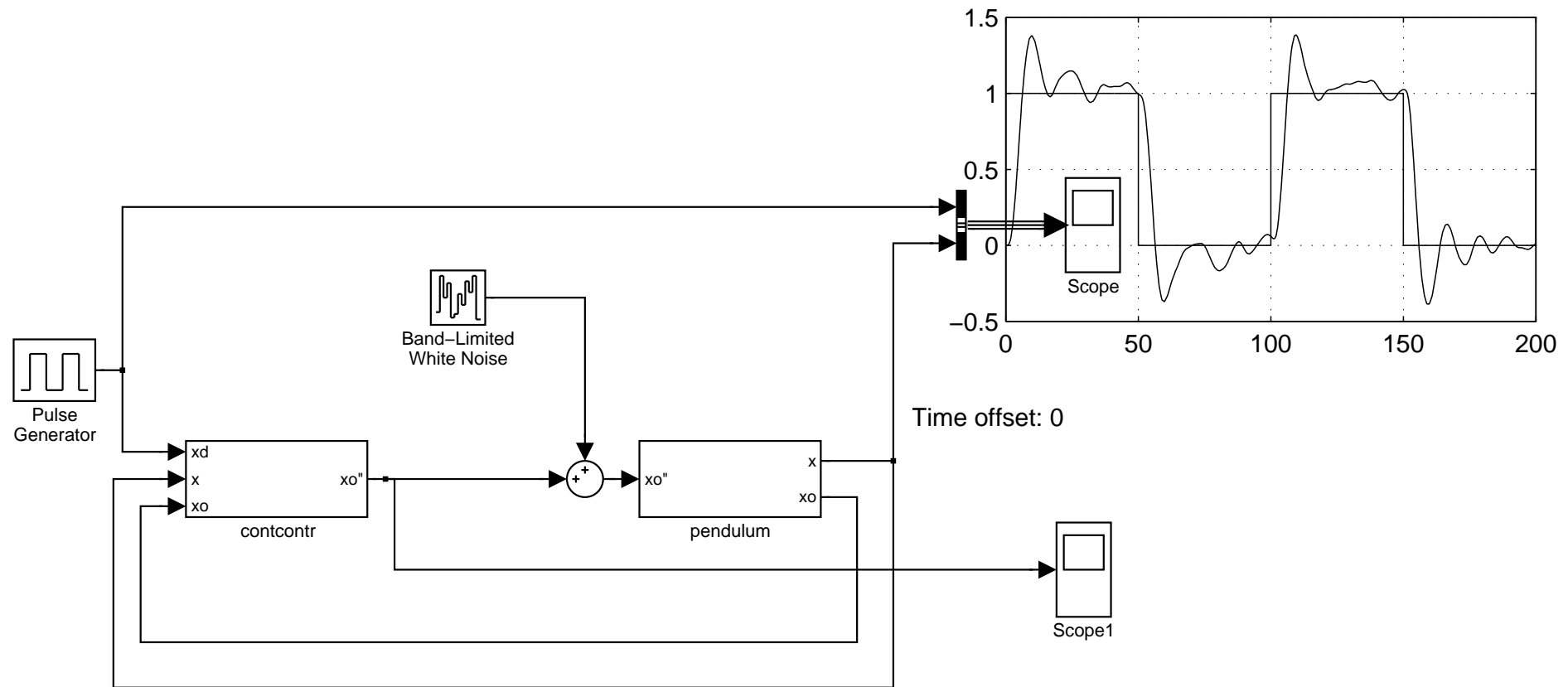
Model-based design in control

Perfect model



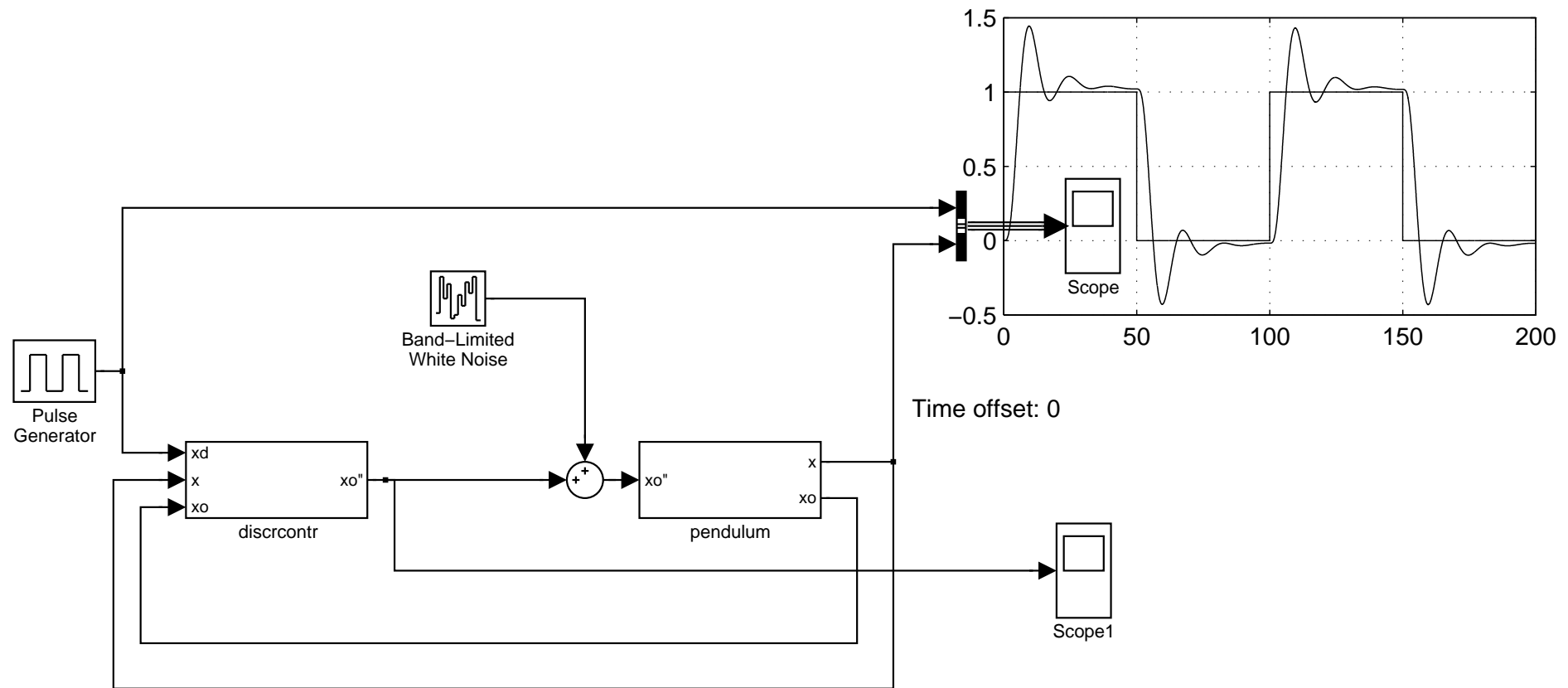
Model-based design in control

Perfect model with noise



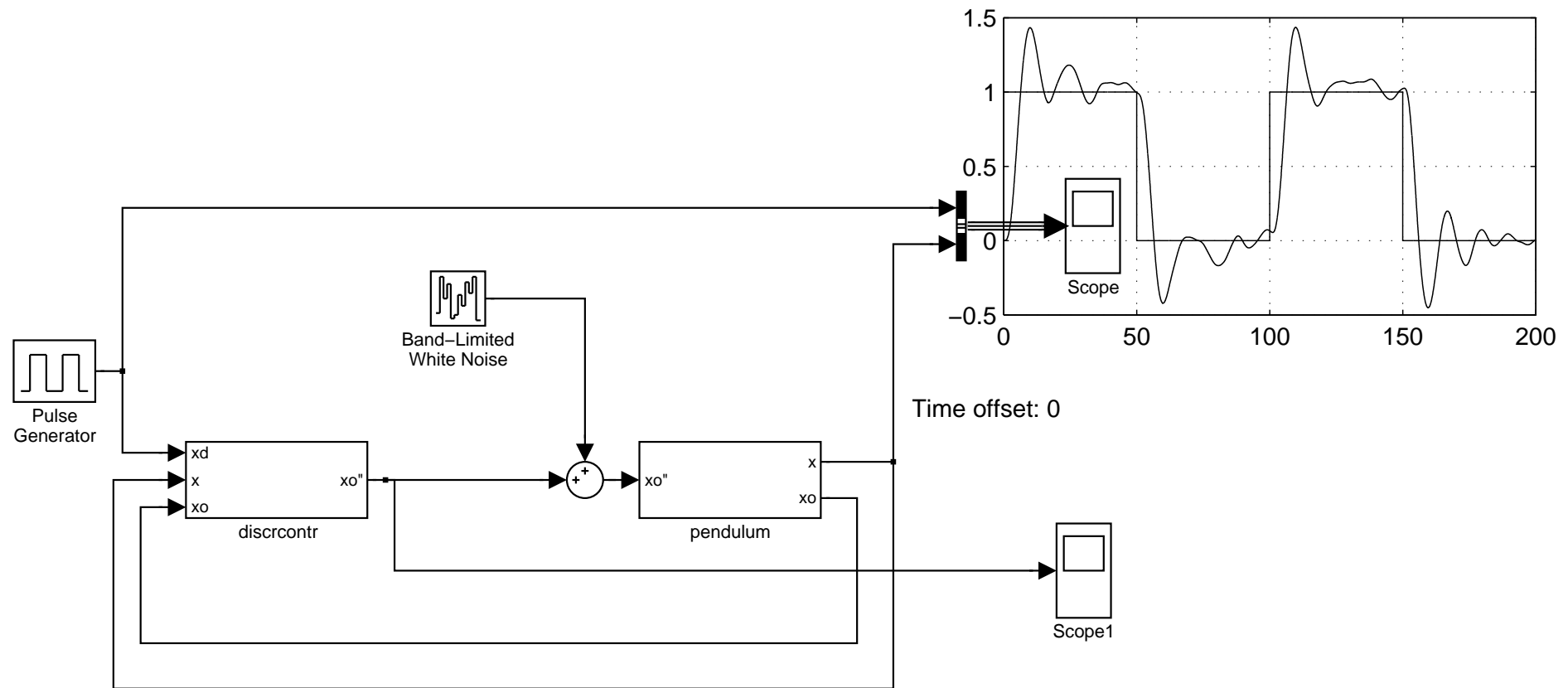
Model-based design in control

Discrete-time controller



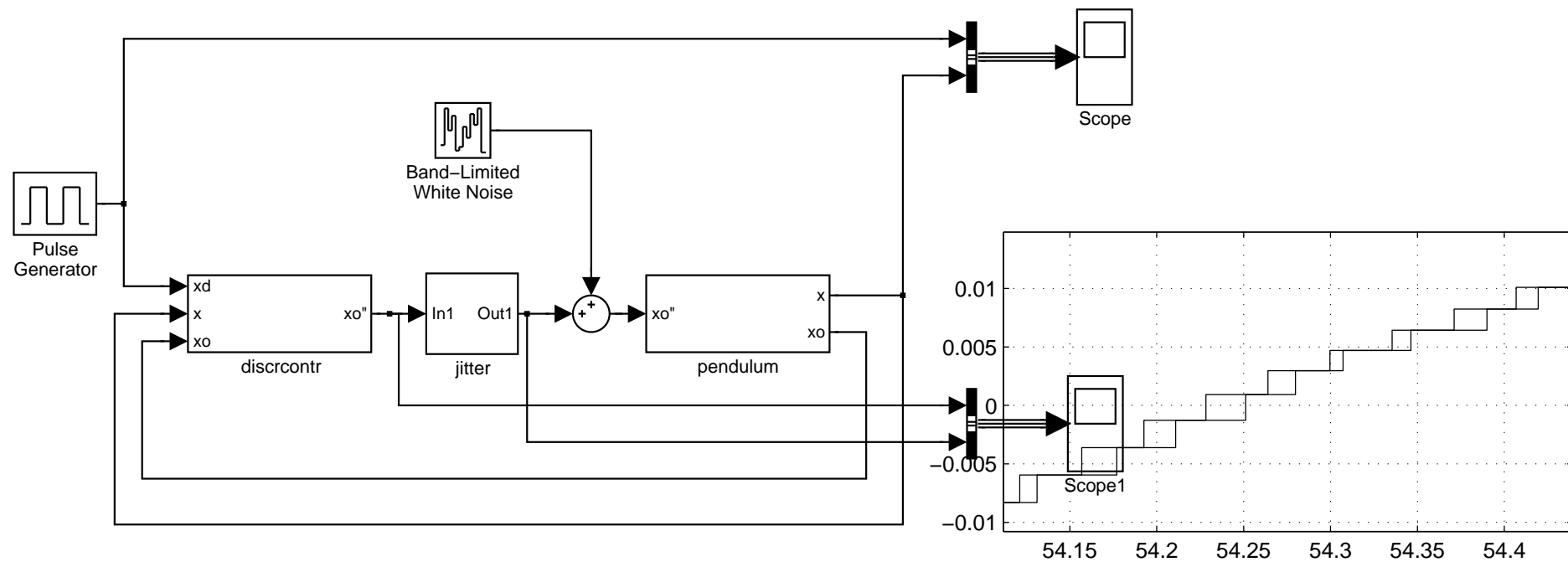
Model-based design in control

Discrete-time controller with noise



Model-based design in control

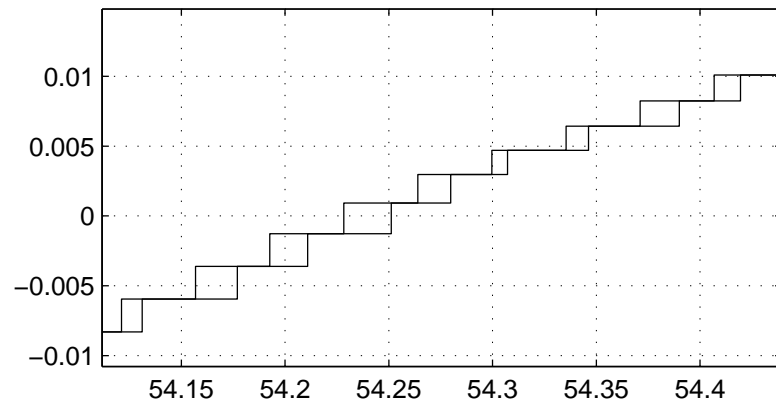
Discrete-time controller with jitter



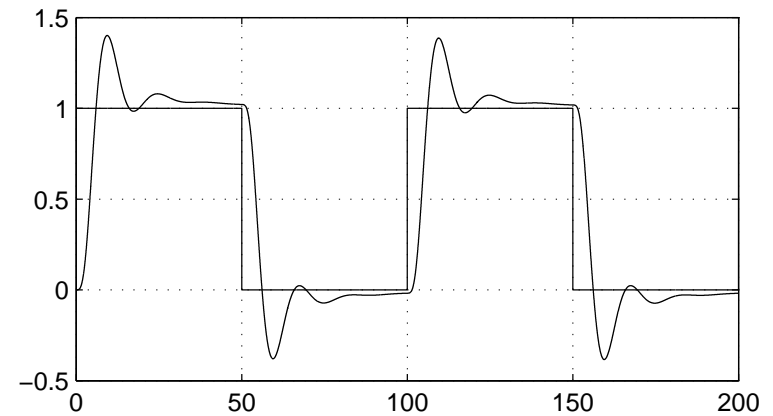
Time offset: 0

Model-based design in control

Discrete-time controller with jitter



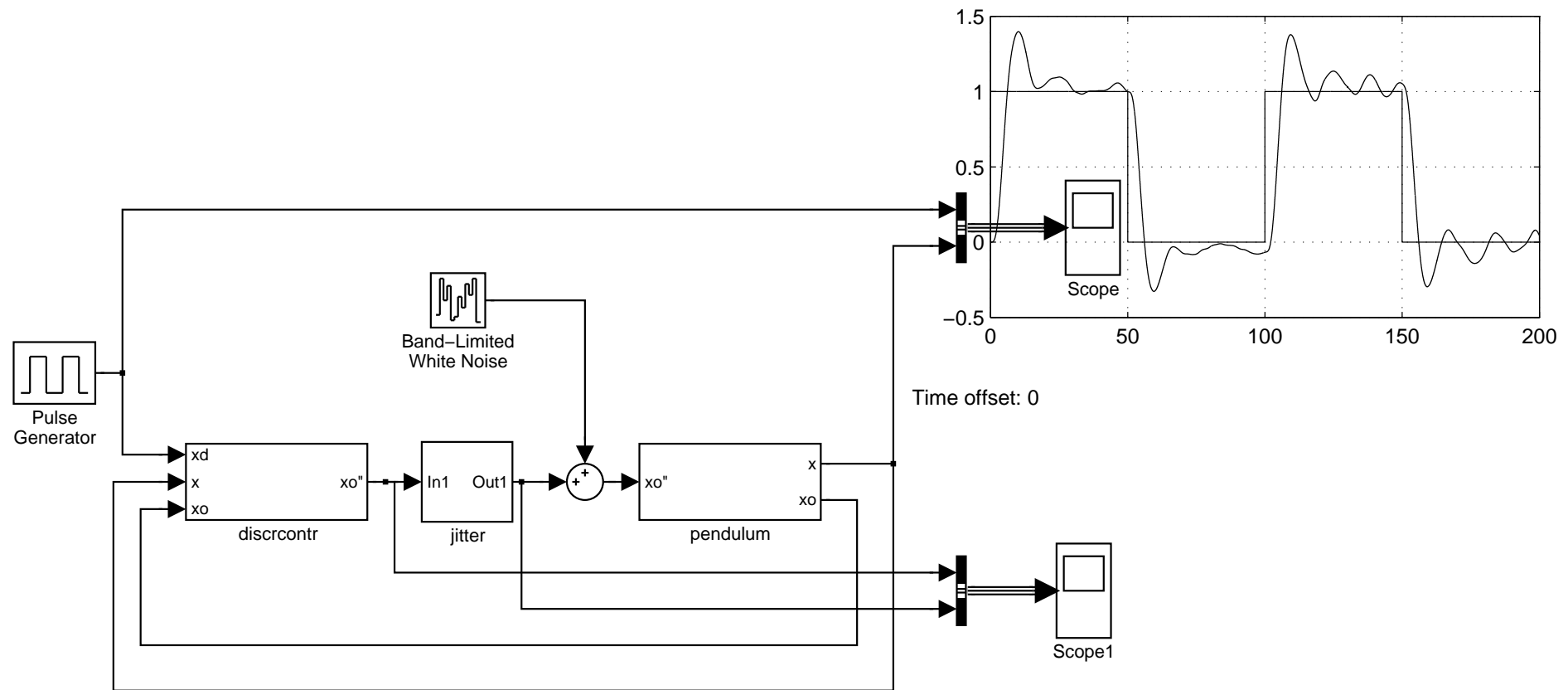
Time offset: 0



Time offset: 0

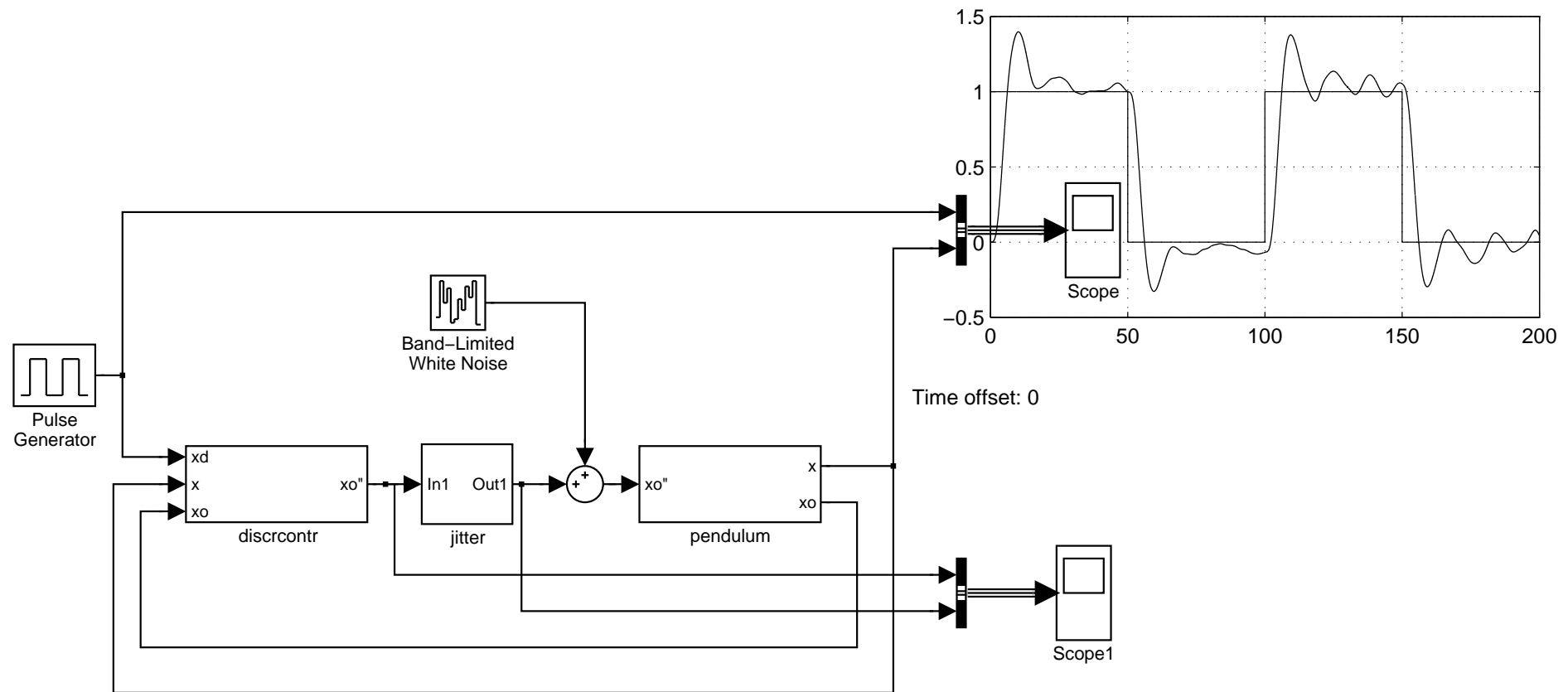
Model-based design in control

Discrete-time controller with jitter and noise



Model-based design in control

Discrete-time controller with jitter and noise



Is this enough? When should we stop adding implementation details?

A Possible Answer

Make computer science and control science converge ??

A suggestion :

Consider the perfect control model as specifying a set of behaviours, those behaviours which are within some “distance” of the perfect model behaviour.

This requires some notion of “distance”, able to account for

- perturbations
 - modelling errors
 - sampling
 - jitter and communication delays
-

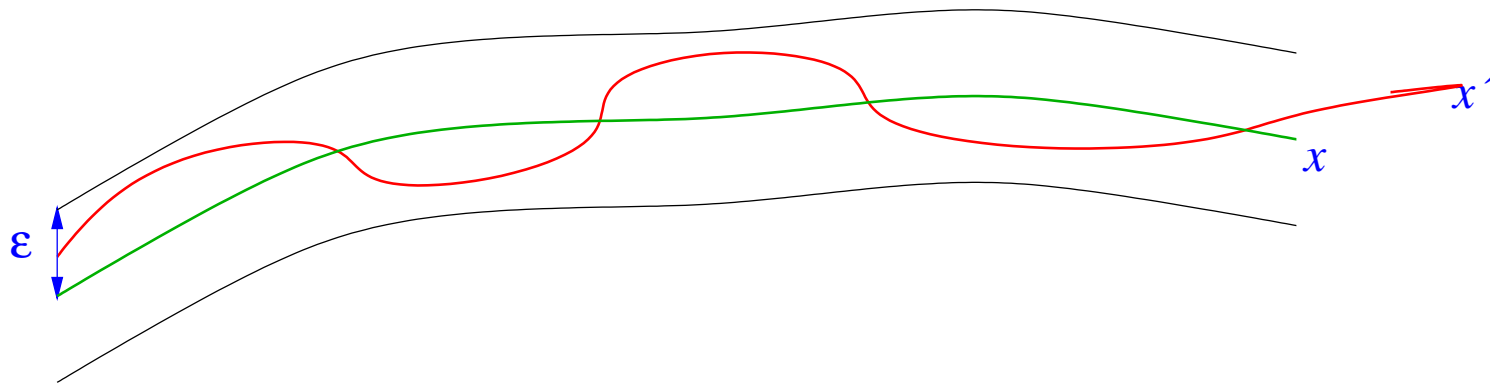
- priorities, distribution, ...

Approximation, Sampling and Voting in Hybrid Computing Systems **Ch. Kossentini and P. Caspi**

Hybrid Systems Computation and Control, HSCC06, Volume 3927 in
Lecture Notes in Computer Science

<http://www-verimag.imag.fr/caspi/PAPIERS/index.html>

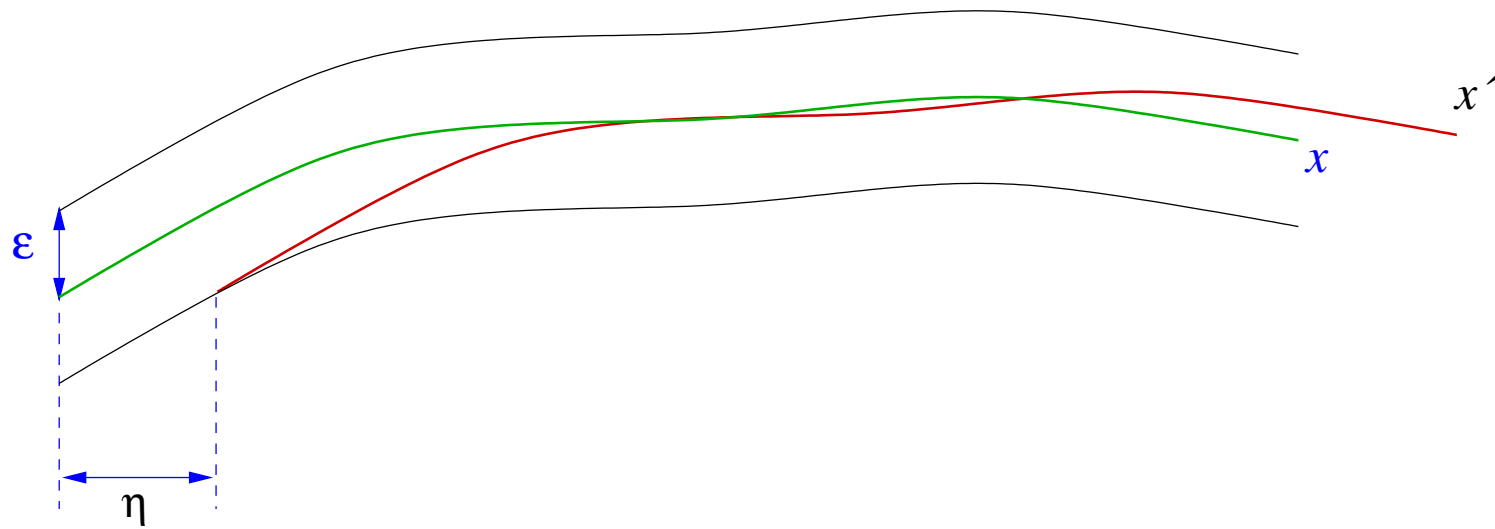
L_∞ Distance



$$\|x - x'\|_\infty \leq \epsilon$$

accounts for bounded modelling errors and perturbations

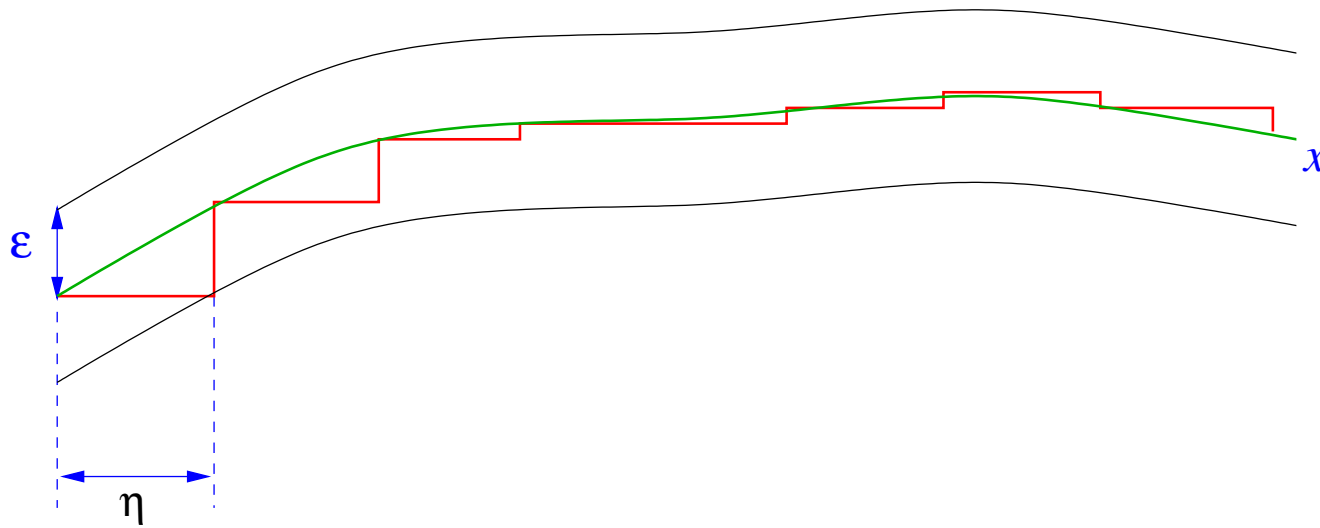
L_∞ Distance



$$\left\{ \begin{array}{l} \tau \leq \eta \Rightarrow \|x - \Delta^\tau x\|_\infty \leq \epsilon \\ \text{where } \Delta^\tau x(t) = x(t - \tau) \end{array} \right.$$

accounts for bounded jitter and communication delays

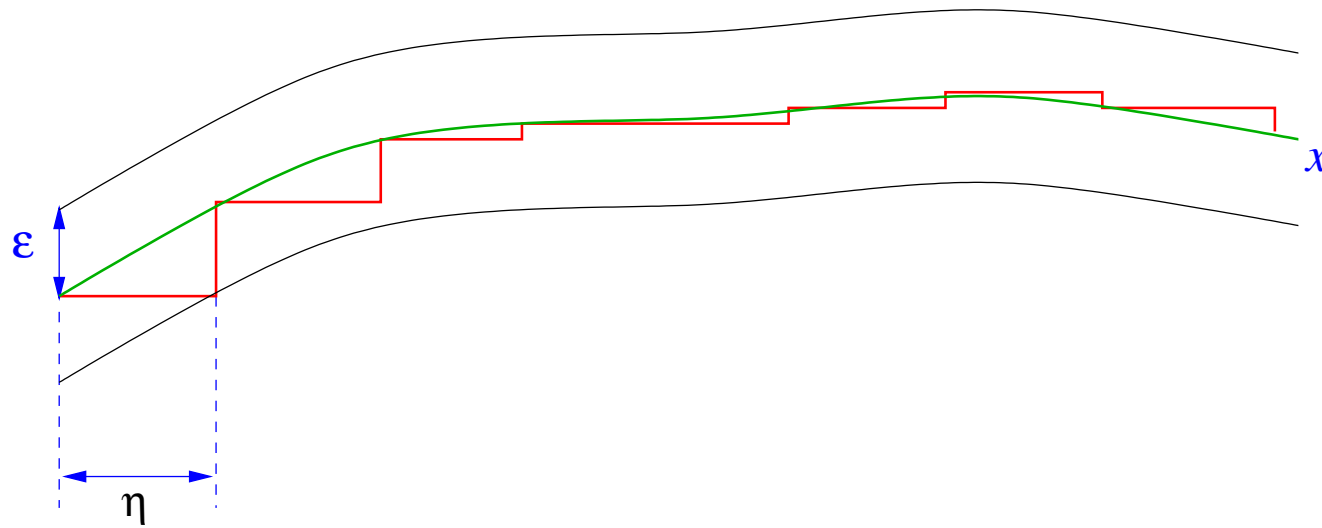
L_∞ Distance



$$\begin{cases} \tau \leq \eta \Rightarrow \|x - S_\tau x\|_\infty \leq \epsilon \\ \text{where } S_\tau x(t) = x(\lfloor \frac{t}{\tau} \rfloor \tau) \end{cases}$$

accounts for periodic sampling

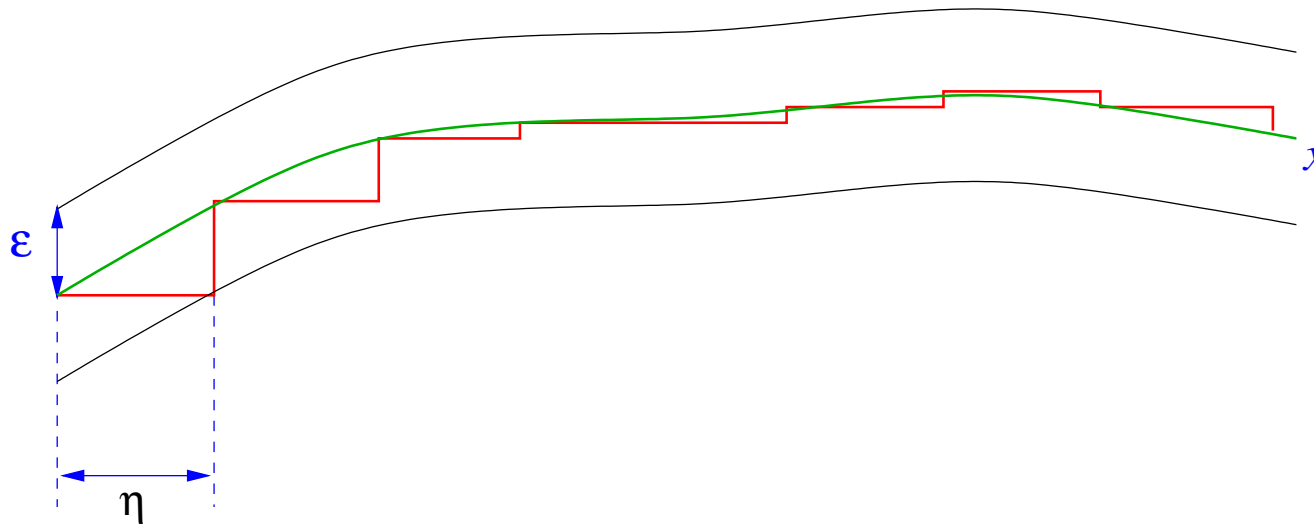
L_∞ Distance



$$\left\{ \begin{array}{l} \|r - Id\|_\infty \leq \eta \Rightarrow \|x - x \circ r\|_\infty \leq \epsilon \\ \text{where } Id(t) = t \\ f \circ g(t) = f(g(t)) \end{array} \right.$$

accounts for communication delays, jitter and sampling; r is called a retiming function

Delay-Error Function

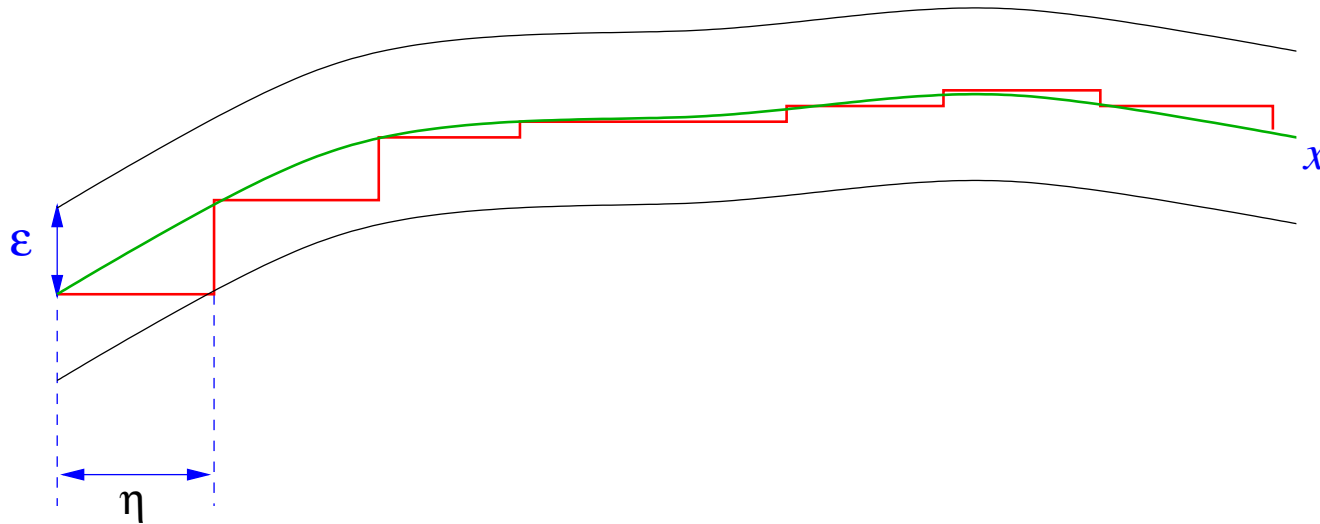


We can introduce the delay-error function η_x :

$$\left\{ \begin{array}{l} \|r - Id\|_{\infty} \leq \eta_x(\epsilon) \Rightarrow \|x - x \circ r\|_{\infty} \leq \epsilon \\ \text{where } Id(t) = t \\ f \circ g(t) = f(g(t)) \end{array} \right.$$

accounts for communication delays, jitter and sampling

Extension to Any Metric

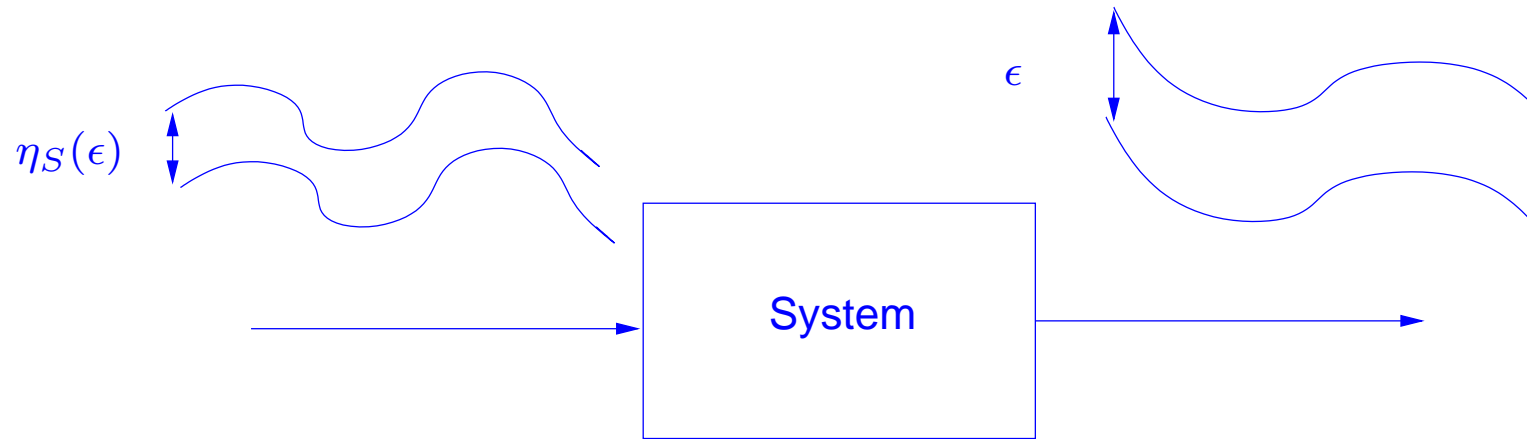


We can introduce the delay-error function η_x :

$$\left\{ \begin{array}{l} \|r - Id\|_{\infty} \leq \eta_x(\epsilon) \Rightarrow \|x - x \circ r\| \leq \epsilon \\ \text{where } Id(t) = t \\ f \circ g(t) = f(g(t)) \end{array} \right.$$

Extension to Systems

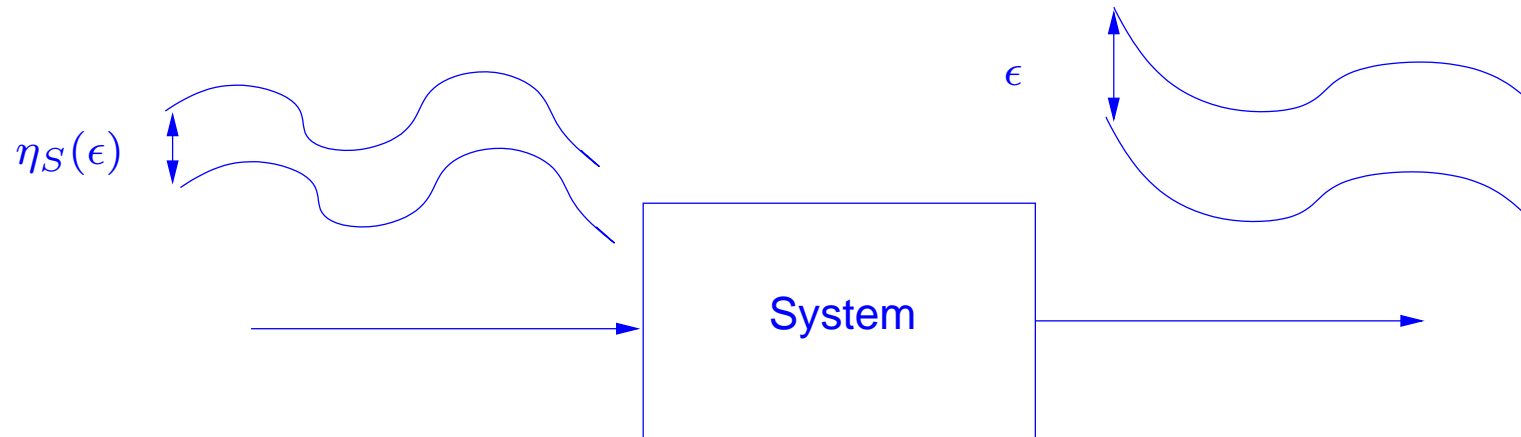
Uniformly Continuous Systems



$$\exists \eta_S > 0, \forall \epsilon > 0, \forall x, x' \quad \|x - x'\| \leq \eta_S(\epsilon) \Rightarrow \|Sx - Sx'\| \leq \epsilon$$

Extension to Systems

Uniformly Continuous Systems



$$\exists \eta_S > 0, \forall \epsilon > 0, \forall x, x' \|x - x'\|_\infty \leq \eta_S(\epsilon) \Rightarrow \|Sx - Sx'\|_\infty \leq \epsilon$$

A UC time invariant system, fed with a UC input yields a UC output

Proof:

Given x UC, S UC, and $\epsilon > 0$, $\forall x'$,

$$\|x - x'\|_\infty \leq \eta_S(\epsilon) \Rightarrow \|(S x) - (S x')\|_\infty \leq \epsilon$$

and $\forall \tau$,

$$|\tau| \leq \eta_x(\eta_S(\epsilon)) \Rightarrow \|x - (\Delta^\tau x)\|_\infty \leq \eta_S(\epsilon)$$

Thus, $\forall \tau$,

$$|\tau| \leq \eta_x(\eta_S(\epsilon)) \Rightarrow \|(S x) - (S (\Delta^\tau x))\|_\infty \leq \epsilon$$

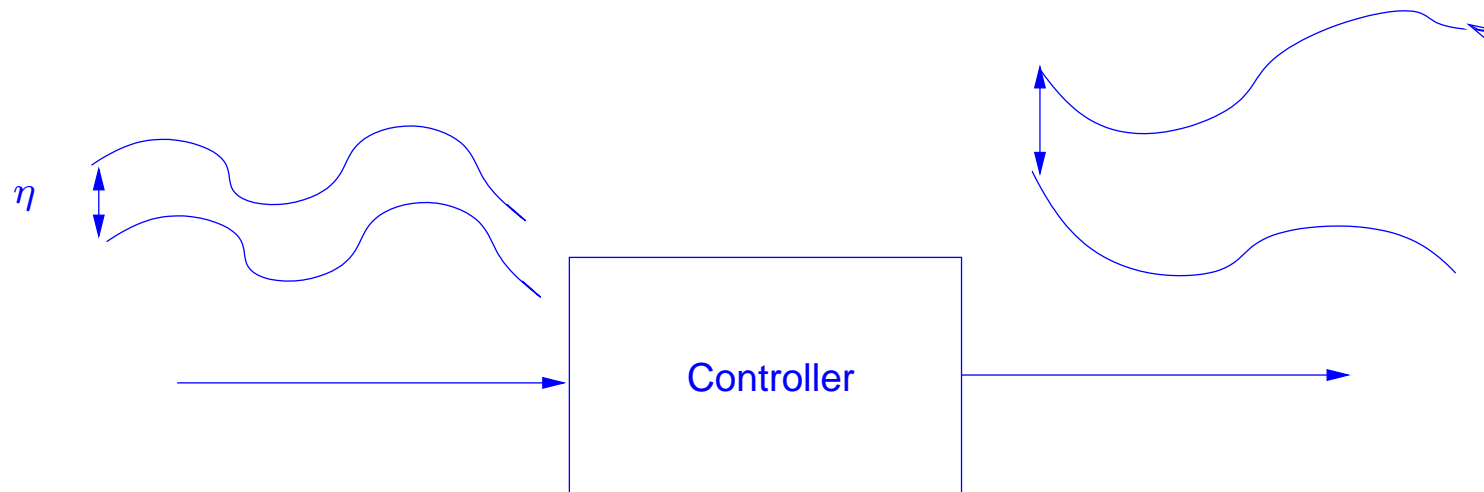
But $S(\Delta^\tau x) = \Delta^\tau(S x)$. We thus get

$$\eta_{Sx} = \eta_x \circ \eta_S$$

Extension to Systems

Actually, things are a bit more complicated as most systems are not UC:

Example : PID

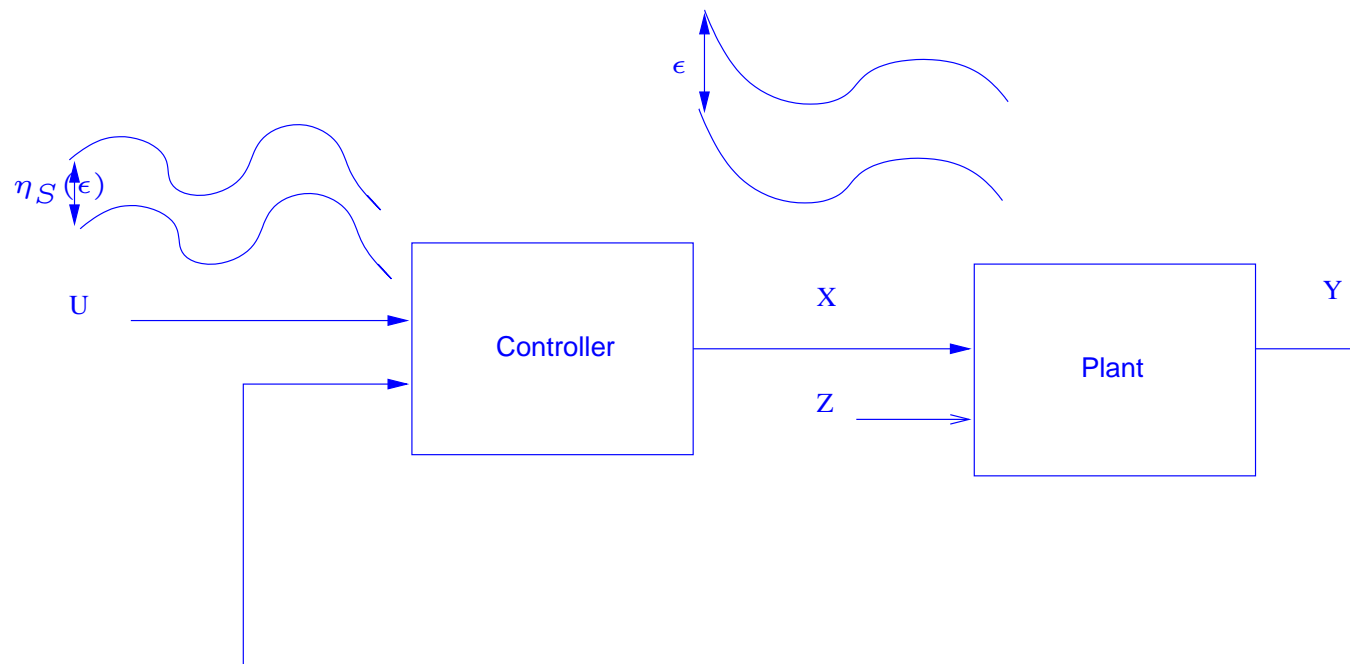


For instance unstable ones

Extension to Systems

Actually, things are a bit more complicated as most systems are not UC:

They become UC in closed loop with the environment



for instance an unstable controller can stabilise an unstable environment.

Conclusion

We need an approximation theory of embedded control systems in order to assess their robustness with respect to implementation details :

- Sampling
- Delays
- Priorities, distribution
- ...

Uniform continuity seems a good framework. Using an abstract functional approach allows us to parametrise it with suitable metrics and even topologies.
