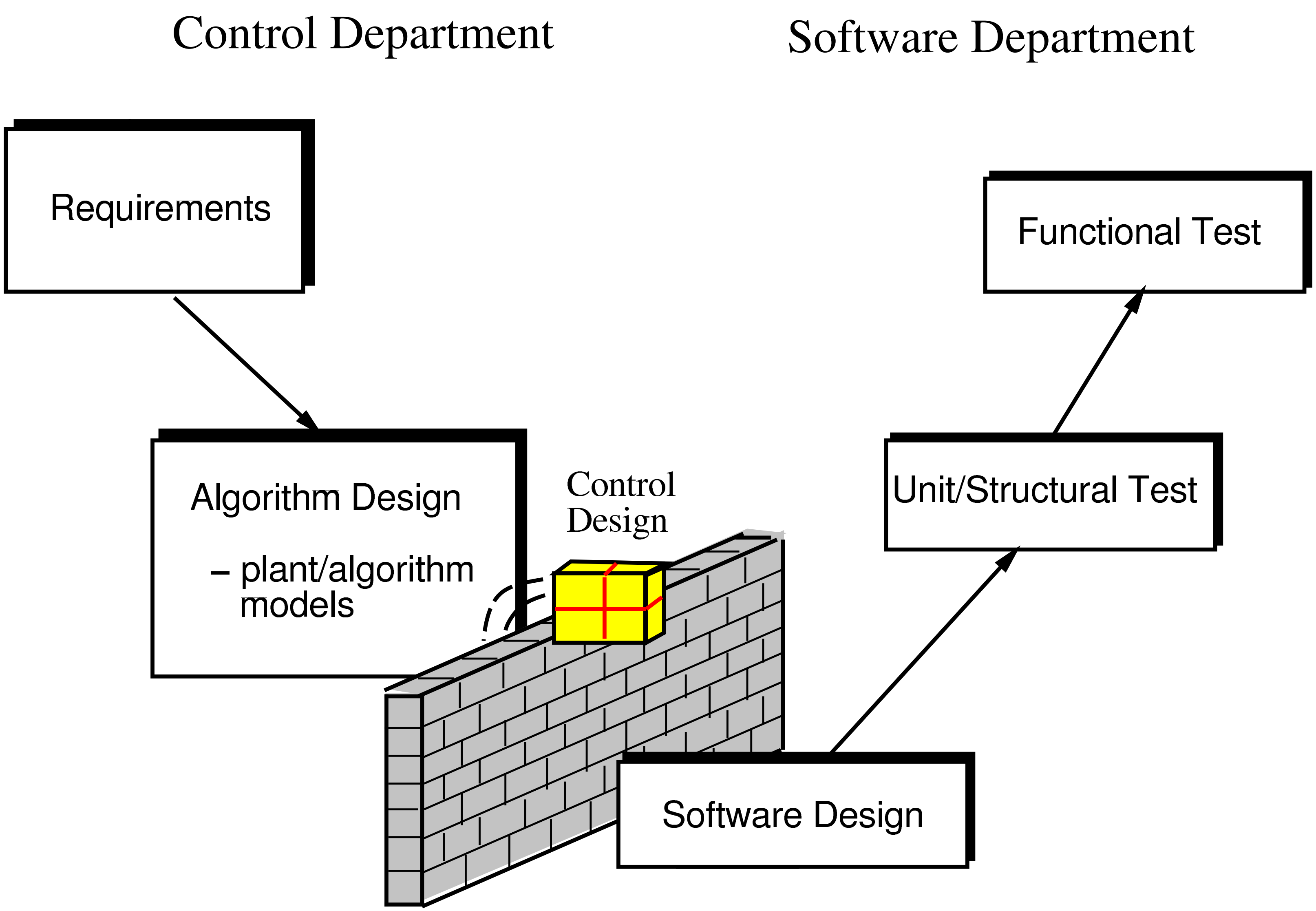
Integrated Control and Scheduling

Anton Cervin & Karl-Erik Årzén
Lund University

Lecture 2 outline

- **Introduction**
- Analysis of controller timing
- Scheduling to reduce delay and jitter

Control system development today



Problems

- The control engineer does not care about the implementation
  - "trivial"
  - "buy a fast computer"
- The software engineer does not understand controller timing
  - "$\tau_i = (T_i, D_i, C_i)$"
  - "hard deadlines"
- Control theory and real-time scheduling theory have evolved as separate subjects for thirty years

In the beginning...

Liu and Layland (1973): "Scheduling algorithms for multiprogramming in a hard-real-time environment." *Journal of the ACM*, **20**:1.

- Rate-monotonic (RM) scheduling
- Earliest-deadline-first (EDF) scheduling
- Motivated by process control
  - Samples "arrive" periodically
  - Control response computed before end of period
  - "Any control loops closed within the computer must be designed to allow at least an extra unit sample delay."

Common assumptions about control tasks

In the simple task model, a task τ_i is described by

- a fixed period T_i
- a fixed, known worst-case execution time C_i
- a hard relative deadline $D_i = T_i$

Is this model suitable for control tasks?

Fixed period?

Not necessarily:

- Different sampling periods could be appropriate for different operating modes
- Some controllers are not sampled against time but are invoked by *events*
- The sampling period could be adjusted on-line by a resource manager (“feedback scheduling”)

Fixed and known WCET?

Not always:

- WCET analysis is a very hard problem
  - May have to use estimates or measurements
- Some controllers switch between modes with very different execution times
  - Hybrid controllers
- Some controllers can explicitly trade off execution time for quality of control
  - "Any-time" optimization algorithms, e.g. model-predictive control (MPC)
  - Long execution time $\Rightarrow$ high quality of control

Hard deadlines?

Often not:

- Controller deadlines are often **firm** rather than hard
  - Often OK to miss a few outputs, but not too many in a row
  - Depends on what happens when a deadline is missed:
    - Task is allowed to complete late – often OK
    - Task is aborted at the deadline – worse
- At the same time, meeting all deadlines does not guarantee stability of the control loop
  - $D_i = T_i$ is motivated by runability conditions only

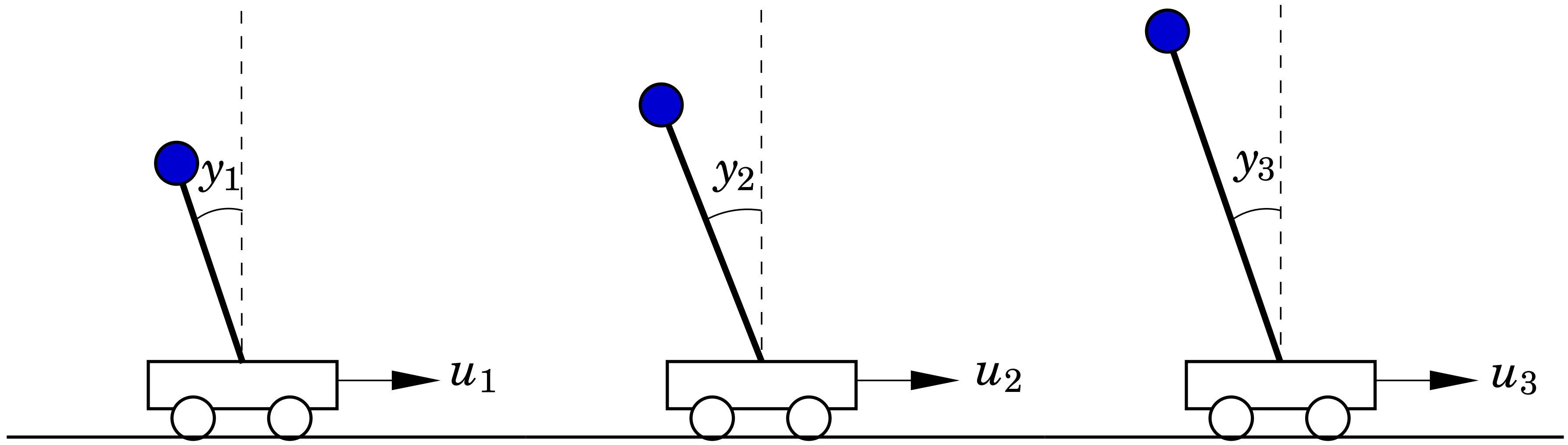
Inputs and outputs?

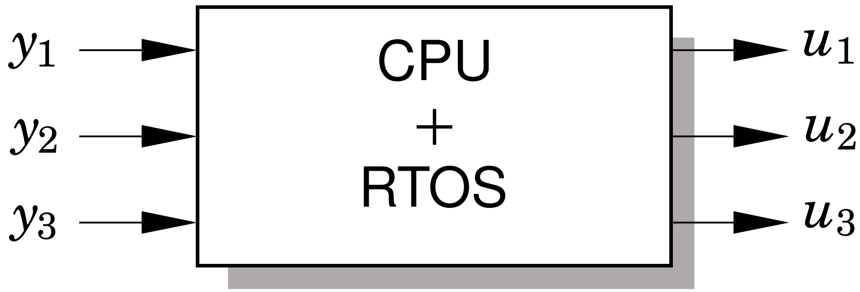
Completely missing from the simple task model:

- When are the inputs (measurement signals) read?
  - Beginning of period?
  - When the task starts?
- When are the outputs (control signals) written?
  - When the task finishes?
  - End of period?

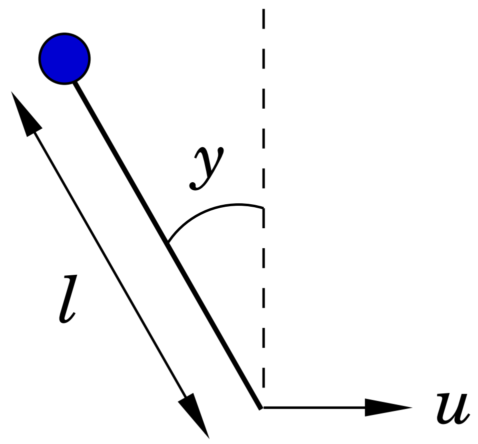
Inverted pendulum example

Control of three inverted pendulums using one CPU:





The pendulums



A simple second-order model is given by

$$\frac{d^2y}{dt^2} = \omega_0^2 \sin y + u\,\omega_0^2 \cos y$$

where $\omega_0 = \sqrt{\frac{g}{l}}$ is the natural frequency of the pendulum.

Lengths $l = \{1, 2, 3\}$ cm $\Rightarrow$ $\omega_0 = \{31, 22, 18\}$ rad/s

Control design

Linearization around the upright equilibrium gives the state-space model

$$\frac{dx}{dt} = \begin{pmatrix} 0 & 1 \\ \omega_0^2 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ \omega_0^2 \end{pmatrix} u$$
$$y = \begin{pmatrix} 1 & 0 \end{pmatrix} x$$

- Model sampled using periods $h = \{10, 14.5, 17.5\}$ ms
- Controllers based on state feedback from observer, designed using pole placement

Control design, Cont'd

- State feedback poles specified in continuous time as

$$s^2 + 1.4\omega_c s + \omega_c^2 = 0$$

$\omega_c = \{53, 38, 31\}$ rad/s

- Observer poles specified in continuous time as

$$s^2 + 1.4\omega_o s + \omega_o^2 = 0$$

$\omega_o = \{106, 75, 61\}$ rad/s

Implementation

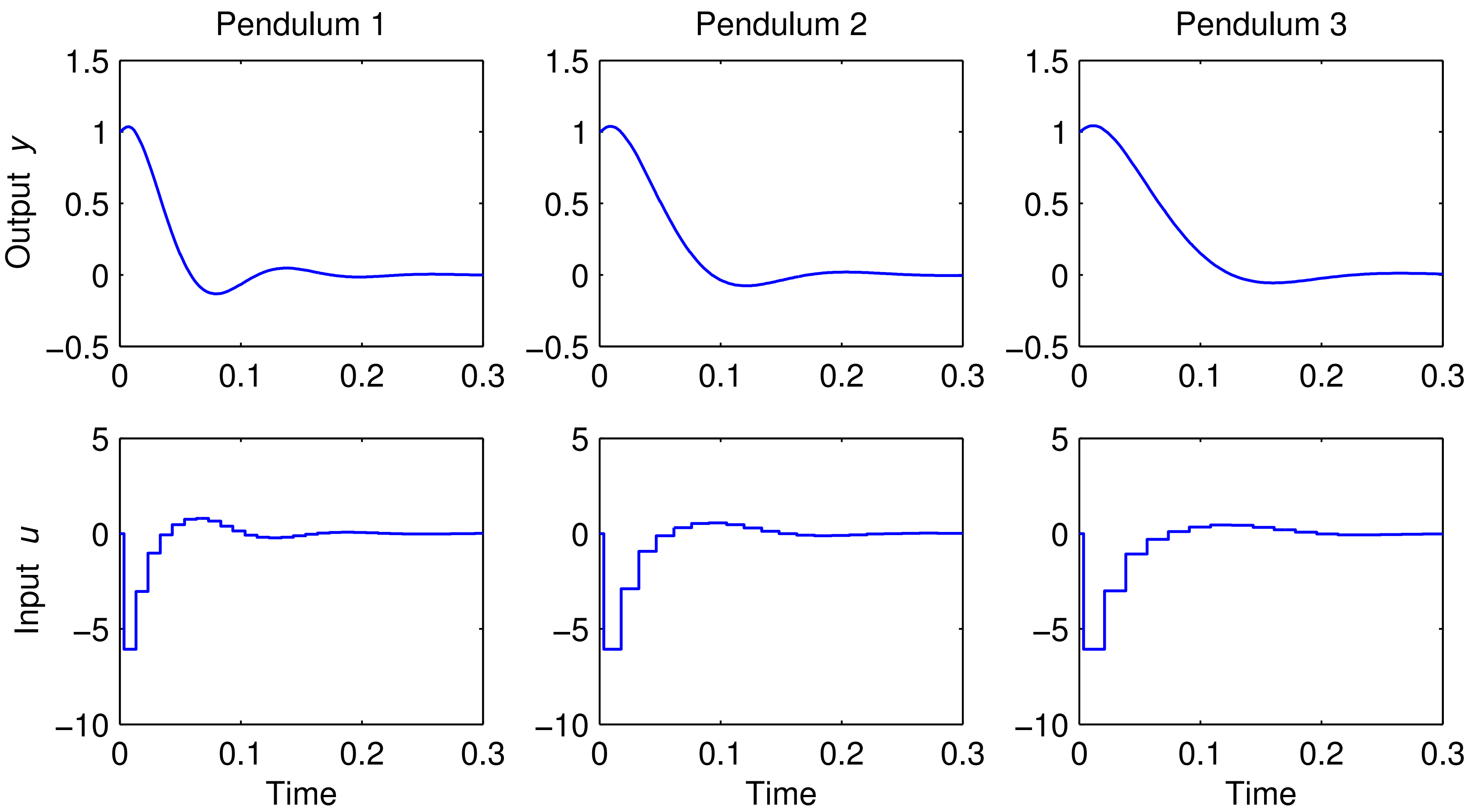
- A periodic timer interrupt samples the plant output and triggers control task
- Each controller i is implemented as a periodic task:

```
t = CurrentTime();
LOOP
   y := AnalogIn();
   u := CalculateControl(y);
   AnalogOut(u);
   t = t + h;
   WaitUntil(t);
END
```

- Assumed execution time: $C = 3.5$ ms

Simulation 1 – Ideal case

Each controller runs on a separate CPU.



Schedulability analysis

- Assume $D_i = T_i$
- CPU utilization $U = \sum_{i=1}^{3} \frac{C_i}{T_i} = 0.79$
- Schedulable under EDF, since $U < 1$
- Schedulable under RM?

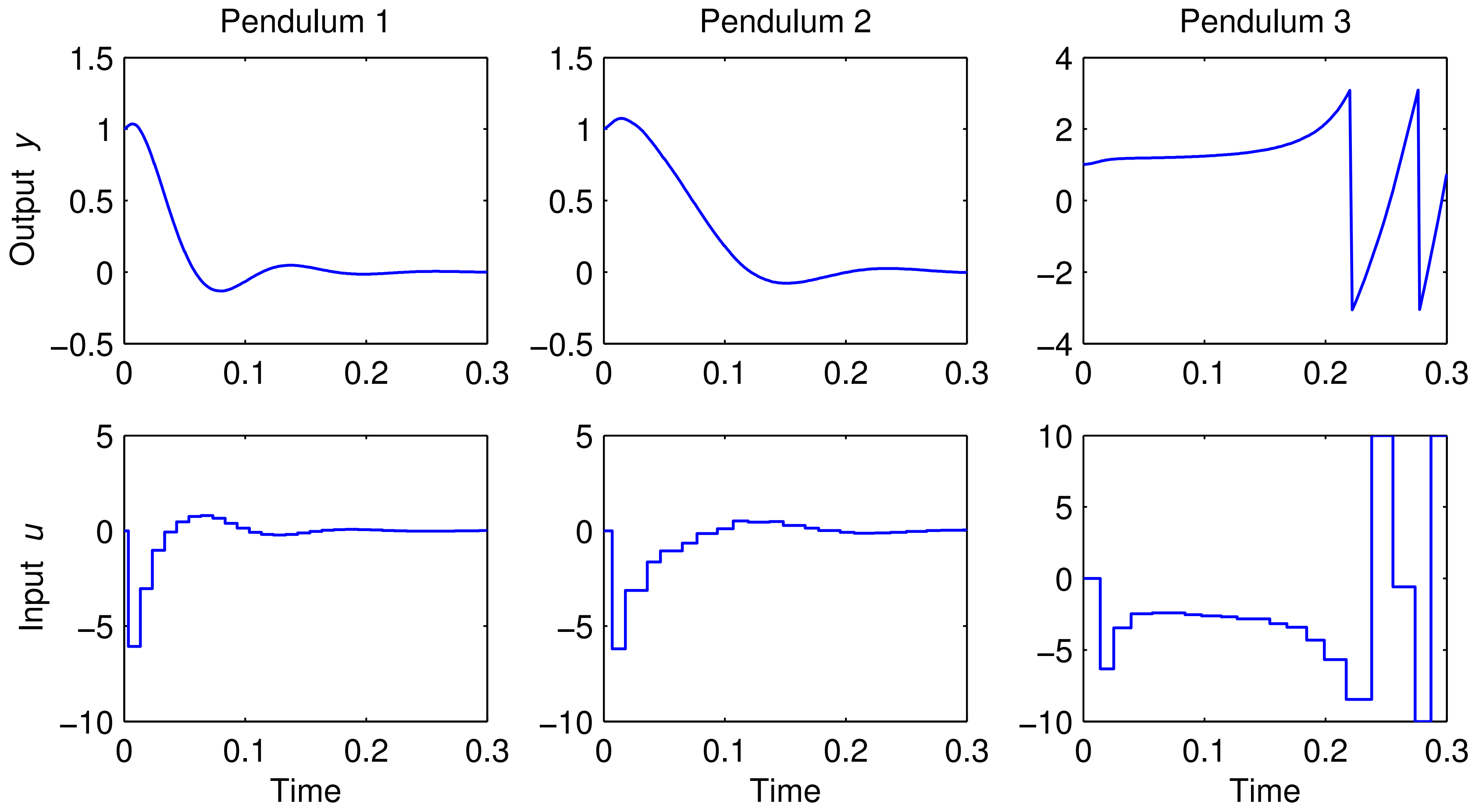
$$U > 3(2^{1/3} - 1) = 0.78 \quad \Rightarrow \quad \text{Cannot say}$$

Compute worst-case response times R_i:

Task	T	D	C	R
1	10	10	3.5	3.5
2	14.5	14.5	3.5	7.0
3	17.5	17.5	3.5	14.0

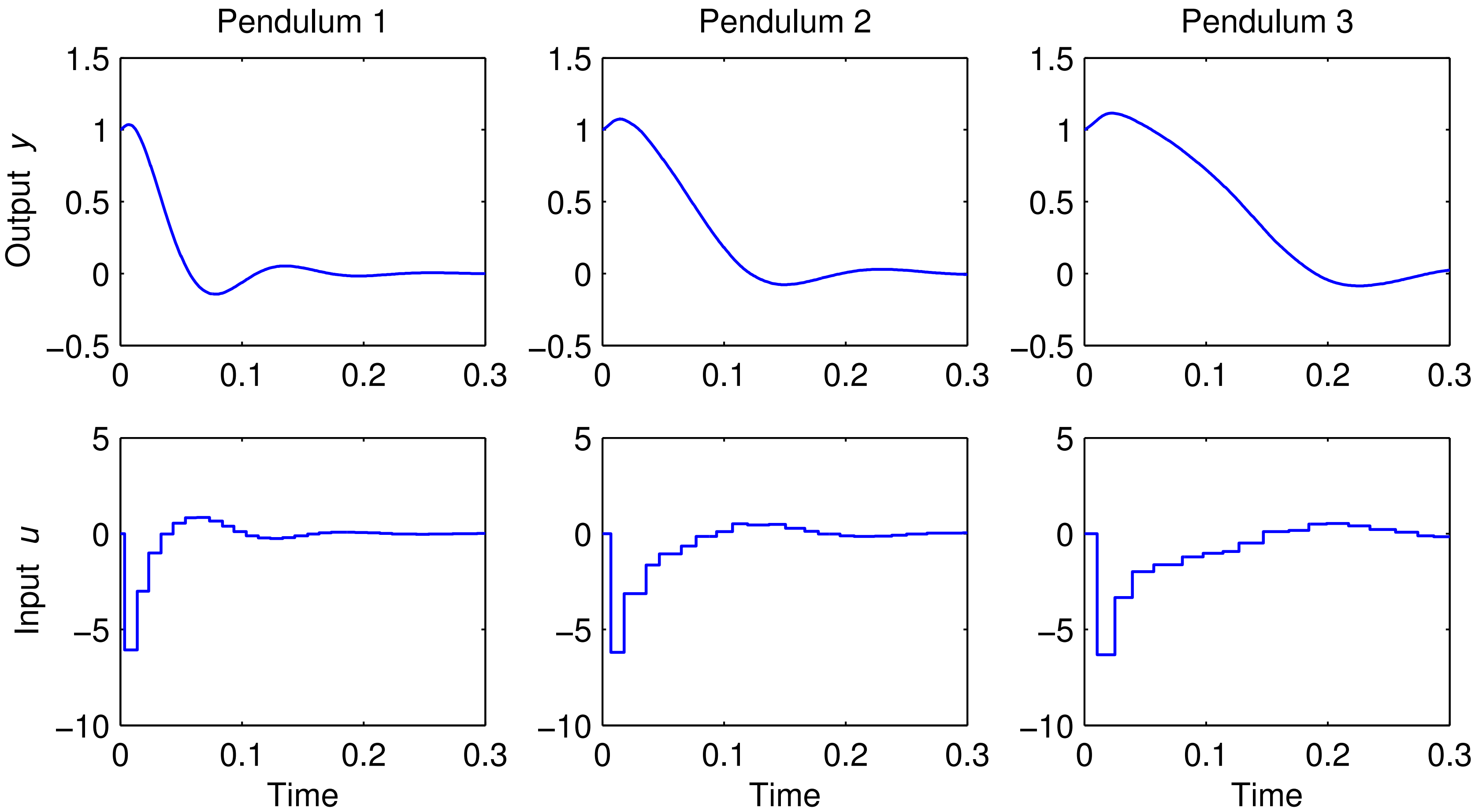
$$\forall i: \; R_i < D_i \;\; \Rightarrow \;\; \text{Yes}$$

Simulation 2 – Rate-monotonic scheduling



- Loop 3 becomes unstable

Simulation 3 – Earliest-deadline-first scheduling



- All loops are OK

Questions

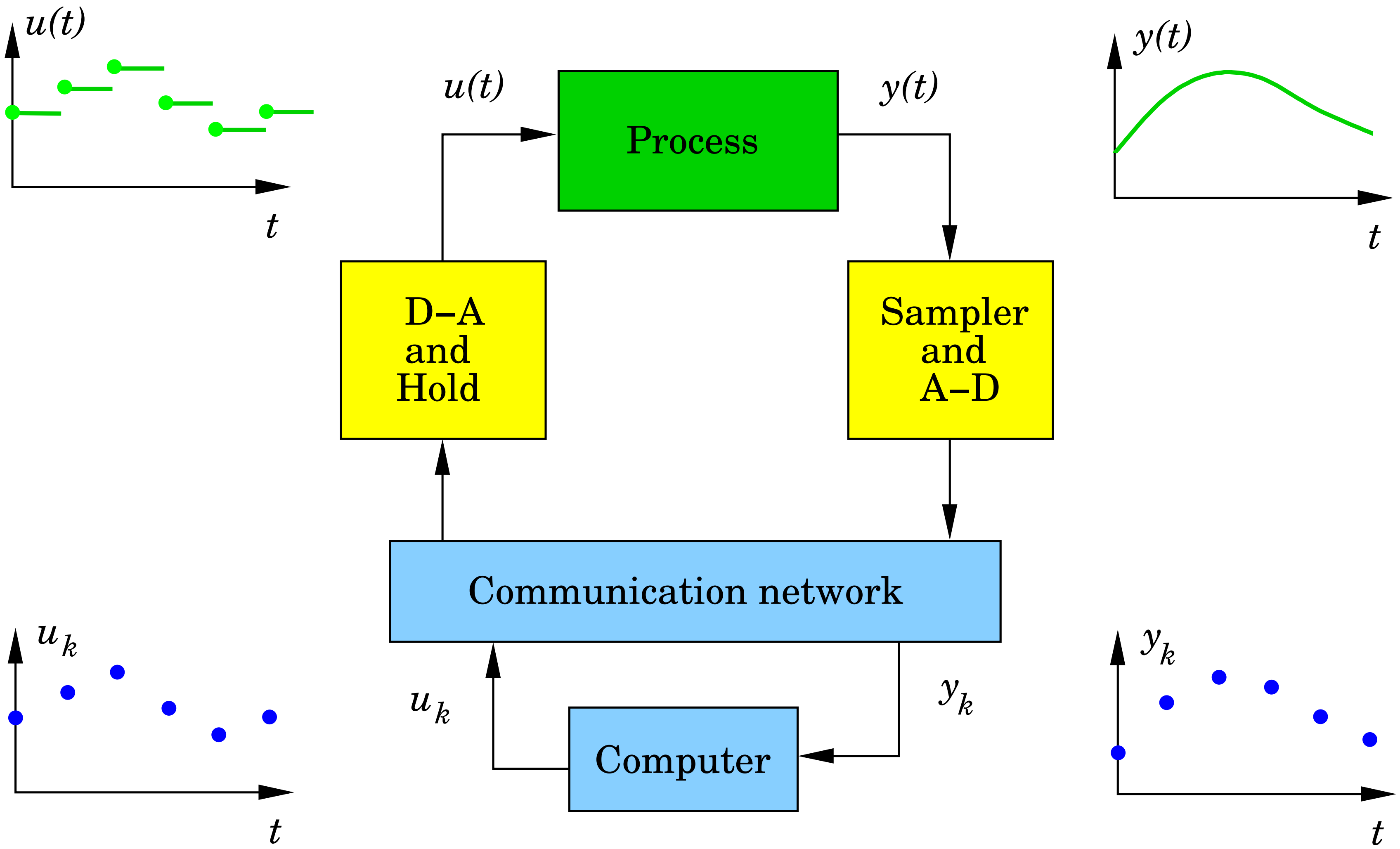
- How can a loop become unstable even though the system is schedulable?
- Why does EDF work better than RM?

Need to study control loop timing

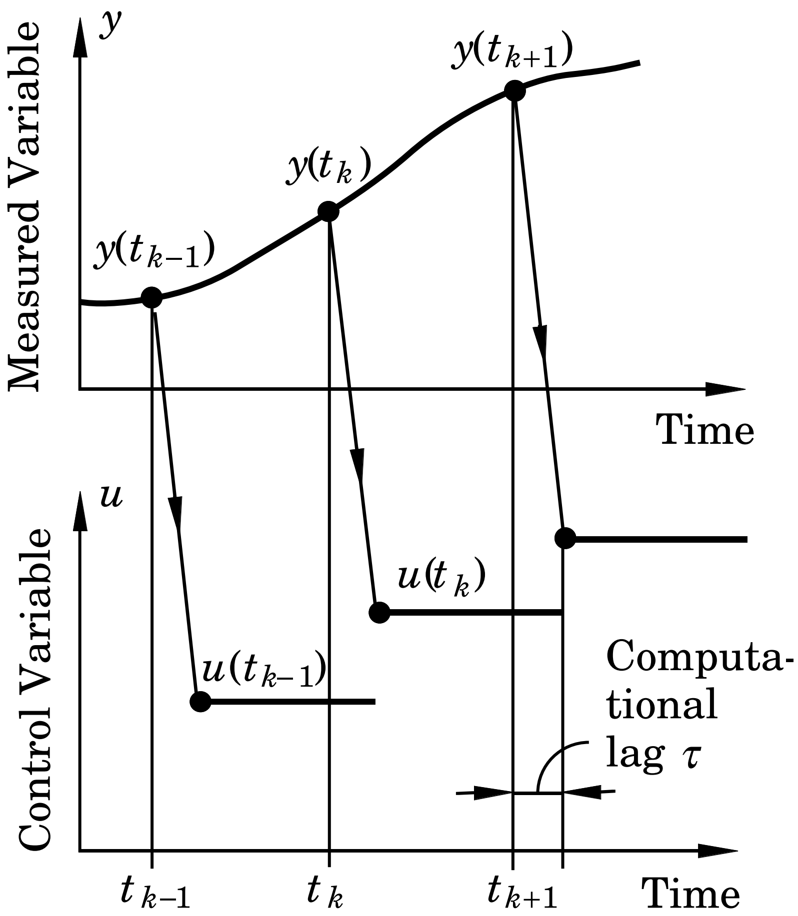
Lecture 2 outline

- Introduction
- **Analysis of controller timing**
- Scheduling to reduce delay and jitter

Sampled-data (networked) control systems



Ideal controller timing

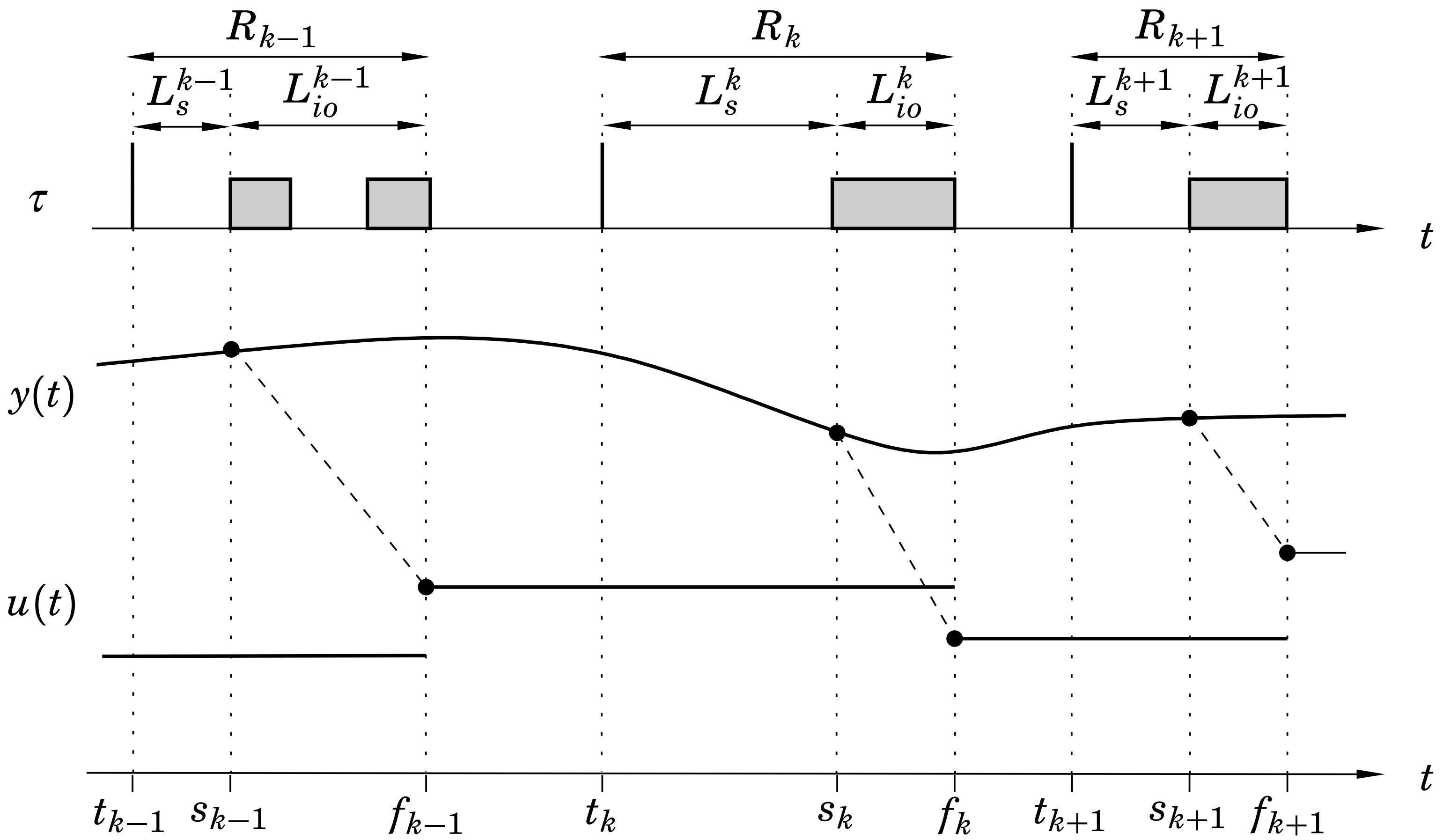


- Process output y sampled periodically at time instants $t_k = kh$
- Control u applied after short and constant time delay τ

Sources of nondeterminism

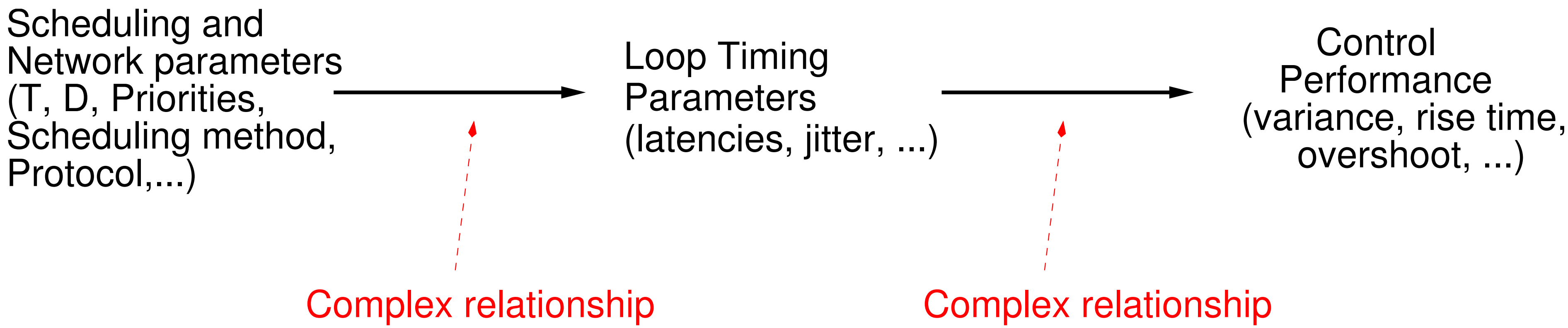
- Jitter in the sampling operation due to poor time resolution or preemption
- Variable communication delay due to the medium access control or the communication protocol
- Variable computational delay due to variable execution time or preemption
- Jitter in the actuation

More realistic controller timing



- Control task τ released at periodic time instances $t_k = kh$
- Output y sampled after time-varying sampling latency L_s
- Control u generated after time-varying input-output latency L_{io}

Real-time and control analysis



Analysis of controller timing

1. Sampling period (h)
2. Control delay (average value of L_{io})
3. Jitter (variability in L_s and L_{io})

1. Sampling interval

Theoretically, the shorter the sampling interval, the better the performance

- When $h \to 0$, we approach continuous (analog) control

Practically, there is a limit as to how fast you can or want to sample

- Hardware limitations
- Limited computational resources
- Numerical problems
- Diminishing returns

Choice of sampling interval

Sampling frequency $\omega_s = 2\pi/h$

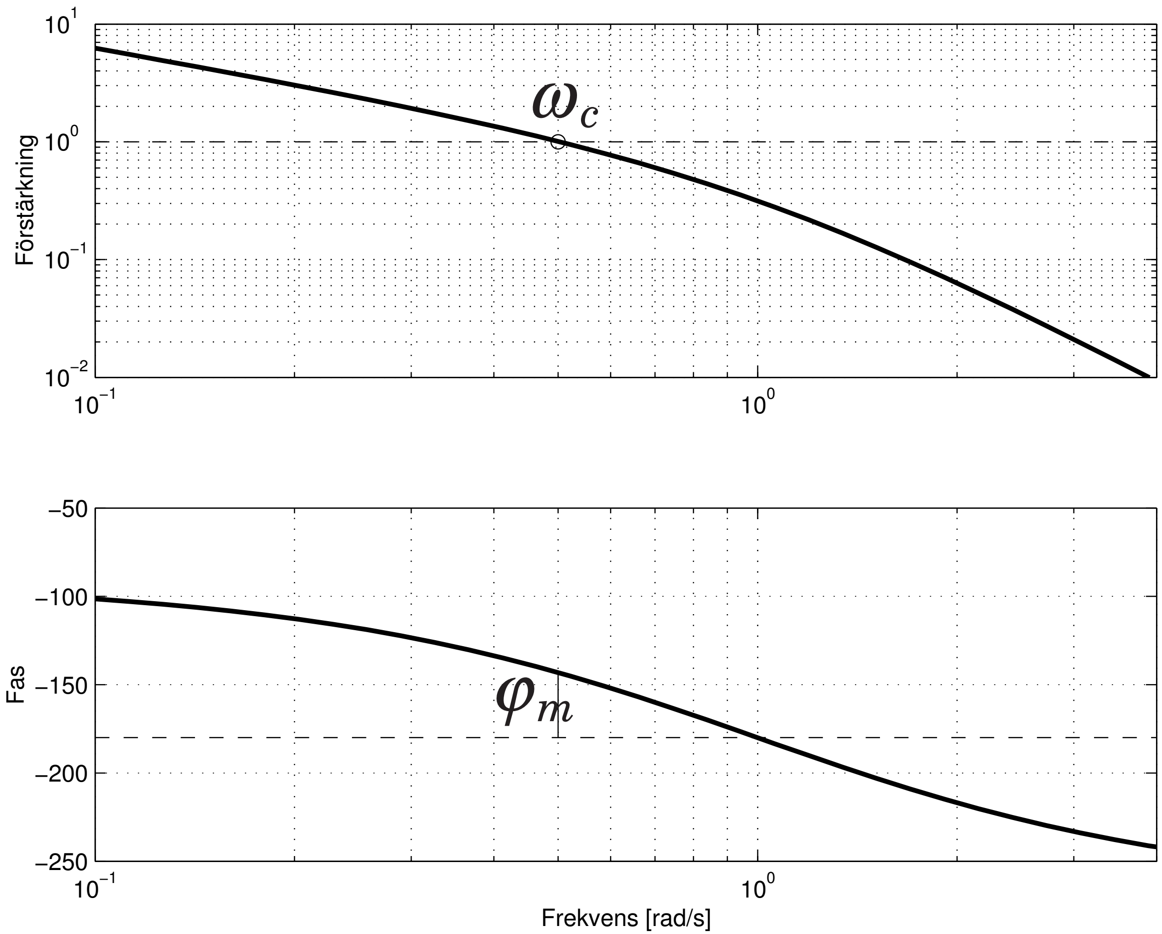
Nyquist frequency $\omega_N = \pi/h$

Nyquist's sampling theorem:

> We must sample at least twice as fast as the highest frequency we are interested in

- What frequencies are we interested in?

Typical loop transfer function $P(i\omega)C(i\omega)$:



- ω_c = cross-over frequency, φ_m = phase margin
- We should have Nyquist frequency $\omega_N \gg \omega_c$

Sampling interval rule of thumb

The sample-and-hold can be approximated by a delay of $h/2$:

$$G_{S\&H}(s) \approx e^{-sh/2}$$

This will decrease the phase margin by

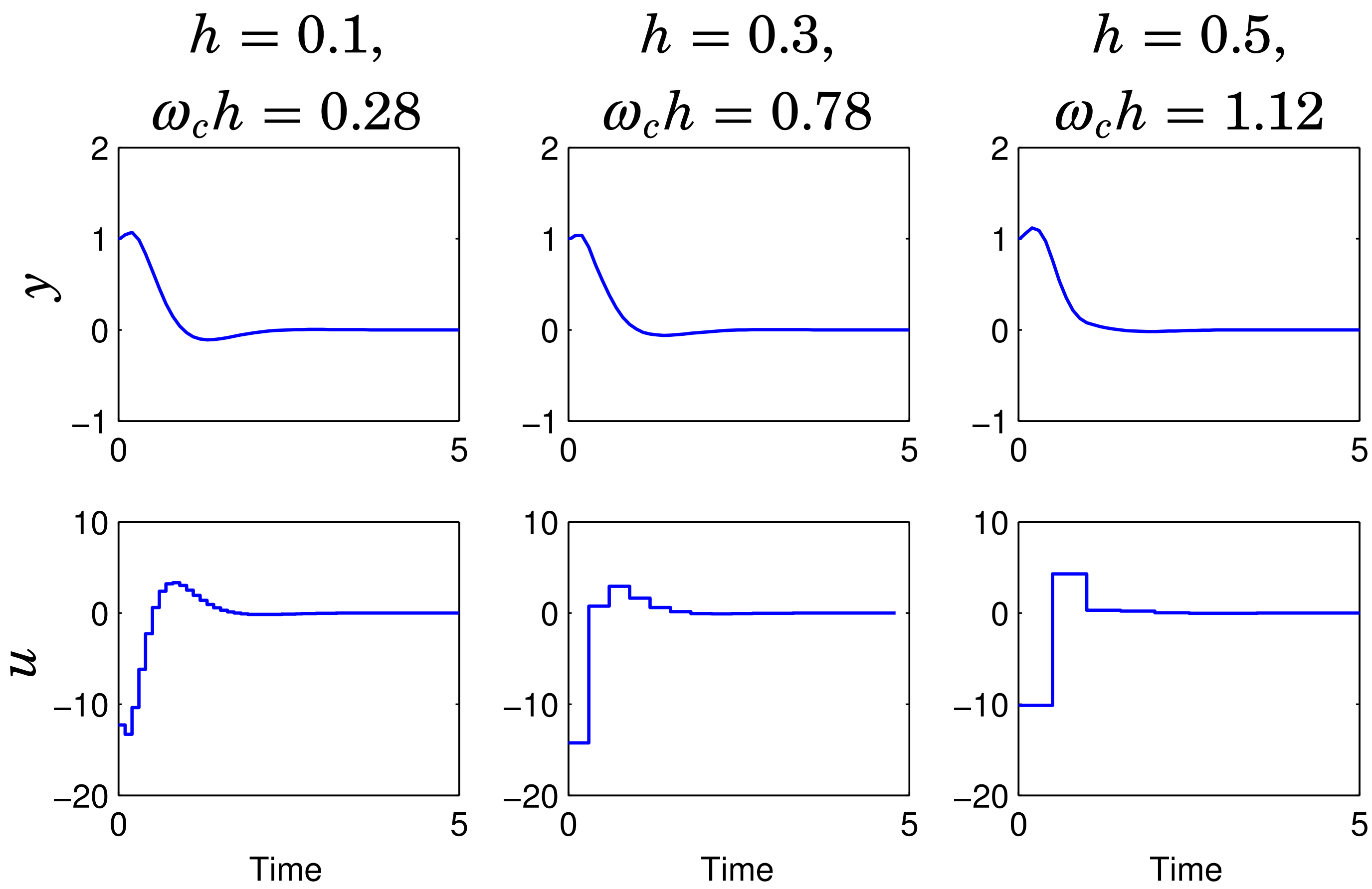
$$\arg G_{S\&H}(i\omega_c) = \arg e^{-i\omega_c h/2} = -\omega_c h/2$$

Assume we can accept a phase loss between 5° and 15°. Then

$$\boxed{0.15 < \omega_c h < 0.5}$$

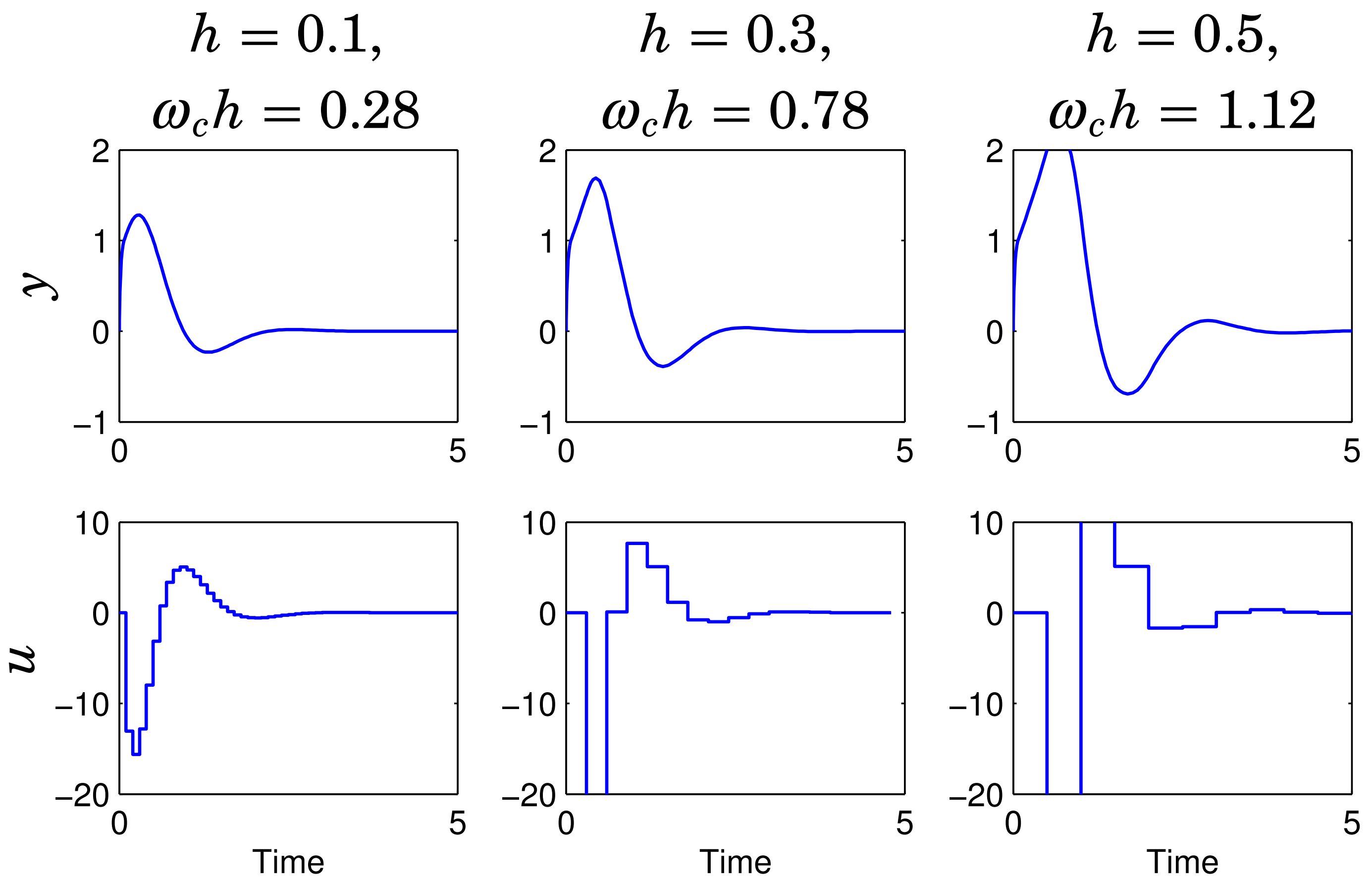
This corresponds to a Nyquist frequency about 6 to 20 times larger than the crossover frequency

Example: control of inverted pendulum



- Large $\omega_c h$ may seem OK, but beware!
  - Digital design assuming perfect model
  - Controller perfectly synchronized with initial disturbance

Pendulum with non-synchronized disturbance



Recall that the controller runs in open loop between samples

2. Control delay

The shorter the delay, the better the achievable performance

Sources of time delays:

- Deadtime in the process: tubes, pipes, conveyor belts
- Deadtime in the controller implementation: computational delay, communication delay

From a theoretical perspective, all (constant) delays in the loop can be lumped into a single control delay τ

Control delay decreases the phase margin

Phase margin loss due to delay τ:

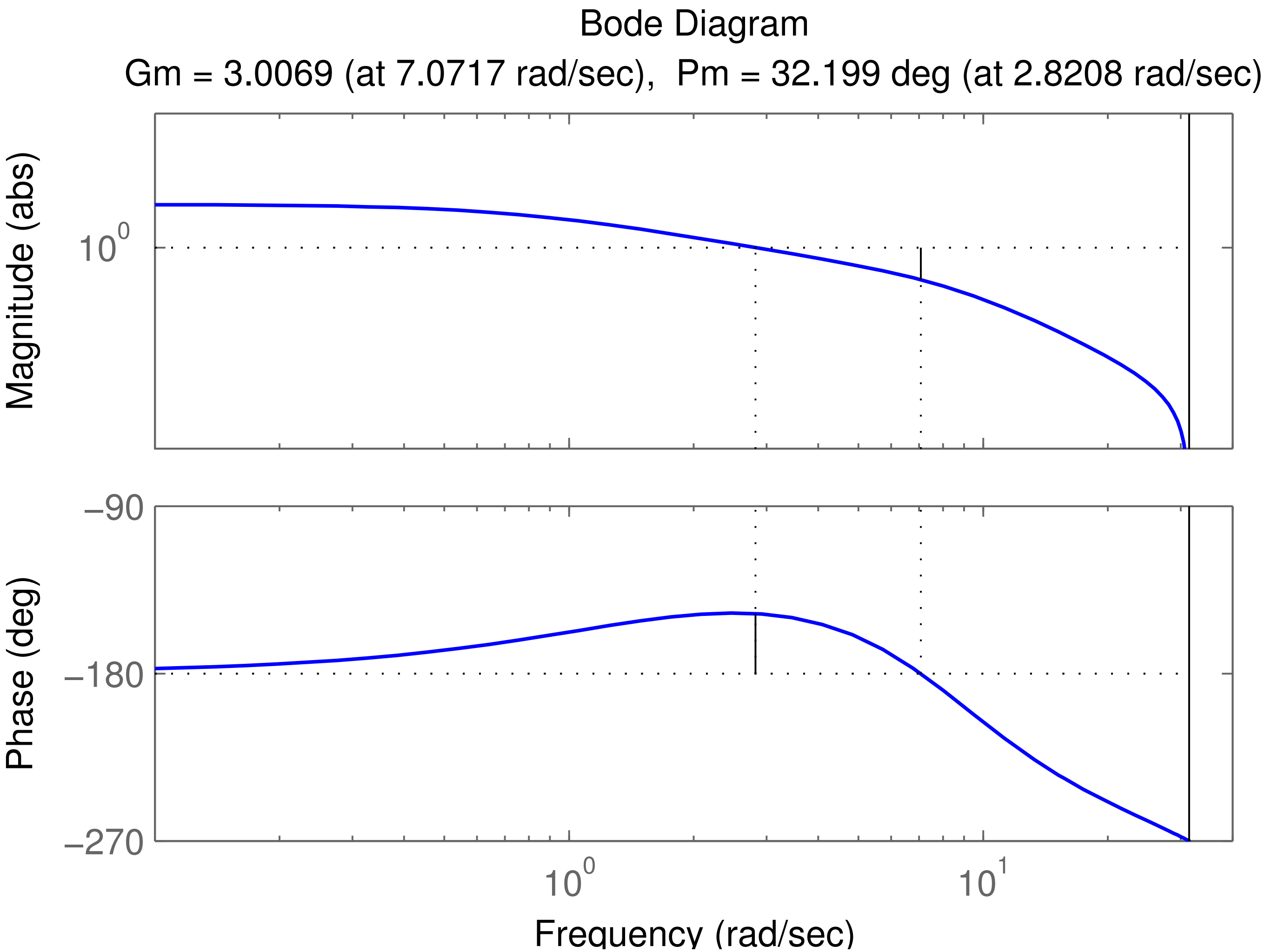
$$\arg e^{-i\omega_c\tau} = -\omega_c\tau$$

Closed-loop system stable if

$$\omega_c\tau < \varphi_m \quad \Leftrightarrow \quad \tau < \frac{\varphi_m}{\omega_c}$$

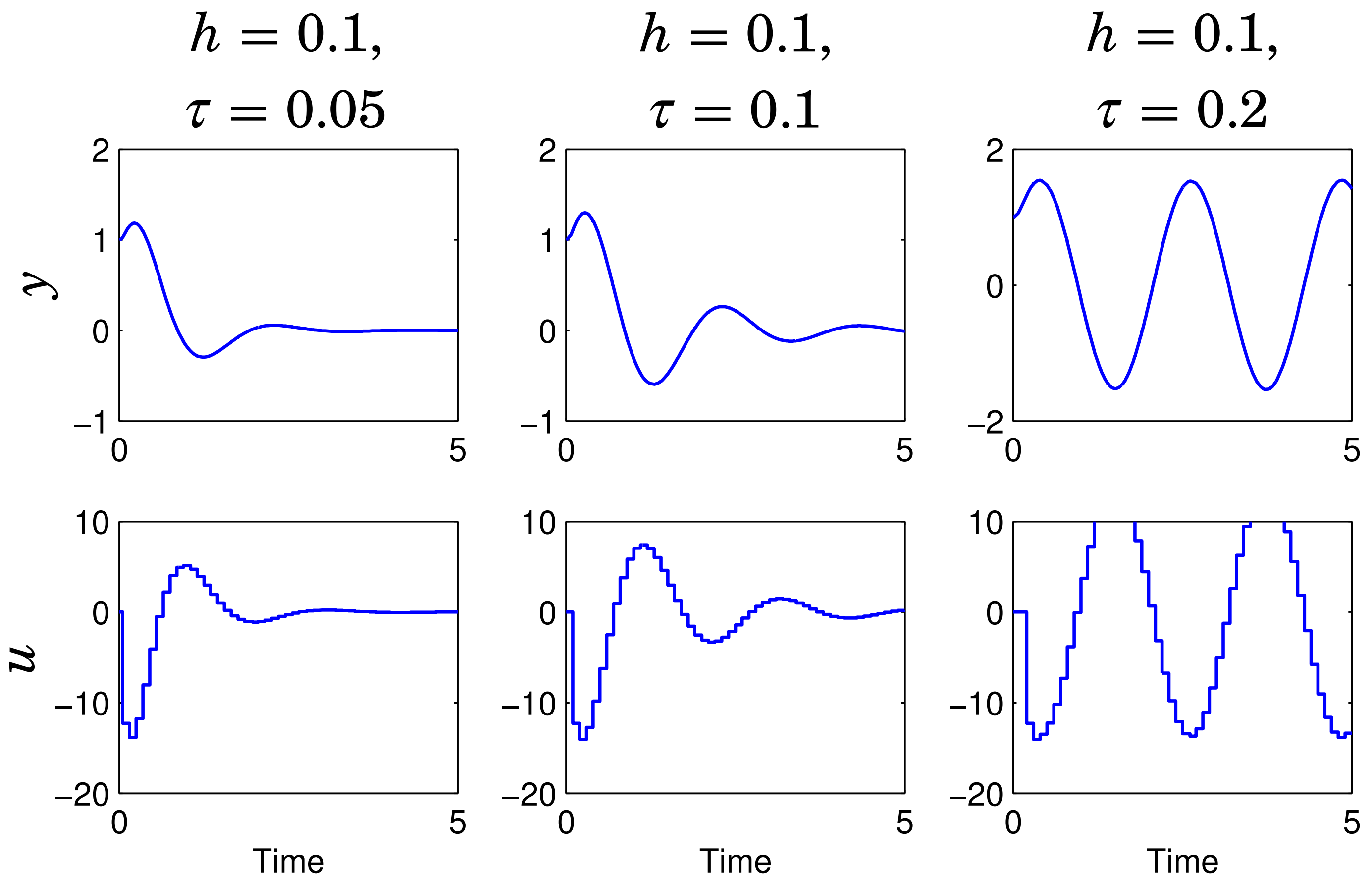
$\tau_m = \dfrac{\varphi_m}{\omega_c}$ is called the **delay margin**

Example: delay margin for pendulum controller



$$\varphi_m = 32^\circ,\ \omega_c = 2.8 \text{ rad/s} \ \Rightarrow\ \tau_m = \frac{32\pi}{180 \cdot 2.8} = 0.2$$

Pendulum controller with control delay



- No delay compensation

Delays in discrete time

Include the control delay in the process model:

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t-\tau), \qquad \tau < h$$

Sampling gives

$$x(kh+h) = \Phi x(kh) + \Gamma_1 u(kh-h) + \Gamma_0 u(kh)$$

where

$$\Gamma_1 = e^{A(h-\tau)} \int_0^{\tau} e^{As} B \, ds$$

$$\Gamma_0 = \int_0^{h-\tau} e^{As} B \, ds$$

State-space model (with extra state $z(kh) = u(kh - h)$)

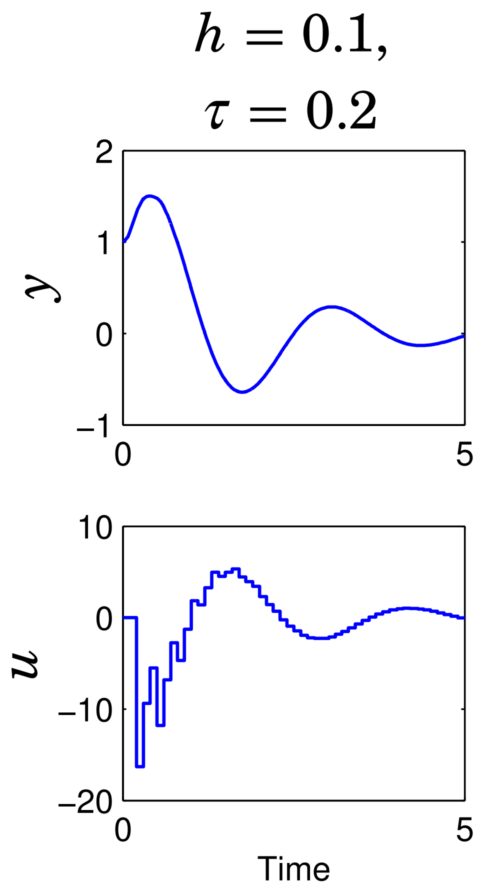
$$\begin{pmatrix} x(kh+h) \\ z(kh+h) \end{pmatrix} = \begin{pmatrix} \Phi & \Gamma_1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x(kh) \\ z(kh) \end{pmatrix} + \begin{pmatrix} \Gamma_0 \\ I \end{pmatrix} u(kh)$$

Can easily be extended to $\tau > h$

Design:

- Apply arbitrary discrete time design using the augmented model
- Remember that the delay imposes a fundamental performance limitation
  - Try to respect the rule of thumb $0.15 < \omega(h + 2\tau) < 0.5$

Pendulum controller with delay compensation



- Shaky response, but stable
- $\omega_c(h + 2\tau) = 1.4$

Minimizing the computational delay

A general controller in state-space representation:

$$
\begin{aligned}
x(k+1) &= Fx(k) + Gy(k) + G_r r(k) \\
u(k) &= Cx(k) + Dy(k) + D_r r(k)
\end{aligned}
$$

Do as little as possible between the input and the output:

```
r = ref.get();
y = yIn.get();
/* Calculate Output */
u := u1 + D*y + Dr*r;
uOut.put(u);
/* Update State */
x := F*x + G*y + Gr*r;
u1 := C*x;
```

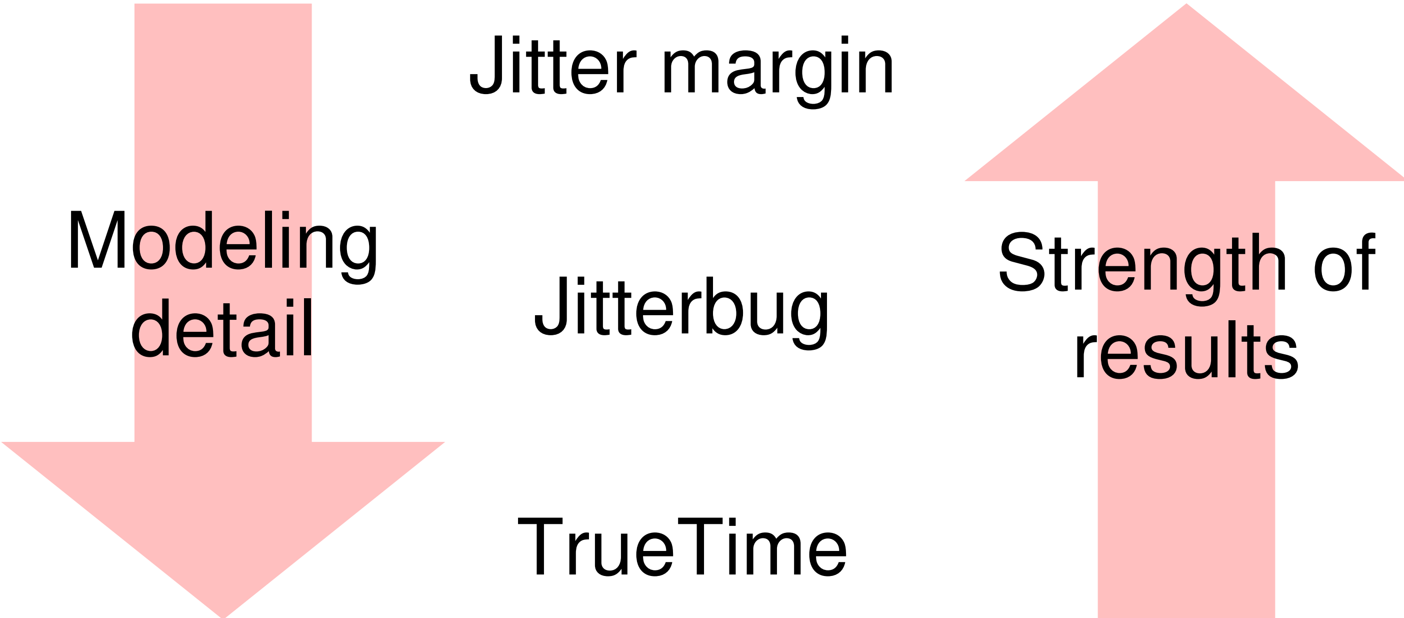
3. Jitter

More difficult to analyze and compensate for.

Some tools for jitter analysis:

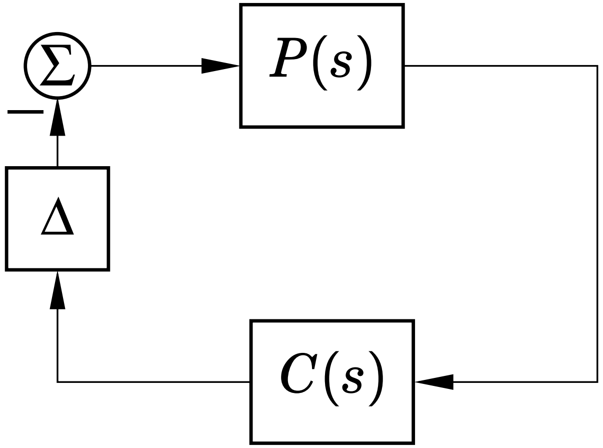
- Robust (worst-case) analysis – e.g. the Jitter margin
- Stochastic (average-case) analysis – e.g. the Jitterbug toolbox
- Simulation – e.g. the TrueTime simulator

Comparison of the tools



The jitter margin

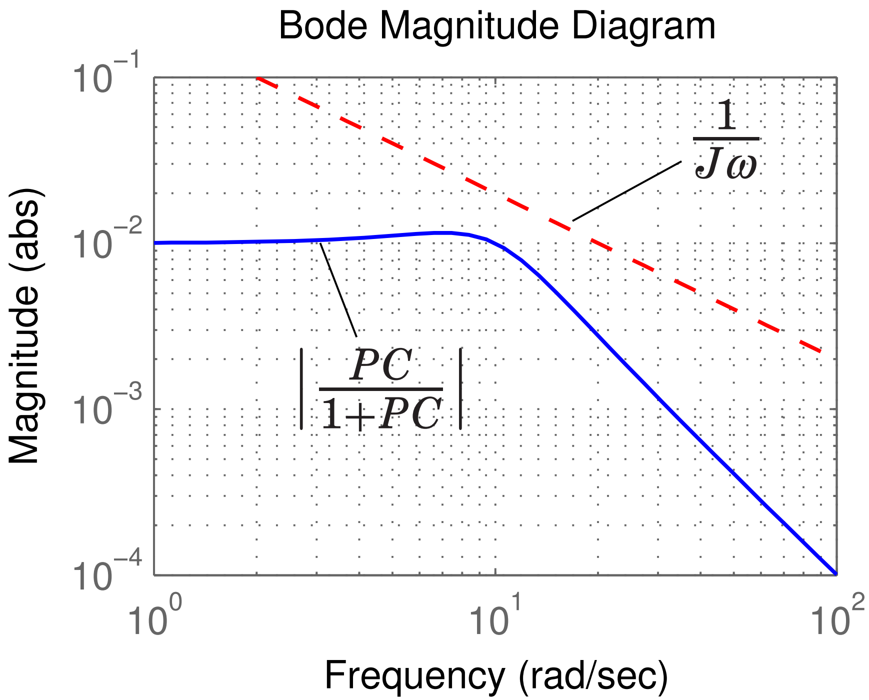
Stability result due to Kao and Lincoln (2004):



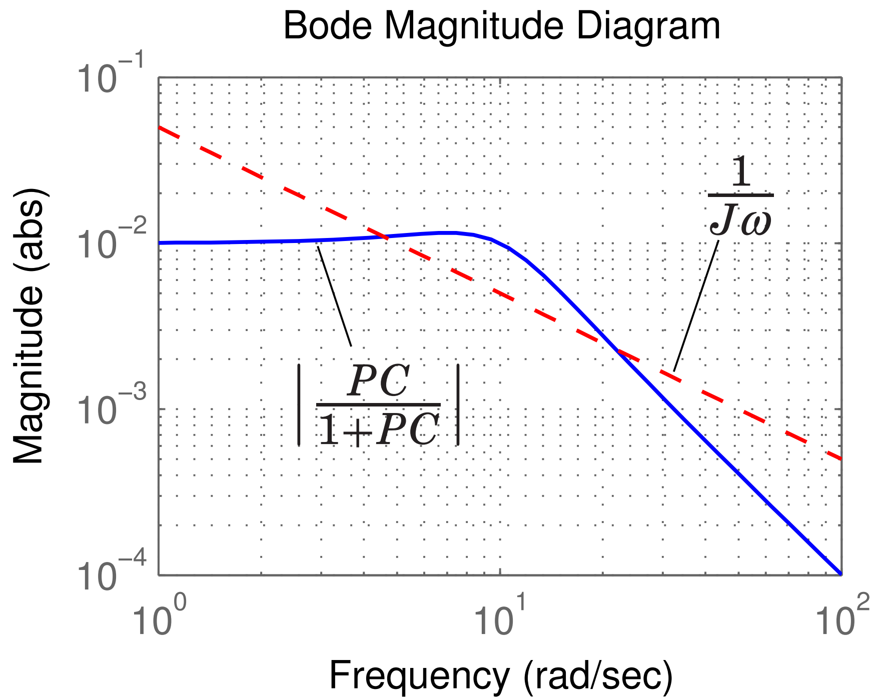
- Continuous-time plant $P(s)$
- Continuous-time controller $C(s)$
- Arbitrarily time-varying delay $\Delta \in [0, J]$
- **Theorem:** closed-loop system stable if

$$\left| \frac{P(i\omega)C(i\omega)}{1 + P(i\omega)C(i\omega)} \right| < \frac{1}{J\omega} \quad \forall \omega \in [0, \infty].$$

Graphical test:



Stable

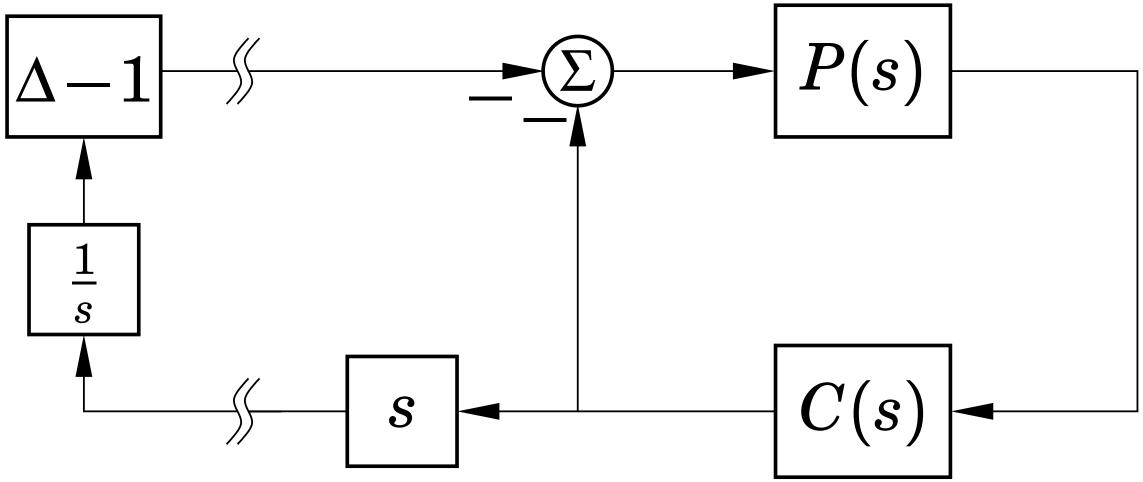


Could be unstable

(Note that the theorem gives a sufficient but not necessary condition for stability)

Proof sketch

Rewrite the control output as one direct path and one error path:



Gain of left part: J

Gain of right part: $\max_{\omega} \left| \frac{i\omega P(i\omega)C(i\omega)}{1 + P(i\omega)C(i\omega)} \right|$

The Small Gain Theorem then gives the result

Stability under jitter – sampled-data case

Now assume continuous-time plant $P(s)$, discrete-time controller $C(z)$ and time-varying delay $\Delta \in [0, J]$

The closed-loop system is stable if

$$\left| \frac{P_{\text{alias}}(\omega)C(e^{i\omega})}{1 + P_{\text{ZOH}}(e^{i\omega})C(e^{i\omega})} \right| < \frac{1}{\sqrt{J}\left|e^{i\omega} - 1\right|}, \quad \forall \omega \in [0, \pi]$$

Here,

- $P_{\text{alias}}(\omega) = \sqrt{\sum_{k=-\infty}^{\infty} \left| P\left(i(\omega + 2\pi k)\frac{1}{h}\right)\right|^2}$
- $P_{\text{ZOH}}(z)$ is the ZOH-discretization of $P(s)$

(For small h, $P_{\text{alias}}(\omega) \approx P_{\text{ZOH}}(e^{i\omega})$)

Jitter analysis – rate-monotonic scheduling

- R_i – worst-case response time of task i

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- R_i^b – *best-case* response time of task i

$$R_i^b = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^b}{T_j} - 1 \right\rceil C_j$$

- J_i – worst-case input-output jitter of task i:

$$J_i = R_i - R_i^b$$

(Analysis for earliest-deadline-first scheduling also exists)

The pendulum example – RM scheduling

Task	T	C	R	R^b	J
1	10	3.5	3.5	3.5	0
2	14.5	3.5	7.0	3.5	3.5
3	17.5	3.5	14.0	3.5	10.5

The pendulum example – RM

- Compute the jitter margin J_m for each task
- $J < J_m \Rightarrow$ Stable

Task	R	$L = R^b$	J	J_m	Stable
1	3.5	3.5	0	4.4	Yes
2	7.0	3.5	3.5	6.4	Yes
3	14.0	3.5	10.5	8.1	Can't say

The pendulum example – EDF

Task	R	$L = R^b$	J	J_m	Stable
1	3.5	3.5	0	4.4	Yes
2	7.5	3.5	4.0	6.4	Yes
3	10.5	3.5	7.0	8.1	Yes

- In general, EDF distributes the jitter more evenly than RM.

Limitations of the jitter margin

- No sampling jitter, only input-output jitter
- Only linear systems
- Sufficient condition only, can be conservative

Coping with sampling jitter

Rule of thumb: Jitter that is less than 10% of the nominal sampling period need not to be compensated for

Two approaches:

- Gain scheduling
- Robust design methods, e.g.
  - H_∞
  - Quantitative Feedback Theory (QFT)
  - μ-design

Gain scheduling

Parametrize the controller parameters in terms of the actual (measured) sampling period h_k

For example:

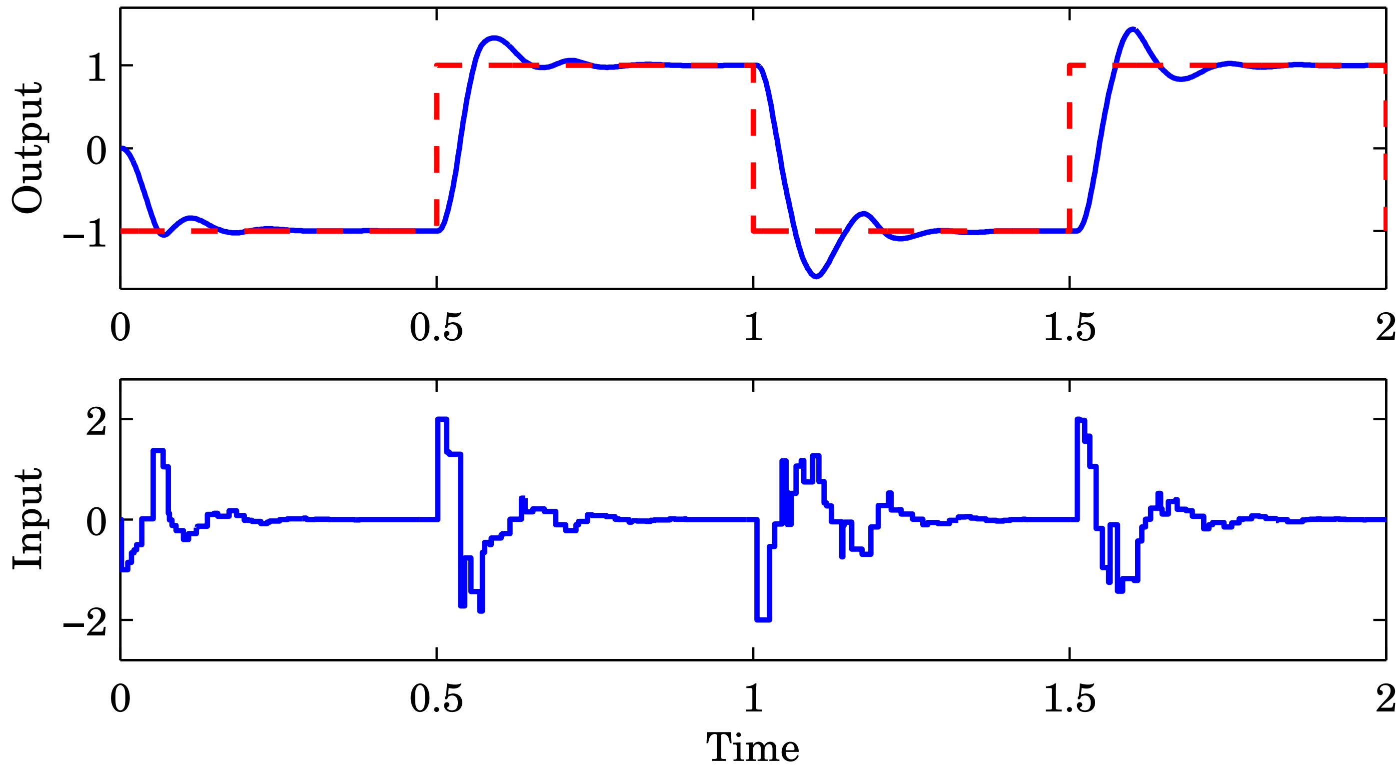
$$\frac{dx(t_k)}{dt} \approx \frac{x(t_k) - x(t_{k-1})}{h_k}$$

Often works well for low order controllers, e.g., PID.

Ad hoc method with no formal guarantees

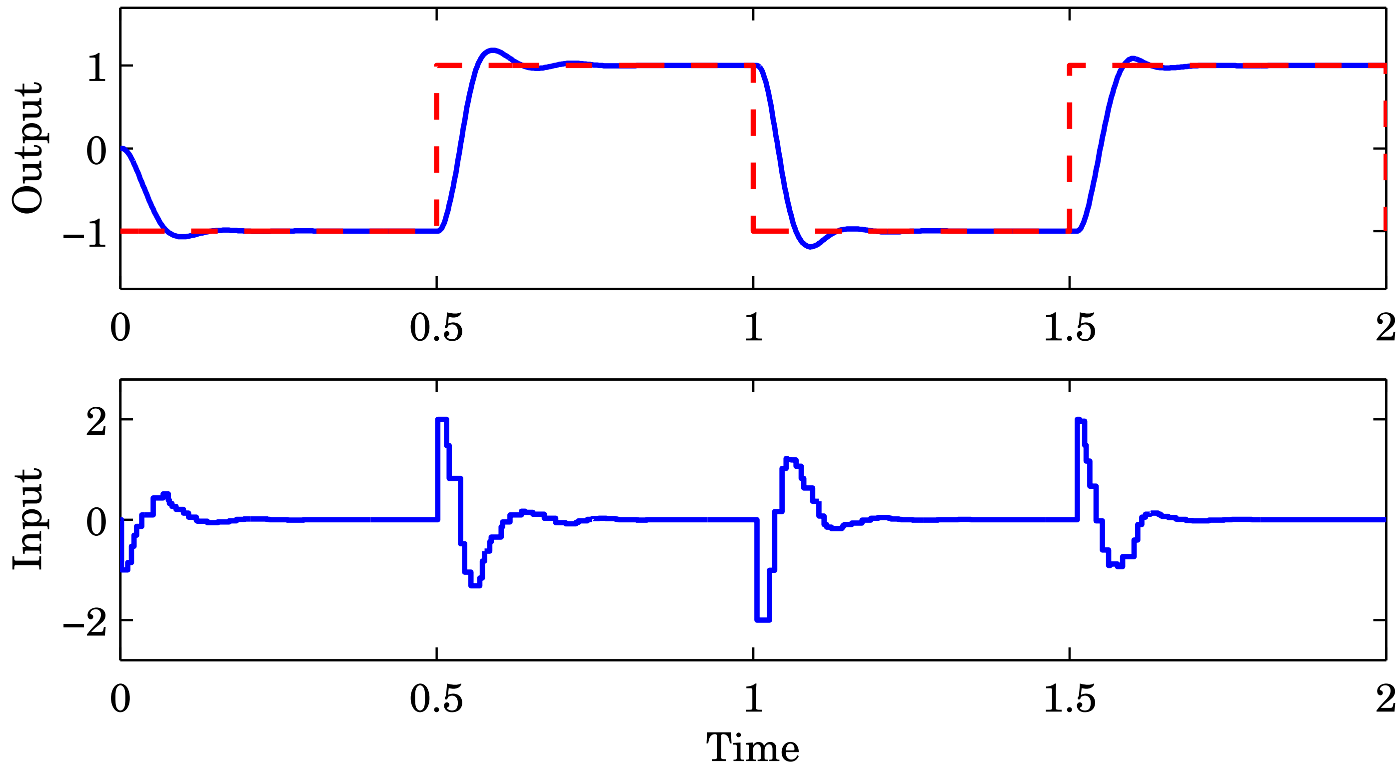
Example: Control of DC servo with sampling jitter

- PD controller designed for $h = 10$ ms
- Actual sampling period varies randomly between 2 and 18 ms



Example: Control of DC servo with sampling jitter

- D-part calculated according to actual sampling interval h_k



Almost no visible performance degradation

Lecture 2 outline

- Introduction
- Analysis of controller timing
- **Scheduling to reduce delay and jitter**

Subtask scheduling

A control algorithm normally consists of four distinct parts:

```
while (1) {
  read_input();
  calculate_output();
  write_output();
  update_state();
  ...
}
```

Idea: schedule the parts as separate (sub)tasks

- reduce delay
- reduce jitter

Subtask scheduling with two subtasks

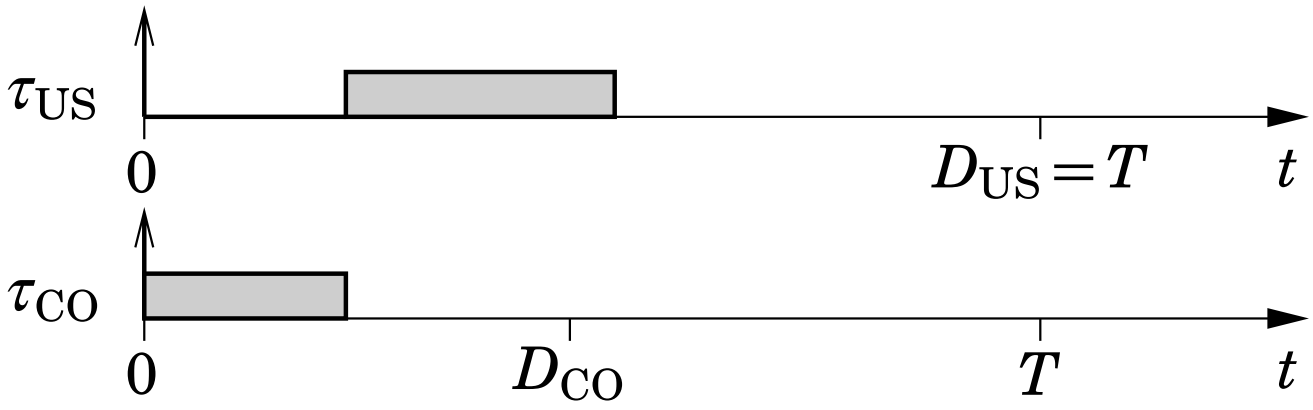
Assume a set of control tasks, where each control task τ is divided into two subtasks:

- τ_{CO} – Read Input, Calculate Output, Write Output; execution time C_{CO}
- τ_{US} – Update State, execution time C_{US}

Many possible scheduling algorithms:

- Deadline-monotonic (DM) scheduling
- EDF scheduling
- $\ldots$

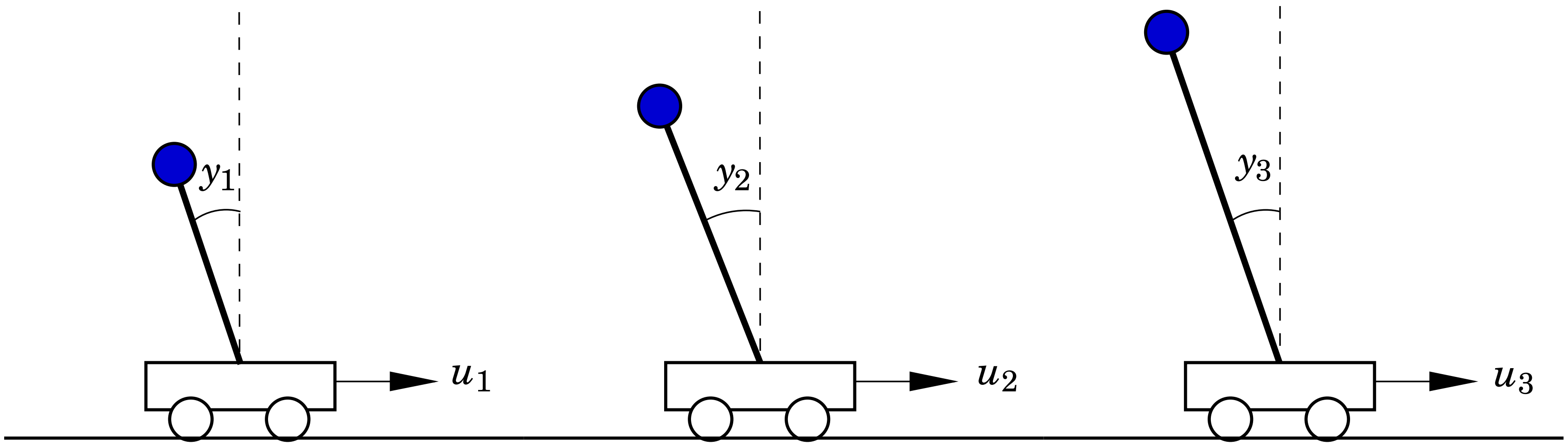
Deadline assignment under DM scheduling

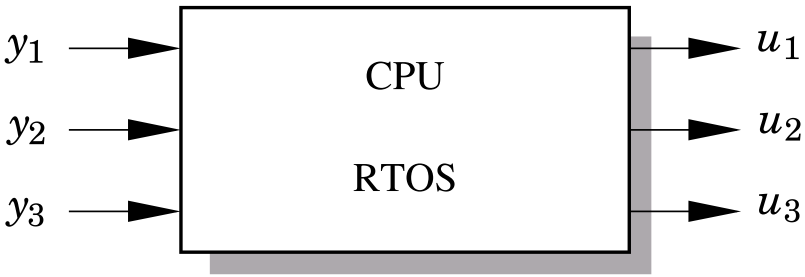


- Assign $D_{US} = T$ for all control tasks
- Want to minimize D_{CO} for each task. Iterative deadline assignment algorithm:

1. Start by assigning $D_{CO} := T - C_{US}$ for all tasks
2. Assign deadline-monotonic priorities to all subtasks
3. Calculate the response time R of each subtask
4. Assign $D_{CO} := R_{CO}$ for all tasks
5. Repeat from 2 until no further improvement.

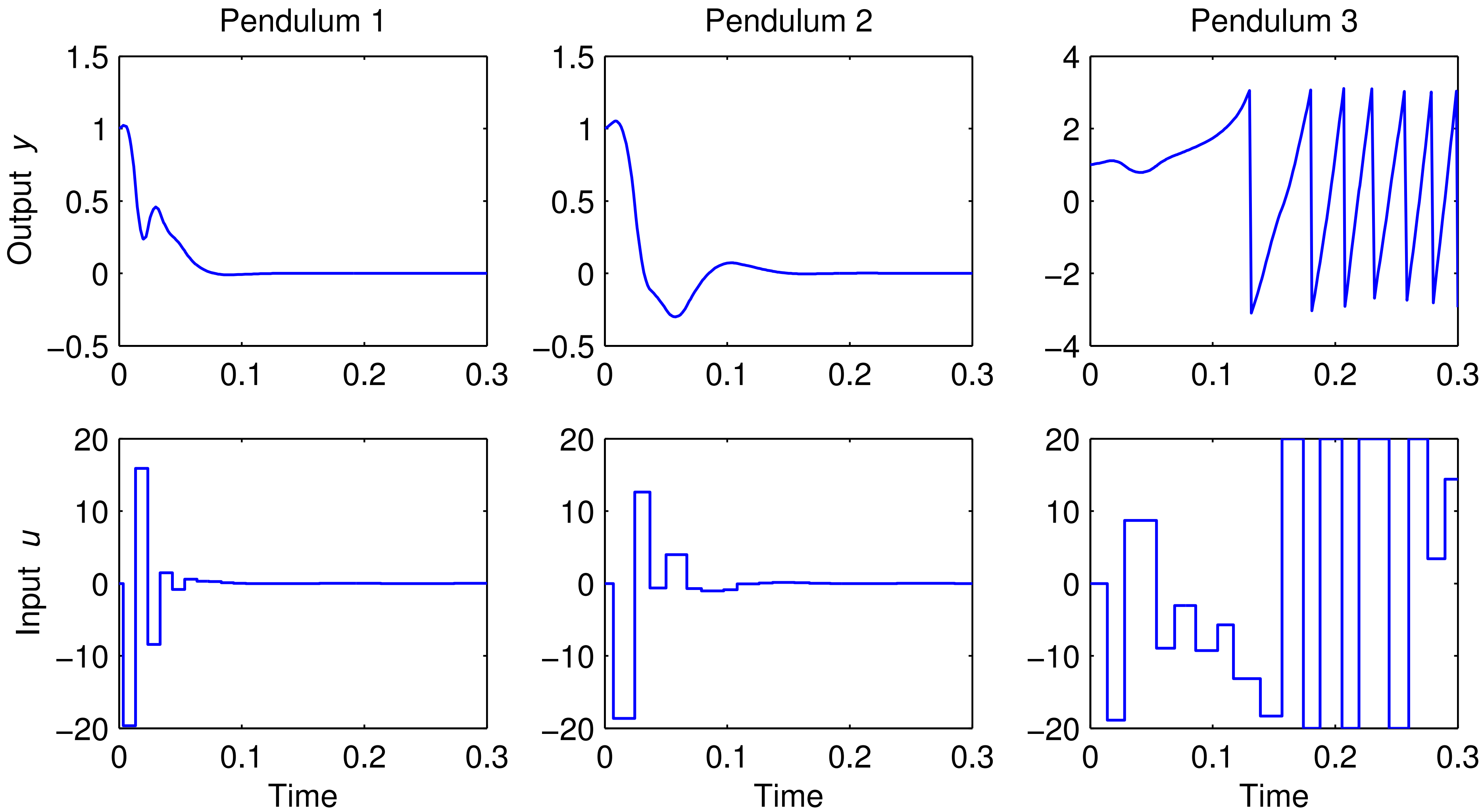
Inverted pendulum example (again)



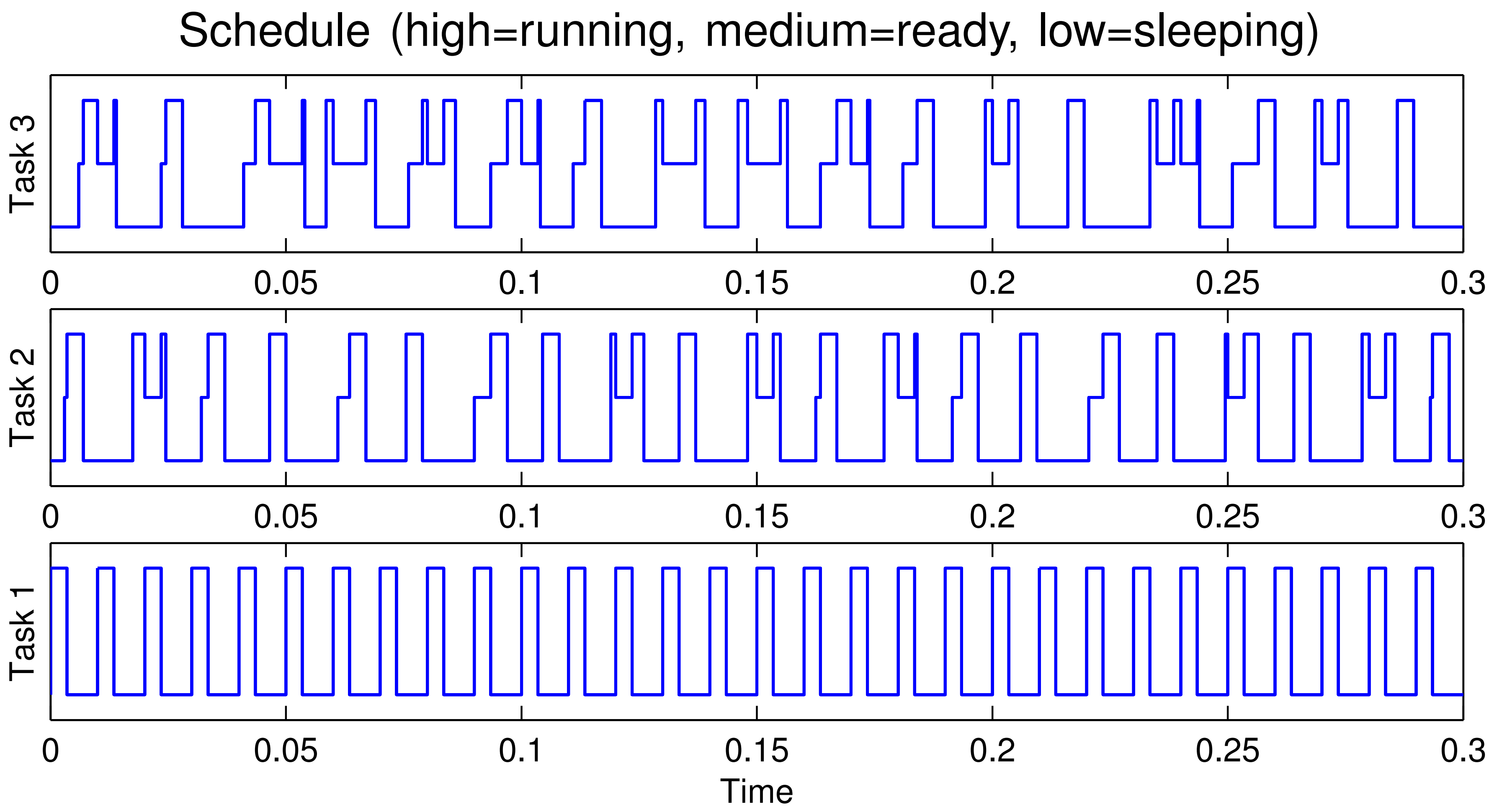


- The same design as before

Simulation under RM scheduling



Schedule under RM scheduling



- Large delay and jitter for controller 3

Subtask scheduling analysis

Each pendulum controller is divided into two subtasks:

- Calculate Output: $C_{\text{CO}} = 1.5$ ms
- Update State: $C_{\text{US}} = 2.0$ ms

First iteration of deadline assignment algorithm:

	T	D	C	R
$\tau_{\text{CO}1}$	10.0	8.0	1.5	1.5
$\tau_{\text{US}1}$	10.0	10.0	2.0	3.5
$\tau_{\text{CO}2}$	14.5	12.5	1.5	5.0
$\tau_{\text{US}2}$	14.5	14.5	2.0	7.0
$\tau_{\text{CO}3}$	17.5	15.5	1.5	8.5
$\tau_{\text{US}3}$	17.5	17.5	2.0	14.0

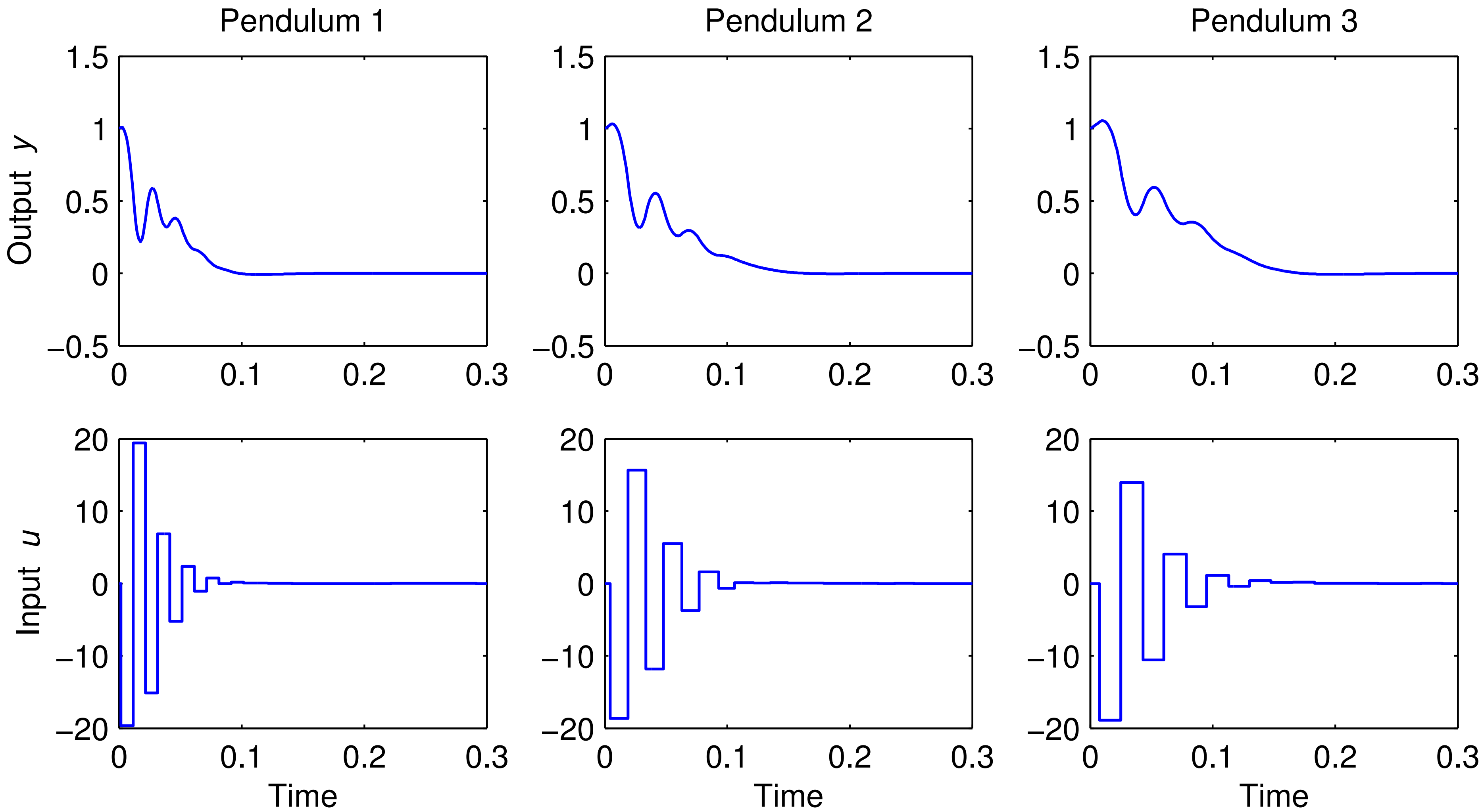
Subtask scheduling analysis

Third iteration (converged):

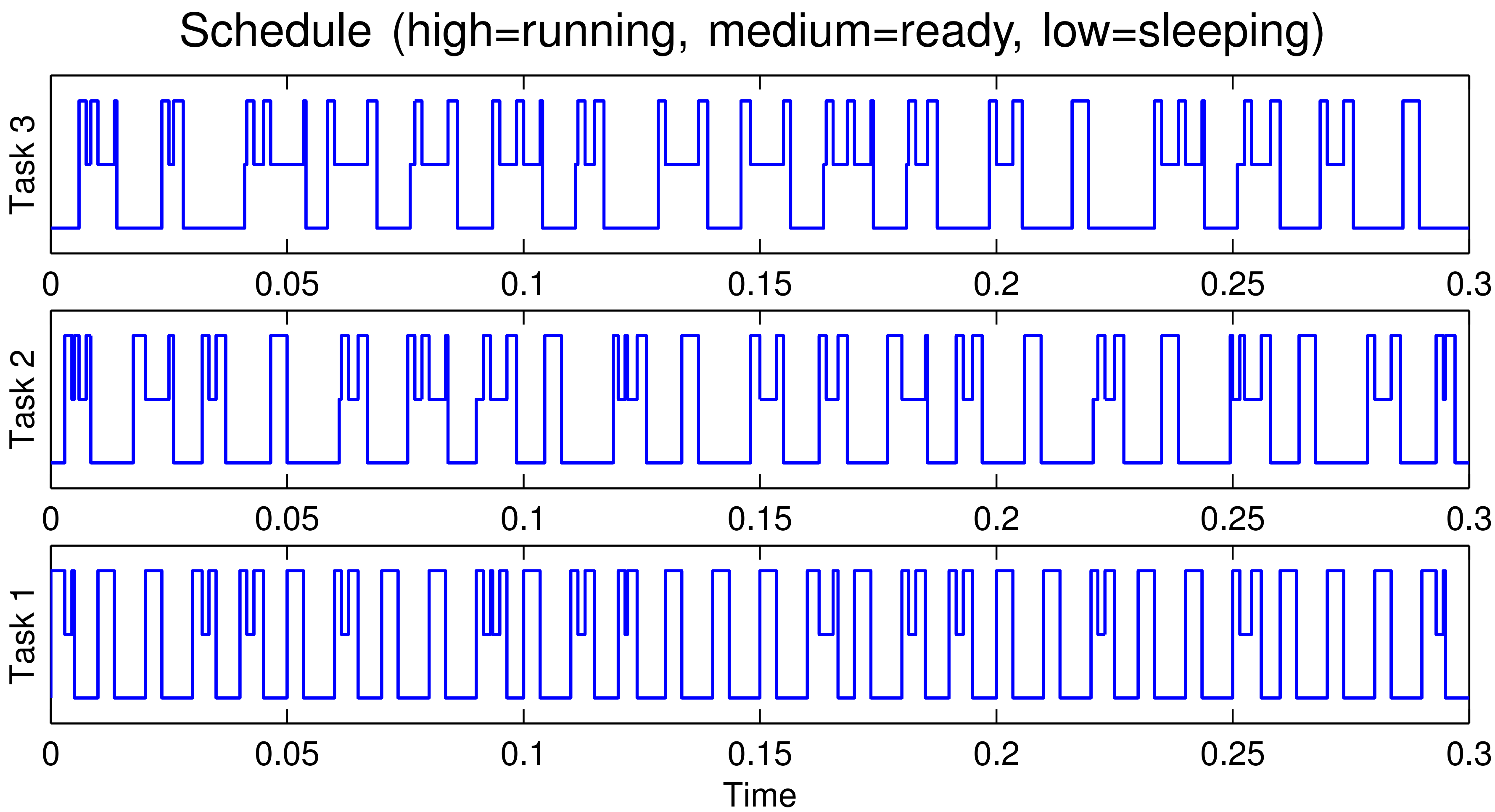
	T	D	C	R
τ_{CO1}	10.0	1.5	1.5	$\boxed{1.5}$
τ_{US1}	10.0	10.0	2.0	6.5
τ_{CO2}	14.5	3.0	1.5	$\boxed{3.0}$
τ_{US2}	14.5	14.5	2.0	8.5
τ_{CO3}	17.5	4.5	1.5	$\boxed{4.5}$
τ_{US3}	17.5	17.5	2.0	14.0

New worst-case input-output latencies: 1.5, 3.0, 4.5 ms.

Simulation under subtask scheduling



Schedule under subtask scheduling



- More context switches