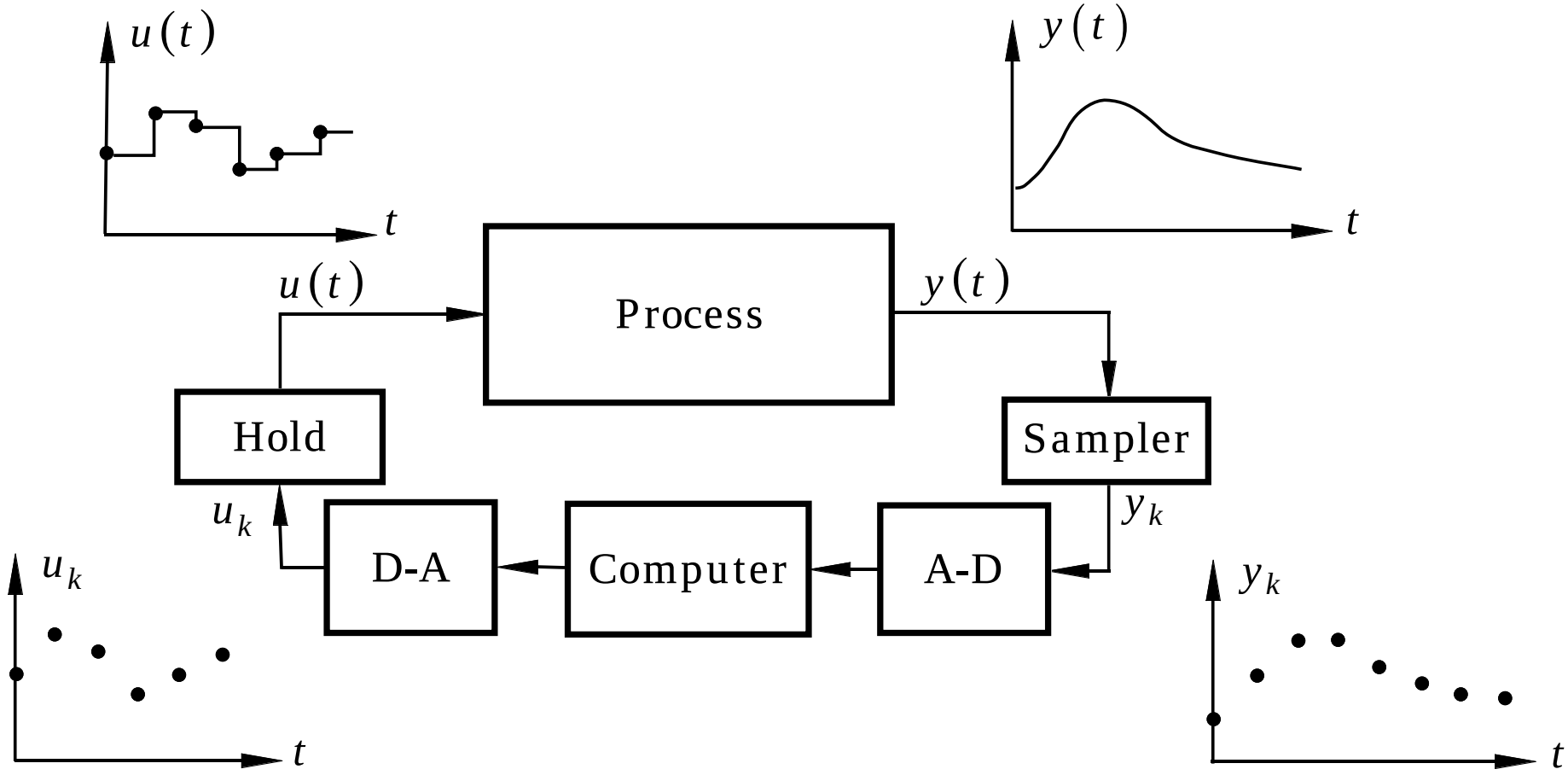
# Computer Implementation of Control Systems

Karl-Erik Årzen, Anton Cervin

# Session outline

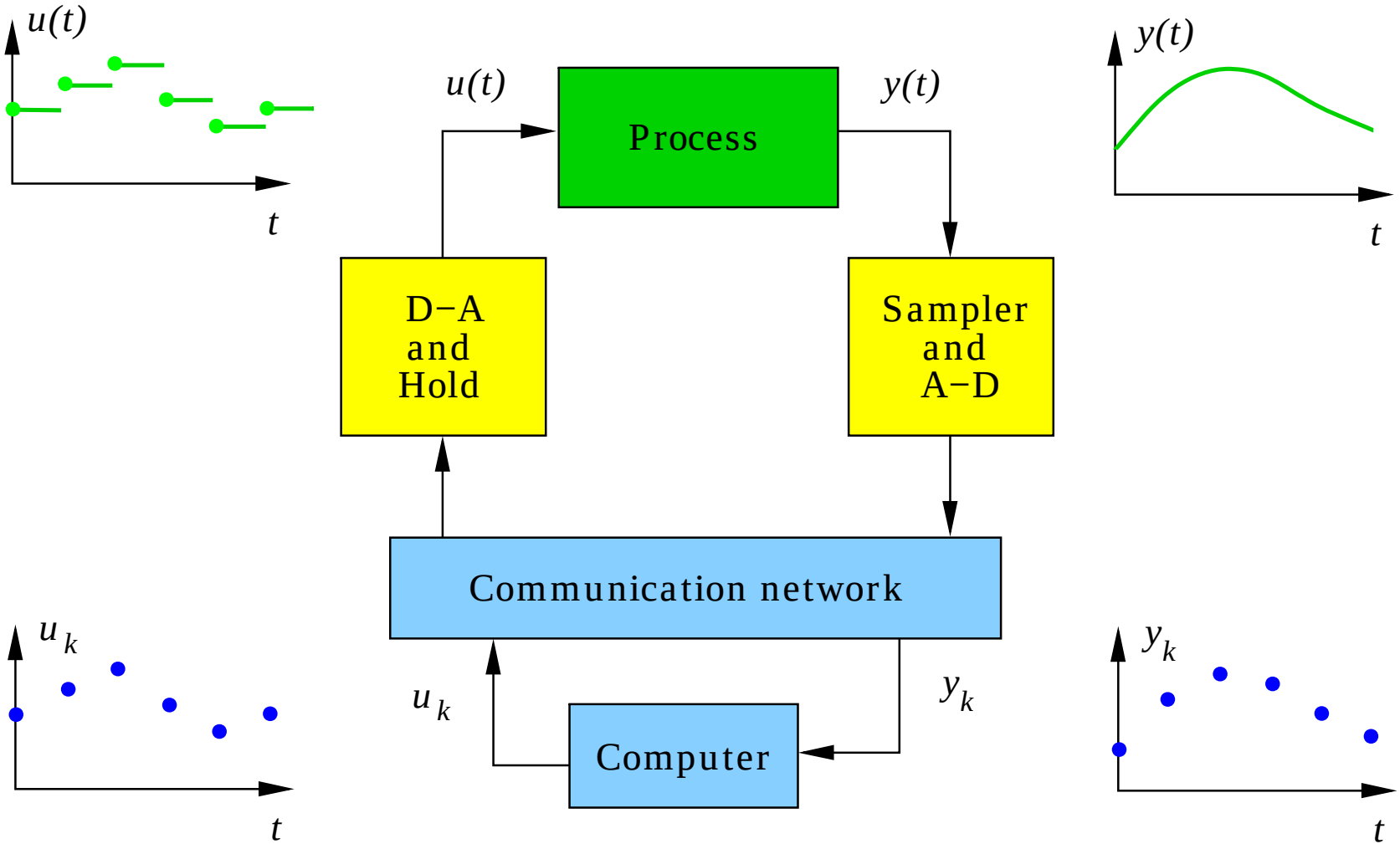
- **Sampled-data control**
- Discretization of continuous-time controllers
- Implementation of PID Controllers

# Sampled-data control systems



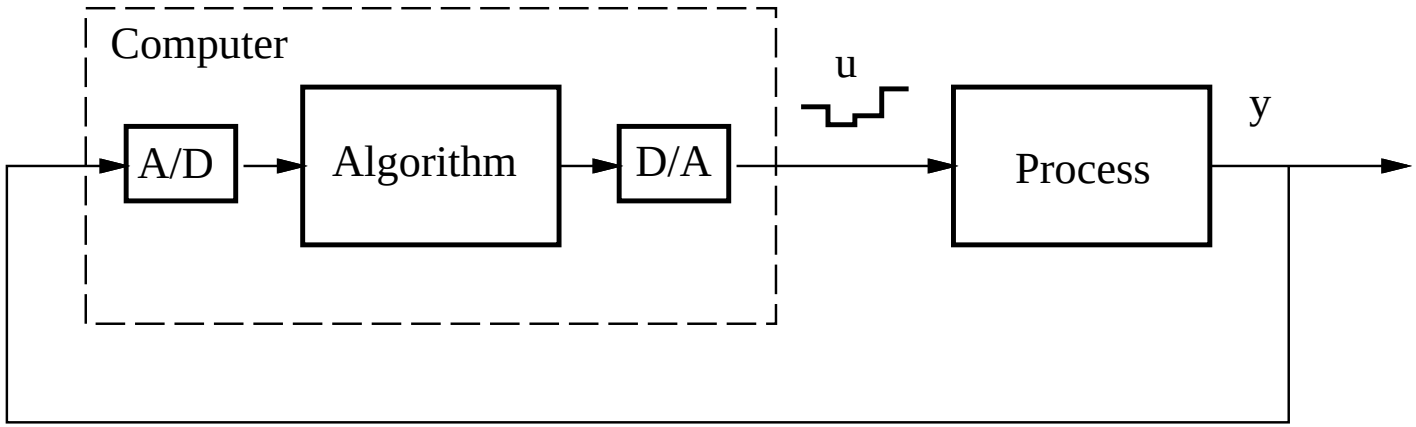

- Mix of continuous-time and discrete-time signals

# Networked control systems

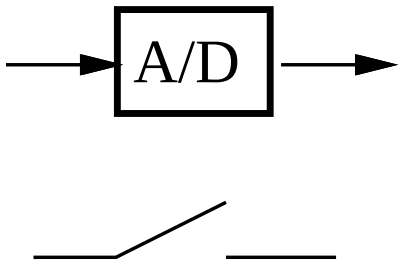



- Extra delay, possibly lost packets

# Sampling



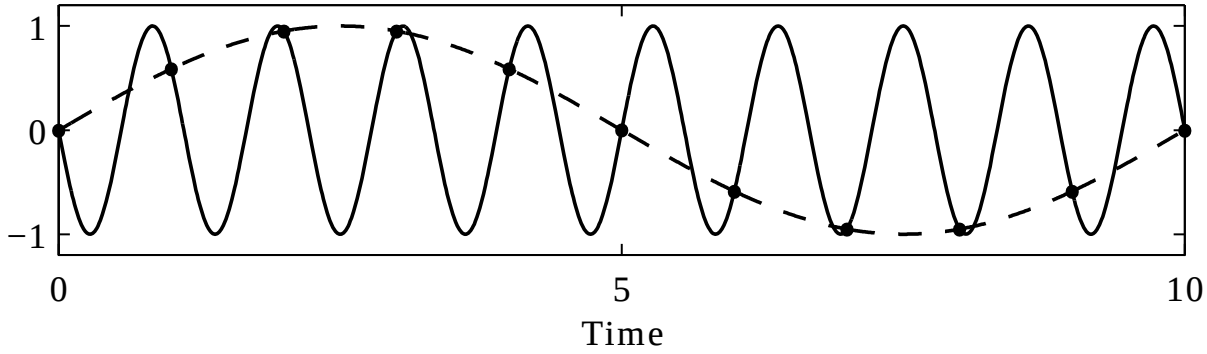

AD-converter acts as sampler




DA-converter acts as a hold device

Normally, zero-order-hold is used $\Rightarrow$ piecewise constant control signals

# Aliasing




$\omega_s = \frac{2\pi}{h} =$ sampling frequency

$\omega_N = \frac{\omega_s}{2} =$ Nyquist frequency

Frequencies above the Nyquist frequency are folded and appear as low-frequency signals.

The fundamental alias for a frequency $f_1$ is given by

$$f = |(f_1 + f_N) \bmod (f_s) - f_N|$$

Above: $f_1 = 0.9, f_s = 1, f_N = 0.5, f = 0.1$

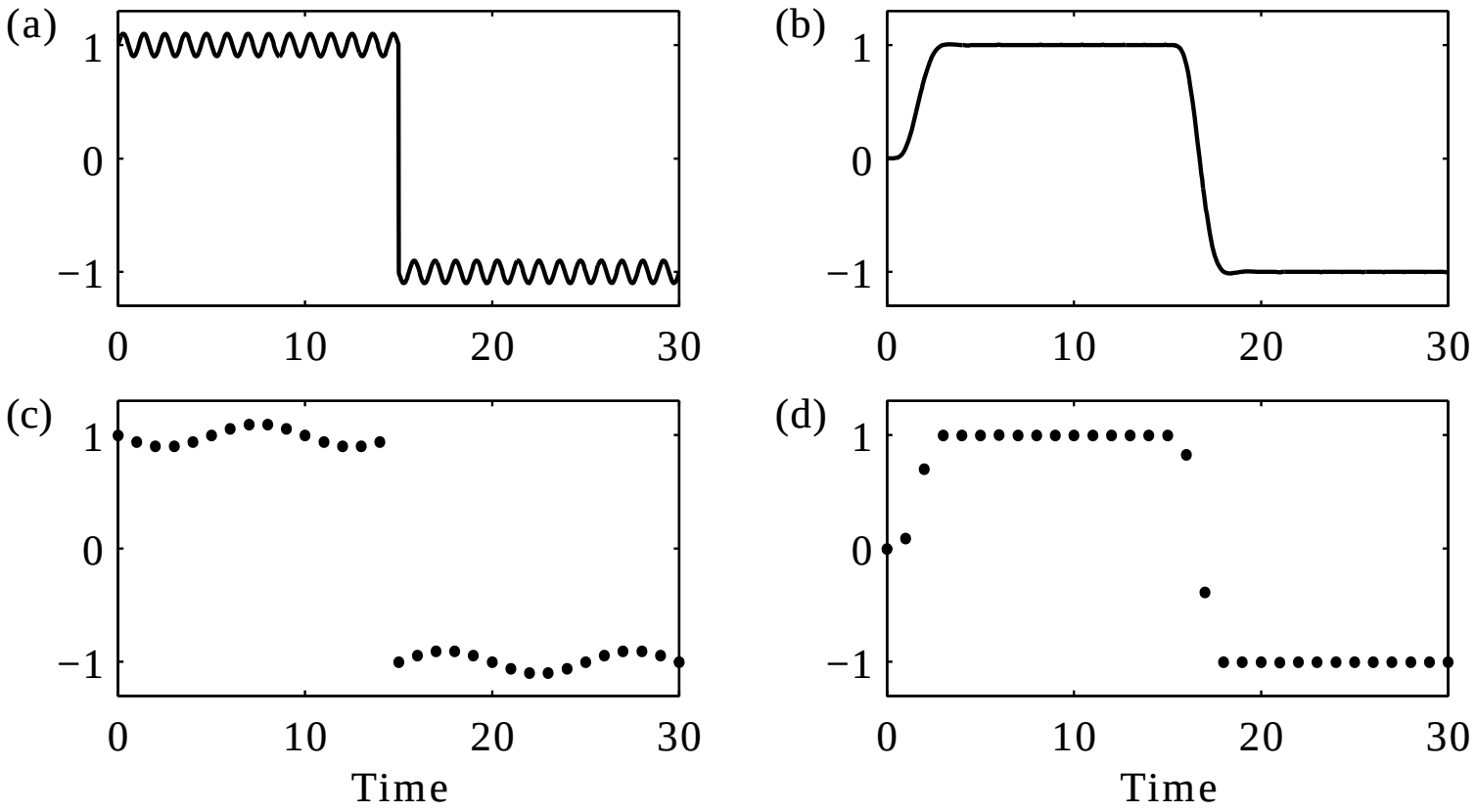
# Anti-aliasing filter

**Analog** low-pass filter that eliminates all frequencies above the Nyquist frequency

- Analog filter
  - 2-6th order Bessel or Butterworth filter
  - Difficulties with changing $h$ (sampling interval)
- Analog + digital filter
  - Fixed, fast sampling with fixed analog filter
  - Downsampling using digital LP-filter
  - Control algorithm at the lower rate
  - Easy to change sampling interval

The filter may have to be included in the control design

# Example – Prefiltering




$\omega_d = 0.9,\ \omega_N = 0.5,\ \omega_{alias} = 0.1$

6th order Bessel with $\omega_B = 0.25$

# Design approaches

Digital controllers can be designed in two different ways:

- Discrete-time design – sampled control theory
  - Sample the continuous system
  - Design a digital controller for the sampled system
    - Z-transform domain
    - state-space domain
- Continuous time design + discretization
  - Design a continuous controller for the continuous system
  - Approximate the continuous design
  - Use fast sampling

# Disk drive example

Control of the arm of a disk drive

$$G(s) = \frac{k}{Js^2}$$

Continuous time controller

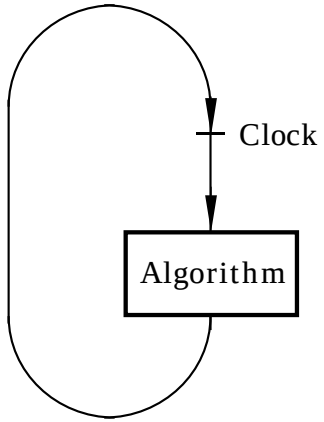
$$U(s) = \frac{bK}{a}U_c(s) - K\frac{s+b}{s+a}Y(s)$$

Discrete time controller (continuous time design + discretization)

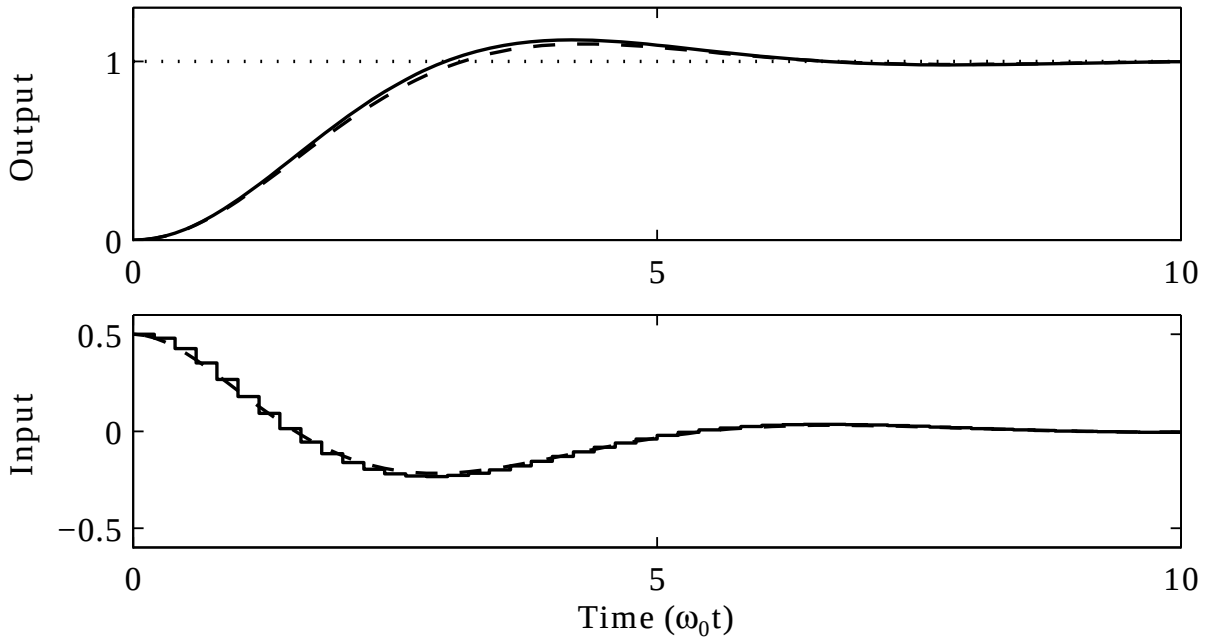
$$u(t_k) = K(\tfrac{b}{a}u_c(t_k) - y(t_k) + x(t_k))$$
$$x(t_k + h) = x(t_k) + h\left((a-b)y(t_k) - ax(t_k)\right)$$

# Disk drive example

```
y: = adin(in2)
u:=K*(b/a*uc-y+x)
dout(u)
x:=x+h*((a-b)*y-a*x)
```

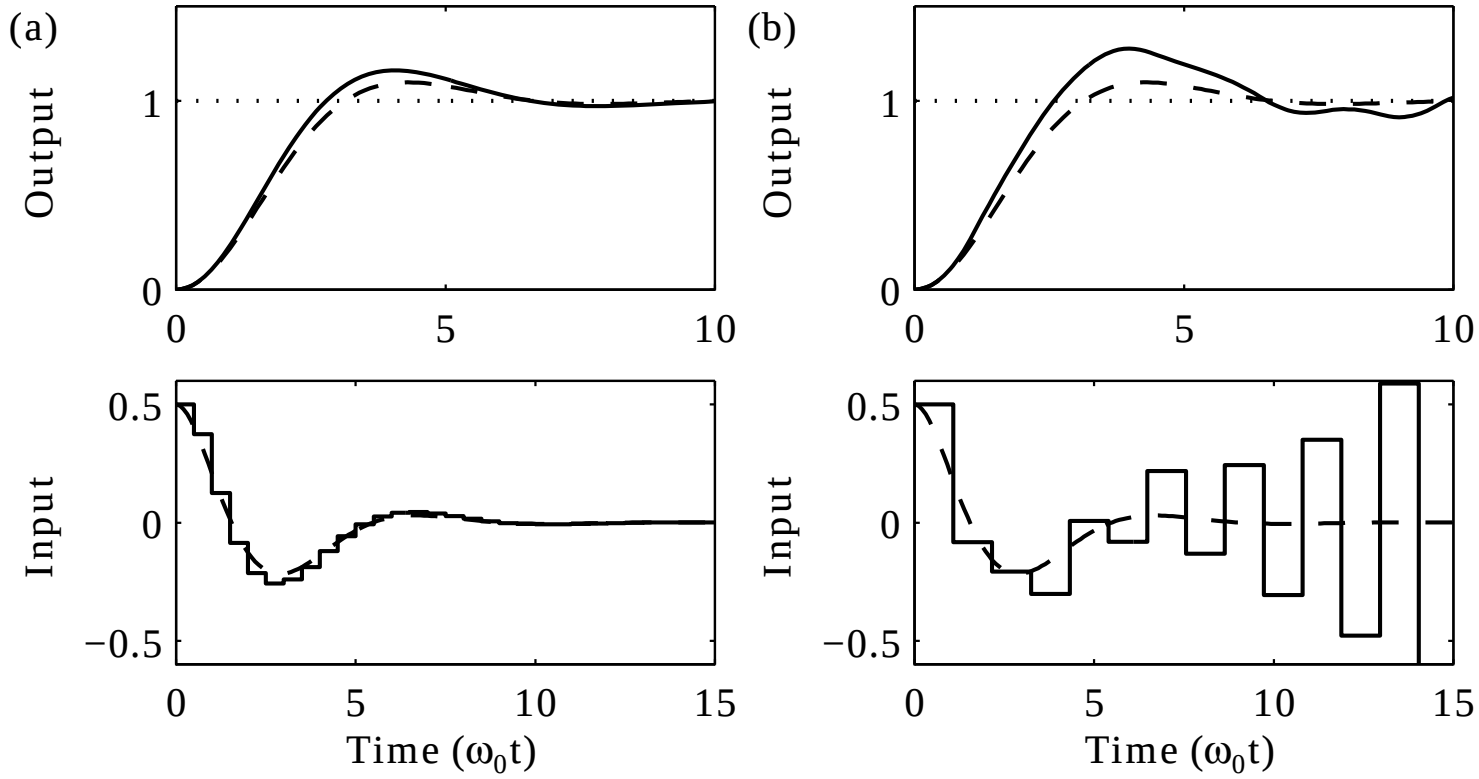



Sampling period $h = 0.2/\omega_0$



# Increased sampling period

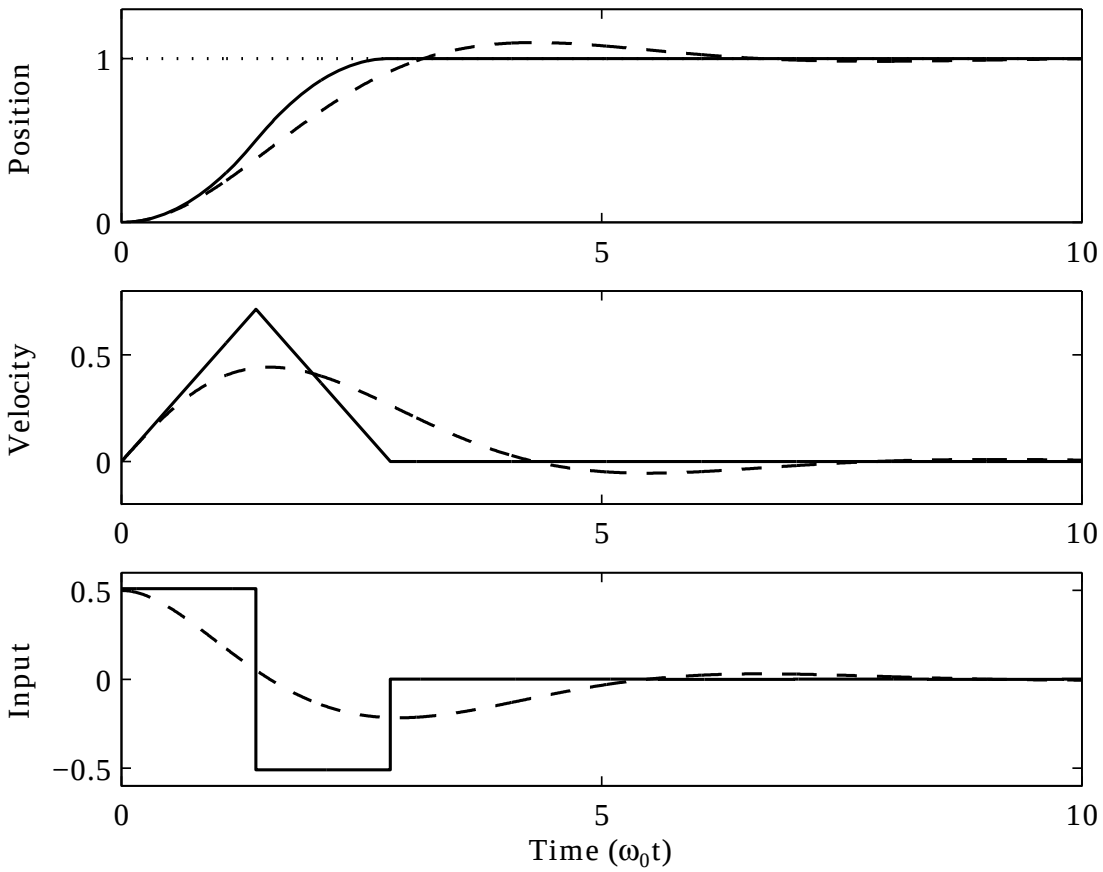
a) $h = 0.5/\omega_0$ b) $h = 1.08/\omega_0$



# Better performance?

Dead-beat control $h = 1.4/\omega_0$

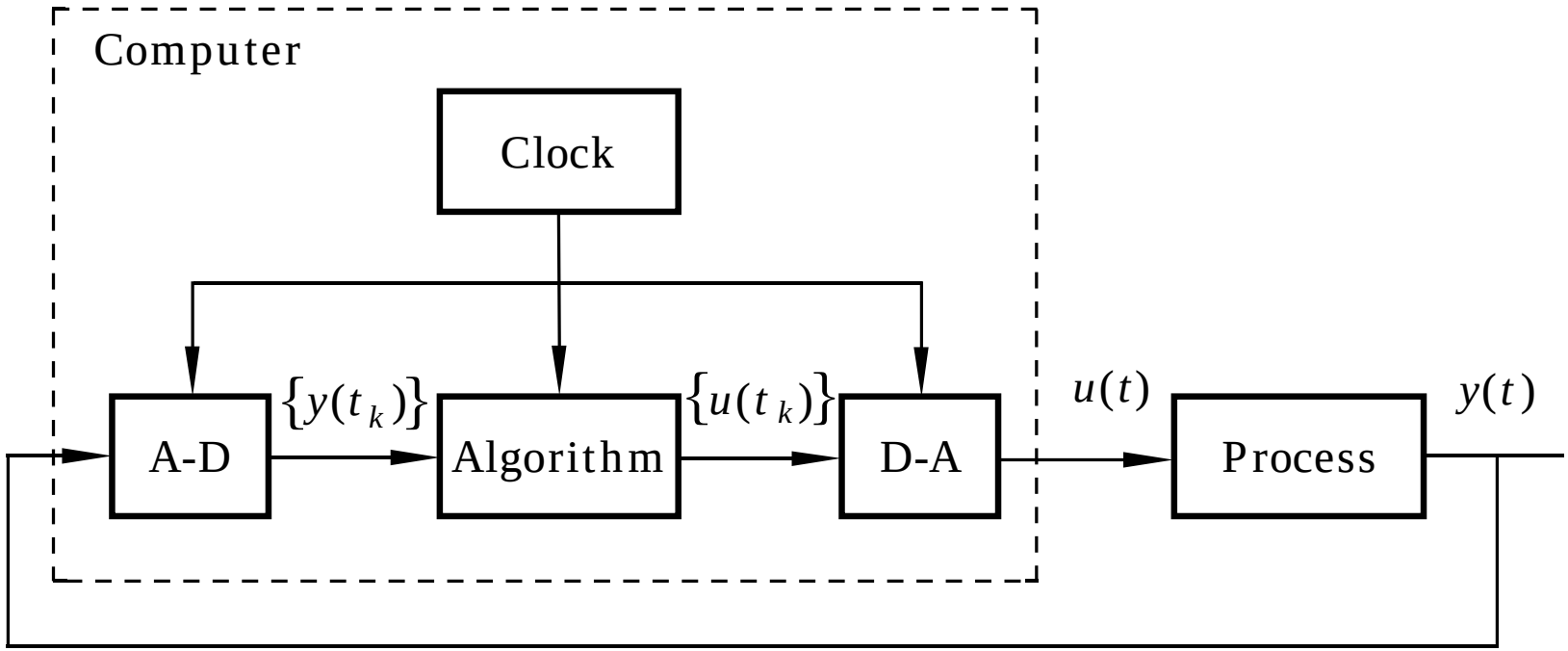
$$u(t_k) = t_0 u_c(t_k) + t_1 u_c(t_{k-1}) - s_0 y(t_k) - s_1 y(t_{k-1}) - r_1 u(t_{k-1})$$



However, long sampling periods also have problems

- open loop between the samples
- disturbance and reference changes that occur between samples will remain undetected until the next sample

# Sampled control theory

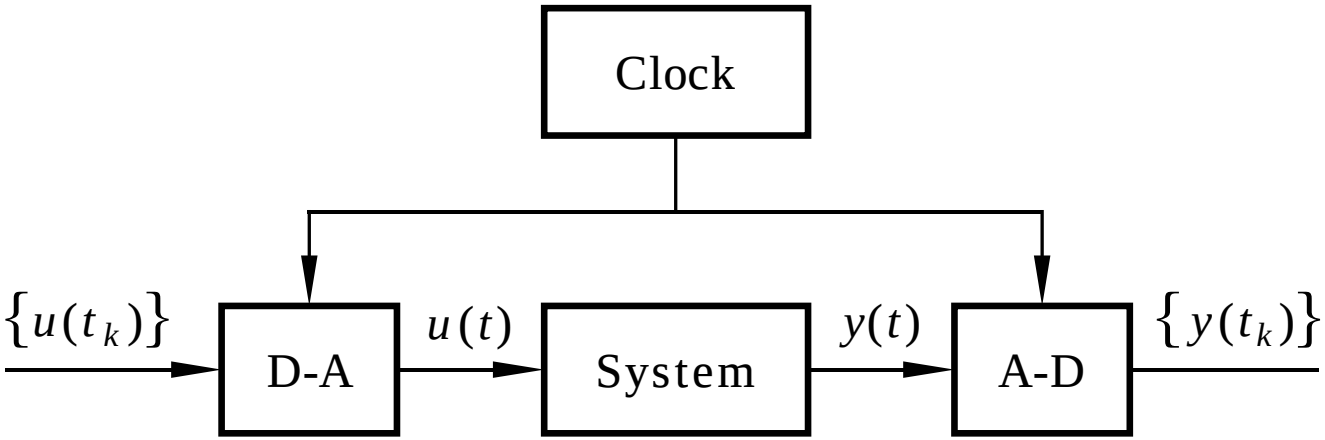



Basic idea: look at the sampling instances only

- System theory analogous to continuous-time systems
- Better performance can be achieved
- Potential problem with intersample behaviour

# Sampling of systems

Look at the system from the point of view of the computer




Zero-order-hold sampling of a system

- Let the inputs be piecewise constant
- Look at the sampling points only
- Solve the system equation

# Sampling a continuous-time system

Process:

$$\frac{dx}{dt} = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

Solve the system equation:

$$x(t) = e^{A(t-t_k)}x(t_k) + \int_{t_k}^{t} e^{A(t-s')}Bu(s')\,ds'$$
$$= e^{A(t-t_k)}x(t_k) + \int_{t_k}^{t} e^{A(t-s')}\,ds'\,Bu(t_k) \quad (u \text{ const.})$$
$$= e^{A(t-t_k)}x(t_k) + \int_{0}^{t-t_k} e^{As}\,ds\,Bu(t_k) \quad \text{(variable change)}$$
$$= \Phi(t,t_k)x(t_k) + \Gamma(t,t_k)u(t_k)$$

# Periodic sampling

Assume periodic sampling, i.e. $t_k = kh$. Then

$$x(kh+h) = \Phi x(kh) + \Gamma u(kh)$$
$$y(kh) = Cx(kh) + Du(kh)$$

where

$$\Phi = e^{Ah}$$
$$\Gamma = \int_0^h e^{As}\, ds\, B$$

Time-invariant linear system!

# Example: Sampling of inverted pendulum

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

$$y = \begin{pmatrix} 1 & 0 \end{pmatrix} x$$

We get

$$\Phi = e^{Ah} = \begin{pmatrix} \cosh h & \sinh h \\ \sinh h & \cosh h \end{pmatrix}$$

$$\Gamma = \int_0^h \begin{pmatrix} \sinh s \\ \cosh s \end{pmatrix} ds = \begin{pmatrix} \cosh h - 1 \\ \sinh h \end{pmatrix}$$

Several ways to calculate $\Phi$ and $\Gamma$. Matlab

# Sampling a system with a time delay

Sampling the system

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

we get the discrete-time system

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

where

$$\Phi = e^{Ah}$$

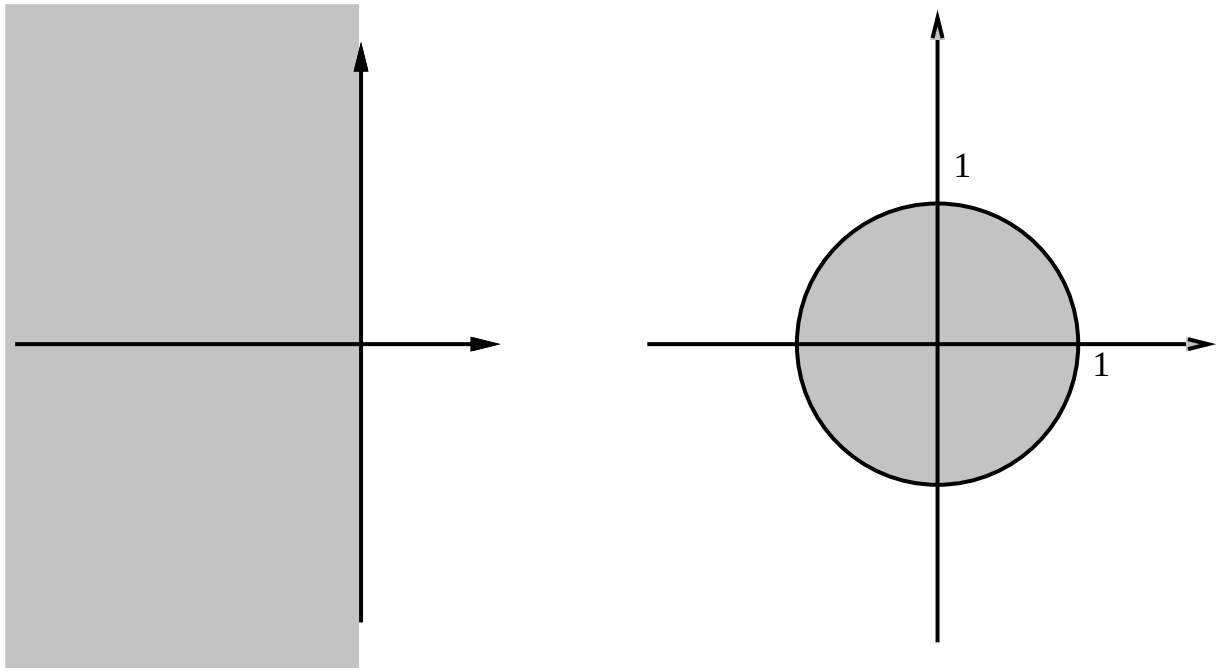
$$\Gamma_0 = \int_0^{h-\tau} e^{As}\, ds\; B$$

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

We get one extra state $\left(u(kh-h)\right)$ in the sampled system

# Stability region

- In continuous time the stability region is the complex left half plane, i.e., the system is stable if all the poles are in the left half plane.
- In discrete time the stability region is the unit circle.



# Digital control design

Similar to analog control design, but

- Z-transform instead of Laplace transform
  - $zX(z) \Leftrightarrow x(t_{k+1})$
  - $z^{-1}X(z) \Leftrightarrow x(t_{k-1})$
- Poles are placed within the unit circle
- The frequency response is more difficult to compute
- The sampling interval $h$ is a new design parameter

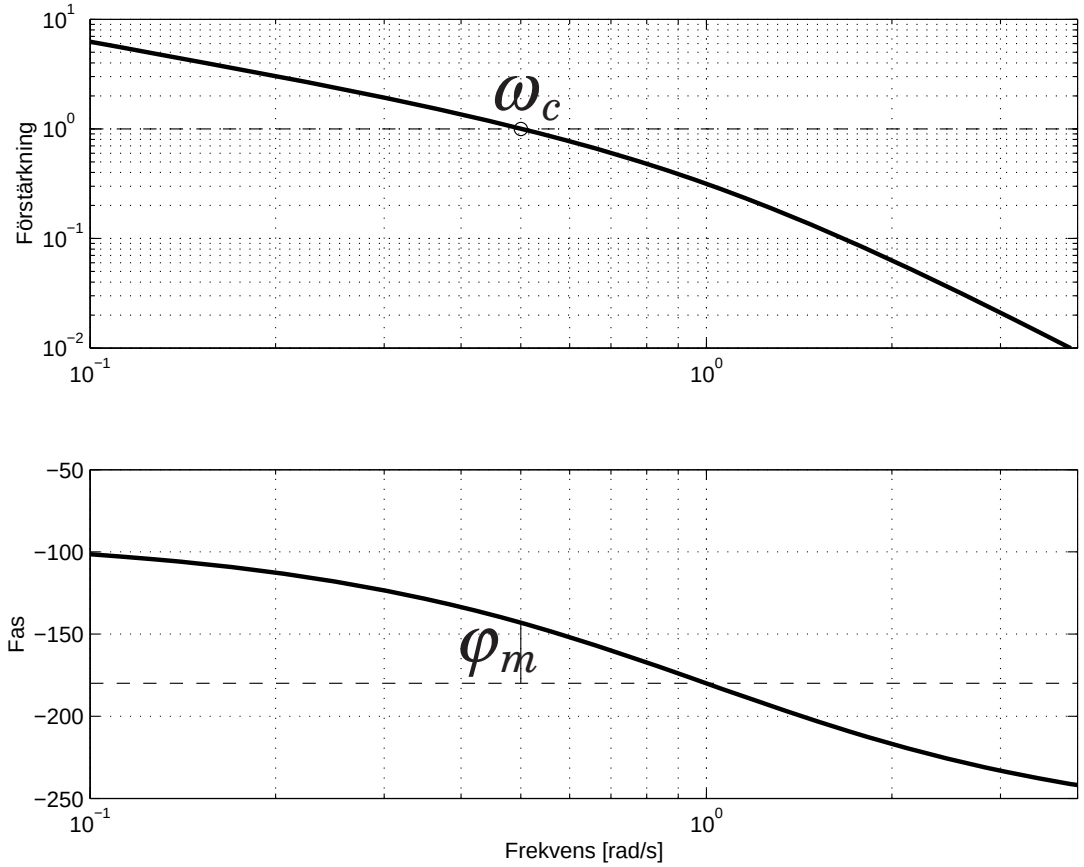
# Choice of sampling interval

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 $L(i\omega) = P(i\omega)C(i\omega)$:




- $\omega_c =$ cross-over frequency, $\varphi_m =$ phase margin
- We should have $\omega_s \gg 2\omega_c$

# Sampling interval rule of thumb

A sample-and-hold (S&H) circuit can be approximated by a delay of $h/2$.

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

This will decrease the phase margin by

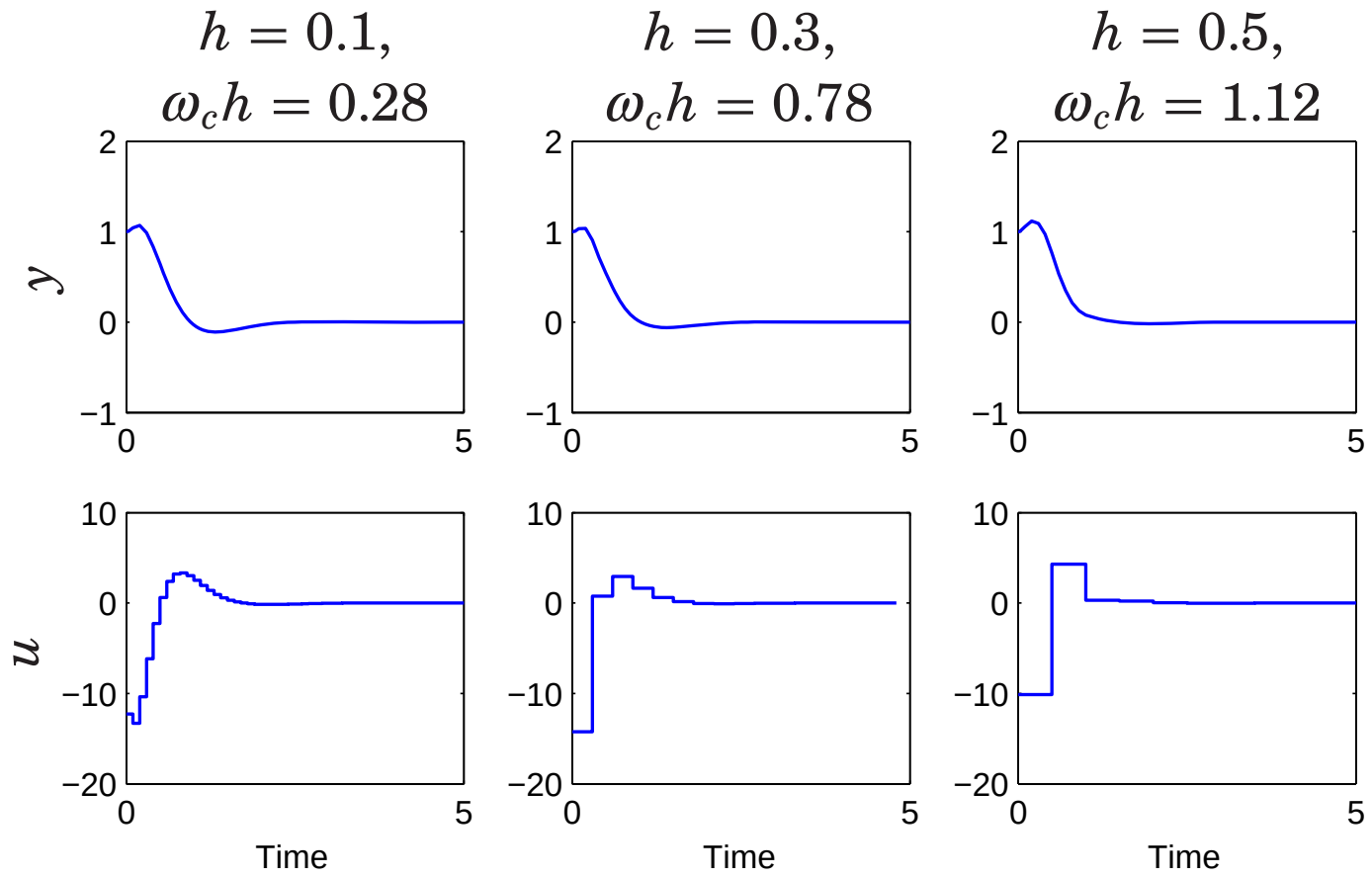
$$\arg G_{S\&H}(i\omega_c) = \arg e^{-i\omega_c h/2} = -\omega_c h/2$$

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

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

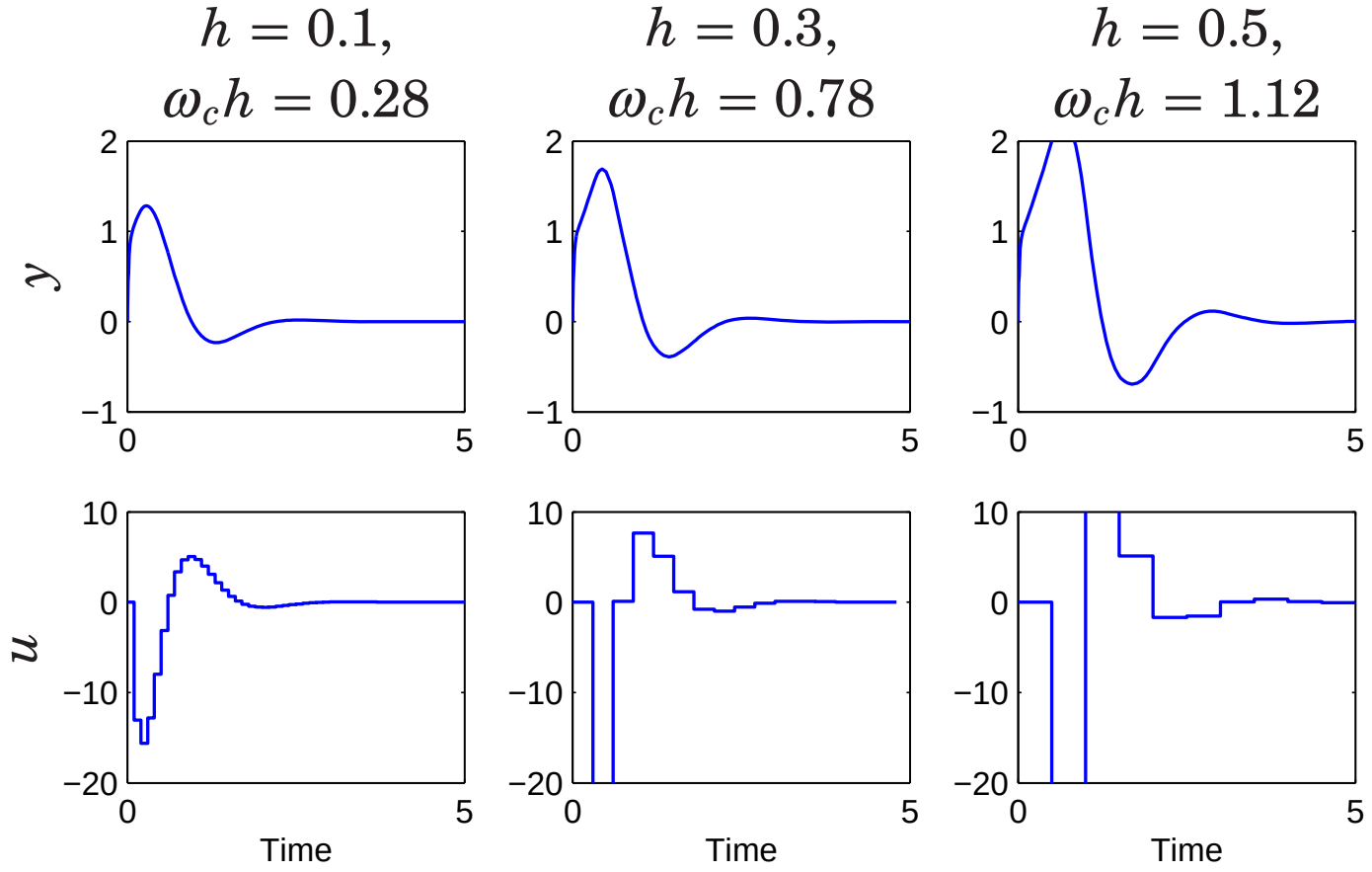
This corresponds to a Nyquist frequency about 6 to 20 times larger than the crossover frequency

# 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



# Accounting for the anti-aliasing filter

Assume we also have a second-order Butterworth anti-aliasing filter with a gain of 0.1 at the Nyquist frequency. The filter gives an additional phase margin loss of $\approx 1.4\omega_c h$.

Again assume we can accept a phase loss of $5^\circ$ to $15^\circ$. Then

$$\boxed{0.05 < \omega_c h < 0.14}$$

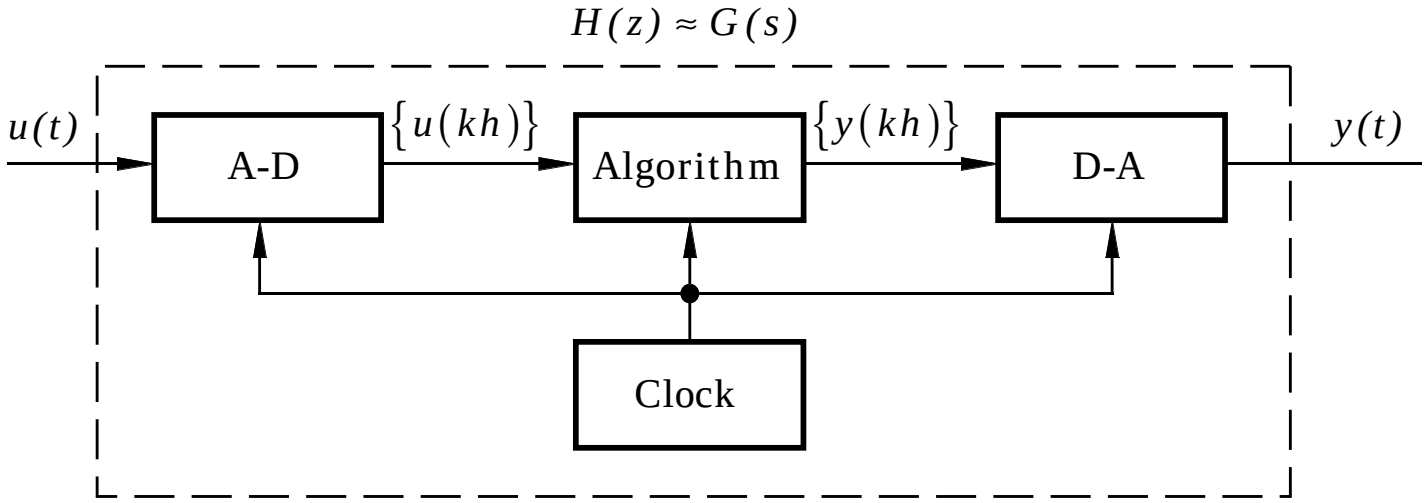
This corresponds to a Nyquist frequency about 23 to 70 times larger than the crossover frequency

# Session outline

- Sampled-data control
- **Discretization of continuous-time controllers**
- Implementation of PID Controllers

# Discretization of continuous-time controllers

Basic idea: Reuse the analog design




Want to get:

- A/D + Algorithm + D/A $\approx G(s)$

Methods:

- Approximate $s$, i.e., $H(z) = G(s')$
- Other discretization methods (Matlab)

# Approximation methods

Forward Difference (Euler's method):

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

$$s' = \frac{z-1}{h}$$

Backward Difference:

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

$$s' = \frac{z-1}{zh}$$

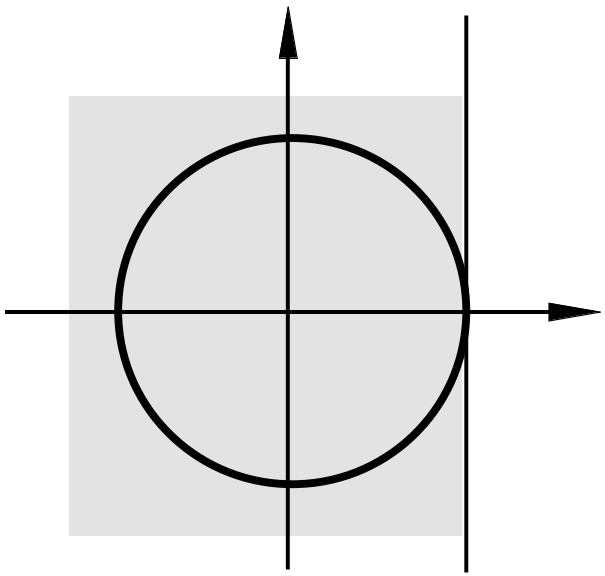
Tustin:

$$\frac{\frac{dx(t)}{dt} + \frac{dx(t_{k+1})}{dt}}{2} \approx \frac{x(t_{k+1}) - x(t_k)}{h}$$

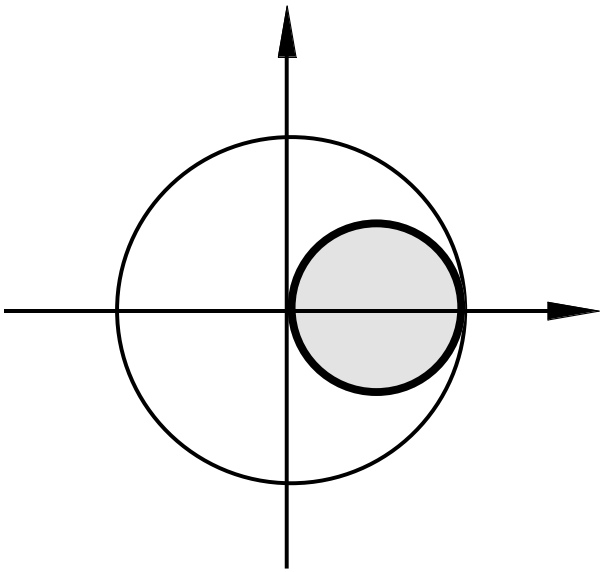
$$s' = \frac{2}{h}\,\frac{z-1}{z+1}$$

# Stability of approximations

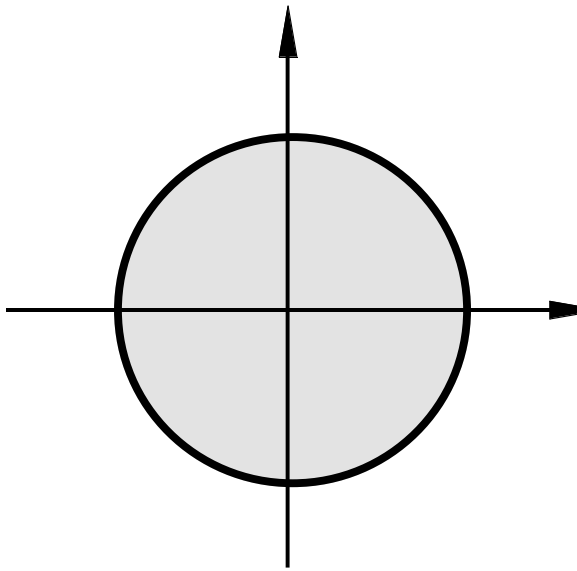
How is the continuous-time stability region (left half plane) mapped?



Forward differences



Backward differences



Tustin

# Discretization example

Controller designed in continuous-time:

$$U(s) = \frac{b}{s+a} E(s)$$

Discretization using Forward Euler ($s' = \frac{z-1}{h}$):

$$u(k) = \frac{b}{(z-1)/h + a} e(k)$$

$$(z - 1 + ha)u(k) = bhe(k)$$

$$u(k+1) = (1 - ha)u(k) + bhe(k)$$

$$u(k) = (1 - ha)u(k-1) + bhe(k-1)$$

Controller stable if $-1 < (1 - ha) < 1$, i.e., $0 < h < 2/a$ (does not imply that the closed loop system is stable, though)

# Controller Synthesis

**Process Model**

$G(s)$

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}$$

**Control Design in Continuous-Time**

- Loop shaping
- Pole placement
- PID
- ....

**Discretize the Controller**

- Euler
- Tustin
- ....

**Discretize the process**

- e.g. ZOH Sampling

$$\begin{cases} x(k+1) = \Phi x(k) + \Gamma u(k) \\ y(k) = Cx(k) + Du(k) \end{cases}$$

**Control Design in Discrete-Time**

- Pole placement
- LQG
- ....

**Difference Equation**

**Software algorithm**

# Session outline

- Sampled-data control
- Discretization of continuous-time controllers
- **Implementation of PID Controllers**

# PID Algorithm

Textbook Algorithm:

$$u(t) = K(e(t) + \frac{1}{T_I}\int^{t} e(\tau)d\tau + T_D\frac{de(t)}{dt})$$

$$U(s) = K(E(s) + \frac{1}{sT_I}E(s) + T_D sE(s))$$

$$= P + I + D$$

# Algorithm Modifications

Modifications are needed to make the controller practically useful

- Limitations of derivative gain
- Derivative weighting
- Setpoint weighting

# Limitations of derivative gain

We do not want to apply derivation to high frequency measurement noise, therefore the following modification is used:

$$sT_D \approx \frac{sT_D}{1 + sT_D/N}$$

$N =$ maximum derivative gain, often $10 - 20$

# Derivative weighting

The setpoint is often constant for long periods of time

Setpoint often changed in steps $\rightarrow$ D-part becomes very large.

Derivative part applied on part of the setpoint or only on the measurement signal.

$$D(s) = \frac{sT_D}{1 + sT_D/N}(\gamma Y_{sp}(s) - Y(s))$$

Often, $\gamma = 0$ in process control, $\gamma = 1$ in servo control

# Setpoint weighting

An advantage to also use weighting on the setpoint.

$$u = K(y_{sp} - y)$$

replaced by

$$u = K(\beta y_{sp} - y)$$

$0 \leq \beta \leq 1$

A way of introducing feedforward from the reference signal (position
a closed loop zero)

Improved set-point responses.

# A better algorithm

$$U(s) = K(\beta y_r - y + \frac{1}{sT_I}E(s) - \frac{T_D s}{1 + sT_D/N}Y(s))$$

Modifications:

- Setpoint weighting ($\beta$) in the proportional term improves set-point response
- Limitation of the derivative gain (low-pass filter) to avoid derivation of measurement noise
- Derivative action only on $y$ to avoid bumps for step changes in the reference signal

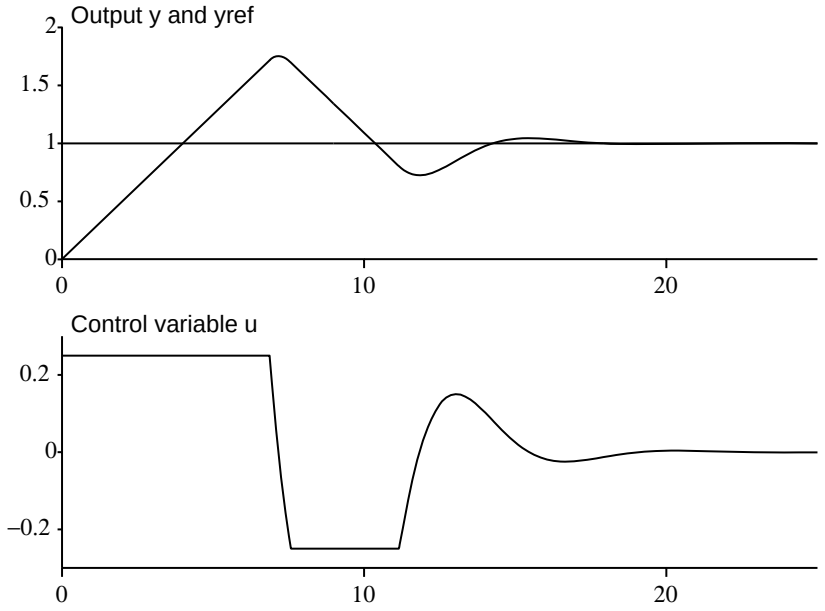
# Control Signal Limitations

All actuators saturate.

Problems for controllers with integration.

When the control signal saturates the integral part will continue to grow – integrator (reset) windup.

When the control signal saturates the integral part will integrate up to a very large value. This may cause large overshoots.



# Anti-Reset Windup

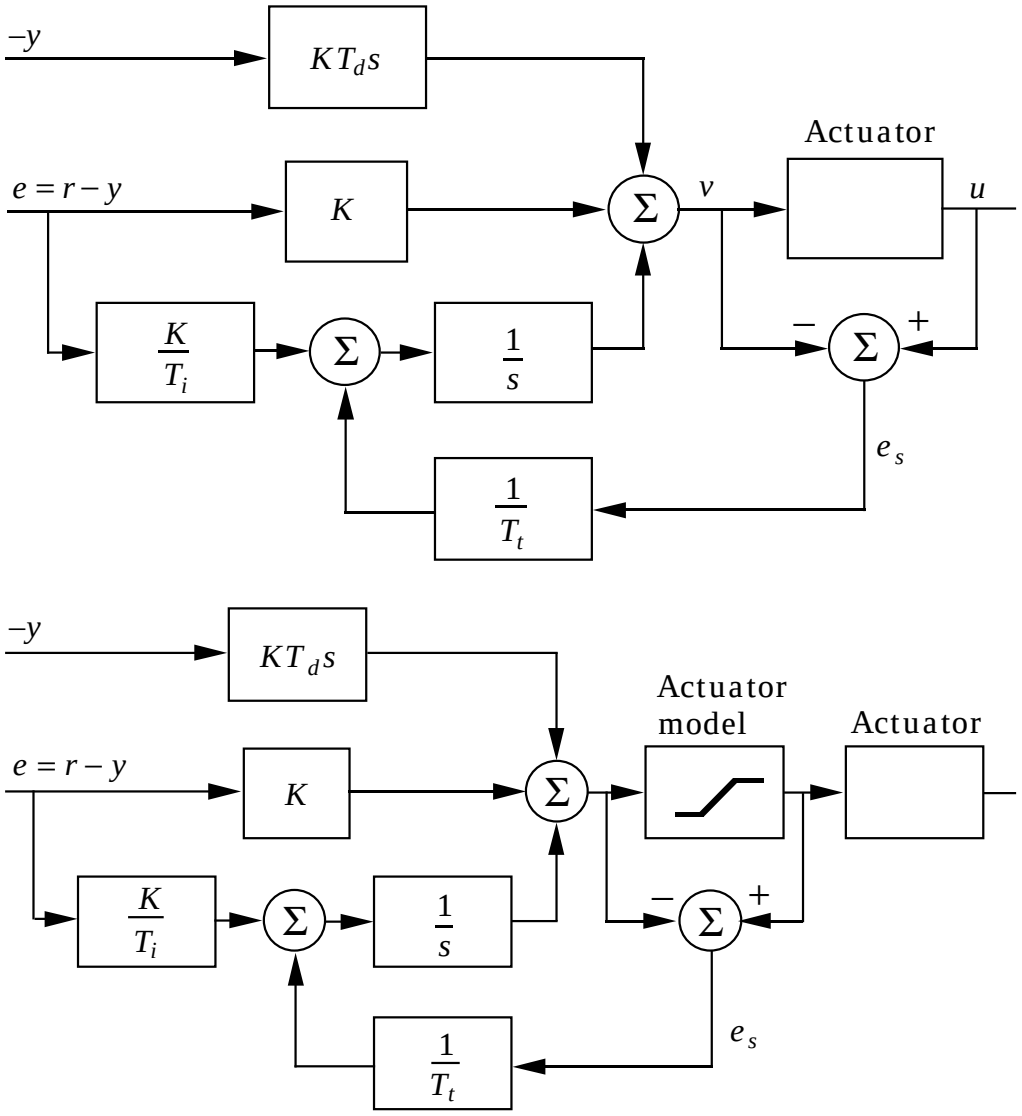
Several solutions exist:

- controllers on velocity form (not discussed here))
- limit the setpoint variations (saturation never reached)
- conditional integration (integration is switched off when the control is far from the steady-state)
- tracking (back-calculation)

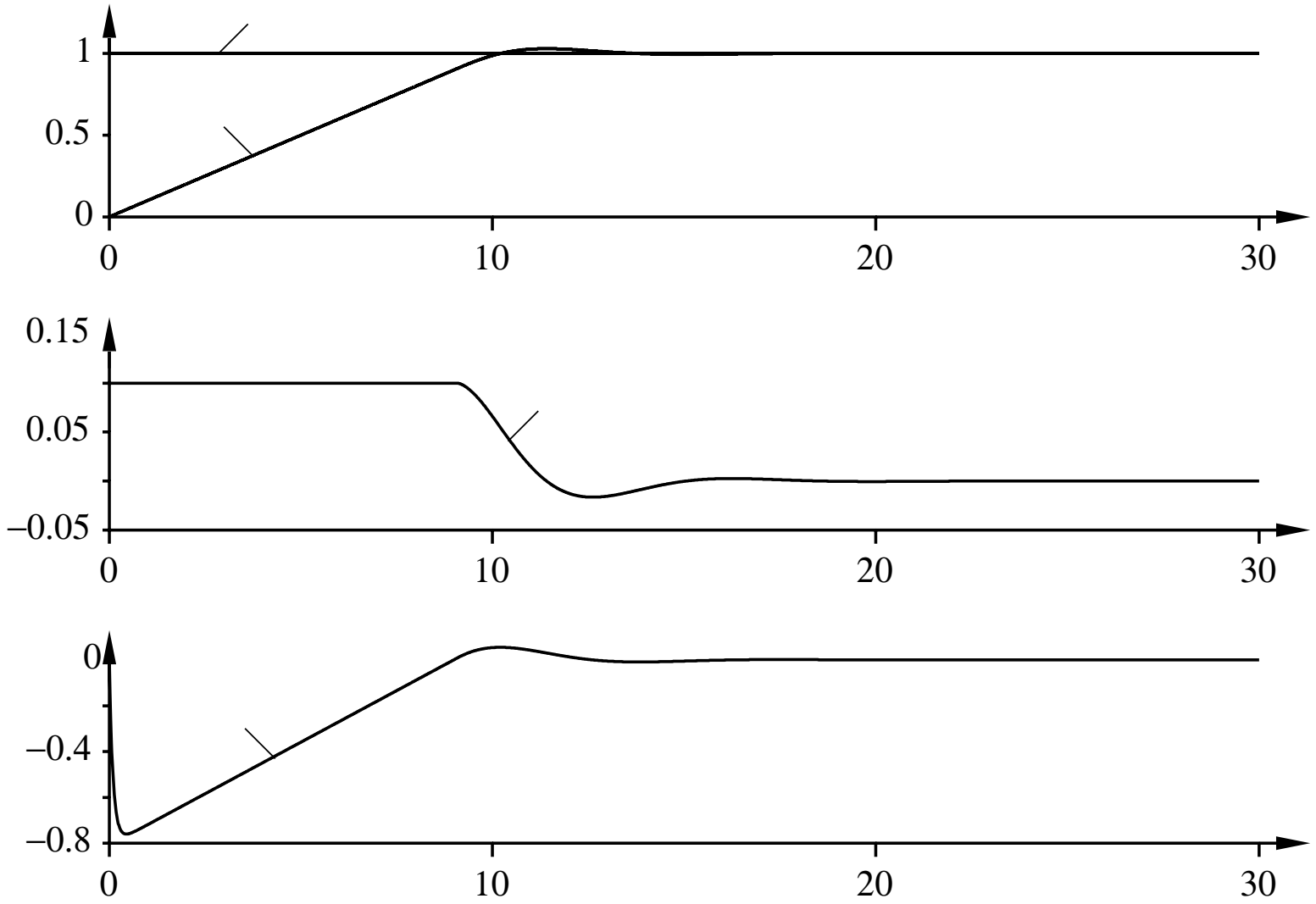
# Tracking

- when the control signal saturates, the integral is recomputed so that its new value gives a control signal at the saturation limit
- to avoid resetting the integral due to, e.g., measurement noise, the re-computation is done dynamically, i.e., through a LP-filter with a time constant $T_r$.

# Tracking


−y
$KT_d s$
Actuator
$e = r - y$
$K$
Σ
$v$
$u$
$\frac{K}{T_i}$
Σ
$\frac{1}{s}$
−
+
Σ
$e_s$
$\frac{1}{T_t}$
−y
$KT_d s$
Actuator model
Actuator
$e = r - y$
$K$
Σ
$\frac{K}{T_i}$
Σ
$\frac{1}{s}$
−
+
Σ
$e_s$
$\frac{1}{T_t}$

# Tracking



# Discretization

**P-part:**

$$u_P(k) = K(\beta y_{sp}(k) - y(k))$$

# Discretization

**I-part:**

$$I(t) = \frac{K}{T_I}\int_0^t e(\tau)d\tau, \frac{dI}{dt} = \frac{K}{T_I}e$$

- Forward difference

$$\frac{I(t_{k+1}) - I(t_k)}{h} = \frac{K}{T_I}e(t_k)$$

```
I(k+1) := I(k) + (K*h/Ti)*e(k)
```

The I-part can be precalculated in UpdateStates

- Backward difference

The I-part cannot be precalculated, i(k) = f(e(k))

- Others

# Discretization

**D-part** (assume $\gamma = 0$):

$$D = K \frac{sT_D}{1 + sT_D/N}(-Y(s))$$

$$\frac{T_D}{N}\frac{dD}{dt} + D = -KT_D\frac{dy}{dt}$$

- Forward difference (unstable for small $T_D$)

# Discretization, cont.

**D-part**:

- Backward difference

$$\frac{T_D}{N}\frac{D(t_k) - D(t_{k-1})}{h} + D(t_k)$$

$$= -KT_D\frac{y(t_k) - y(t_{k-1})}{h}$$

$$D(t_k) = \frac{T_D}{T_D + Nh}D(t_{k-1})$$

$$-\frac{KT_DN}{T_D + Nh}(y(t_k) - y(t_{k-1}))$$

# Discretization

**Tracking:**

```
v := P + I + D;
u := sat(v,umax,umin);
I := I + (K*h/Ti)*e + (h/Tr)*(u - v);
```

# PID code

PID-controller with anti-reset windup

```
y = yIn.get(); // A-D conversion
e = yref - y;
D = ad * D - bd * (y - yold);
v = K*(beta*yref - y) + I + D;
u = sat(v,umax,umin)}
uOut.put(u);   // D-A conversion
I = I + (K*h/Ti)*e + (h/Tr)*(u - v);
yold = y
```

$ad$ and $bd$ are precalculated parameters given by the backward difference approximation of the D-term.

Execution time for CalculateOutput can be minimized even further.

# Alternative PID Implementation

The PID controller described so far has constant gain, $K(1+N)$, a high frequencies, i.e., no roll-off at high frequencies.

An alternative is to instead of having a low pass filter only on the derivative part use a second-order low-pass filter on the measured signal before it enters a PID controller with a pure derivative part.

$$Y_f(s) = \frac{1}{T_f^2 s^2 + 1.4 T_f s + 1} Y(s)$$

$$U(s) = K(\beta Y_{ref}(s) - Y_f(s) + \frac{1}{T_I s}(Y_{ref}(s) - Y_f(s)) - T_D s Y_f(s))$$

# Class SimplePID

```
public class SimplePID {
  private double u,e,v,y;
  private double K,Ti,Td,Beta,Tr,N,h;
  private double ad,bd;
  private double D,I,yOld;

  public SimplePID(double nK, double nTi, double NTd,
              double nBeta, double nTr, double nN, double nh) {
    updateParameters(nK,nTi,nTd,nBeta,nTr,nN,nh);
  }

  public void updateParameters(double nK, double nTi, double NTd,
                           double nBeta, double nTr, double nN, double nh) {
    K = nK; Ti = nTi; Td = nTd; Beta = nBeta;
    Tr = nTr
    N = nN;
    h = nh;
    ad = Td / (Td + N*h);
    bd = K*ad*N;
  }
```

```
  public double calculateOutput(double yref, double newY) {

    y = newY;
    e = yref - y;
    D = ad*D - bd*(y - yOld);
    v = K*(Beta*yref - y) + I + D;
    return v;
  }

  public void updateState(double u) {

    I = I + (K*h/Ti)*e + (h/Tr)*(u - v);
    yOld = y;
  }

}
```

# Extract from Regul

```
public class Regul extends Thread {
  private SimplePID pid;

  public Regul() {
    pid = new SimplePID(1,10,0,1,10,5,0.1);
  }

  public void run() {
    // Other stuff

    while (true) {
      y = getY();
      yref = getYref():
      u = pid.calculateOutput(yref,y);
      u = limit(u);
      setU(u);
      pid.updateState(u);
      // Timing Code
    }
  }
}
```

# Task Models

# Further reading


- B. Wittenmark, K. J. Åström, K.-E. Årzén: “Computer Control: An Overview.” IFAC Professional Brief, 2002. (Available at http://www.control.lth.se)
- K. J. Åström, B. Wittenmark: *Computer-Controlled Systems,* Third Ed. Prentice Hall, 1997.
- K. J. Åström, Tore Hägglund: *Advanced PID Control.* The Instrumentation, Systems, and Automation Society, 2005.