Real-Time Control

Anton Cervin

Department of Automatic Control
Lund University
Sweden

Today's outline

- Part I – Introduction to Automatic Control
- Part II – Sampled-Data and Networked Control
- Part III – Integrated Control and Scheduling
- Part IV – Computer Exercise

Further Reading

- B. Wittenmark, K. J. Åström, K.-E. Årzén: "Computer Control: An Overview." IFAC Professional Brief, 2002. (93 pages, available at http://www.control.lth.se)
- K. J. Åström, Tore Hägglund: "Advanced PID Control." The Instrumentation, Systems, and Automation Society, 2005.
- A. Cervin: "Integrated Control and Real-Time Scheduling." PhD Thesis, Lund University, 2003. (Available at http://www.control.lth.se)

Real-Time Control

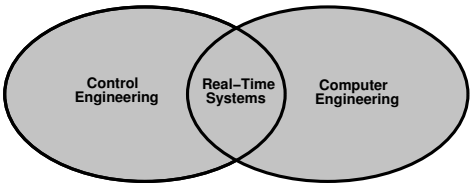
Part I – Introduction to Automatic Control

Part I – Outline

1. Introduction
2. Basic concepts
3. Modeling and design
4. Empirical PID control

1. Introduction

Real-time and control



- All control systems are real-time systems
- Many hard real-time systems are control systems

Real-time and control

- Control engineers need real-time systems to implement their systems.
- Computer engineers need control theory to build "controllable" systems
- Interesting research problems in the interface
- Much can be lost without integration

Automatic control

The silent technology:

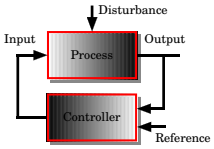
- Widely used
- Very successful
- Seldom talked about, except when disaster strikes!

Automatic control

Use of **models** and **feedback**

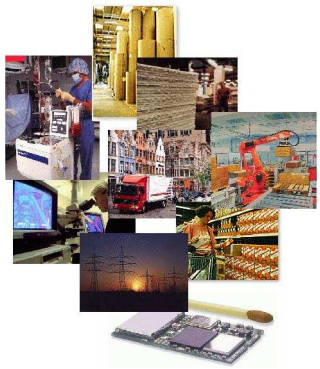
Activities:

- Modeling
- Analysis and simulation
- Control design
- Implementation

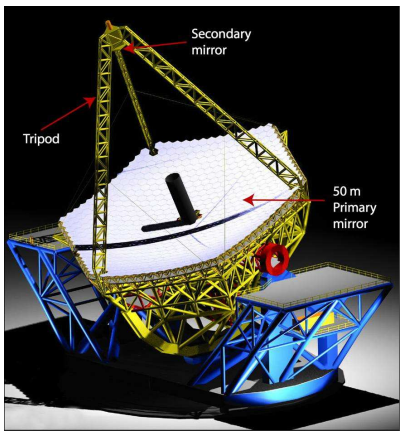


Applications

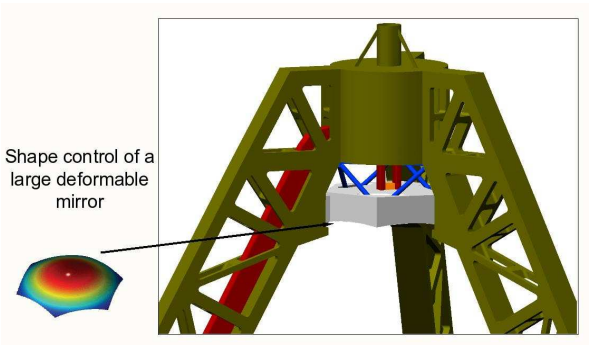
- Automotive systems
- Robotics
- Biotechnology
- Power systems
- Process control
- Communications
- Consumer electronics
- ...



Example: The EURO 50 telescope



The deformable mirror

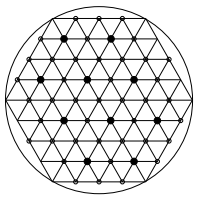


- Compensate for atmospheric disturbances 1000 times/sec

Control of deformable mirror

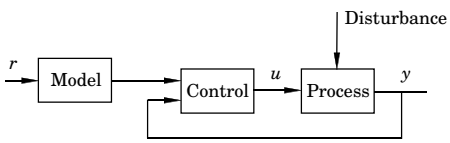
Control the system $M\ddot{x} + C\dot{x} + Kx = Fu$ to the equilibrium $Kx = Fu_r$ using measurements of $y = Ex$ and $\dot{y}$.

- Large number of sensors and actuators (2000-3000)
- Computational limitations (1kHz)
- Cannot measure and control at the same spot
- Large uncertainty in C



2. Basic Concepts

Basic setting



Must handle two tasks:

- Follow reference signals, r
- Compensate for disturbances

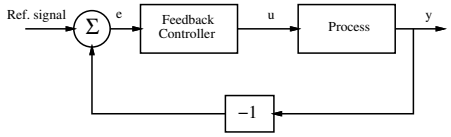
How to

- do several things with the control signal u

The feedback principle

A very powerful idea, that often leads to revolutionary changes in the way systems are designed.

The primary paradigm in automatic control.

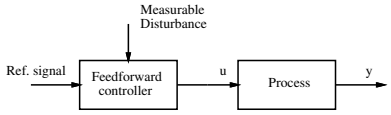


- Base corrective action on an error that has occurred
- Closed loop

Properties of feedback

+ Reduces influence of disturbances
+ Reduces effect of process variations
+ Does not require exact models
− Feeds sensor noise into the system
− May lead to instability, e.g.:
  - if the controller has too high gain
  - if the feedback loop contains too large time delays
    - from the process
    - from the controller implementation

The feedforward principle



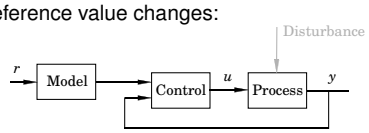
- Take corrective action before an error has occurred
- Measure the disturbance and compensate for it
- Use the fact that the reference signal is known and adjust the control signal to the reference signal
- Open loop

Properties of feedforward

+ Reduces effect of disturbances that cannot be reduced by feedback
+ Measurable signals that are related to disturbances
+ Allows faster set-point changes, without introducing control errors
− Requires good models
− Requires stable systems

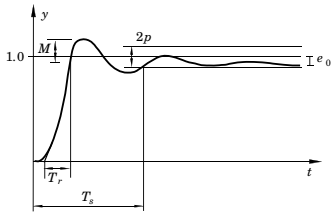
The servo problem

Focus on reference value changes:



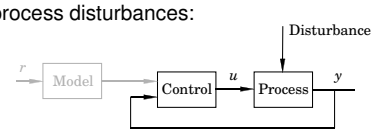
Typical design criteria:

- Rise time, T_r
- Overshoot, M
- Settling time, T_s
- Steady-state error, e_0
- ...



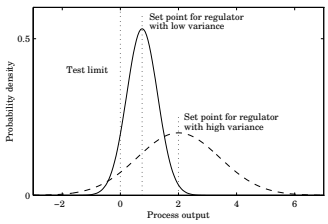
The regulator problem

Focus on process disturbances:

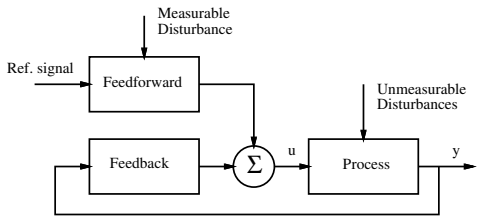


Typical design criteria:

- Output variance
- Control signal variance



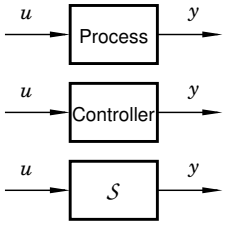
Putting it all together



Combination of **feedback** and **feedforward**

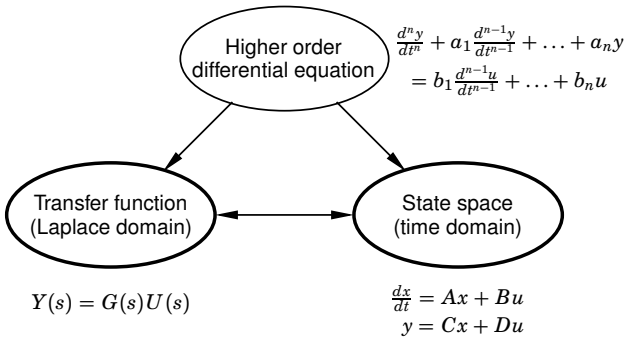
3. Modeling and Design

Modeling

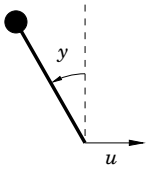


- View all subsystems as “boxes” with inputs and outputs
- Linear, time-invariant (LTI) dynamical systems
- Continuous or discrete time

Continuous-time systems



Example: Inverted pendulum



Nonlinear differential equation from physical modeling:

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

Linearized model around $y^0 = 0$ $(\sin y \approx y, \cos y \approx 1)$:

$$\frac{d^2y}{dt^2} = \omega_0^2 y + ku$$

Inverted pendulum in state space form

Introduce state variables

- $x_1 = y$ (pendulum angle)
- $x_2 = \frac{dy}{dt}$ (pendulum angular velocity)

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

Inverted pendulum in transfer function form

Apply Laplace transform to differential equation:

$$s^2 Y(s) = \omega_0^2 Y(s) + kU(s)$$
$$G(s) = \frac{Y(s)}{U(s)} = \frac{k}{s^2 - \omega_0^2}$$

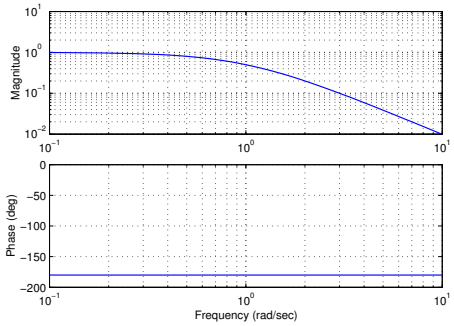
Or, from state space to transfer function:

$$G(s) = C(sI - A)^{-1}B = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} s & -1 \\ -\omega_0^2 & s \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ k \end{pmatrix} = \frac{k}{s^2 - \omega_0^2}$$

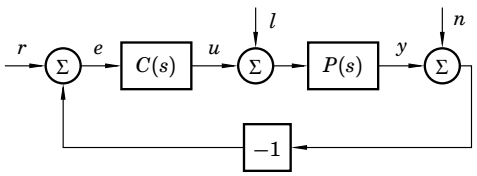
Frequency response: $G(i\omega)$

Frequency response

Plot $|G(i\omega)|$ and $\arg G(i\omega)$ for $\omega \in [0, \infty]$



Model-based design



Given $P(s)$, determine $C(s)$ such that the specifications on the closed-loop system are met. Common approaches:

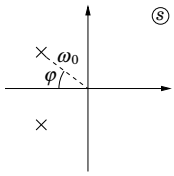
- Frequency domain design (loop shaping)
- Pole placement design
  - transfer function domain
  - state space domain
- Optimization-based methods (H_∞, LQG, ...)

Pole placement – transfer function domain

- Determine the required form of $C(s) = \frac{b_1 s^{n-1} + \ldots + b_n}{s^n + a_1 s^{n-1} + \ldots + a_n}$
- Calculate the closed loop system:

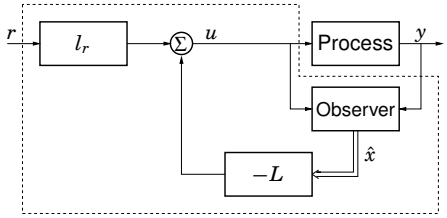
$$G_{cl}(s) = \frac{P(s)C(s)}{1 + P(s)C(s)}$$

- Choose the coefficients of $C(s)$ such that you get the desired closed-loop poles



- Large ω_0 ⇔ fast system
- Large φ ⇔ poorly damped system

Pole placement – state space domain



State feedback from an observer:

$$\frac{d\hat{x}}{dt} = A\hat{x} + Bu + K(y - C\hat{x})$$
$$u = -L\hat{x} + l_r r$$

Choose gain vectors L and K to give desired closed-loop poles

4. Empirical PID Control

PID control

- The oldest controller type
- The most widely used
  - Pulp & Paper 86%
  - Steel 93%
  - Oil refineries 93%
- Much to learn!!

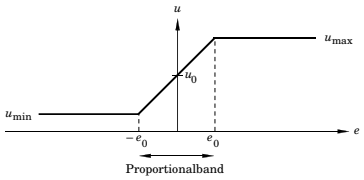
The textbook algorithm

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right)$$

$$U(s) = K\left(E(s) + \frac{1}{sT_i}E(s) + T_d sE(s)\right)$$

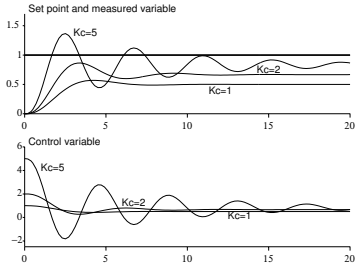
$$= P + I + D$$

Proportional term



$$u = \begin{cases} u_{\max} & e > e_0 \\ Ke + u_0 & -e_0 < e < e_0 \\ u_{\min} & e < -e_0 \end{cases}$$

Properties of P-control



- stationary error
- increased K means faster speed, increased noise sensitivity, worse stability

Errors with P-control

Control signal:

$$u = Ke + u_0$$

Error:

$$e = \frac{u - u_0}{K}$$

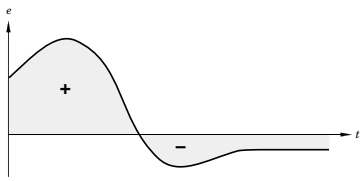
Error removed if:

- K equals infinity
- $u_0 = u$

Solution: Automatic way to obtain u_0

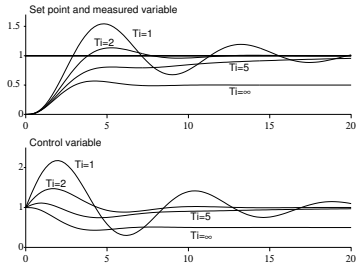
Integral term

$$u = Ke + u_0$$
$$u = K\left(e + \frac{1}{T_i}\int e(t)dt\right) \quad \text{(PI)}$$



Stationary error present → $\int edt$ increases → u increases → y increases → the error is not stationary

Properties of PI-control

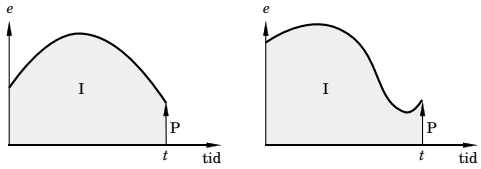


- removes stationary error
- smaller T_i implies worse stability, faster steady-state error removal

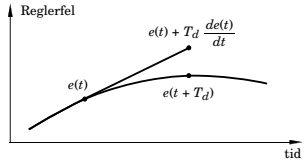
Prediction

A PI-controller contains no prediction

The same control signal is obtained for both these cases:



Derivative part



P:

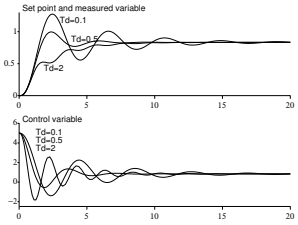
$$u(t) = Ke(t)$$

PD:

$$u(t) = K\left(e(t) + T_d\frac{de(t)}{dt}\right) \approx Ke(t + T_d)$$

T_d = Prediction horizon

Properties of PD-control



- T_d too small, no influence
- T_d too large, decreased performance

In industrial practice the D-term is often turned off.

Algorithm modifications

Modifications are needed to make the controller practically useful

- Limitations of derivative gain
- Derivative weighting
- Reference weighting
- Handle control signal limitations

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 reference is often constant for long periods of time

Reference often changed in steps → D-part becomes very large.

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

$$D(s) = \frac{sT_d}{1 + sT_d/N}(\gamma R(s) - Y(s))$$

Often, $\gamma = 0$

Reference weighting

An advantage to also use weighting on the reference.

$$u = K(r - y)$$

replaced by

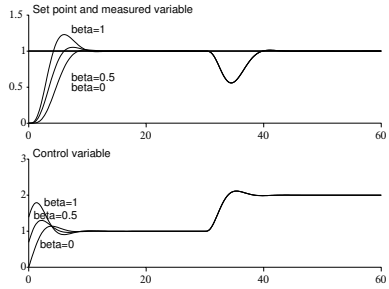
$$u = K(\beta r - y)$$

$0 \leq \beta \leq 1$

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

Improved set-point responses.

Reference weighting



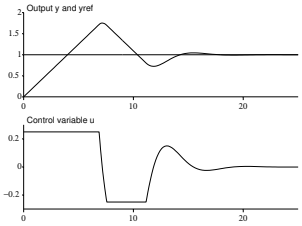
Control signal limitations

All actuators saturate.

Problems for controllers with integration.

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

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



Anti-windup

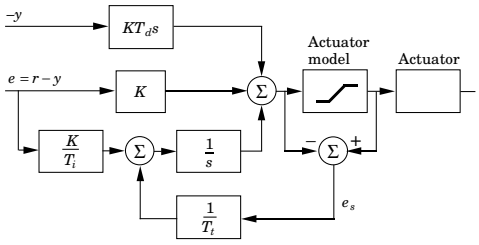
Several solutions exist:

- limit the reference variations (saturation never reached)
- conditional integration (integration is switched off when the control is far from the steady-state)
- tracking (back-calculation)

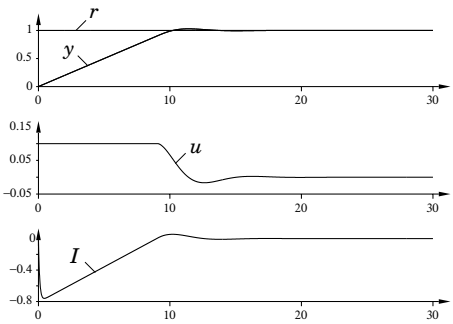
Tracking

- 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 recomputation is done dynamically, i.e., through a LP-filter with a time constant T_t.

Tracking



Tracking



Tuning

Parameters: $K, T_i, T_d, N, \beta, \gamma, T_t$

Methods:

- empirically, rules of thumb, tuning charts
- model-based tuning, e.g., pole-placement
- automatic tuning experiment
  - Ziegler-Nichols method
    - step response method
    - ultimate sensitivity method
  - relay method

Industrial reality

Canadian paper mill audit. Average paper mill: 2000 loops, 97% use PI, remaining 3% are PID, adaptive, ...

- default settings used
- poor performance due to bad tuning
- poor performance due to actuator problems

Real-Time Control

Part II – Sampled-Data and Networked Control

Anton Cervin

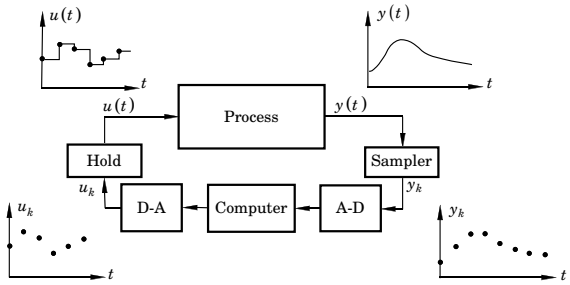
Department of Automatic Control
Lund University
Sweden

Outline – Part II

1. Introduction
2. Design of digital controllers
   - Sampled control theory
   - Approximation of continuous-time design
     - Discretization of the PID controller
   - Choice of sampling interval
3. Delay and jitter

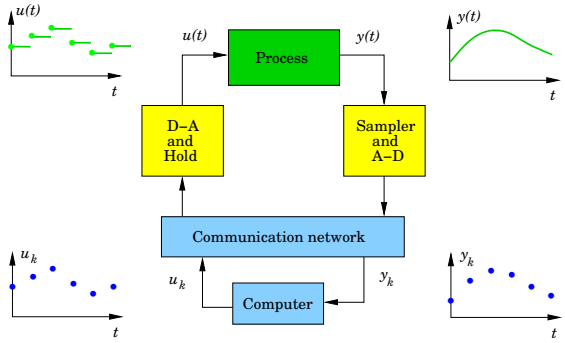
1. Introduction

Sampled-data control systems



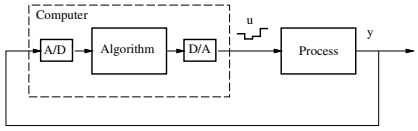
- Mix of continuous-time and discrete-time signals

Networked control systems

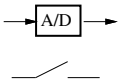


- Extra delay, possibly lost packets

Sampling



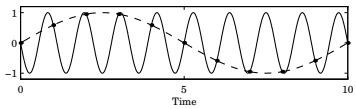
AD-converter acts as sampler



DA-converter acts as a hold device

Normally, zero-order-hold is used ⇒ piecewise constant control signals

Aliasing



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

$\omega_N = \omega_s/2$ = Nyquist frequency

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

The fundamental alias frequency for a frequency f_1 is given by

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

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

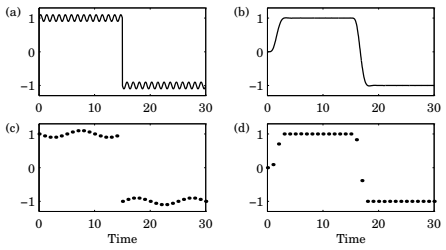
Anti-aliasing filter

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 $\omega_B = 0.25$

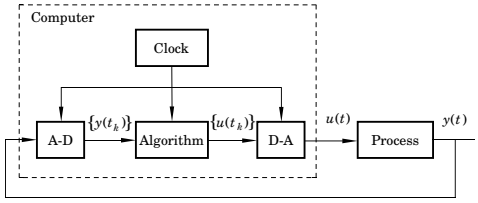
2. Design of digital controllers

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

Sampled control theory

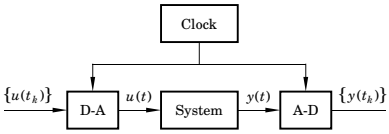


Basic idea: look at the sampling instances only

- 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 equation

Sampling a continuous-time system

System description

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

Solve the system equation

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

Periodic sampling

Assume periodic sampling, i.e. $t_k = k \cdot h$, then

$$\begin{aligned} x(kh+h) &= \Phi x(kh) + \Gamma u(kh) \\ y(kh) &= Cx(kh) + Du(kh) \end{aligned}$$

where

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

Time-invariant linear system!

Example: Sampling of inverted pendulum

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

We get

$$\begin{aligned} \Phi &= e^{Ah} = \begin{pmatrix} \cosh h & \sinh h \\ \sinh h & \cosh h \end{pmatrix} \\ \Gamma &= \int_0^h \begin{pmatrix} \sinh s \\ \cosh s \end{pmatrix} ds = \begin{pmatrix} \cosh h - 1 \\ \sinh h \end{pmatrix} \end{aligned}$$

Several ways to calculate Φ and Γ. 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

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

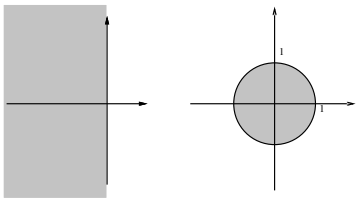
$$\Gamma_0 = \int_0^{h-\tau} e^{As}\, ds\; B$$

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

We get one extra state $(u(kh-h))$ 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.



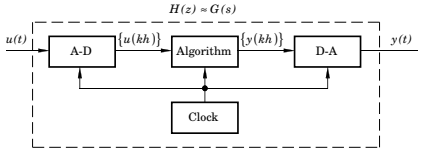
Digital control design

Similar to the continuous-time case, we can choose between

- frequency-domain design (loop shaping)
- pole-placement design
  - transfer function domain
  - state space domain
  - the poles are placed inside the unit circle
- optimal design methods (e.g. LQG)

Approximation of continuous-time design

Basic idea: Reuse the design



$G(s)$ is designed based on analog techniques

Want to get:

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

Methods:

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

Approximation methods

Forward Difference (Euler's method)

$$\frac{dx(t)}{dt} \approx \frac{x(t+h) - x(t)}{h}$$

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

Backward Difference

$$\frac{dx(t)}{dt} \approx \frac{x(t) - x(t-h)}{h}$$

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

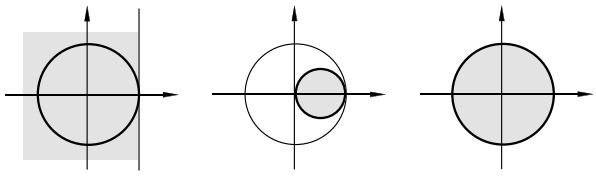
Tustin

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

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

Stability of approximations

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



Forward differences | Backward differences | Tustin

Discretization of the PID controller

Continuous PID controller with set-point weighting β and $\gamma = 0$:

$$U(s) = K\left(\beta R(s) - Y(s) + \frac{1}{sT_i}\left(R(s) - Y(s)\right) - \frac{sT_d}{1 + sT_d/N}\, Y(s)\right)$$

Discretization

P-part:

$$P(k) = K(\beta r(k) - y(k))$$

Discretization

I-part:

$$I(t) = \frac{K}{T_I}\int_0^t (r(\tau) - y(\tau))d\tau$$

$$\frac{dI}{dt} = \frac{K}{T_I}(r(t) - y(t))$$

- Forward difference

$$\frac{I(k+1) - I(k)}{h} = \frac{K}{T_I}(r(k) - y(k))$$

```
I(k+1) := I(k) + (K*h/Ti)*(r(k)-y(k))
```

The I-part can be precalculated

- Backward difference
  The I-part cannot be precalculated, $I(k) = f(r(k), y(k))$

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)
- Backward difference

$$\frac{T_D}{N}\frac{D(k) - D(k-1)}{h} + D(k) = -KT_D\frac{y(k) - y(k-1)}{h}$$

$$D(k) = \frac{T_D}{T_D + Nh}D(k-1) - \frac{KT_DN}{T_D + Nh}(y(k) - y(k-1))$$

Discretization

Tracking:

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

PID code

PID-controller with anti-windup ($\gamma = 0$).

```
r = ref.get();
y = yIn.get();
D = ad * D - bd * (y - yold);
v = K*(beta*r - y) + I + D;
u = sat(v,umax,umin);
uOut.put(u);
I = I + (K*h/Ti)*(r - y) + (h/Tt)*(u - v);
yold = y;
```

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

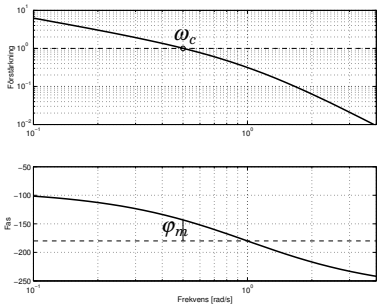
Choice of sampling interval

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



- ω_c = cross-over frequency, φ_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

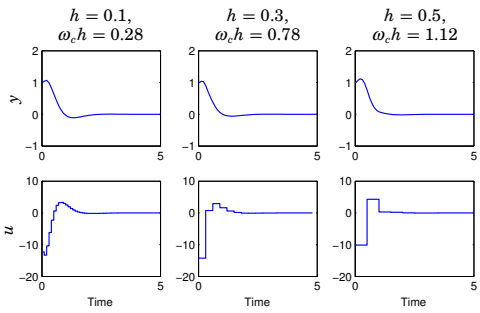
$$\arg G_{S\&H}(i\omega_c) = \arg e^{-i\omega_c h/2} = -\omega_c h/2$$

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

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

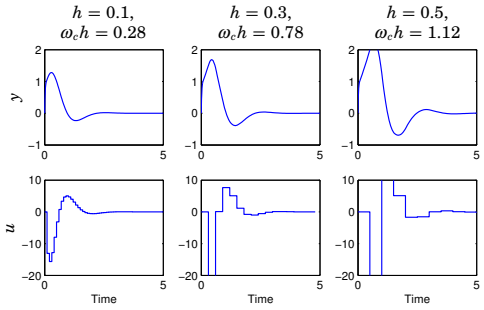
This corresponds to a Nyquist frequency about 6 to 20 times larger than the crossover frequency

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 $5°$ to $15°$. 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

3. Delay and jitter

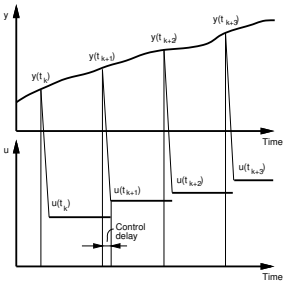
Computational delay

Problem: $u(k)$ cannot be generated instantaneously at time k when $y(k)$ is sampled

Computational delay (control delay) due to execution time

```
LOOP
  wait for clock interrupt;
  read analog input;
  perform calculations;
  set analog output;
END;
```

Control delay



Minimizing the computational delay

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;
```

Sampling interval and delay rule of thumb

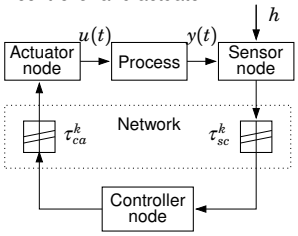
Assume that the delay is τ. This gives an additional phase margin loss of $-\omega_c \tau$. Extending our first rule of thumb we get

$$\boxed{0.15 < \omega_c(h + 2\tau) < 0.5}$$

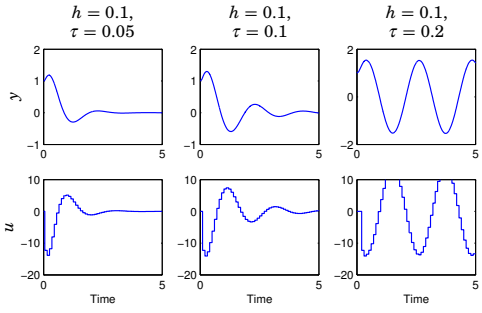
- If the delay is too large, we **must** decrease the speed of the controlled system (i.e. the cross-over frequency ω_c)
  - The delay imposes a fundamental performance limitation

Other sources of time delays

- Deadtime in the process
  - deadtime after the actuator
  - deadtime before the sensor
- Communication delays
  - between sensor and controller
  - between controller and actuator



Pendulum controller with time delay



- No delay compensation

Delay margin

Suppose the loop transfer function without delay has

- cross-over frequency ω_c
- phase margin φ_m

Phase margin loss due to delay:

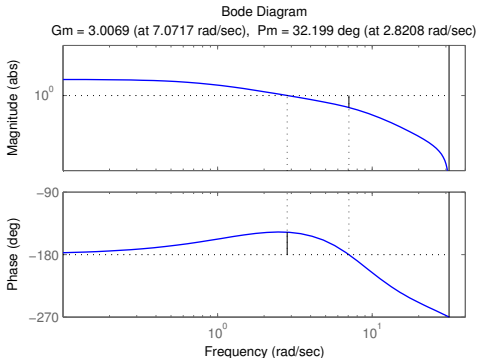
$$\arg e^{-i\omega_c\tau} = -\omega_c\tau$$

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**

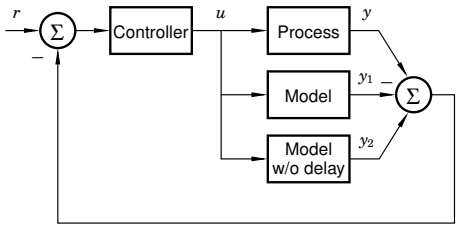
Example: delay margin for pendulum controller



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

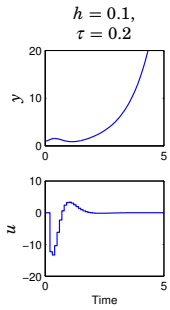
Delay compensation using Smith predictor

Idea: control against simulated model without delay:



- Requires **accurate** and **stable** model

Pendulum controller with Smith predictor



- The controller thinks that it is doing the right thing
- Based on feedforward rather than feedback

Delay compensation using pole placement

- Sample the model with the time delay
  - Gives a model with $d = \left\lceil \frac{\tau}{h} \right\rceil$ extra states
- Place all closed-loop poles within the unit circle
  - As a first attempt, place the extra poles in the origin
- Try to respect the rule of thumb

$$0.15 < \omega_c(h + 2\tau) < 0.5$$

Delay compensation in state space form

Assume a constant delay $\tau \leq h$.

Sampled model:

$$x(k+1) = \Phi x(k) + \Gamma_0 u(k) + \Gamma_1 u(k-1)$$

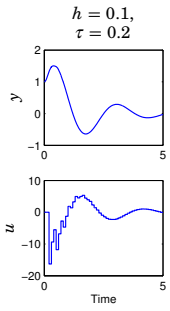
Observer and state feedback:

$$\hat{x}(k) = (I - KC)\Big(\Phi\hat{x}(k-1) + \Gamma_0 u(k-1) + \Gamma_1 u(k-2)\Big) + Ky(k)$$

$$u(k) = -L\begin{pmatrix}\hat{x}(k)\\ u(k-1)\end{pmatrix}$$

(Extension to $\tau > h$ is straightforward)

Pendulum controller with delay compensation



- Shaky response, nervous control signal
- $\omega_c(h + 2\tau) = 1.4$

Delay jitter

- In general, it is the **average** value of the delay that determines the degree of control performance degradation.
- However, jitter (i.e. time variation) in the delay makes it harder to compensate for.
- Jitter can be removed at the expense of more delay, using buffers.
  - Trade-off between delay and jitter
  - Most often, a short but jittery delay is preferred over a long but constant delay
  - But, counter-examples exist!

Jitter compensation in state space form

Assume a **time-varying** delay $\tau_k \leq h$.

Sampled model:

$$x(k+1) = \Phi x(k) + \Gamma_0(\tau_k)u(k) + \Gamma_1(\tau_k)u(k)$$

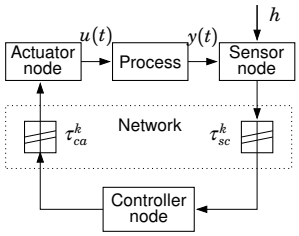
Problem: current delay τ_k is not known at time k! Solution:

- Base the L calculation on the **average** delay, $\mathrm{E}\{\tau\}$
- Measure the actual delay at the output
- Make the observer time-varying

$$\hat{x}(k) = (I - KC)\Big(\Phi\hat{x}(k-1) + \Gamma_0(\tau_k)u(k-1) + \Gamma_1(\tau_k)u(k-2)\Big) + Ky(k)$$

$$u(k) = -L\begin{pmatrix}\hat{x}(k)\\ u(k-1)\end{pmatrix}$$

Jitter compensation in networked systems



Part of the current loop delay (τ_{sc}) can now be measured!

- Time-varying state feedback L_k based on $\tau_{sc}^k + \mathrm{E}\{\tau_{ca}\}$
- Let the actuator node record the total delay
- The total delay is communicated back to the controller
- Make the observer time-varying as before

Real-Time Control

Part III – Integrated Control and Scheduling

Anton Cervin

Department of Automatic Control
Lund University
Sweden

Outline

1. Task models for control

The simple task model

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 deadline $D_i = T_i$

Is this model suitable for control tasks?

Fixed period?

Not necessarily:

- Some controllers are not sampled against time
  - Engine controllers
- Some controllers may switch between different modes with different sampling intervals
  - Hybrid controllers
- The control task could be triggered sporadically by measurements arriving over the network

Fixed and known WCET?

Not always:

- WCET analysis is a very hard problem
  - May have to use estimates or measurements
- Some controllers may switch between different modes with different execution times
  - Hybrid controllers
- Some controllers can explicitly trade off execution time for quality of control
  - "Any-time" algorithms
  - Model-predictive controllers (MPC)
  - Long execution time $\Leftrightarrow$ High quality of control

Hard deadlines?

Most often not:

- Controller deadlines are *firm* rather than hard
  - OK to miss a few outputs, but not too many in a row
- $D_i = T_i$ is a quite arbitrary choice
- Really depends on what happens when a deadline is missed
  - Late completion – often OK
  - Aborted computation (no new output) – worse

Inputs and outputs?

Completely missing from the simple task model:

- When are the inputs (measurement signals) read?
  - Beginning of period?
  - Beginning of task execution?
- When are the outputs (control signals) written?
  - End of period?
  - End of task execution?
  - Other time?

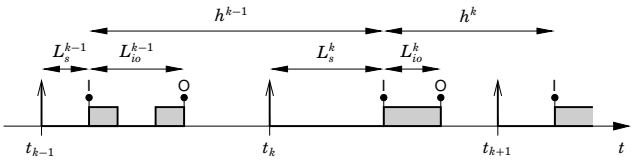
A simple control task

```
t = now();
while (1) {
  read_input();
  control_algorithm();
  write_output();
  t = t + T;
  wait_until(t);
}
```

- The input and output operations may be synchronous or asynchronous
- Trade-off between delay and jitter

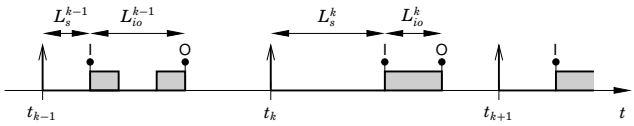
Control task timing

Assuming task-triggered inputs and outputs:



- L_s^k – sampling latency in period k
- L_{io}^k – input-output latency in period k
- h^k – actual sampling interval in period k

Jitter



- Absolute sampling jitter: $J_s \stackrel{\text{def}}{=} \max_k L_s^k - \min_k L_s^k$
- Absolute input-output jitter: $J_{io} \stackrel{\text{def}}{=} \max_k L_{io}^k - \min_k L_{io}^k$
- J_s and J_{io} can be found using scheduling theory

Computing the jitter

Under fixed-priority scheduling, exact response-time analysis can be used:

- worst-case analysis [Joseph & Pandya, 1986]
- best-case analysis [Redell & Sanfridson, 2002]:

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

Under EDF, response-time analysis is more complicated (and the exact best-case is not known)

Computing the jitter

Sampling jitter:

- Replace task τ_i by "sampling task" $\tilde{\tau}_i$ with $\tilde{C}_i = \epsilon$
- $\max L_{si} = \tilde{R}_i$
- $\min L_{si} = 0$
- $J_{si} = \tilde{R}_i$

Input-output jitter:

- $\max L_{ioi} = R_i$
- $\min L_{ioi} = R_i^b$
- $J_{ioi} = R_i - R_i^b$

Subtask scheduling

A control algorithm normally consists of two parts:

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

Idea: schedule the two parts as separate tasks

- reduce delay
- reduce jitter

Task models

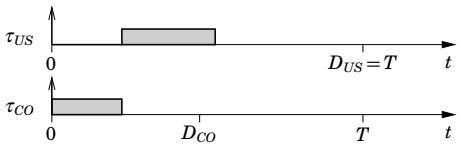
Each control task τ is divided into two subtasks:

- τ_{CO} – Calculate output
- τ_{US} – Update state
- Input and output operations are ignored in the analysis

Two possible task models:

- Priority-constrained – deadline-monotonic scheduling
- Offset model – EDF scheduling

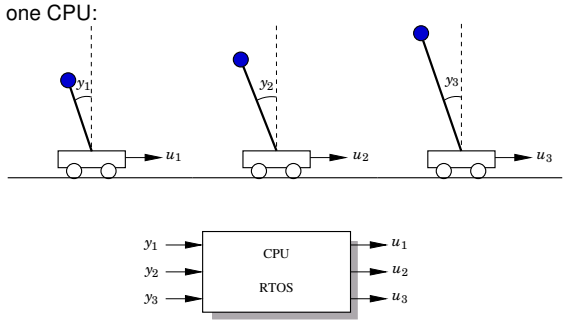
Deadline assignment



- $D_{CO} < D_{US}$
- We want to minimize D_{CO}. Iterative 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.

Example

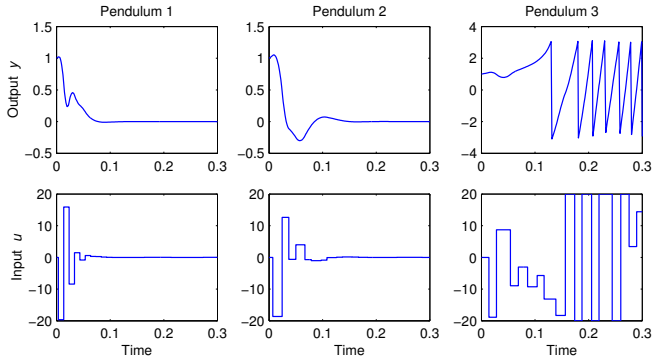
Suppose you want to control three inverted pendulums using one CPU:



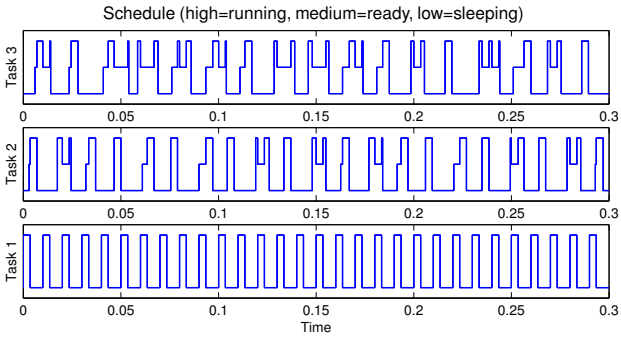
Example

- Discrete-time LQG controllers
- Execution time: $C_i = 3.5$ ms
- Sampling intervals: $(h_1, h_2, h_3) = (10, 14.5, 17.5)$ ms
- Control delay of 3.5 ms assumed in the design

Simulation under RM scheduling



Simulation under RM scheduling



- Large delay and jitter for controller 3

Subtask scheduling analysis

Each pendulum controller is divided into two subtasks:

- Calculate output: $C_{CO} = 1.5$ ms
- Update state: $C_{US} = 2.0$ ms

First iteration of algorithm:

	T	D	C	R
τ_{CO1}	10.0	8.0	1.5	1.5
τ_{US1}	10.0	10.0	2.0	3.5
τ_{CO2}	14.5	12.5	1.5	5.0
τ_{US2}	14.5	14.5	2.0	7.0
τ_{CO3}	17.5	15.5	1.5	8.5
τ_{US3}	17.5	17.5	2.0	14.0

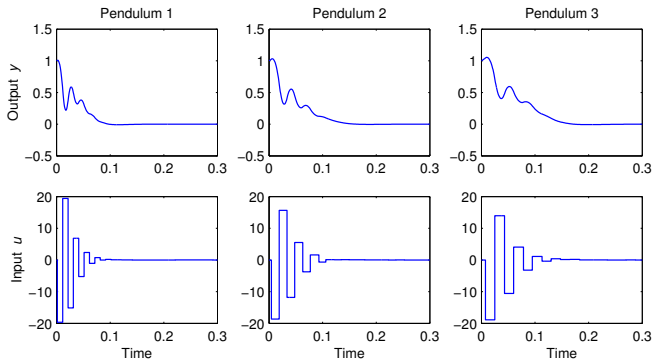
Subtask scheduling analysis

Third iteration (converged):

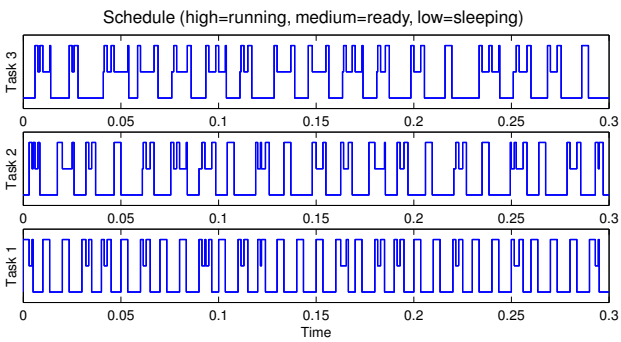
	T	D	C	R
τ_{CO1}	10.0	1.5	1.5	1.5
τ_{US1}	10.0	10.0	2.0	6.5
τ_{CO2}	14.5	3.0	1.5	3.0
τ_{US2}	14.5	14.5	2.0	8.5
τ_{CO3}	17.5	4.5	1.5	4.5
τ_{US3}	17.5	17.5	2.0	14.0

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

Simulation subtask scheduling



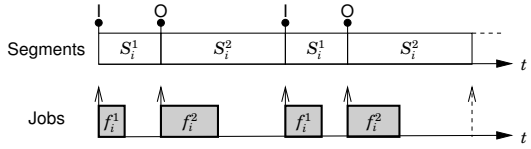
Simulation subtask scheduling



- More context switches

Extension: the Control Server model

- Schedule the inputs and outputs using kernel events (interrupts)
- Schedule the jobs of the subtasks (segments) using a modified Constant Bandwidth Server



A control server task

The control server has been implemented in Shark. Example task:

```
while (1) {
  // first segment
  r = read_input(0);
  y = read_input(1);
  u = calculate_output(r,y);
  write_output(0,u);
  task_endsegment();

  // second segment
  update_state();
  task_endsegment();
}
```

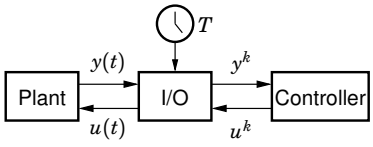
2. Handling overruns

Question

What to do in case of a controller execution overrun?

- **Abort** the computation?
- Continue the computation in the next period, but **skip** the next sample?
- Continue the computation in the next period, but **queue** the next sample?

Simple analysis of overruns



- Continuous-time plant
- Discrete-time controller with time-varying response time
- Synchronized, time-triggered inputs and outputs
  - Real-time control model assumed in Liu and Layland (1973)
  - Control server model
  - At least one sample input-output delay

Motivation

- Robustness against design faults
  - Very difficult to predict the worst-case response time
  - Treat the overrun as an exception and deal with it
  - Feedback in the computer system
- More efficient use of resources
  - The worst-case response time may be very rare
  - Trade-off between sampling period and risk of overruns

Stochastic Control Analysis

- Assume response-time probability density function $f_c(x)$
- Assume response times independent between samples
- Formulate jump linear system

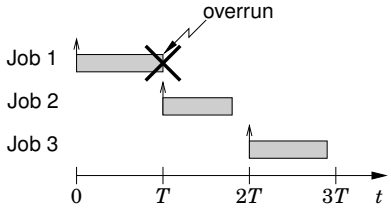
$$x(k+1) = A_i x(k) + B_i v(k)$$

  - x – plant, I/O and controller states
  - v – sampled noise process
  - i – current Markov node
- Evaluate quadratic cost function (performance index)

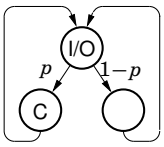
$$J = \lim_{t\to\infty} \frac{1}{t} \int_0^t x^T(\tau) Q x(\tau)\, d\tau$$

  - Well-known theory (1960s)
  - Jitterbug toolbox (Lincoln and Cervin, 2002)

The Abort Strategy



The Abort Strategy – Markov Chain



- Probability of no overrun:

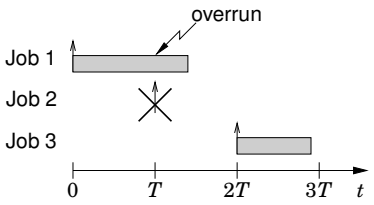
$$p = \int_0^T f_c(x)dx$$

The Abort Strategy – Implementation

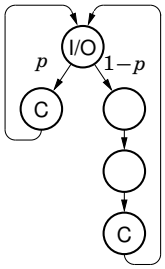
```
t := Clock();
loop
  select
    delay until t + T;
  then abort
    Read_input();
    Compute_control();
    Write_output();
  end
  t := t + T;
  delay until t;
end
```

- Possible in ADA, Real-Time Java, Shark(??)

The Skip Strategy



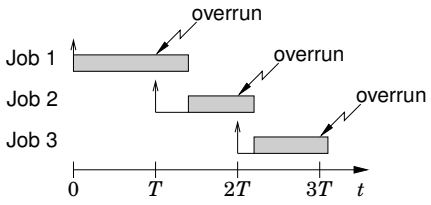
The Skip Strategy



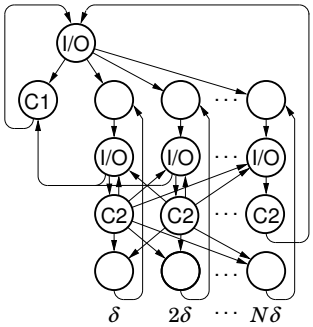
The Skip Strategy – Implementation

```
t := Clock();
late := false;
loop
  if not late then
    Read_input();
    Compute_control();
    Write_output();
  end
  t := t + T;
  if Clock() < t
    delay until t;
    late = false;
  else
    late = true;
  end
end loop
```

The Queue Strategy



The Queue Strategy – Markov Chain



- Queued execution time discretized with interval $\delta = T/N$
- C1 – ctrl using current sample, C2 – ctrl using old sample

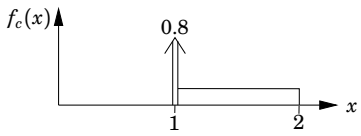
The Queue Strategy – Implementation

```
t := Clock();
loop
  Read_input();
  Compute_control();
  Write_output();
  t := t + T;
  delay until t;
end loop
```

- The textbook implementation of a control loop

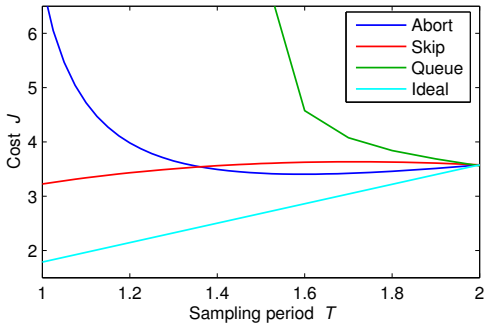
Example

- Plant: $P(s) = \frac{1}{s}$
- LQG controller with period T assuming one-sample delay and cost function $J = \lim_{t\to\infty} \frac{1}{t} \int_0^t y^2(\tau)\, d\tau$
- Response-time probability density function:



- Evaluate J for different $T \in [1, 2]$ and different strategies
- Ideal case without overruns: $J = \frac{\sqrt{3}+9}{6}\, T \approx 1.79T$

Example – Results



Example – Conclusions

- The Queue strategy performs the worst
  - Domino effect (c.f. EDF scheduling)
- The Skip strategy is the most robust one
  - Stable for all periods
  - Optimal period = nominal period ⇒ easy to design the controller (holds for many examples)
  - Easy to implement
  - Easy to analyze

Real-Time Control

Part IV – Computer Control Exercise

Anton Cervin

Department of Automatic Control
Lund University
Sweden

The Problem

Control of a (simulated) ball and beam process

Instructions

1. Design a continuous-time PID controller

$$C(s) = K\left(1 + \frac{1}{sT_i} + sT_d\right)$$

   that gives a fast ($\omega_0 \approx 3$ rad/s) and well-damped step response. Use pole placement design (see below)
2. Discretize the controller, after introducing
   - maximum derivative gain N
   - reference weighting β
   - tracking with time-constant T_t
3. Select a suitable sampling interval
4. Write a program that implements the controller
5. Try the controller against the simulated ball and beam and tune its parameters

Process Description

- Input u: beam angle (e.g. actuated by a servo motor)
- Output y: ball position (measured using e.g. a camera)

The process dynamics are given by

$$P(s) = \frac{10}{s^2}$$

To make it more interesting, the simulated process also has

- control signal limitation $u \in [-5.0,\ 5.0]$
- some process and measurement disturbances
- some unmodeled dynamics

Pole placement design

1. Compute the closed-loop transfer function

$$G_{cl}(s) = \frac{P(s)C(s)}{1 + P(s)C(s)} = \frac{p(s)}{q(s)}$$

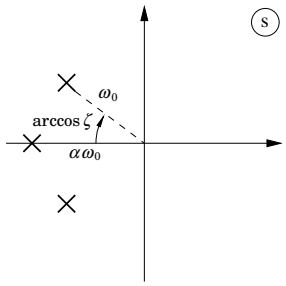
   (i.e. simplify the expression for $G_{cl}(s)$ to get a quotient between two polynomials in s)
2. The desired characteristic polynomial can be expressed as

$$(s + \alpha\omega_0)(s^2 + 2\zeta\omega_0 s + \omega_0^2)$$

   Set this equal to $q(s)$ and solve for K, T_i and T_d as expressions in ω_0, ζ and α!

Pole placement

Geometrical interpretation of the characteristic polynomial $(s + \alpha\omega_0)(s^2 + 2\zeta\omega_0 s + \omega_0^2)$:



Controller Implementation

- Implement the controller as a periodic thread in Shark
- After `#include <ballbeam.h>`, the following commands are available:

```
double get_position();  // read position of the ball
void set_angle(double); // set the angle of the beam
double get_reference(); // read the reference value
```