EL2450 Hybrid and Embedded Control

Lecture 4: Computer realization of controllers

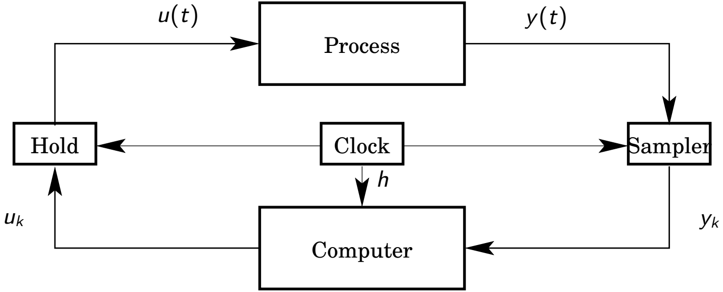
- Approximation of continuous-time designs
- Digital PID structures
- Realization of controllers
- Quantization
- Choice of sampling time

Today's Goal

You should be able to

- Transform continuous-time controller into discrete time
- Identify and select appropriate controller realizations
- Model and analyze quantization in computations and AD/DA converters
- Decide on reasonable sampling time

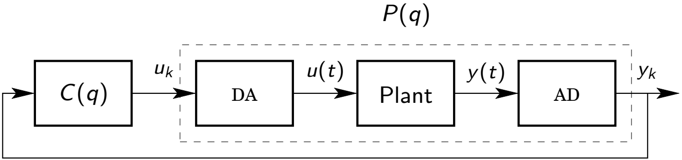
Time-Triggered Control System



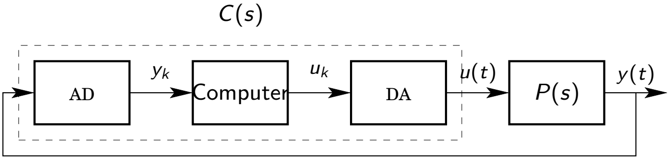
- **Q:** How apply continuous-time design methods to sampled control system?
- **A:** Make cuts at $u(t)$ and $y(t)$ (instead of at u_k and y_k)

Designs in Discrete or Continuous Time

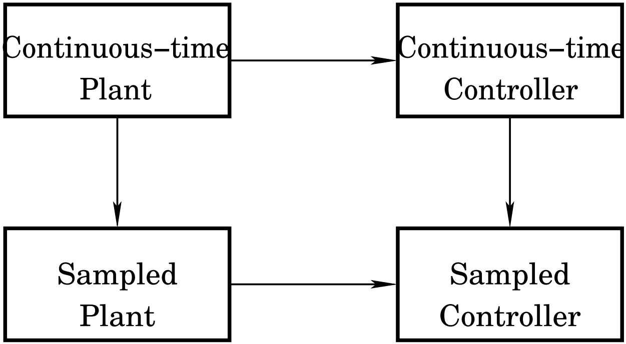
Sample plant, then derive a discrete-time controller:



Derive continuous-time controller, then transform to discrete-time algorithm:

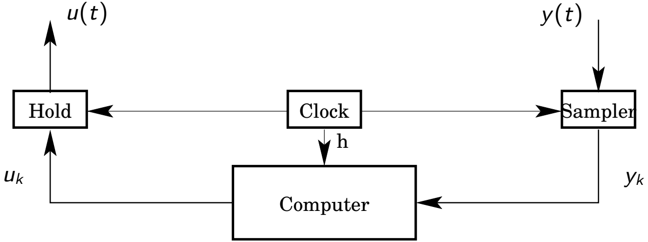


Designs in Discrete or Continuous Time



Approximating $C(s)$ by a Computer

Transform continuous-time design into discrete time:
Find $C(z)$ that approximates $C(s)$



Approximation of Continuous-Time Derivatives

Forward difference:

$$\frac{dx(t)}{dt} \approx \frac{x(t+h) - x(t)}{h} = \frac{q-1}{h} x(t)$$

Backward difference:

$$\frac{dx(t)}{dt} \approx \frac{x(t) - x(t-h)}{h} = \frac{q-1}{qh} x(t)$$

Tustin's approximation:

$$\frac{dx(t)}{dt} \approx \frac{2}{h} \cdot \frac{q-1}{q+1} x(t)$$

From $G(s)$ to $H(z)$

Pulse-transfer function $H(z)$ corresponding to transfer function $G(s)$ is

$$H(z) = G(s')$$

where

$$s' = \frac{z-1}{h} \qquad \text{(Forward difference)}$$

$$s' = \frac{z-1}{zh} \qquad \text{(Backward difference)}$$

$$s' = \frac{2}{h} \cdot \frac{z-1}{z+1} \qquad \text{(Tustin's approximation)}$$

Example

Discrete-time approximation of

$$G(s) = \frac{1}{s+1}$$

gives pulse-transfer functions

$$H_1(z) = \frac{h}{z-1+h} \qquad \text{(Forward difference)}$$

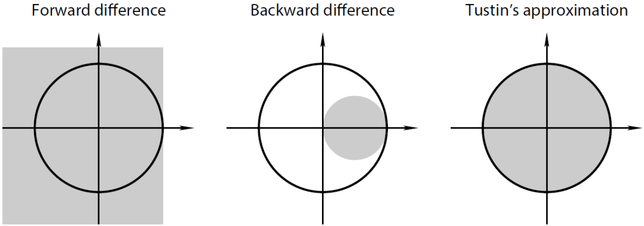
$$H_2(z) = \frac{h(1+h)^{-1}z}{z-(1+h)^{-1}} \qquad \text{(Backward difference)}$$

$$H_3(z) = \frac{h(2+h)^{-1}(z+1)}{z+(h-2)(h+2)^{-1}} \qquad \text{(Tustin's approximation)}$$

Note that H_1 is unstable for large h, but H_2 and H_3 are always stable.

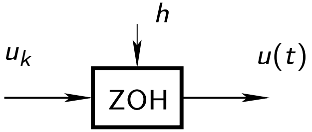
Stability Regions for $H(z)$

The stability region $\{s \in \mathbb{C} : \operatorname{Re} s < 0\}$ of continuous-time $G(s)$ is mapped to the regions below for discrete-time $H(z) = G(s')$:



- Forward difference may generate unstable $H(z)$ even if $G(s)$ is stable
- Backward difference may yield stable $H(z)$ even for unstable $G(s)$
- Tustin maps stable to stable

Transfer Function of a ZOH Circuit



Impulse response of $1/s$ is a step. Impulse response of e^{-s}/s is a delayed step. Hence, a ZOH circuit can be represented as

$$H(s) = \frac{1 - e^{-sh}}{s}$$

since pulses can be represented as a series of impulse responses.

- For small $h > 0$: $H(s) \approx \dfrac{1 - 1 + sh - s^2h^2/2 + \ldots}{s} = h - \dfrac{sh^2}{2} + \ldots$
- Steady-state gain $H(0) = h$

Transfer Function of a Sampler

Recall the relationship between the Fourier transforms of the continuous-time and sampled signals:

$$F_s(\omega) = \frac{1}{h} \sum_{k=-\infty}^{\infty} F(\omega + k\omega_s)$$

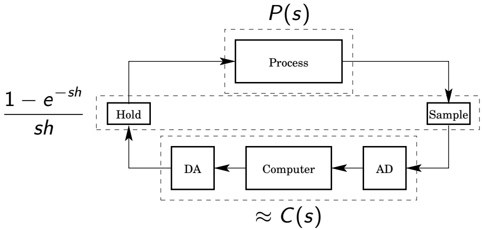
So there is a gain of $1/h$ between the frequency responses.

ZOH together with sampler hence contribute to the loop with

$$\frac{1 - e^{-sh}}{sh}$$

Continuous-Time Design of Sampled Controller

1. Design $C(s)$ based on $P(s)$ (or on $P(s)\dfrac{1-e^{-sh}}{sh}$ if h not small)
2. Derive discrete-time approximation of $C(s)$
3. Implement discrete-time algorithm



Continuous-time PID Controller

Continuous-time PID controller with $e = r - y$:

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

Implementation of pure derivative not desirable since it yields large amplification of measurement noise. Approximation:

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

Practical continuous-time PID

Continuous-time PID controller:

$$U(s) = K\left[bR(s) - Y(s) + \frac{1}{sT_i}\Big(R(s) - Y(s)\Big) - \frac{sT_d}{1 + sT_d/N}Y(s)\right]$$
$$= P(s) + I(s) + D(s)$$

Note the parameter $b \in [0, 1]$ and that r is not differentiated in the D-part. Using $p = \frac{d}{dt}$:

$$u(t) = K\left[br(t) - y(t) + \frac{1}{T_i p}\Big(r(t) - y(t)\Big) - \frac{T_d p}{1 + T_d p/N}y(t)\right]$$
$$= P(t) + I(t) + D(t)$$

Discretization of P-part

$$P(kh) = K\big[br(kh) - y(kh)\big]$$

Discretization of I-part

$$I(t) = \frac{K}{T_i}\int^t e(s)ds = \frac{K}{T_i p}e(t)$$

Forward approximation gives

$$I(kh+h) = I(kh) + \frac{Kh}{T_i}e(kh)$$

Discretization of D-part

$$\frac{T_d}{N} \cdot \frac{dD(t)}{dt} + D(t) = -KT_d \frac{dy(t)}{dt}$$

Backward approximation gives

$$\frac{T_d}{Nh}[D(kh) - D(kh-h)] + D(kh) = -\frac{KT_d}{h}[y(kh) - y(kh-h)]$$

so

$$D(kh) = \frac{T_d}{T_d + Nh} D(kh-h) - \frac{KT_d N}{T_d + Nh}[y(kh) - y(kh-h)]$$

Discretized PID Control

With P, I, D as given above, then

$$u(kh) = P(kh) + I(kh) + D(kh)$$

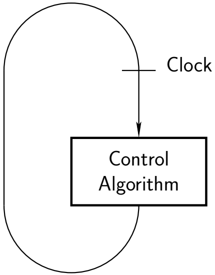
Note

- The given approximations (backward + forward) enable independent calculations of P, I, D
- Other approximations are possible (e.g., Tustin)
- Sometimes implementation on incremental form with output

$$\Delta u(kh) = u(kh) - u(kh - h)$$

Computer Code for Sampled-Data Controller

```
nexttime = getCurrentTime();
while (true) {
  AD_conversion();
  calculateOutput();
  DA_conversion();
  updateState();
  nexttime = nexttime + h;
  sleepUntil(nexttime);
}
```



- `calculateOutput` between `AD_conversion` and `DA_conversion` gives minimum computational time delay in control loop
- Controller states (such as observer state) are updated separately
- `sleepUntil` supposed to be supported by operating system

CalculateOutput and UpdateState

For a second-order controller on state space form

$$\begin{aligned} z(kh+h) &= Az(kh) + By(kh) \\ u(kh) &= Cz(kh) + Dy(kh) \end{aligned}$$

`calculateOutput()` is essentially

```
u = c1*z1 + c2*z2 + d*y;
```

and `updateState()` is essentially

```
z1 = a11*z1 + a12*z2 + b1*y;
z2 = a21*z1 + a22*z2 + b2*y;
```

Java Code for PID Controller

```
public static void main(String[] args) {
  double uc, y, u;
  PID pid = new PID();        // Construct PID controller object
  long time = System.getCurrentTimeMillis(); // Get current time
  while (true) {
    y = readY();
    r = readR();
    u = pid.calculateOutput(r,y);
    writeU(u);
    pid.updateStates;
    time = time + par.pid.h*1000;
    Thread.waitUntil(time);                  // Wait until 'time'
  }
}
```

```
public double calculateOutput(double r, double y) {
  signals.r = r;
  signals.y = y;
  double P = par.K*(par.b*r-y);
  states.D = par.ad*states.D-par.bd*(y - states.yold);
  signals.v = P + states.I + states.D;
  if (signals.v < par.ulow) {
    signals.u = par.ulow;
  } else {
    if (signals.v > par.uhigh) {
      signals.u = par.uhigh;
    } else {
      signals.u = signals.v;
    }
  }
  return signals.u;
}

public void updateStates() {
  states.I = states.I + par.bi*(signals.r - signals.y)
             + par.ar*(signals.u - signals.v); // Integral part
  states.yold = signals.y;
}
```

Comments on PID Computer Code

- Controller state update is separated from controller output calculations
- Controller parameters are pre-calculated (e.g., `par.bi`, `par.ad`)
- A *thread* (e.g., `Thread.waitUntil`) is a real-time task, and supposed to be supported by the operating system

Well-Conditioned Realizations

Consider controller

$$u(k) = \frac{b_o + b_1 q^{-1} + \cdots + b_m q^{-m}}{1 + a_1 q^{-1} + \cdots + a_n q^{-n}} y(k)$$

This can be realized in different forms, which influence

- Sensitivity to parameter and state quantization
- Number of storage elements and parameters (memory)
- Parameter range

Sensitivity Analysis

Consider characteristic polynomial for $i \in \{1, \ldots, n\}$:

$$A(z, a_i) = z^n + a_1 z^{n-1} + \cdots + a_n = (z - p_1) \ldots (z - p_n)$$

Suppose perturbation $a_i + \delta a_i$ gives $p_k + \delta p_k$. Then,

$$\begin{aligned} 0 &= A(p_k + \delta p_k, a_i + \delta a_i) \\ &= A(p_k, a_i) + \frac{dA(p_k, a_i)}{dz} \delta p_k + \frac{dA(p_k, a_i)}{da_i} \delta a_i + \ldots \end{aligned}$$

For small perturbations,

$$\delta p_k \approx -\frac{dA/da_i}{dA/dz}(p_k, a_i) \delta a_i$$

$$\frac{dA(p_k, a_i)}{da_i} = p_k^{n-i}, \qquad \frac{dA(p_k, a_i)}{dz} = \prod_{j \neq k}(p_k - p_j)$$

Hence, if $p_j \neq p_k$,

$$\delta p_k \approx -\frac{p_k^{n-i}}{\prod_{j \neq k}(p_k - p_j)} \delta a_i$$

Note

- Sensitive to parameter quantization if poles are close and/or close to one
- For $|p_k| < 1$, largest perturbation is obtained for $i = n$, i.e., sensitivity largest for perturbations in a_n

Bad Implementation I: Direct Form

$$u(k) = \sum_{i=0}^{m} b_i y(k-i) - \sum_{i=1}^{n} a_i u(k-i)$$

- Not minimal form: $m+n$ variables but only n states
- Parameters in implementation equal to characteristic polynomial: sensitive to computational errors if n large and poles close to one or close with each other

Bad Implementation II: Canonical Form

Canonical forms, such as observable canonical form:

$$
\begin{aligned}
x(k+1) &= \begin{bmatrix} -a_1 & 1 & 0 & \dots & 0 \\ -a_2 & 0 & 1 & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ -a_{n-1} & 0 & 0 & \dots & 1 \\ -a_n & 0 & 0 & \dots & 0 \end{bmatrix} x(k) + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} y(k) \\
u(k) &= \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \end{bmatrix} x(k)
\end{aligned}
$$

- Similar to in direct form: parameters in implementation equal to characteristic polynomial, so sensitive

Good Implementation

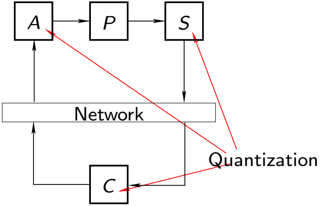
Controller with n_r distinct poles and n_c complex pairs:

$$
\begin{aligned}
z_i(k+1) &= \lambda_i z_i(k) + \beta_i y(k), && i = 1, \dots, n_r \\
v_i(k+1) &= \begin{bmatrix} \sigma_i & \omega_i \\ -\omega_i & \sigma_i \end{bmatrix} v_i(k) + \begin{bmatrix} \gamma_{i1} \\ \gamma_{i2} \end{bmatrix} y(k), && i = 1, \dots, n_c \\
u(k) &= \sum_{i=1}^{n_r} \alpha_i z_i(k) + \sum_{i=1}^{n_c} \delta_i^T v_i(k)
\end{aligned}
$$

- Robust to perturbations in parameters and states
- Parallel form implementation, as a cascade of first and second-order filters (see exercise 5.1)

Quantization

- Quantization in AD converters
- Quantization of controller parameters
- Roundoff, overflow, and underflow in operations (addition etc.)
- Quantization in DA converters



Quantization in AD and DA Converters

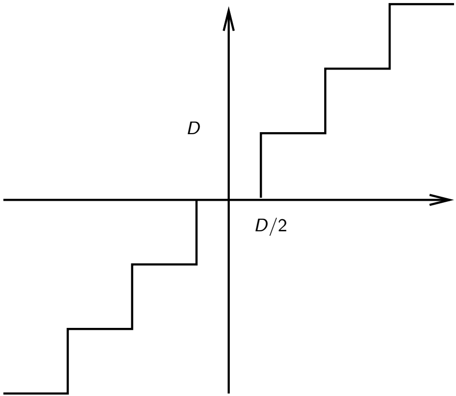
AD Converter

- Typical accuracy of 8, 10, 12, and 14 bits
- 8 bits correspond to $1/2^8 \approx 0.4\%$ resolution
- 14 bits correspond to $1/2^{14} \approx 0.006\%$ resolution

DA Converter

- Typical accuracy of 10 bits

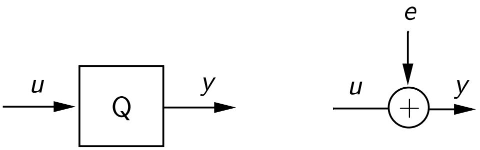
Uniform Quantization



Linear Model of Quantization

Model quantization error as a uniformly distributed stochastic signal e independent of u with

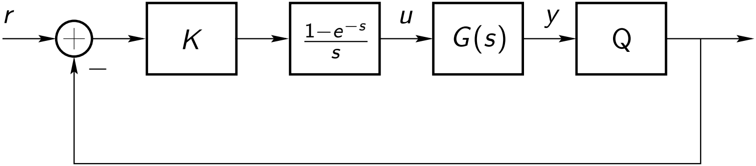
$$\mathrm{Var}(e) = \int_{-\infty}^{\infty} e^2 f_e \, de = \int_{-D/2}^{D/2} \frac{e^2}{D} \, de = \frac{D^2}{12}$$

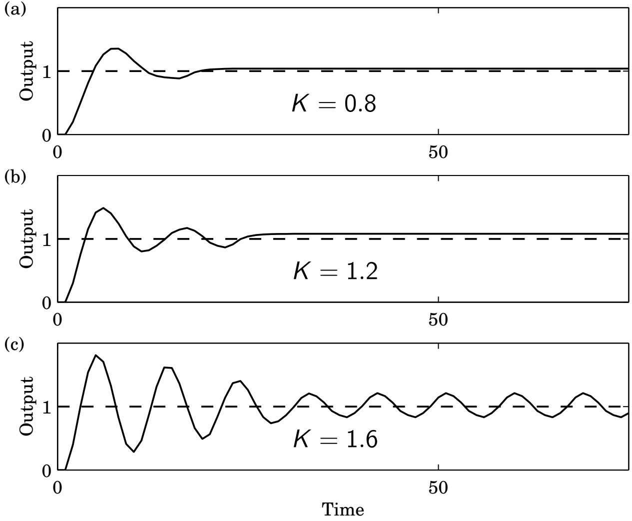


Works if D is small compared to the variations in u

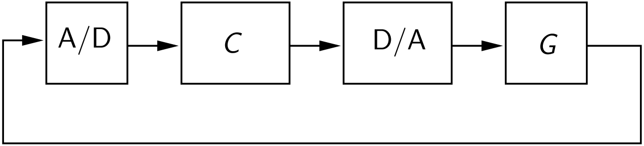
Example: Sensor Quantization

- Quantization of process output with $D = 0.2$
- Quantizer generates stable oscillation for $K = 1.6$
- Can be predicted using nonlinear control methods

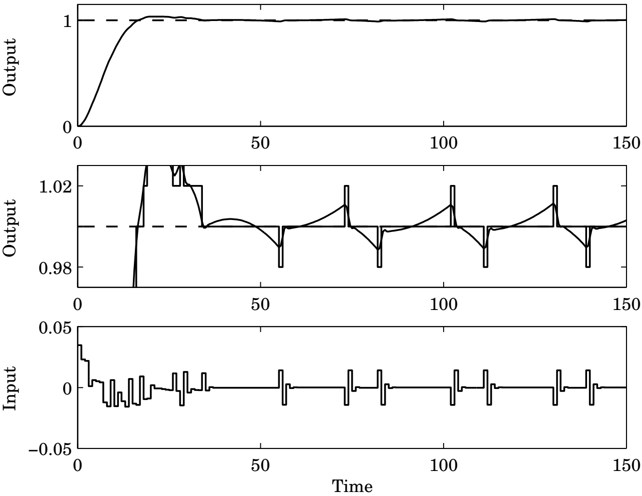



(a)
Output
1
0
0
50
$K = 0.8$
(b)
Output
1
0
0
50
$K = 1.2$
(c)
Output
1
0
0
50
$K = 1.6$
Time

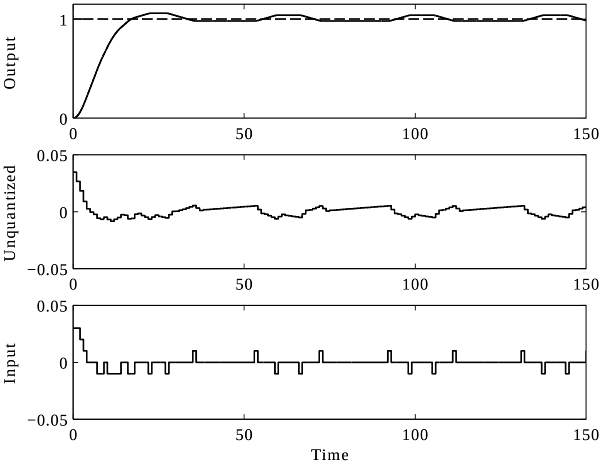
Example: Quantization in AD or DA



Quantization $D = 0.02$ in AD converter:



Quantization $D = 0.01$ in DA converter:



Choosing Sampling Time

Sampling time $h > 0$ might be

- Considered as an independent design parameter
- Fixed by the application or implementation platform
- Uncertain due to asynchronous sampling and hold
- Time varying due to communication and computation variations

Note

- Sampling time might heavily influence closed-loop performance
- Sometimes overlooked in the design
- Lack of systematic methods for choosing sampling time
- Use computer simulations off-line

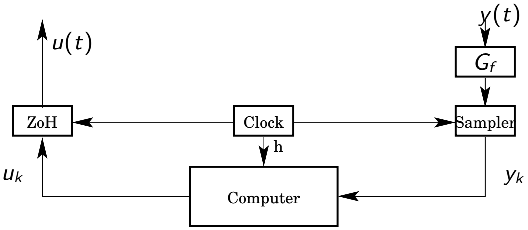
Sampling Time Rule of Thumb

- Choose h such that there will be 4 to 10 samples per rise time

For a second-order system with natural frequency ω_0 this gives
$0.2 < \omega_0 h < 0.6$
The sampling frequency ω_s should thus fulfill $10\omega_0 < \omega_s < 30\omega_0$, so the sampling frequency for control is much higher than the frequency $2\omega_0$ needed for reconstructing a sampled signal of frequency ω_0

The rule of the thumb is built on extensive experience, but not much theory

Another Sampling Time Rule of Thumb



Choose h such that the zero-order-hold (ZoH) and anti-aliasing filter (G_f) give phase margin decrease of 5 to 15 deg

With cross-over frequency ω_c for the continuous-time plant, this results (under certain assumptions) in

$$h\omega_c \approx 0.05 \quad \text{to} \quad 0.14$$

Why Not $h \to 0$?

If a plant

$$\dot{x} = Ax + Bu$$

is sampled fast ($h \approx 0$), then the discrete-time dynamics

$$x(k+1) = \Phi x(k) + \Gamma u(k)$$

looks approximately like integrators to the controller because

$$\Phi = e^{Ah} \approx I + Ah$$

It will then require high numerical accuracy in the controller (computer) to be able to distinguish the difference.

Next Lecture

- Quantization in state feedback
- Network delays and data drops
- Compensation for delay variations