

### Lecture 8: From Analog to Digital Controllers, PID Control

[IFAC PB Ch 6, Ch 8, RTCS Ch 10]

- Discrete-time approximation of continuous-time controllers
  - State-space domain
  - Frequency domain
- The PID Controller

### Design Approaches

![Diagram: Design Approaches]

- Control Design in Continuous Time
  - Discretized Controller
  - Difference Equation
  - Software Algorithm
- Control Design in Discrete Time

### Implementing a Continuous-Time Controller Using a Computer

![Diagram: Implementing a Controller]

Want to find discrete-time Algorithm such that

\[ A-D + \text{Algorithm} + D-A \approx \text{Continuous Controller} \]

### Differentiation and Tustin Approximations

Forward difference (Euler's method):

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

Backward difference:

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

Tustin’s approximation (trapezoidal method, bilinear transformation):

\[
\frac{x(t+h) + x(t)}{2} \approx \frac{x(t+h) - x(t)}{h}
\]

### State-Space Domain

Assume that the controller is given in state-space form

\[
\frac{dx}{dt} = A x + B u
\]

where \( x \) is the controller state, \( y \) is the controller output, and \( u \) is the controller input.

Forward or backward approximation of the derivative
The discrete-time approximation

Alternative: Write as differential equation first:

\[ \frac{dx(t)}{dt} = \frac{x(k+1) - x(k)}{h} \]

leads to

\[ \frac{x(k+1) - x(k)}{h} = Ax(k) + Bu(k) \]

which gives

\[ x(k+1) = (I + hA)x(k) + hBu(k) \]
\[ y(k) = Cx(k) + Du(k) \]

Forward difference

\[ \frac{dx(t)}{dt} = \frac{x(k+1) - x(k)}{h} \]

Backward difference

\[ \frac{dx(t)}{dt} = \frac{x(k) - x(k-1)}{h} \]

Example: Discretization

Assume that the following simple controller (filter) has been designed in continuous-time:

\[ U(s) = \frac{1}{s + 2} E(s) \]

Discretize this controller using Forward Euler approximation, i.e. replace \( s \) with \( \frac{s}{h} \):

\[ U(z) = \frac{1}{1 + \frac{s}{h} + 2} E(z) \]

\[ U(z) = \frac{h}{z - 1 + 2h} E(z) \]

\[ u(k+1) - u(k) = hE(z) \]
\[ u(k) = (1 - 2h)u(k-1) + he(k-1) \]

Properties of the Approximation \( H(z) \approx G(z) \)

Where do stable poles of \( G(s) \) get mapped?
Frequency Distortion

Simple approximations such as Tustin introduce frequency distortion.

Important for controllers or filters designed to have certain characteristics at a particular frequency, e.g., a band-pass filter or a notch (band-stop) filter.

Tustin:

\[ H(e^{j\omega h}) \approx G \left( \frac{2 e^{j\omega h} - 1}{h e^{j\omega h} + 1} \right) \]

The argument of \( G \) can be written as

\[ 2 e^{j\omega h} - 1 = \frac{2 e^{j\omega h/2} - e^{-j\omega h/2}}{e^{j\omega h/2} + e^{-j\omega h/2}} = \frac{2i \omega h}{2 - e^{-j\omega h/2}} \]

\[ = \frac{2i \omega h}{2} - e^{-j\omega h/2} \]

\[ = \frac{2i \omega h}{2} + e^{-j\omega h/2} \]

\[ = \frac{2i \omega h}{2} \tan \left( \frac{\omega h}{2} \right) \]

Frequency Distortion, Cont’d

If the continuous-time system affects signals at frequency \( \omega' \), the sampled system will instead affect signals at \( \omega \) where

\[ \omega = \frac{2}{h} \tan \left( \frac{\omega h}{2} \right) \]

i.e.,

\[ \omega = \frac{2}{h} \tan^{-1} \left( \frac{\omega h}{2} \right) \approx \omega' \left( 1 - \frac{(\omega' h)^2}{12} \right) \]

No distortion at \( \omega = 0 \)

Distortion is small if \( \omega h \) is small

Prewarping to Reduce Frequency Distortion

Choose one point \( \omega_1 \). Approximate using

\[ s' = \frac{\omega_1}{\tan(\omega_1 h/2)} \]

\[ \frac{z - 1}{z + 1} \]

This implies that \( H(e^{j\omega_1 h}) = G(i\omega_1) \). Plain Tustin is obtained for \( \omega_1 = 0 \) since tan \( \omega_1 h/2 \approx \omega_1 h/2 \) for small \( \omega \).

Comparison of Approximations (1)

\[ G(s) = \frac{(s + 1)^2(s^2 + 2s + 400)}{(s + 5)^2(s^2 + 2s + 100)(s^2 + 3s + 2500)} \]

\[ h = 0.03 \]

Comparison of Approximations (2)

Sample and Hold-Based Approximations

Sample the controller in the same way as the physical plant model is sampled

- Zero-order hold or Step invariance method
- First-order hold or Ramp invariance method

For a controller, the assumption that the input is piece-wise constant (ZOH) or piece-wise linear (FOH) does not hold!

However, the ramp invariance method normally gives good results with little frequency distortion
Comparison of Approximations (3)

Matlab

\[ \text{SYSD = C2D(SYSC,TS,METHOD)} \]

converts the continuous system \( \text{SYSC} \) to a discrete-time system \( \text{SYSD} \) with sample time \( \text{TS} \). The string \( \text{METHOD} \) selects the discretization method among the following:

- ‘zoh’ Zero-order hold on the inputs.
- ‘foh’ Linear interpolation of inputs (triangle approx.)
- ‘tustin’ Bilinear (Tustin) approximation.
- ‘prewarp’ Tustin approximation with frequency prewarping.

The critical frequency \( \text{Wc} \) is specified last as in \( \text{C2D(SysC,Ts,’prewarp’,Wc)} \)

- ‘matched’ Matched pole-zero method (for SISO systems only).

Design Approaches: Which Way?

Sampled-Control Design:
- When the plant model is already on discrete-time form
  - obtained from system identification
- When the control design assumes a discrete-time model
  - e.g., model-predictive control
- When fast sampling not possible

Discretization of Continuous Design:
- Empirical control design
  - not model-based
  - e.g., PID control
- Nonlinear continuous-time model

In most other cases it is mainly a matter of taste.

An Example: PID Control

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

The Textbook Algorithm

\[ u(t) = K \left( e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right) \]

\[ U(s) = KE(s) + K \frac{1}{sT_i} E(s) + KT_d s E(s) \]

\[ = P + I + D \]
Proportional Term

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

Properties of P-Control

- Stationary error
- Increased \( K \) means faster speed, worse stability, increased noise sensitivity

Error with P-control

Control signal:

\[ u = Ke + u_0 \]

Error:

\[ e = \frac{u - u_0}{K} \]

Error removed if:
1. \( K = \infty \)
2. \( u_0 = u \)

Solution: Automatic way to obtain \( u_0 \)

Integral Term

\[ u = Ke + u_0 \] (P)

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

Properties of PI-Control

- Removes stationary error
- Smaller \( T_i \) implies faster steady-state error removal, worse stability
Prediction

A PI-controller contains no prediction
The same control signal is obtained for both these cases:

$$e(t)$$  
$$e(t + T_d)$$

$$P:$$
$$u(t) = Ke(t)$$

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

$$T_d = \text{Prediction horizon}$$

Derivative Part

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.

Alternative Forms

So far we have described the direct (position) version of the PID controller on parallel form

Other forms:
- Series form
  $$U = K' \left(1 + \frac{1}{sT_i'}\right) (1 + sT_d') E$$
  $$= K' \left(\frac{T_p'}{T_i'} + \frac{1}{sT_i'} + sT_d'\right) E$$

Different parameter values

Practical Modifications

Modifications are needed to make the PID controller practically useful
- Limitations of derivative gain
- Derivative weighting
- Setpoint weighting
- Handle control signal limitations
**Limitation 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 → 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 (step reference changes), \( \gamma = 1 \) in servo control (smooth reference trajectories)

**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.

**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 (\( \Delta u \) is set to 0 if \( u \) saturates)
- 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 recomputation is done dynamically, i.e., through a LP-filter with a time constant $T_i(T_s)$.

**Discretization**

**P-part:**

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

**I-part:**

$$I(t) = \frac{K}{T_i} \int_0^t e(\tau) d\tau$$

$$\frac{dI}{dt} = \frac{K}{T_i} e(t)$$

- Forward difference
  
  $I_{k+1} = \frac{I(k) - I(t_k)}{h} = \frac{K}{T_i} e(t_k)$

- Backward difference
  
  $I(k+1) := I(k) + (K \ast h / T_i) * e(k)$

The I-part can be precalculated in UpdateStates

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

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

$$\frac{T_d}{N} \frac{dB}{dt} + D = -K T_d \frac{dy}{dt}$$

- Forward difference (unstable for small $T_d$ / large $h$)
- Backward difference

$$\frac{T_d}{N} \frac{D(t_k) - D(t_{k-1})}{h} + D(t_k) = -K T_d \frac{y(t_k) - y(t_{k-1})}{h}$$

$$D(t_k) = \frac{T_d}{J_d + N K} D(t_{k-1}) - \frac{K T_d N}{J_d + N K} (y(t_k) - y(t_{k-1}))$$
Discretization

Tracking:

\[ v := P + I + D; \]
\[ u := \text{sat}(v, u_{\text{max}}, u_{\text{min}}); \]
\[ I := I + \left( \frac{K \cdot h}{T_i} \right) e + \left( \frac{h}{T_r} \right) (u - v); \]

Tuning

Parameters: \( K, T_i, T_d, N, \beta, \gamma, T\)

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

Bumpless Transfer

Avoid bumps in control signal when

- changing operating mode (manual - auto - manual)
- changing parameters
- changing between different controllers

Key Issue: Make sure that the controller states have the correct values, i.e., the same values before and after the change

Bumpless Mode Changes

Incremental Form:

\[ \text{MCU} \]

\[ \frac{1}{T} \]

\[ \frac{1}{T_i} \]

\[ \frac{1}{T_d} \]

\[ \frac{1}{T_m} \]

\[ \text{Inc PID} \]

\[ u \]

Bumpless Parameter Changes

A change in a parameter when in stationarity should not result in a bump in the control signal.

For example:

\[ v := P + I + D; \]
\[ I := I + \left( \frac{K \cdot h}{T_i} \right) e; \]

or

\[ v := P + \left( \frac{K}{T_i} \right) I + D; \]
\[ I := I + h e; \]

The latter results in a bump in \( u \) if \( K \) or \( T_i \) are changed.
Bumpless Parameter Changes

More involved situation when setpoint weighting is used. The quantity $P + I$ should be invariant to parameter changes.

$$I_{new} = I_{old} + K_{old} (\beta_{old} y_{sp} - y) - K_{new} (\beta_{new} y_{sp} - y)$$

Switching Controllers

Similar to changing between manual and auto
Let the controllers run in parallel
Let the controller that is not active track the one that is active.
Alternatively, execute only the active controller and initialize the new controller to its correct value when switching (saves CPU).

PID Code

PID-controller with anti-reset windup and manual and auto modes ($y = 0$).

```java
y = yIn.get();
e = yref - y;
v = K*(beta*yref - y) + I + D;
if (mode == auto) {
  u = sat(v, umax, umin);
} else u = sat(uman, umax, umin);
uOut.put(u);
I = I + (K*h/Ti)*e + (h/Tr)*(u - v);
```

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

Class SimplePID

```java
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

```java
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 = getRef();
u = pid.calculateOutput(yref, y);
u = limit(u);
      pid.updateState(u);
    }
  }
}
```