Approximation of Analog Controllers, PID Control

Real-Time Systems, Lecture 8

Karl-Erik Årzén

2 February 2017

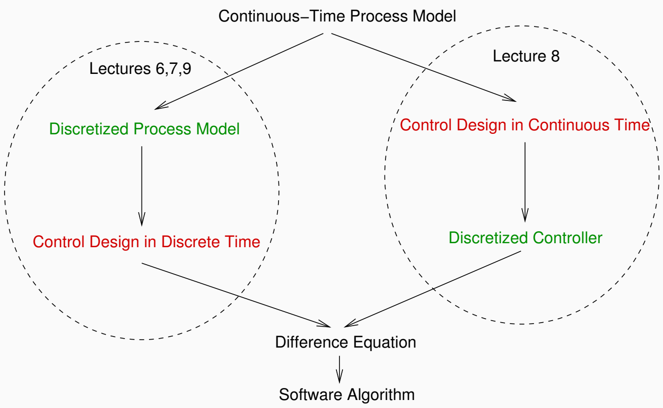
Lund University, Department of Automatic Control

Lecture 8

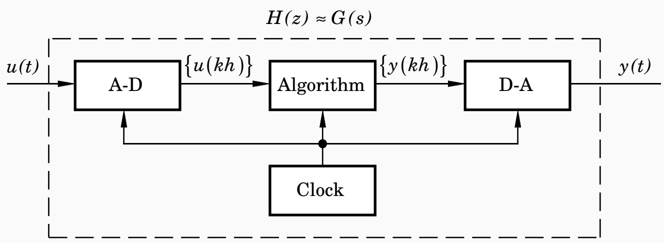
[IFAC PB Chapters 6–7, RTCS Chapter 10]

- Discrete-time approximation of continuous-time controllers
  - Differential equations
  - State-space systems
  - Transfer functions
- The PID controller

Design Approaches for Computer Control



Digital Implementation of an Analog Controller



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

We want to find digital algorithm such that

$$\text{A-D} + \text{Algorithm} + \text{D-A} \approx G(s)$$

NOTE: Involves approximation!

Approximation Methods

- Difference and Tustin approximations
- Step invariance approximation (ZOH)
- Ramp invariance approximation (FOH)
- (Pole-zero matching)

(Tustin and the three last methods are available in Matlab's `c2d` command)

Difference and Tustin Approximations of Differential Equations

Forward difference (explicit Euler, Euler's method):

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

Backward difference (implicit Euler):

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

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

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

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

Difference Approximations of State-Space Systems

Assume that the controller is given in state-space form as

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

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

Forward and backward differences suitable for hand calculations

Difference Approximations of State-Space Systems

Forward difference:

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

leads to

$$\frac{x(k+1) - x(k)}{h} = Ax(k) + Bu(k)$$
$$y(k) = Cx(k) + Du(k)$$

which gives

$$x(k+1) = (I + hA)x(k) + hBu(k)$$
$$y(k) = Cx(k) + Du(k)$$

Difference Approximations of State-Space Systems

Backward difference:

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

first gives

$$x(k) = (I - hA)^{-1}x(k-h) + (I - hA)^{-1}hBu(k)$$
$$y(k) = Cx(k) + Du(k)$$

which after a variable shift $x'(k) = x(k-h)$ gives

$$x'(k+1) = (I - hA)^{-1}x'(k) + (I - hA)^{-1}hBu(k)$$
$$y(k) = C(I - hA)^{-1}x'(k) + (C(I - hA)^{-1}hB + D)u(k)$$

Difference and Tustin Approximations of Transfer Functions

Assume that the controller is given as a transfer function $G(s)$

The discrete-time approximation $H(z)$ is given by

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

where

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

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

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

Example: Discretization

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

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

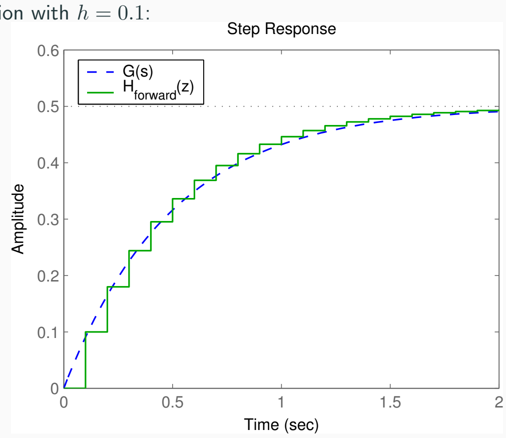
Discretize the controller using forward difference with sampling interval h and write the result as a difference equation.

Solution: Replace s with $\frac{z-1}{h}$:

$$U(z) = \frac{1}{\frac{z-1}{h} + 2}E(z)$$
$$U(z) = \frac{h}{z - 1 + 2h}E(z)$$
$$(z - 1 + 2h)U(z) = hE(z)$$
$$u(k+1) - (1-2h)u(k) = he(k)$$
$$u(k) = (1-2h)u(k-1) + he(k-1)$$

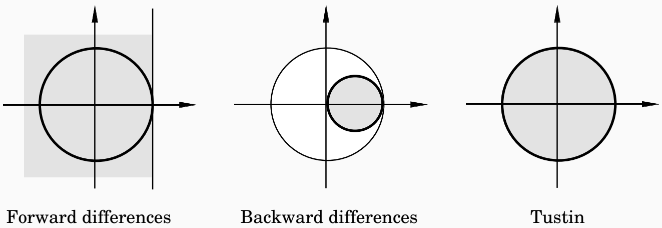
Example: Discretization

Simulation with $h = 0.1$:



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

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:

$$H(e^{i\omega h}) \approx G\left(\frac{2}{h}\frac{e^{i\omega h}-1}{e^{i\omega h}+1}\right)$$

The argument of G can be written as

$$\frac{2}{h}\frac{e^{i\omega h}-1}{e^{i\omega h}+1} = \frac{2}{h}\frac{e^{i\omega h/2}-e^{-i\omega h/2}}{e^{i\omega h/2}+e^{-i\omega h/2}} = \frac{2i}{h}\tan\left(\frac{\omega h}{2}\right)$$

Frequency Distortion, Cont'd

If the continuous-time system affects signals at frequency ω', the sampled system will instead affect signals at ω where

$$\omega' = \frac{2}{h}\tan\left(\frac{\omega h}{2}\right)$$

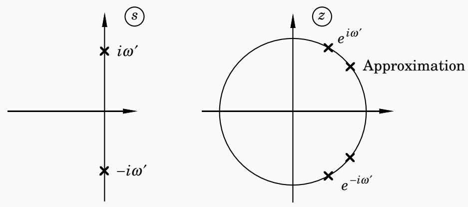
i.e.,

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

No distortion at $\omega = 0$

Distortion is small if ωh is small

Prewarping to Reduce Frequency Distortion



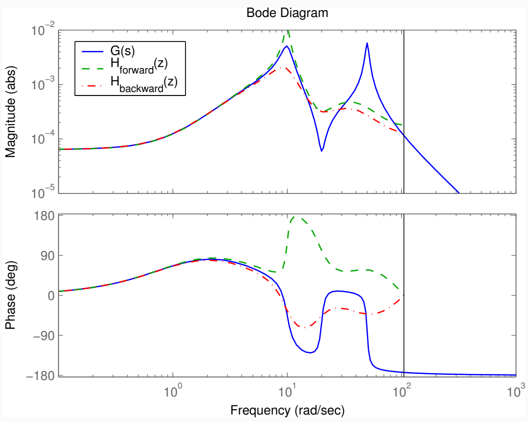
Choose one point ω_1. Approximate using

$$s' = \frac{\omega_1}{\tan(\omega_1 h/2)} \cdot \frac{z-1}{z+1}$$

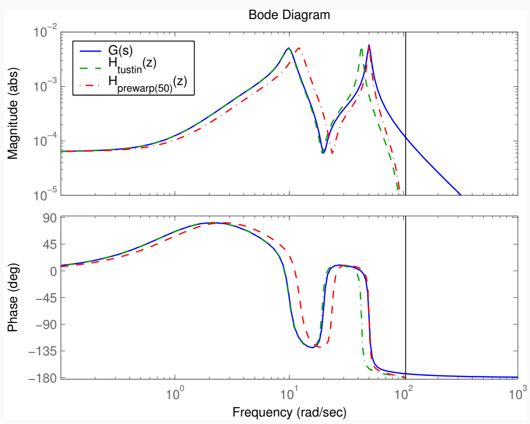
This implies that $H\left(e^{i\omega_1 h}\right) = G(i\omega_1)$. Plain Tustin is obtained for $\omega_1 = 0$ since $\tan\left(\frac{\omega_1 h}{2}\right) \approx \frac{\omega_1 h}{2}$ for small ω.

Comparison of Approximations (1)

$$G(s) = \frac{(s+1)^2(s^2+2s+400)}{(s+5)^2(s^2+2s+100)(s^2+3s+2500)}, \qquad h = 0.03$$



Comparison of Approximations (2)



Step and Ramp Invariance Approximations

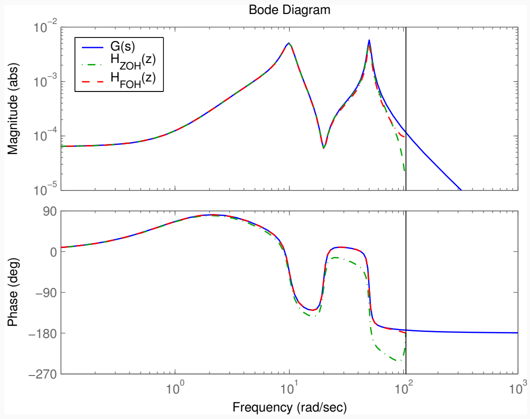
Sample the controller in the same way as the physical plant model is sampled

- Step invariance approximation or Zero-order hold sampling
- Ramp invariance approximation or First-order hold sampling

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

However, the ramp invariance approximation method usually gives very good results with little frequency distortion

Comparison of Approximations (3)



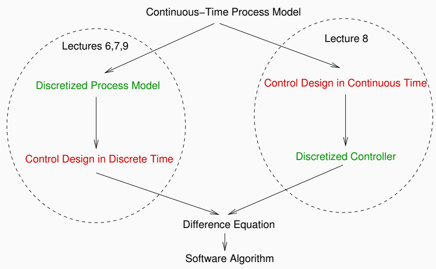
Matlab

```
C2D  Converts continuous-time dynamic system to discrete time.

   SYSD = C2D(SYSC,TS,METHOD) computes a discrete-time model SYSD with
   sample time TS that approximates the continuous-time model SYSC.
   The string METHOD selects the discretization method among the following:
      'zoh'       Zero-order hold on the inputs
      'foh'       Linear interpolation of inputs
      'impulse'   Impulse-invariant discretization
      'tustin'    Bilinear (Tustin) approximation.
      'matched'   Matched pole-zero method (for SISO systems only).
   The default is 'zoh' when METHOD is omitted. The sample time TS should
   be specified in the time units of SYSC (see "TimeUnit" property).

   C2D(SYSC,TS,OPTIONS) gives access to additional discretization options.
   Use C2DOPTIONS to create and configure the option set OPTIONS. For
   example, you can specify a prewarping frequency for the Tustin method by:
      opt = c2dOptions('Method','tustin','PrewarpFrequency',.5);
      sysd = c2d(sysc,.1,opt);
```

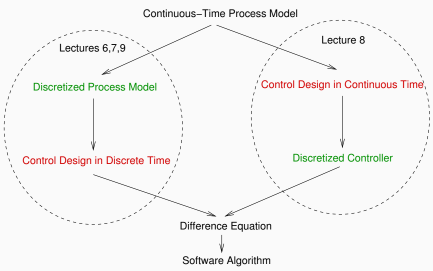

Design Approaches: Which Way?



Sampled control design:

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

Design Approaches: Which Way?



Approximation of analog 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.

PID Control

Based on a survey of over eleven thousand controllers in the refining, chemicals and pulp and paper industries, 97% of regulatory controllers utilize PID feedback.

[Desborough Honeywell, 2000]

- The oldest controller type
- The most widely used
- Much to learn!

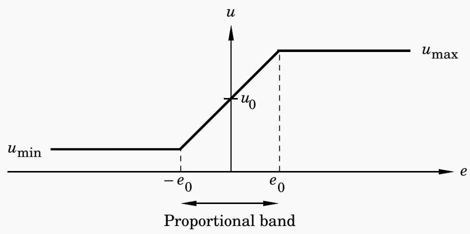
The Textbook Algorithm

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

$$U(s) = KE(s) + \frac{K}{sT_i}E(s) + KT_d sE(s)$$

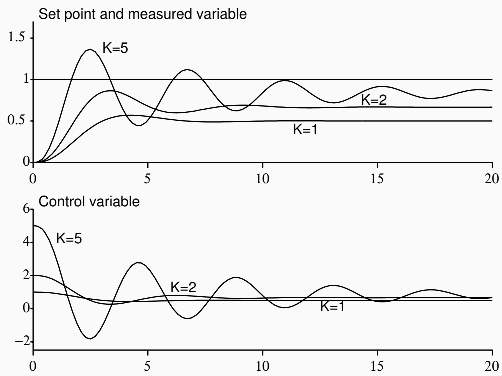
$$= P + I + D$$

Proportional Term



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

Properties of P-Control



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

Stationary Error with P-control

Control signal:

$$u = Ke + u_0$$

Error:

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

Stationary error removed if:

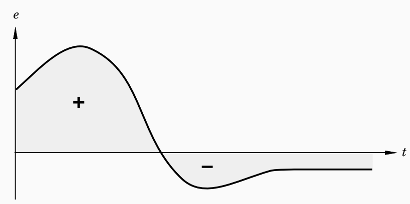
1. $K = \infty$
2. $u_0 = u$

Solution: Automatic way to obtain u_0

Integral Term

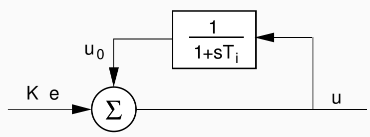
$$u = Ke + u_0 \quad \text{(P)}$$

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



Stationary error present $\rightarrow \int e\,dt$ increases $\rightarrow u$ increases $\rightarrow$ y increases $\rightarrow$ the error is not stationary

Automatic Reset

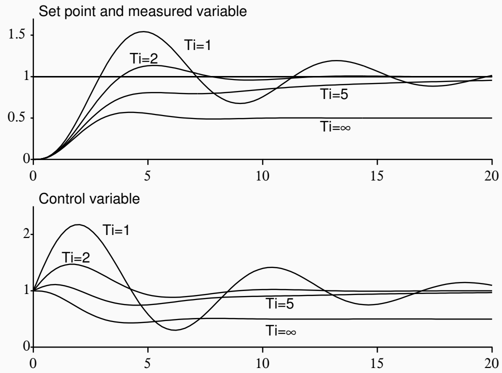


$$U = KE + \frac{1}{1+sT_i}U$$

$$\left(1 - \frac{1}{1+sT_i}\right)U = \frac{sT_i}{1+sT_i}U = KE$$

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

Properties of PI-Control

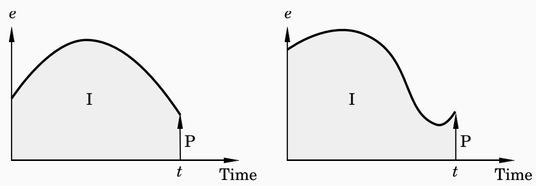


- Removes stationary error
- Smaller T_i implies faster steady-state error removal, worse stability

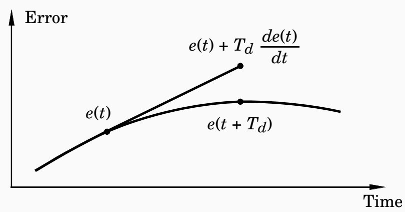
Prediction

A PI-controller contains no prediction

The same control signal is obtained for both these cases:



Derivative Part



P:

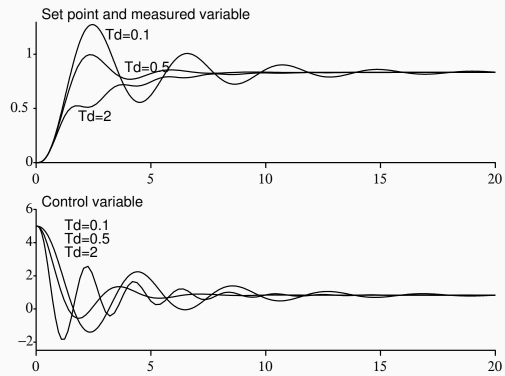
$$u(t) = Ke(t)$$

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.

Alternative Forms

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

Other forms:

- Series form

$$U = K'\left(1 + \frac{1}{sT_i'}\right)\left(1 + sT_d'\right)E$$
$$= K'\left(1 + \frac{T_d'}{T_i'} + \frac{1}{sT_i'} + sT_d'\right)E$$

Different parameter values

Less general than the parallel form

Alternative Forms

- Incremental (velocity) form

$$U = \frac{1}{s}\Delta U$$
$$\Delta U = K\left(s + \frac{1}{T_i} + \frac{s^2 T_d}{1 + sT_d/N}\right)E$$

Integration external to the algorithm (e.g. step motor) or internal

Practical Modifications

Modifications are needed to make the PID controller practically useful

- Limitations of derivative gain
- Derivative weighting
- Setpoint weighting
- Anti-reset windup

Limitation of Derivative Gain

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

$$D(s) = sKT_d \approx \frac{sKT_d}{1 + sT_d/N}$$

$N =$ maximum derivative gain, often $5 - 20$

Another option is to add second-order filter to the measurement signal:

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

$$U(s) = K\left(1 + \frac{1}{sT_i} + sT_d\right)(Y_{sp}(s) - Y_f(s))$$

$T_f =$ filter time constant

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 (step reference changes), $\gamma = 1$ in servo control (smooth reference trajectories)

Setpoint Weighting

Sometimes advantageous to also use weighting on the setpoint.

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

is replaced by

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

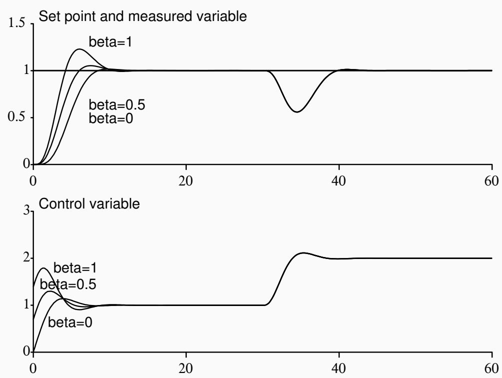
$\beta \geq 0$

Assumes that the I-part is also turned on! (otherwise the controller will try to follow the wrong reference value)

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

Improved set-point responses

Setpoint Weighting

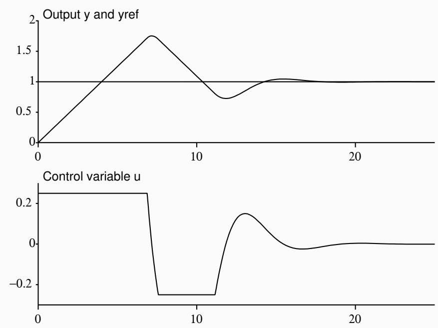


Control Signal Limitations

All actuators saturate. Problems for controllers with integration.

When the control signal saturates the integral part will continue to grow – integrator (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 in velocity form (Δu is set to 0 if u saturates)
- limited the setpoint variations (saturation never reached)
- conditional integration (integration is switched off when the control is far from the steady-state)
- tracking (back-calculation)

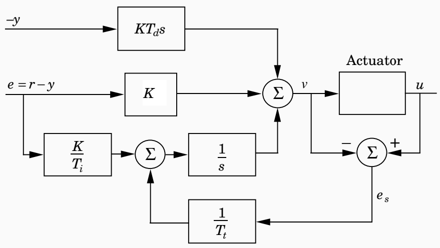
Tracking

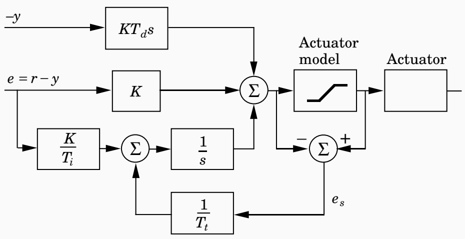
General technique that can also be used to handle controller mode switches, etc.

The controller output v tracks the actual output u

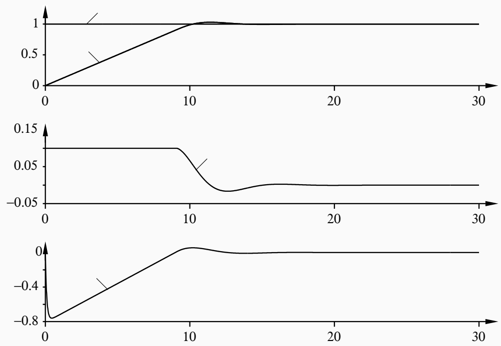
- When the control signal saturates, the integral is recomputed so that its new value gives a control signal at the saturation limit
- To avoid resetting the integral due to, e.g., measurement noise, the recomputation is done dynamically, i.e., through a LP-filter with a time constant T_t (T_r).

Tracking





Tracking



Discretization

Discretize the P, I, and D parts separately

Maintains the interpretation

P-part:

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

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))

Discretization

(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/large h)
- 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);
```


Bumpless Transfer

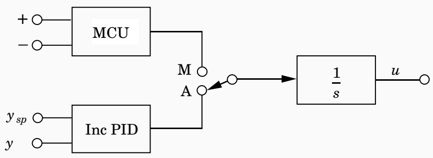
Avoid bumps in control signal when

- 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

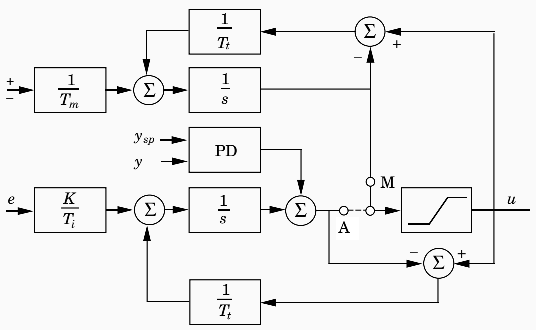
Bumpless Mode Changes

Incremental Form:



Bumpless Mode Changes

Direct Position form:



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 +(K*h/Ti)*e;
```

vs

```
v := P + (K/Ti)*I + D;
I := I + h*e;
```

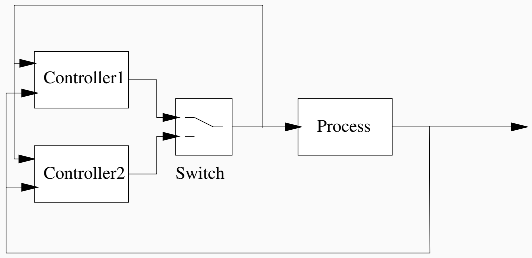
The latter results in a bump in u if K or T_i are changed.

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 Between 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 ($\gamma = 0$):

```
y = yIn.get();
e = yref - y;
D = ad * D - bd * (y - yold);
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);
if (increment)
  uinc = 1;
else if (decrement)
  uinc = -1;
else uinc = 0;
uman = uman + (h/Tm) * uinc + (h/Tr) * (u - uman);
yold = y;
```

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

[Java] 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;
  }
```


[Java] Class SimplePID

```
  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;
  }

}
```


[Java] 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
    }
  }
}
```