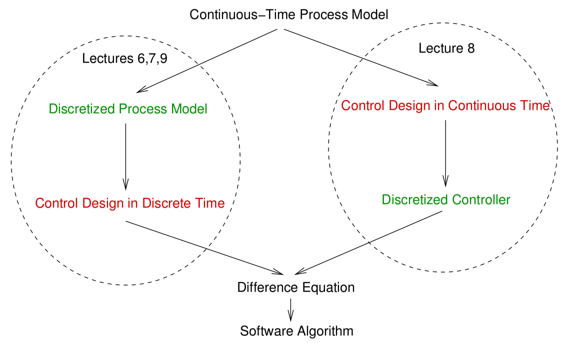
Lecture 8: From Analog to Digital Controllers, PID Control

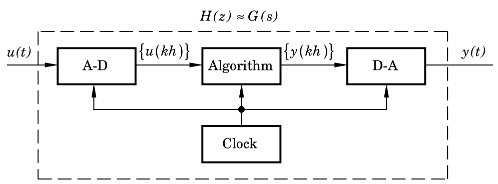
[IFAC PB Ch 6, Ch 8, RTCS Ch 10]

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

Design Approaches



Implementing a Continuous-Time Controller Using a Computer



Want to find discrete-time Algorithm such that

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

Methods:

- Differentiation and Tustin approximations
  - State-space domain
  - Frequency domain
- Step invariance (ZOH)
- Ramp invariance (FOH)
- Pole-zero matching

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

Differentiation and Tustin Approximations

Forward difference (Euler's forward method):

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

Backward difference (Euler backward):

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

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

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

State-Space Domain

Assume that the controller is given in state-space form

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

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

Forward or backward approximation of the derivative

Forward difference

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

leads to

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

which gives

$$\begin{aligned} x(k+1) &= (I + hA)x(k) + hBu(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned}$$

Backward difference

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

first gives

$$\begin{aligned} x(k) &= (I - hA)^{-1}x(k-h) + (I - hA)^{-1}hBu(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned}$$

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

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

Frequency Domain

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

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

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

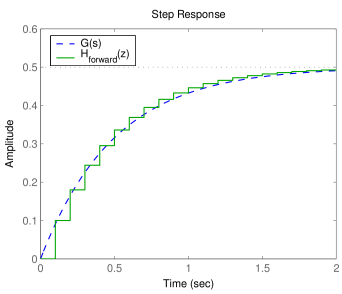
Discretize this controller using Forward Euler approximation, i.e. replace s with $\frac{z-1}{h}$:

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

Alternative: Write as differential equation first:

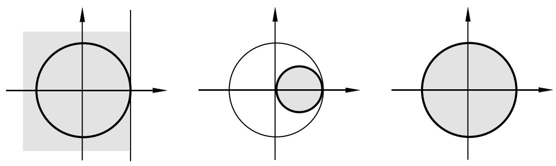
$$\begin{aligned} \frac{du}{dt} + 2u(t) &= e(t) \\ \frac{u(k+1) - u(k)}{h} + 2u(k) &= e(k) \\ u(k+1) - u(k) + 2hu(k) &= he(k) \\ u(k) &= (1-2h)u(k-1) + he(k-1) \end{aligned}$$

Simulation ($h = 0.1$):



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

Where do stable poles of $G(s)$ get mapped?



Forward differences   Backward differences   Tustin

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

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

Extra Slide: Basic Math

$$e^a e^b = e^{a+b}$$
$$e^0 = 1$$
$$\tan a = \frac{\sin a}{\cos a}$$
$$\cos a = \frac{1}{2}(e^{ia}+e^{-ia})$$
$$\sin a = \frac{1}{2i}(e^{ia}-e^{-ia})$$
$$\tan^{-1} x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots$$

Frequency Distortion, Cont'd

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

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

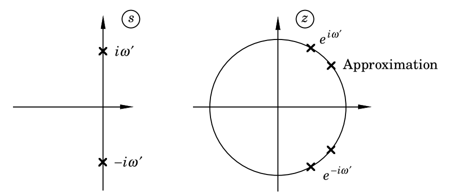
i.e.,

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

No distortion at $\omega = 0$

Distortion is small if ωh is small

Prewarping to Reduce Frequency Distortion



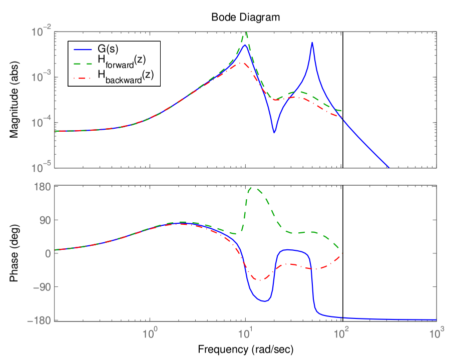
Choose one point ω_1. Approximate using

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

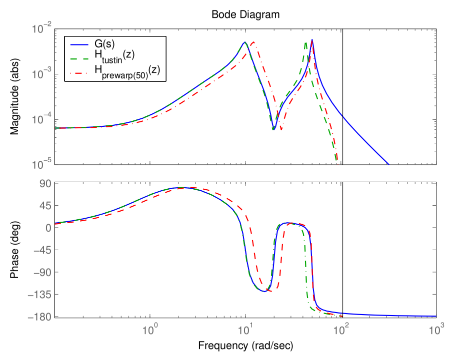
This implies that $H\left(e^{i\omega_1 h}\right) = G(i\omega_1)$. Plain Tustin is obtained for $\omega_1 = 0$ since $\tan\left(\frac{\omega_1 h}{2}\right) \approx \frac{\omega_1 h}{2}$ for small ω.

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)



Sample and Hold-Based Approximations

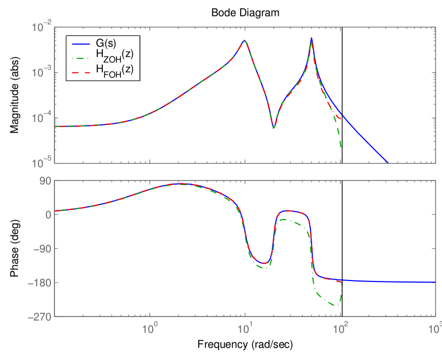
Sample the controller in the same way as the physical plant model is sampled

- 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

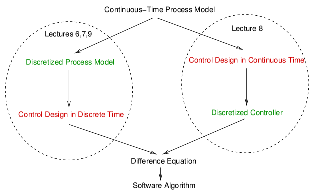
Comparison of Approximations (3)



Matlab

```
SYSD = C2D(SYSC,TS,METHOD)  converts the continuous
system SYSC to a discrete-time system SYSD with
sample time TS.  The string METHOD selects the
discretization method among the following:
 'zoh'       Zero-order hold on the inputs.
 'foh'       Linear interpolation of inputs
             (triangle appx.)
 'tustin'    Bilinear (Tustin) approximation.
 'prewarp'   Tustin approximation with frequency
             prewarping.
             The critical frequency Wc is specified
             last as in C2D(SysC,Ts,'prewarp',Wc)
 'matched'   Matched pole-zero method
             (for SISO systems only).
```

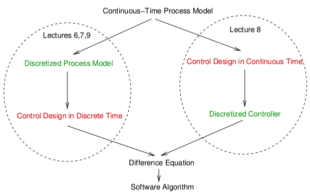

Design Approaches: Which Way?



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

Design Approaches: Which Way?



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!

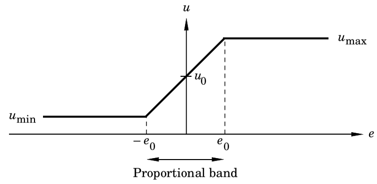
The Textbook Algorithm

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

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

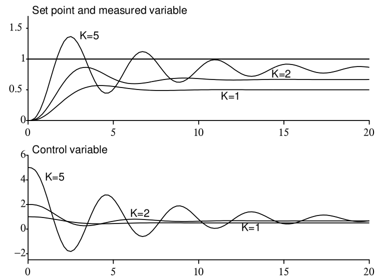
$$= P + I + D$$

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, 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:

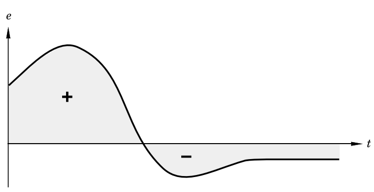
1. $K = \infty$
2. $u_0 = u$

Solution: Automatic way to obtain u_0

Integral Term

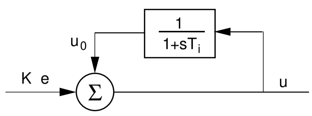
$$u = Ke + u_0 \qquad \text{(P)}$$

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



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

Automatic Reset

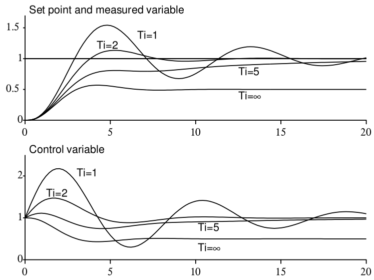


$$U = KE + \frac{1}{1+sT_i}U$$

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

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

Properties of PI-Control

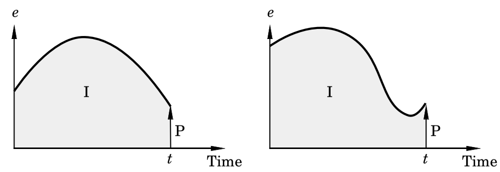


- Removes stationary error
- Smaller T_i implies faster steady-state error removal, worse stability

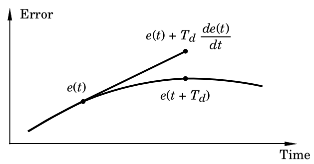
Prediction

A PI-controller contains no prediction

The same control signal is obtained for both these cases:



Derivative Part



P:

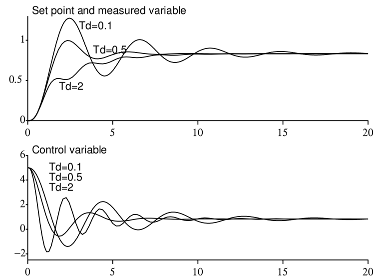
$$u(t) = K e(t)$$

PD:

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

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 on parallel form

Other forms:

- Series form

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

Different parameter values

- Incremental (velocity) form

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

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
- 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

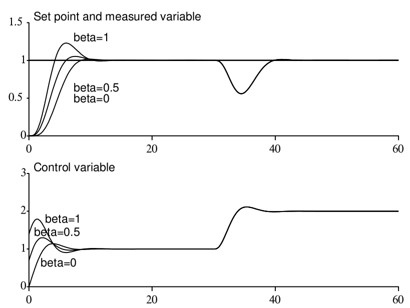
$$u = K(\beta y_{sp} - y)$$

$0 \leq \beta \leq 1$

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

Improved set-point responses.

Setpoint Weighting

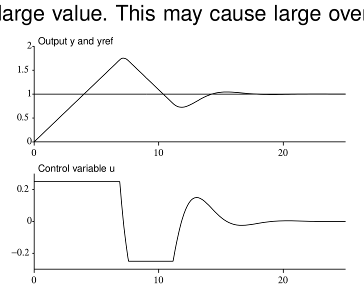


Control Signal Limitations

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

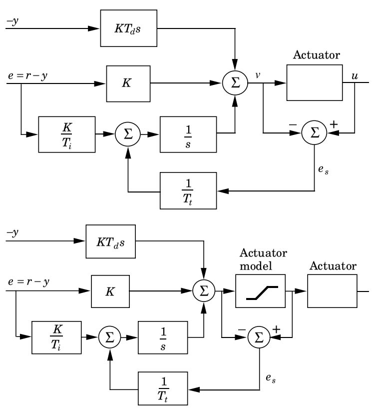
Several solutions exist:

- controllers on velocity form (Δu 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)

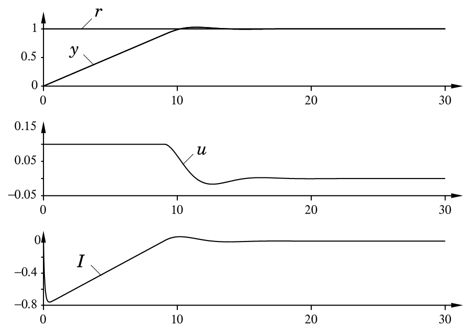
Tracking

- 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 $T_t(T_r)$.

Tracking



Tracking



New Slide: Discretization

Two approaches:

- Discretize the entire PID controller at the same time using some approximation method. Assuming that $\beta = \gamma = 0$

$$\begin{aligned}\mathrm{PID}(s) &= K\left(1 + \frac{1}{T_I s} + \frac{T_D s}{1 + sT_D/N}\right) \\ &= \frac{K(T_I T_D(1 + 1/N)s^2 + (T_I + T_D/N)s + 1)}{T_I s(1 + sT_D/N)}\end{aligned}$$

  - Only two states
  - Lose the interpretation of the individual parts
- Discrete the P, I and D parts separately
  - Requires one more state
  - Maintains the interpretation
  - The approach used here

Discretization

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

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/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

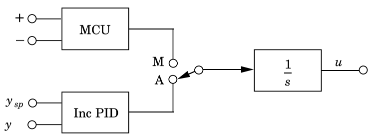
Avoid bumps in control signal when

- 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

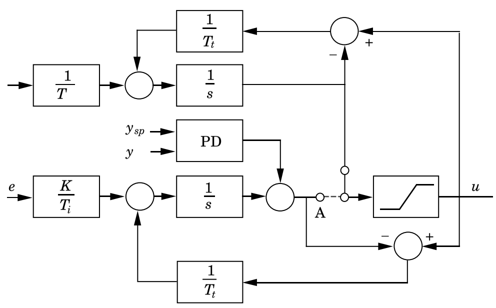
Bumpless Mode Changes

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;
```

or

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

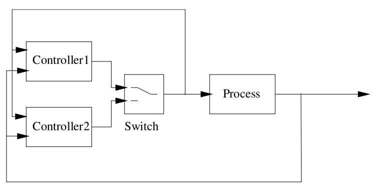
The latter results in a bump in u if K or T_i are changed.

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 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.

Class SimplePID

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

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

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



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

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

  public void updateState(double u) {

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

}
```

Extract from Regul

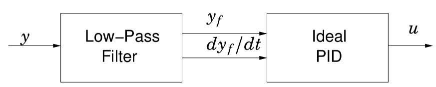
```
public class Regul extends Thread {
  private SimplePID pid;

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

  public void run() {
    // Other stuff

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


Alternative PID Realization

The PID controller presented so far does not suppress high-frequency noise very well (constant gain for high frequencies)

Alternative:

- use a second-order low-pass on the measurement signal
- use the filtered measurement signal, y_f, as an input to a PID with an ideal derivative (without low-pass filter)
- implement the low-pass filter so that dy_f/dt is directly obtainable from the filter



Alternative PID: Low-pass filer

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

- Relative damping: $\zeta = 1/\sqrt{2}$
- Filter constant $T_f = T_I/N$ (PI) or $T_f = T_D/N$ (PID), where N ranges from 2 to 20.
- State-space representation: $x_1(t) = y_f(t)$ and $x_2(t) = dy_f(t)/dt$

$$\begin{aligned}\frac{dx_1(t)}{dt} &= x_2(t)\\ \frac{dx_2(t)}{dt} &= -\frac{1.4}{T_f}x_2(t) - \frac{1}{T_f^2}x_1(t) + \frac{1}{T_f^2}y(t)\end{aligned}$$

Alternative PID: Low-pass filer

Discretize using backward Euler gives

$$\begin{aligned}x_1[k] &= (1 - \frac{h^2}{den})x_1[k-1] + \frac{hT_f^2}{den}x_2[k-1] + \frac{h^2}{den}y[k]\\ x_2[k] &= \frac{1}{den}(T_f^2 x_2[k-1] - hx_1[k-1] + hy[k])\\ den &= (T_f^2 + 1.4hT_f + h^2)\end{aligned}$$

Alternative PID: Ideal PID

Since $dy_f(t)/dt = x_2(t)$ the discretization and the pseudo-code for the ideal PID becomes very simple, The total PID code (without anti-windup) including the filter is shown below:

```
x1 = p1*x1old + p2*x2old + p3*y;
x2 = p4*x2old + p5*(y - x1old);
v = K*(Beta*yref - x1) + I - K*Td*x2;
u = sat(v);
output u
I = I + (K*h/Ti)*(yref - x1);
x1old = x1; x2old = x2;
```

with the precalculated parameters

```
den = Tf*Tf + 1.4*h*Tf + h*h;
p1 = 1 - h*h/den;
p2 = h*Tf*Tf/den;
p3 = h*h/den;
p4 = Tf*Tf/den; // equals p2/h
p5 = h/den; // equals p3/h
```