# MANUAL

# for

# COURSE ON I/A SERIES

# CONTROLLERS

Originator:  E.R Draga
Author:      H.L.C.M. Stapper

Revision:   6.0
Date:        23-01-2204

**The Foxboro Controller Family**

**1- Function of this document**

This document is intended as a support and a guideline for a course about the wide variety of controller functions that are offered by Foxboro
In the official Foxboro documentation for control and I/O blocks, the (Laplace) formula for the different MODOPT (mode options) of controllers is given (for PIDA block. see figure 1 – 1). Through simple mathematical manipulation these formula's have been re-arranged, so that it was possible to represent them in a block diagram that can be understood by the engineer (a minimum basic knowledge on control engineering and the Laplace representations of process dynamics is expected).
The resulting block diagrams are tested and results are shown in trends.
Based on the above found theoretical knowledge, it can be better understood how the tuning parameters must be set to get optimal control.

**2- Introduction**

In process control, there is always a combination of process dynamics and controller dynamics (see figure 2-1). The dynamics of the process or the controller are described with differential equations. For an example of a process dynamic, using differential calculations, see figures 2-2 and 2-3.
Making calculations with differential equations is difficult and for most engineers problematic.
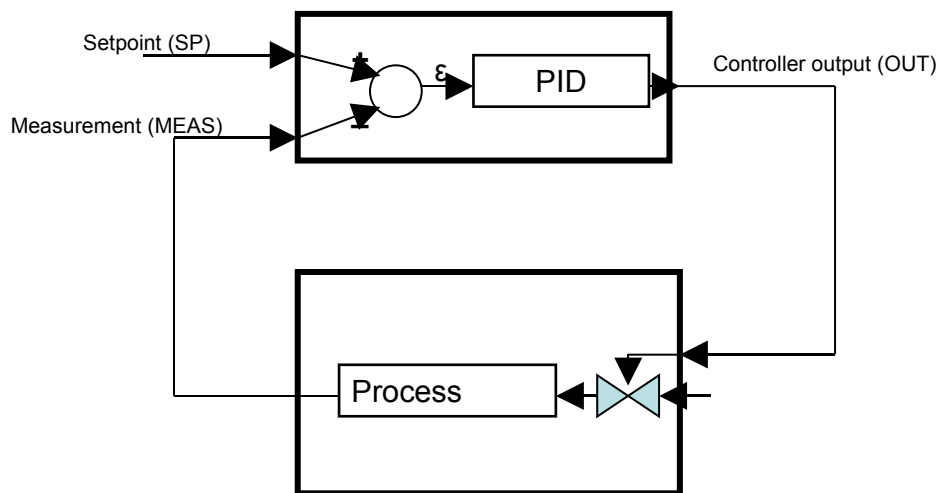
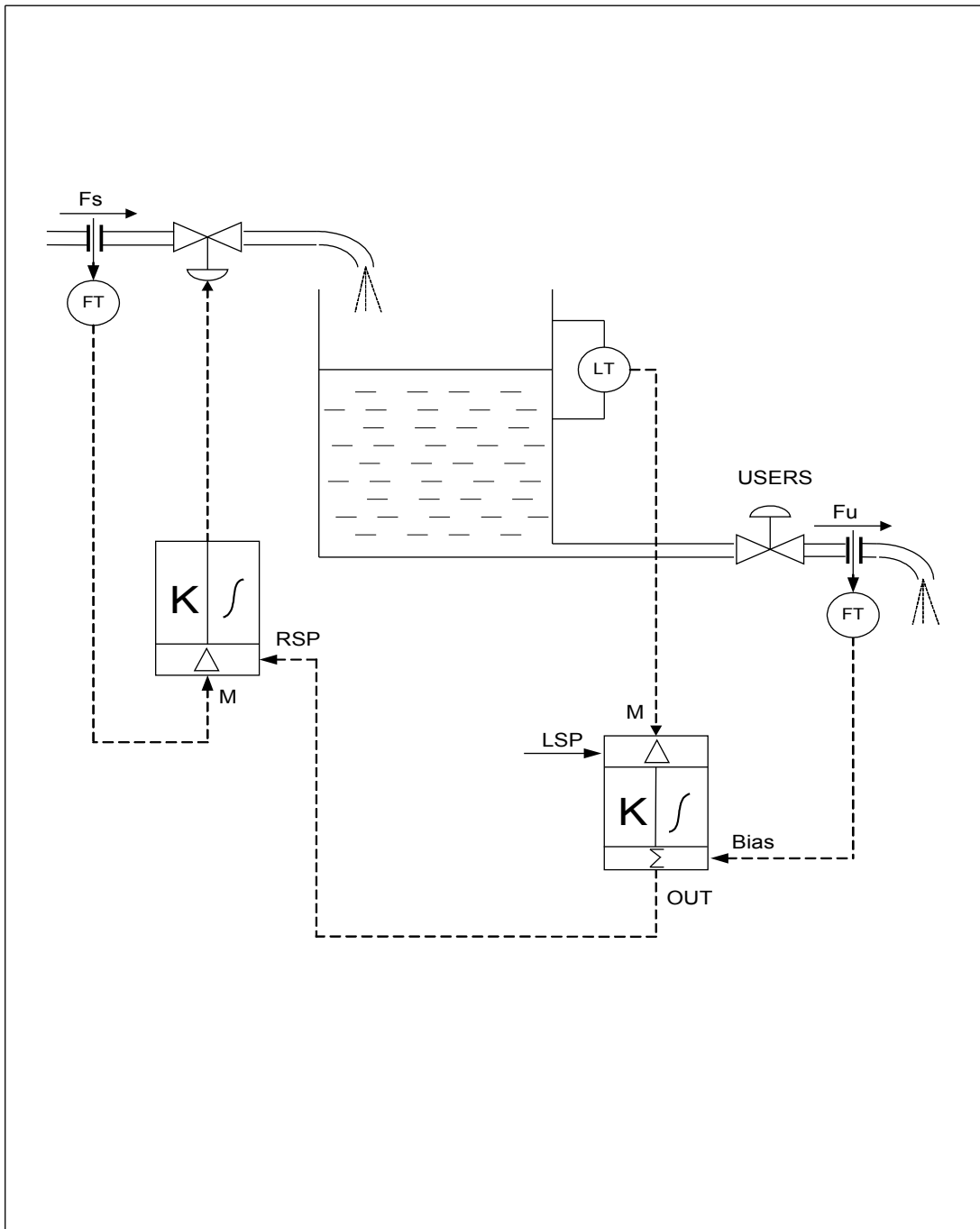Figure 2.1: Block structure of dynamic process and controller



**Figure 2- 2: Example of process with level control (cascade)**

Process Dynamic for vessel level

$$\frac{dV}{dt} = (F_s - F_u) \qquad (1)$$

$$\frac{dV}{dt} = \frac{d(A*L)}{dt} = A*\frac{dL}{dt} \qquad (2)$$

From (1) and (2)

$$A*\frac{dL}{dt} = (F_s - F_u) = \Delta F \quad \ggg$$

$$L = \frac{1}{A} * \int \Delta F(t)*dt \quad \gg \qquad$$ **This process behaves as an integrator**

*Figure 2- 3: Example of process dynamics*

invensys

Process Systems

ISO 9001 en VCA**
DNV
GECERTIFICEERD
BEDRIJF

Fortunately, it was LAPLACE who has given us the possibility to transform the differential equations into simple algebraic equations which can be treated in the standard mathematical manner.

In order to follow this course, the engineer should understand the meaning of the LAPLACE transformed differential equations. These simplified examples are (see also figures 2-4 and 2-5):

| Dynamic | Differential equation | LAPLACE representation |
|---|---|---|
| Integrator | $y = \dfrac{1}{T} * \int x * dt$ | $y = \dfrac{1}{Ts} * x$ |
| Differentiator | $y = \dfrac{dx}{dt}$ | $y = sx$ |
| First order process | $\dfrac{T * dy}{dt} + y = K * x$ | $y = \dfrac{K}{1 + Ts} * x$ |
| Second order process | $\tau_1 \dfrac{d^2 y}{dt^2} + \tau_2 \dfrac{dy}{dt} + y = K * x$ | $y = \dfrac{K}{1 + \tau_2 s + \tau_1 s^2} * x$ |
| Time delay | $y = x(t - \tau)$ | $y = x * e^{-s\tau}$ |

During this course, all calculations will be made, using the LAPLACE transformed equations. This is done because it makes it easy to do mathematical calculations and to understand the results.

As examples, some schemes are give in the figures 2-6 and 2-7, where the mathematical calculations with LAPLACE functions (H1(s) and H2(s)) are given.

$$\left[\frac{d}{dt}\right] \rightarrow \ s \rightarrow$$

Output

time

$$\int dt \rightarrow \ \frac{1}{s} \rightarrow$$

Output

time
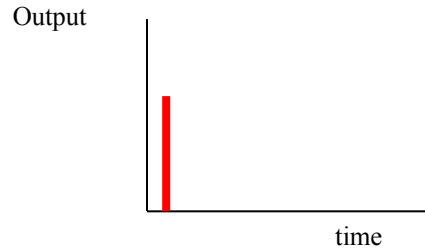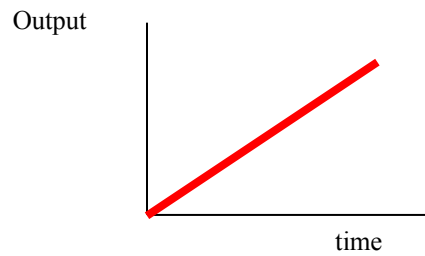
**First order process**

$$\frac{1}{1+ T * s}$$
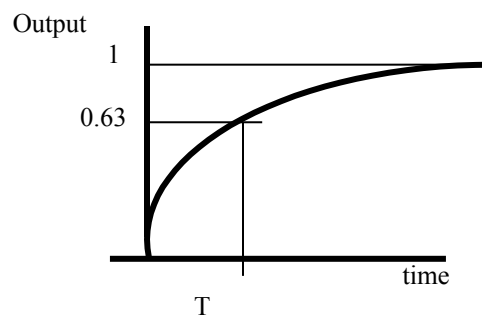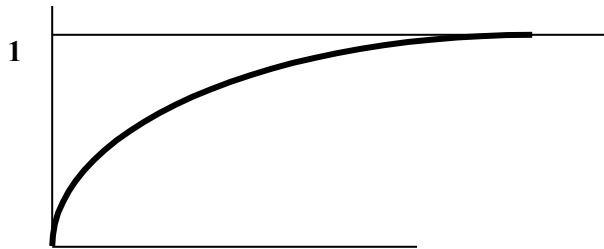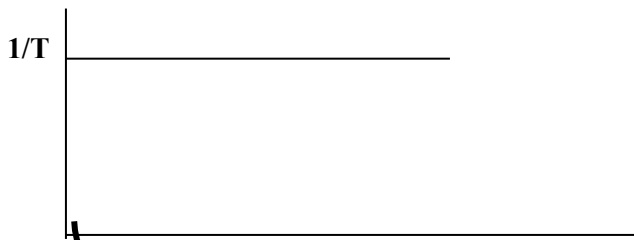
Output

1

0.63

time

T

**Figure 2- 4: differentiator,  integrator  and first order process**

## LeadLag Action



**Lag action**

$$y(t) = (1 - e^{-\frac{t}{T}}) * x \rightarrow T\frac{dy}{dt} + y = x \rightarrow \frac{y}{x} = \frac{1}{1 + sT}$$



**Lead action**

$$y(t) = \frac{1}{T} * e^{-\frac{t}{T}} \rightarrow T\frac{dy}{dt} + y = \frac{dx}{dt} \rightarrow \frac{y}{x} = \frac{s}{1 + Ts}$$
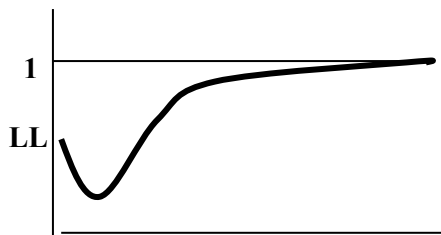


**LeadLag action**

$$\frac{y}{x} = \frac{1 + (LL * T)s}{1 + sT} = \frac{1}{1 + Ts} + \frac{(LL * T)s}{1 + sT}$$

**Figure 2- 5: Laplace transforms of Lag/Lead/and LLAG function**

**Parallel Configuration**
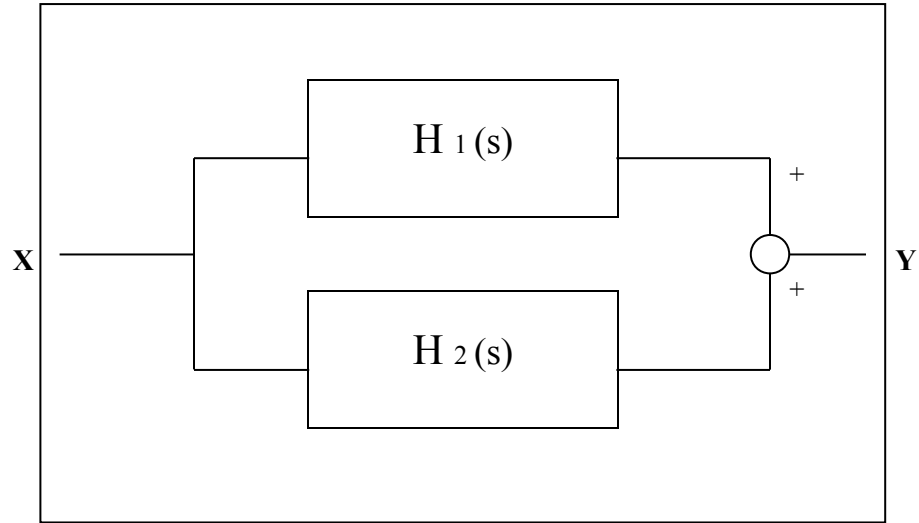


$$H(s) = H_1(s) + H_2(s)$$

**Figure 2- 6: Function of parallel processes**

## Feedback function



$$Y = \varepsilon * H_1(s)$$
$$Z = Y * H_2(s)$$

$$\varepsilon = X - Z$$

$$\frac{Y}{H_1(s)} = X - Y * H_2(s)$$

$$Y * \{\frac{1}{H_1(s)} + H_2(s)\} = X$$

$$\frac{Y}{X} = \frac{H_1(s)}{1 + H_1(s) * H_2(s)}$$

**Figure 2- 7: Feedback scheme**

**3- The process GAIN (Ko)**

In control blocks of the I/A system, the signals are scaled between 0 and 100%. The process gain (Ko) is important when tuning a controller. Below follows the procedure for determining the process gain:
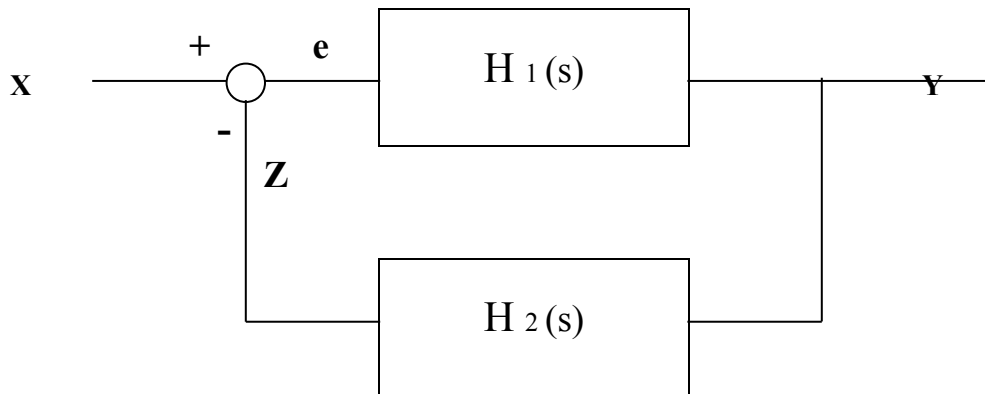- the controller must be configured with MEASUREMENT low and high scale; the difference between high and low scale is called the **range of the controller**.
- When changing the controller output ( position of control valve, always between 0 and 100%) with x % the actual measurement of the process should change with x % of the controller range in order to have a process gain, Ko = 1.
- When actual measurement changes with y % of controller range on a change of x % of the control valve (= controller output) the process gain is y/x.

Example:
Controller measurement low scale is 1000 ltr/hr and the high scale is 4000 ltr/hr; the controller measurement range is 3000 ltr/hr.
The process gain is = 1 when the actual measurement changes with 300 ltr/hr on a change of 10 % of controller output (10 % change in valve position). If controller output changes 10 % and the actual measurement changes with 600 ltr/hr (being 20 % of controller range)
the process gain (Ko) is 20/10=2.

**It is important to know the process GAIN for initial tuning of the controller: one should keep in mind that process GAIN multiplied by controller GAIN (100/PB) should be approximately equal to 1.**

In many actual processes the process GAIN is not one and the same value for different process conditions. The process itself and in many cases the control valve is not linear over the whole range of the process range. For instance the figure 3.1 shows a non linear behavior of a control (flow) valve. For such flow control process, it can be seen that process gain for low flows is high, bigger than the process gain for higher flows.
As the process GAIN for this flow process is not a fixed value, it will become difficult to find the optimum controller tuning parameters, as these are different for low or for high flows. One option for such control problem is to find optimum controller tuning parameters for the situation where the process GAIN is high and to accept a slow control behavior for the situation where the process GAIN is low. If it is done the other way around, the control will become unstable for process conditions with high GAIN.

A second option is to change the controller parameters for different process conditions: one could do this in two, three or more steps (depending on how accurate control is required).

One more remark for the above control valve behavior: it can be seen that for valve positions > 70% the effect on flow is very small: there is a possibility that the controller output could integrate to values >> 70% (for instance to 100%). In this situation this could be seen as a form of integral wind-up: it would take a long time to bring the controller output back from 100% to 70% and during all this time there would be little or no effect on the process, as the flow does not change much between 100% and 70 %. Therefore it is advised to give an output limit to the controller to prevent this form of integral wind-up.

In this course it is assumed that the process gain (Ko) is equal to 1, unless it is specifically stated differently. This is important to keep in mind when tuning the controller, in particular the tuning of the Proportional Band.

**1000 ltr/hr**

**Flow**

**0 ltr/hr**

|  |  |  |
| --- | --- | --- |
| 0% | 50% | 100% |

**Klepstand**
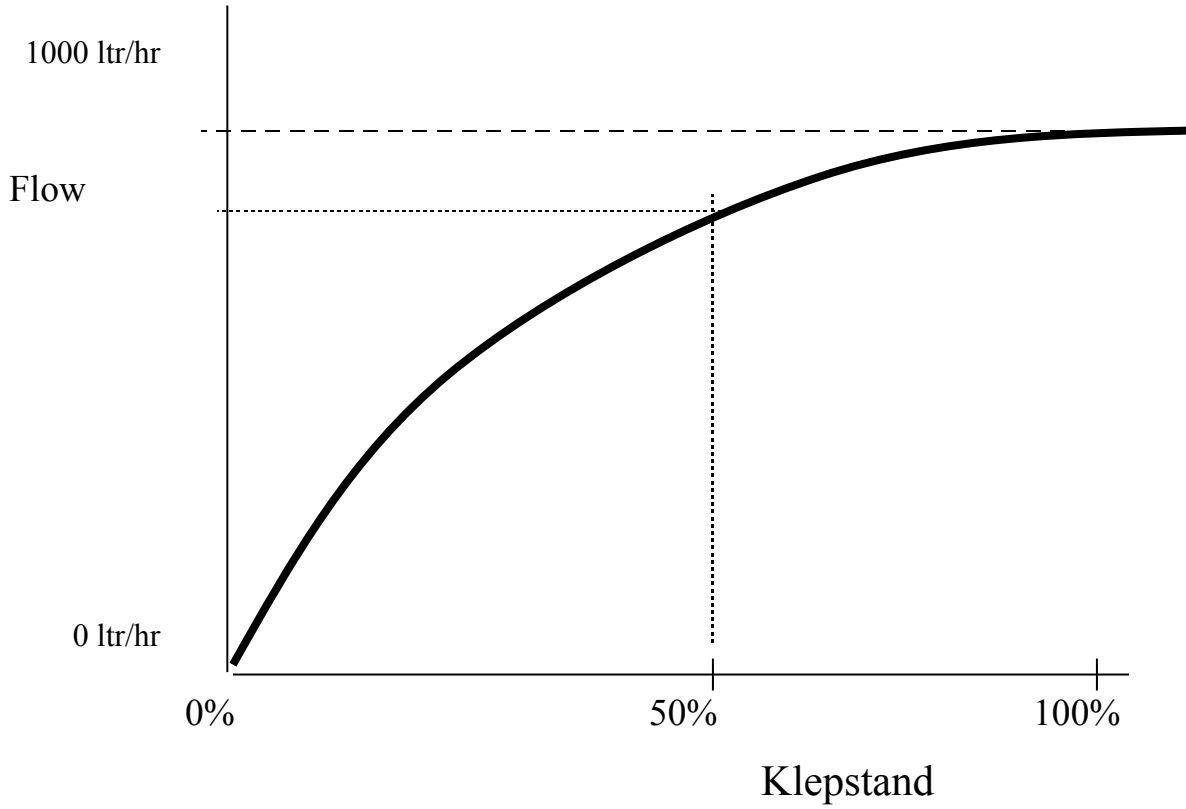
**Figuur 3.1: Non lineair behavior of a flow valve**

### 4- The Foxboro PI algorithm.

Applicable for all I/A series controllers (PID/PIDA/PIDE etc.)

The Foxboro PI algorithm uses the so called "positive feedback" configuration; in figure 4 – 1A and B  this positive feedback mechanism is shown. In this open loop control scheme it can be seen that each control cycle the output signal is increased with the same amount as the proportional action.
The following will happen: when the setpoint for an open loop controller is changed with a step = 1, than in the next block processing cycle of the controller the error will be 1 and the controller output will change also with 1 (provided the PB=100). In the next cycle of the block, the error is still 1 and the value of output will become  z (=1) + output (=1)   = 2.  In the next cycle output will be equal to z(=1) + output(=2) = 3,  etc. etc.

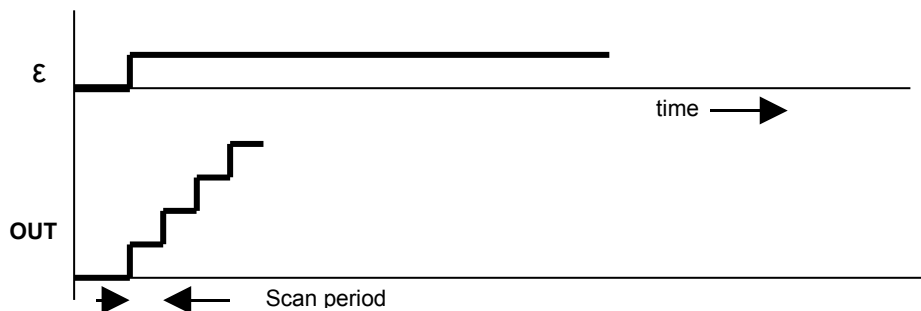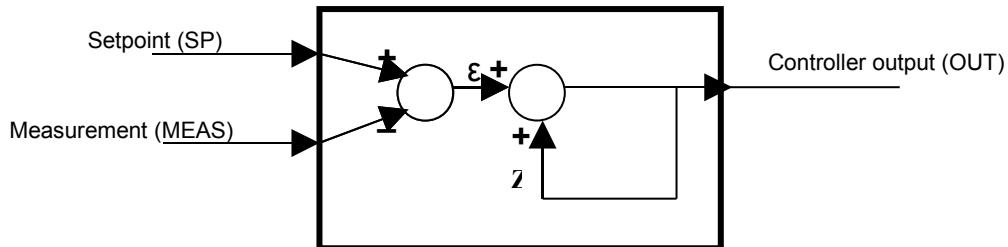|          | $0^{th}$ cycle | $1^{st}$ cycle | $2^{nd}$ cycle | $3^{rd}$ cycle | $4^{th}$ cycle | Etc. |
|----------|------|------|------|------|------|------|
| setpoint | 0    | 1    | 1    | 1    | 1    |      |
| error    | 0    | 1    | 1    | 1    | 1    |      |
| z        | 0    | 0    | 1    | 2    | 3    |      |
| output   | 0    | 1    | 2    | 3    | 4    |      |





**Fig. 4 – 1A: Open loop control by PI controller without and with filter in feedback**
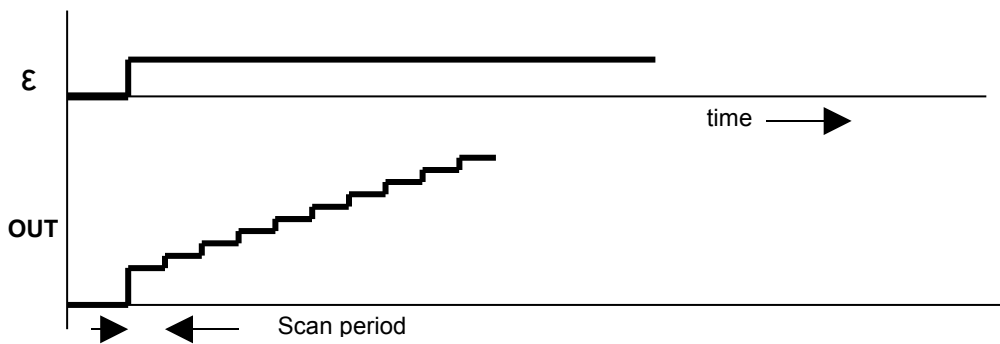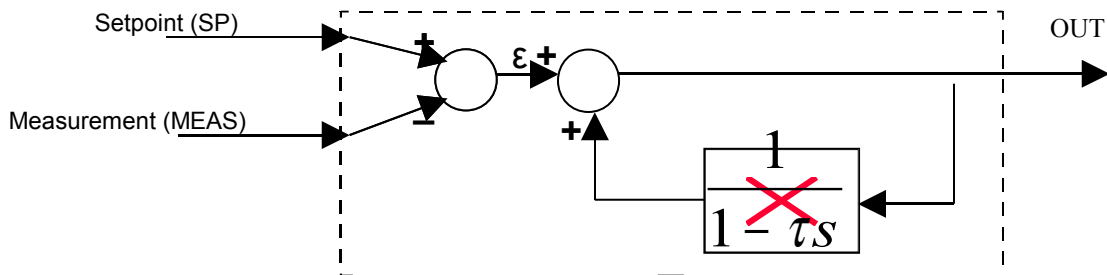
**Fig. 4 – 1B: Open loop control by PI controller without and with filter in feedback**

This would be too much integral action for most control situations. In order to reduce the amount of integral action, a first order filter has been introduced in the positive feedback loop (see figure 4 – 1B and 4 - 2). The formulae for Hf is:

$$y = \{1 + \frac{1}{T_r s}\} * x$$

In this formulae the "1" stands for the proportional action and $\frac{1}{T_r s}$ is the integral action.

If feedback is not connected, the controller has only P-action.
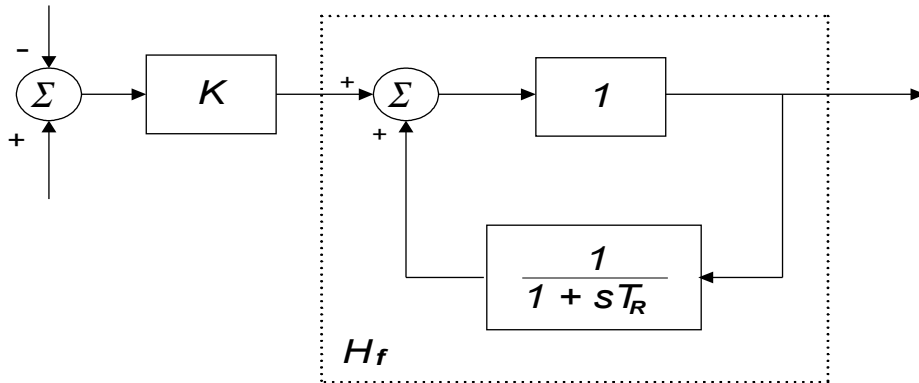
$$H_f = \cfrac{1}{1 - \cfrac{1}{1 + sT_R}} = 1 + \frac{1}{sT_R}$$
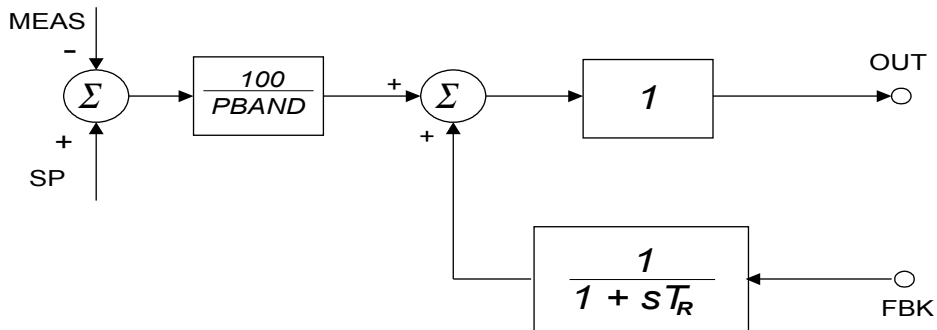
**Figure 4- 2:    BLOCK SCHEME FOR PI CONTROLLER
(with (standard) filter in positive feedback)**

In this figure it can bee seen that the feedback parameter must be linked to the "output signal" in order to make the controller a PI controller. (if link is not made, the controller behaves like a P only controller.
The Foxboro controllers provide the possibility to link the feedback parameter to other signals than the controller output itself; this provides fundamental improvements of the controller function.

invensys
Process Systems

ISO 9001 en VCA**
DNV
GECERTIFICEERD
BEDRIJF

The first possible improvement is the connection of the process measurement from the slave controller to the feedback input parameter of the master controller (see figure 4-3). The great advantage of this solution is that the master controller does not have a problem with "reset windup"; in case the slave controller is put in local or for any other reasons the slave process can not reach the requested remote setpoint (for instance, the temperature of the heating medium is too low to reach the requested temperature). When this happens with a standard PI controller the integral action of the Master controller would continue to increase the controller output (being the remote setpoint of the slave).

When the measurement of the slave process is used as feedback, this wind-up will not happen.
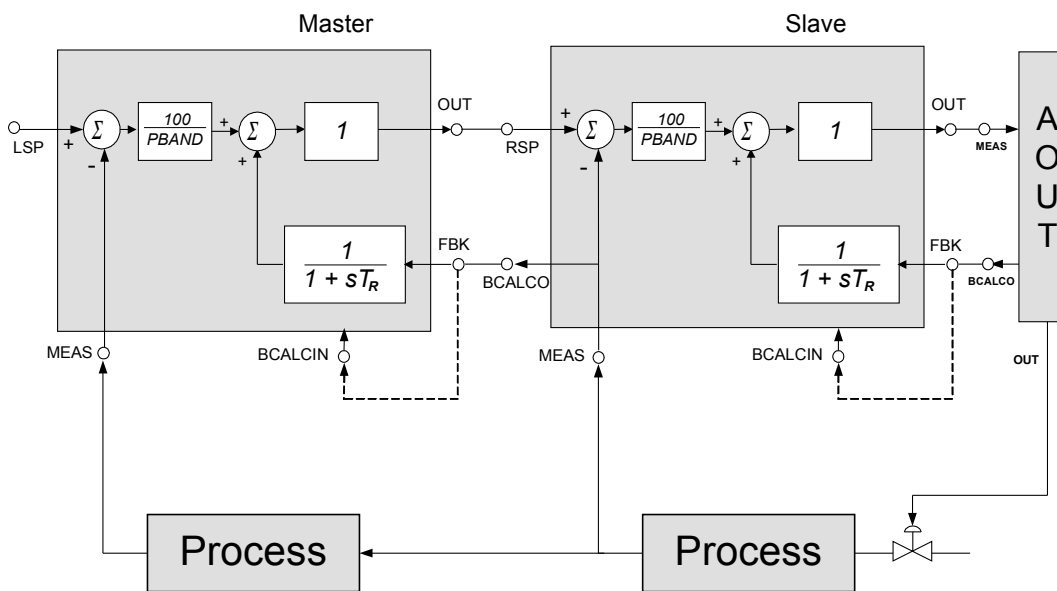


**Figure 4-3: Cascade controller showing the advantages of external feedback.**
**(Master controller has measurement of slave as feedback and not**
**Its own output signal)**

A second situation is the use of the conversion output (direct from ECB) of the AOUT block as feedback to the controller; this has the advantage that the output of the controller will always follow this output (instead of giving a reset wind-up), also when the AOUT is put in MANUAL.

A third example is the control scheme with more process variables and only one manipulated variable (the valve). This results in the so called "AUTO SELECTOR" control configuration. Figure 4-4 shows the auto selector configuration of a fuel-oil control loop. In the normal situation the fuel-oil valve is manipulated by the flow controller, controlling the flow to the required setpoint (usually a remote setpoint ) However, when demand is lowered and the setpoint for oil-flow is also lowered, there may come a point where the pressure of the oil before the burner drops below the required minimum pressure setpoint. When this happens, the pressure controller will request the oil-valve to further open and this request will be honored via the high signal selector. The takeover from flow controller to pressure controller will be 100% bumpless. When the pressure controller is in charge, the output signal of the flow controller will follow, as the feedback signal to the controller is taken after the high selector. When the flow controller is in charge, the pressure controller will follow the selector switch output signal.
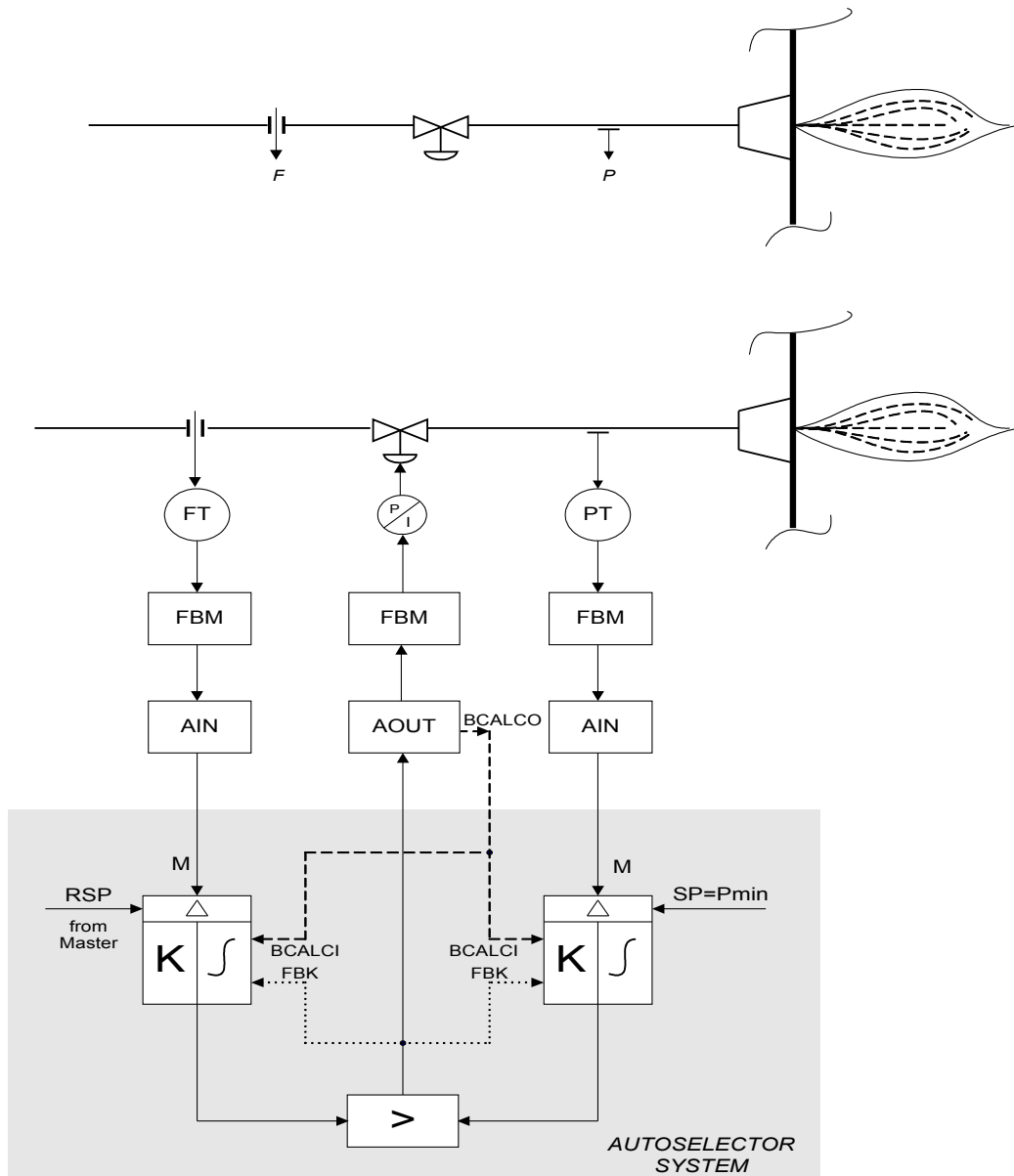
**Figure 4-4: Auto selector control scheme: this scheme prevents a pressure to become to low when requested flow is low.**
**Both controllers do get the output of the selector switch as feedback signal; this ensures that both controllers will always follow the output signal to the valve.**

### 5- Foxboro (PID) controllers with External BIAS

Applicable for all I/A series controllers (PID/PIDA/PIDE etc.)

The External BIAS parameter offers an extra function on all type of  I/A Series controllers.
It is specially created for control schemes with FEEDFORWARD control.
The calculated feedforward signal (this is the calculated output of the controller, based on the expected behavior of the process) is connected to the Ext-BIAS parameter of the controller and this signal is added to the output of the controller. The Ext-BIAS signal  is at the same time subtracted from the feedback signal (see fig 5-1). Although this output signal manipulation is done inside the controller, it can be seen in fig 5-1, that the old PI algorithm is not modified (see the part inside the dotted lines Therefore the Ext-BIAS signal is added to the output signal, without further effect on the PI control algorithm. This is exactly what is required with a feedforward signal: it should be added to the controller output, without any further effect on controller behavior.
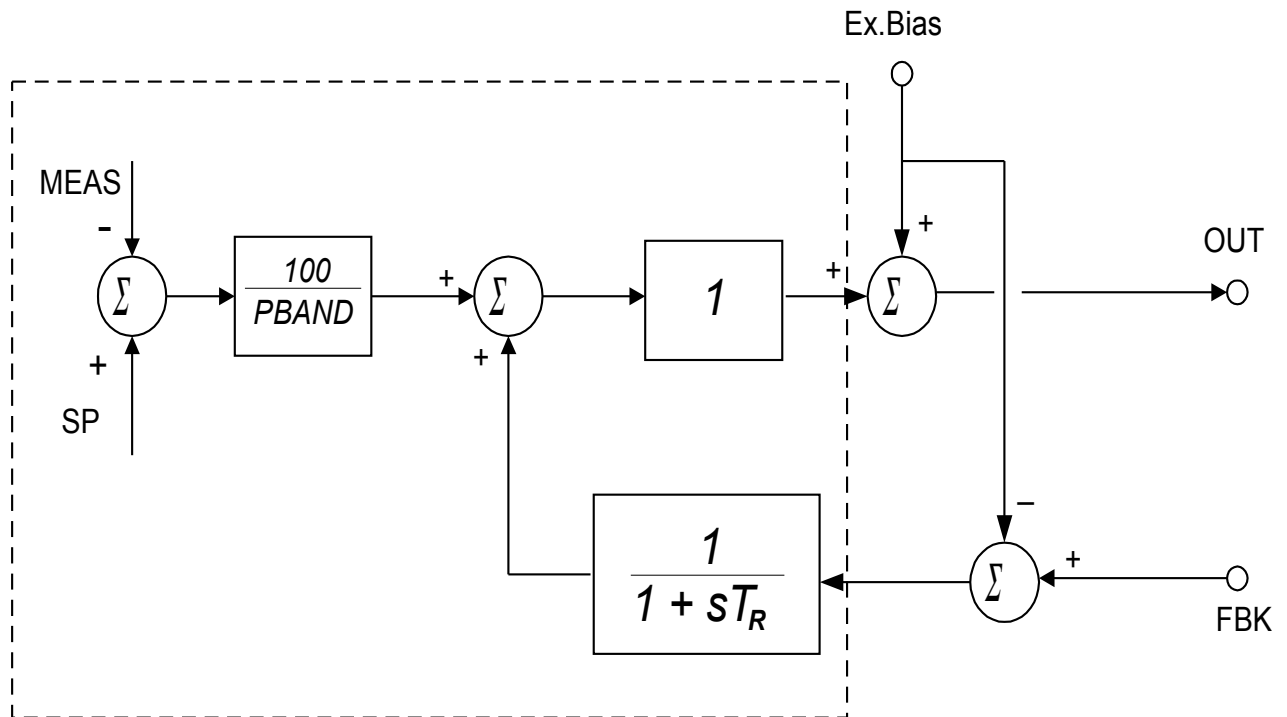


**Figure 5-1: External BIAS function as standard function of Foxboro controllers**

The major advantage of using external bias input  is that the feedforward (bias) goes to the control valve via the controller. When the controller is taken on MANUAL, it is possible to manipulate the valve position from the controller block. By doing so, it is part of the solution that there will be bumpless transfer when going from MANUAL to AUTO. This is not the case when the feedforward signal is added to the controller output signal in some calculation block between controller and AOUT block to the valve.

For an example of feedforward control with Cascade Control Loop, see figure 5-2

**Figure 5-2: Use of External BIAS in cascade loop with Feedforward signal**

A typical example of a process with an "additive laod" is a vessel with an inflow and an outflow. When the level in the vessel is controlled through manipulation of the outflow, a variation in the inflow can be used as a feedforward signal for the levelcontroller. Every ltr/hr extra in the inflow should be compensated by an extra ltr/hr in the outflow. The feedforward signal can be linked to the External BIAS of the level controller (see fig 5-3)



**Figure 5 – 3: Example of "additive load" and Feedforward compensation via external BIAS**

### 6- Foxboro Controller with MULTIN

The MULTIN function allows the user to multiply the output signal of a controller with an external value (just like the Ex-BIAS, except for the MULTIN signal there is a multiplication instead of an addition). Figure 6-1 shows the schematic for this function. It can be seen that the controller output signal is multiplied with a signal from outside (MULTIN) and that the feedback signal is divided by the same signal. In this manner the PI algorithm can function as if there is no multiplication done.

The MULTIN function is specifically developed for ratio type control; The ratio calculation is executed inside the controller.
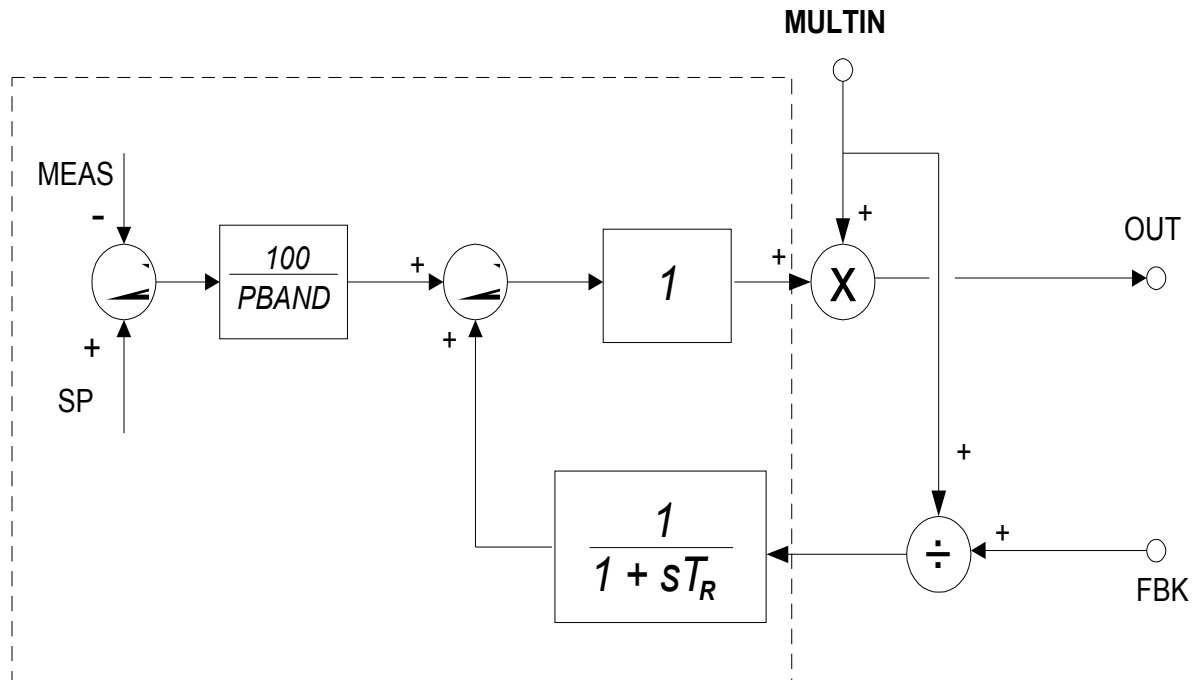


**Figure 6-1: MULTIN  function as standard function of Foxboro controllers.**

A simple example for such "multiplicaive load"compensation is a process where two flows, one cold and one hot are being mixed, with the goal to reach a certain temperature. If one of the flows is increased with factor 2, the other flow should also be increased with factor 2 (flows should stay in same ratio). The control scheme for such process can be found in fig 6.2, where can be seen how the MULTIN of TIC is used to calculate the feedforward compensated remote setpoint for the FIC of the cold fluid.

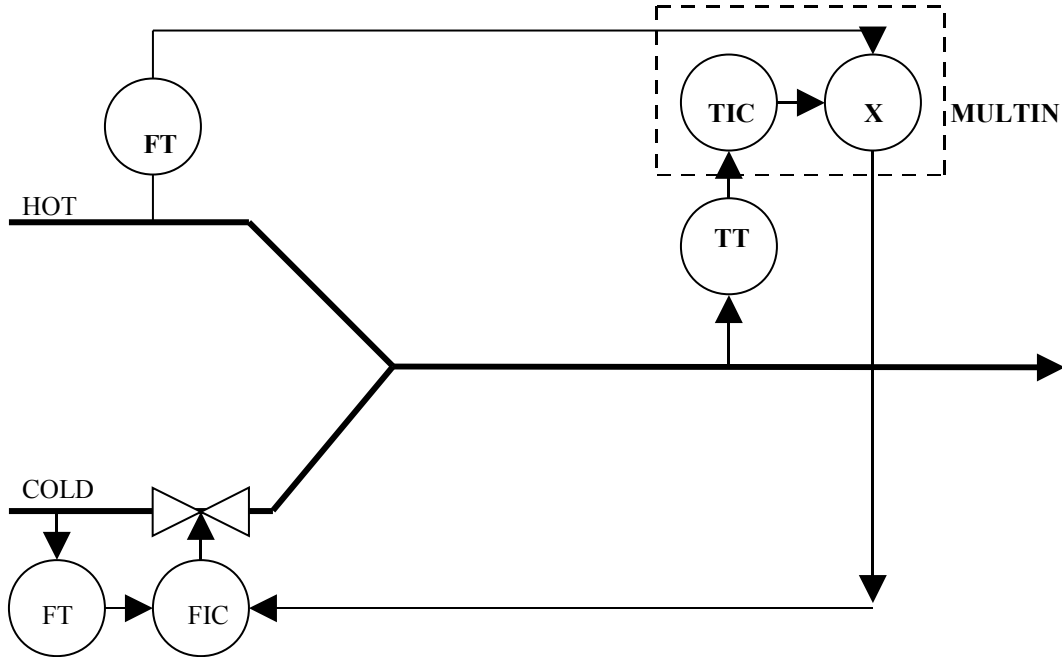**Figure 6 – 2: Example of Multiplicative or Ratio Control**

A more complex example can be seen through the figures 6-3 and 6-4

In figure 6-3  the standard solution for ratio control is shown. The output of the pH controller will change the multiplication factor of the ratio block. When feed changes, the flow of reagent will change in order to keep pH in the reactor the same. Only when the pH of the feed changes (or when there is taking place a reaction in the reactor) the controller will take action and change the ratio factor for the RATIO block.

In figure 6-4 the same control scheme is shown, however the RATIO calculation has been included in the PIDA block (pH controller).

The advantage of using this solution is identical as the advantage for the Ex-BIAS. All provisions for bumpless change-over from AUTO to MANUAL and reverse are part of the standard functions of the controller block.

An added advantage is that in the MULTIN signal more than one disturbance can be included. For instance, when the pH of the feed is measured, it would be possible to first multiply the feed signal with the measured pH and use the result as input to the MULTIN parameter of the controller. In this manner an even more optimal feed forward control solution has been created.

A second application for the MULTIN function is in processes were the throughput has big variations. In such processes the process dynamics do change greatly: the MULTIN parameter can be used to increase or decrease the controller output signal as a function of the throughput of the process. This is an attractive alternative for changing the controller PB or integral time.
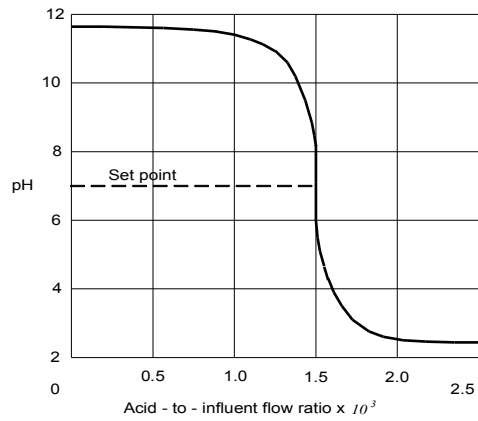


*The gain of the pH curve is the pH deviation caused by a change in reagent flow*
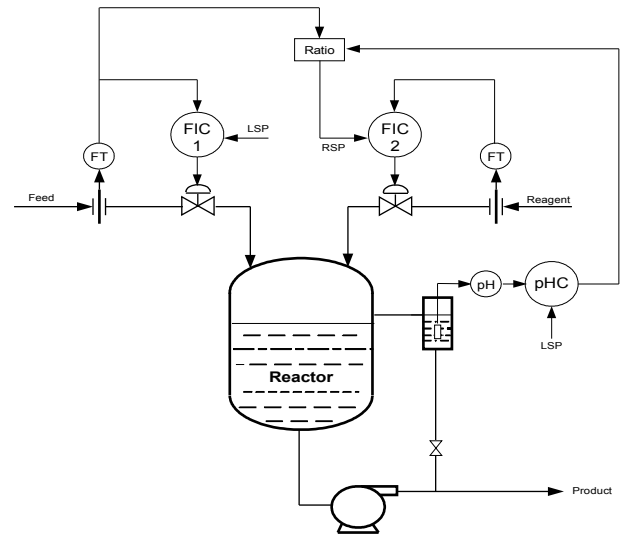


*When controling a pH in the reactor, flow ratio should be trimmmed by a nonlinear contrller*

**Figure 6- 3: Standard RATIO control**
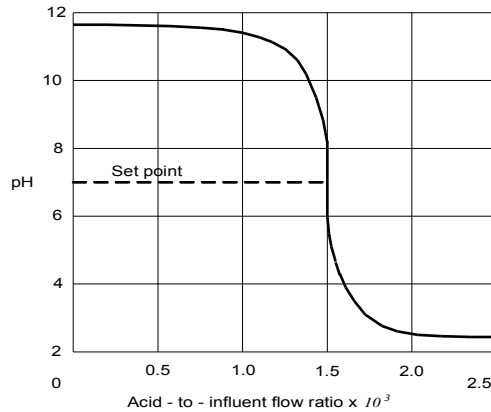
The gain of the pH curve is the pH deviation
caused by a change in reagent flow
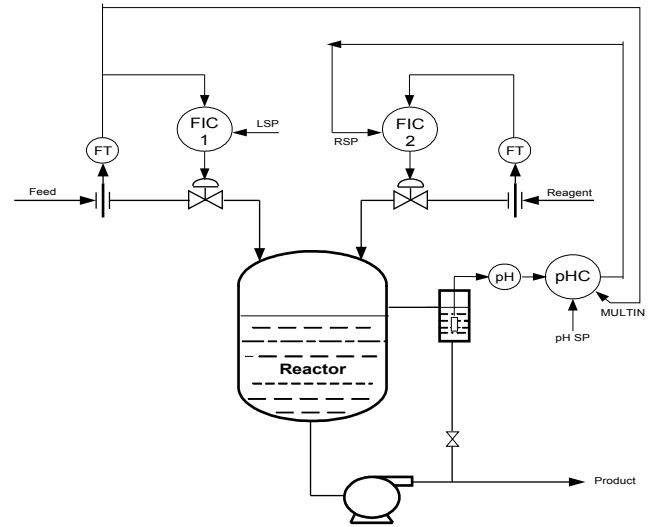


When controling a pH in the reactor, flow ratio
should be trimmmed by a nonlinear contrller

**Figure 6- 4: Same control function as shown in figure 6- 2, except the ratio calculation function has been replaced with**

**the MULTIN function in the Ph Controller. The pH Controller now has as output the multiplication of the**

**following two signals: the "internal controller output signal" X measurement of main flow in.**

**7- Proportional (P) and Proportional + Integral (PI) control in digital control systems:**
       **lessons to learn.**

In digital control systems, the control algorithm is executed each block execution period (scan period). This results in a different control behavior compared to an analogue controller, in particular for very fast processes, like a (non compressible) flow control system.
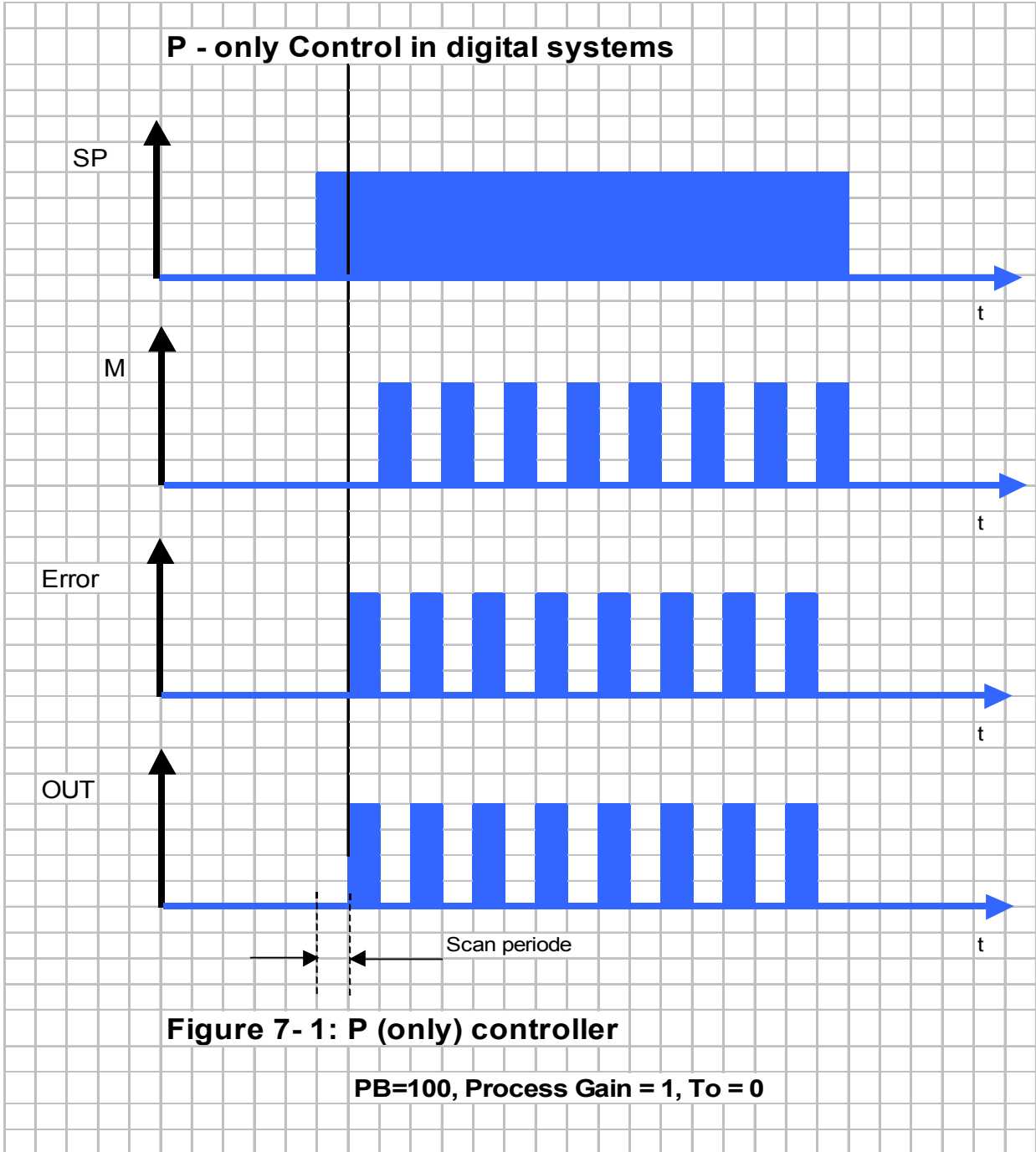
For a process with process gain, $Ko = 1$ and with a proportional only controller with $PB = 100$, the MEASUREMENT will follow the change in de controller output for 100% after one processing cycle. This will result in the next processing cycle in an error of 0 (zero) and a resulting controller output back to start position. When controller output is back to start position, this will result, one control cycle later, in the MEASUREMENT going back to the start position which in turn gives an error equal to the error of two control cycle ago. The result is a block pattern in the controller output and in the MEASUREMENT. (see figure 7- 1).

**Foxboro . SimSci . Triconex . Wonderware**

# P - only Control in digital systems



**Figure 7- 1: P (only) controller**

**PB=100, Process Gain = 1, To = 0**

The above described block pattern in control loops with "P"-only controller is undesirable; it can easily be solved by the introduction of a filter in the measurement of the flow. This filter can have a time constant = to 3 scan periods. Having such filter in the measurement will result in a control behavior as shown in figure 7- 2.



**P - only Control in digital systems**

SP

delta

M

0,5 delta

Error

OUT

Scan periode

**Figure 7- 2:** **P (only) controller, with first order filter in Measurement**

**PB=100, Process Gain = 1,**

**Filter time constant, To = 3 scan periods**
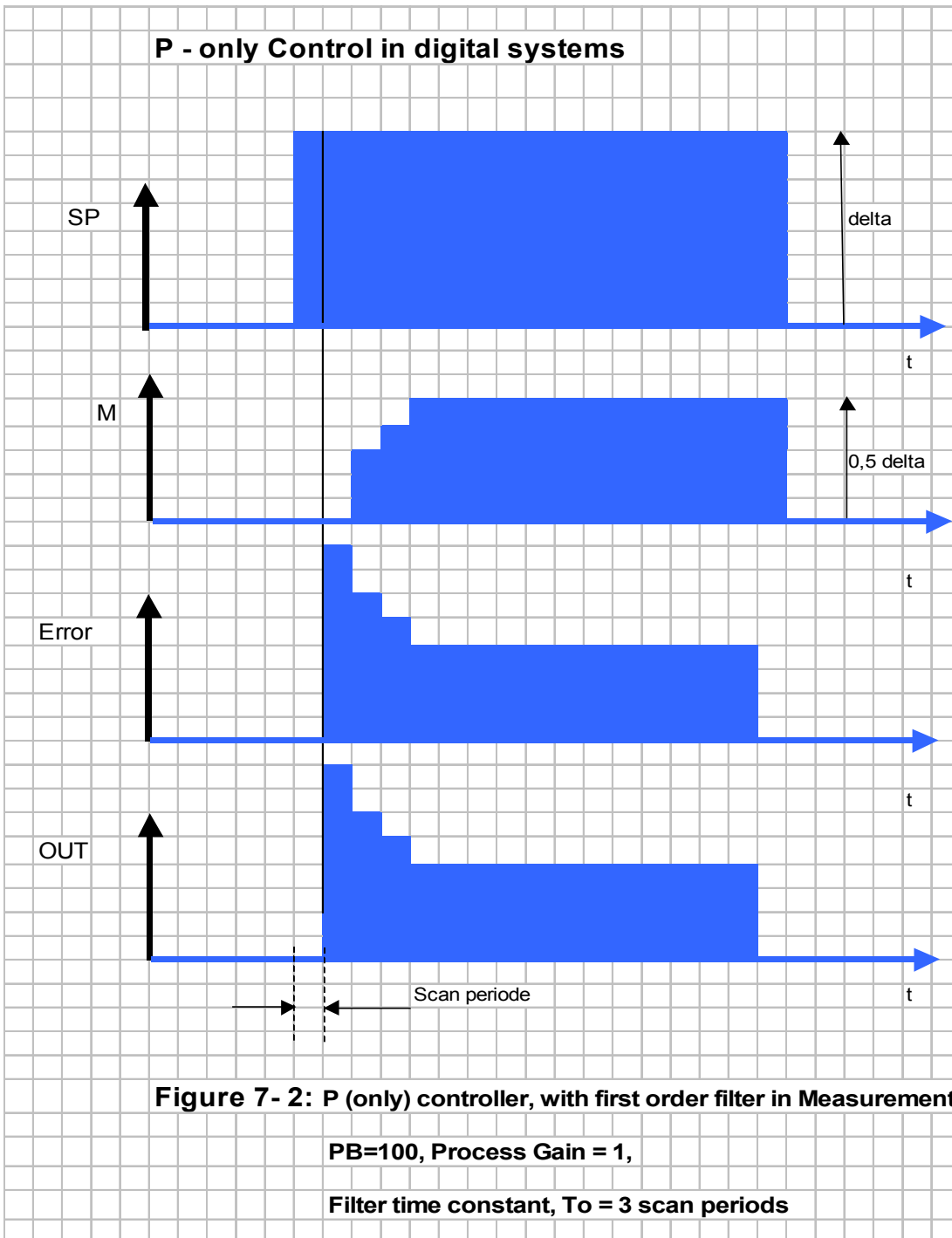
The disadvantage of a P-only controller is that there is always an offset. The offset can be reduced in two manners: first by using an external BIAS and second by decreasing the Proportional Band (PB). These two methods should be combined.

The external bias should be set so that the controller output is equal to the most occurring operating condition. When setpoint is equal to the most occurring operating condition, the P action of the controller will be = zero (controller output = external bias: this results in a process condition (MEASUREMENT) equal to the requested most occurring operating condition).

Under this condition, with P-action = zero, the offset is also zero.

Making the PB smaller will reduce the offset, however this will make the controller "nervous".
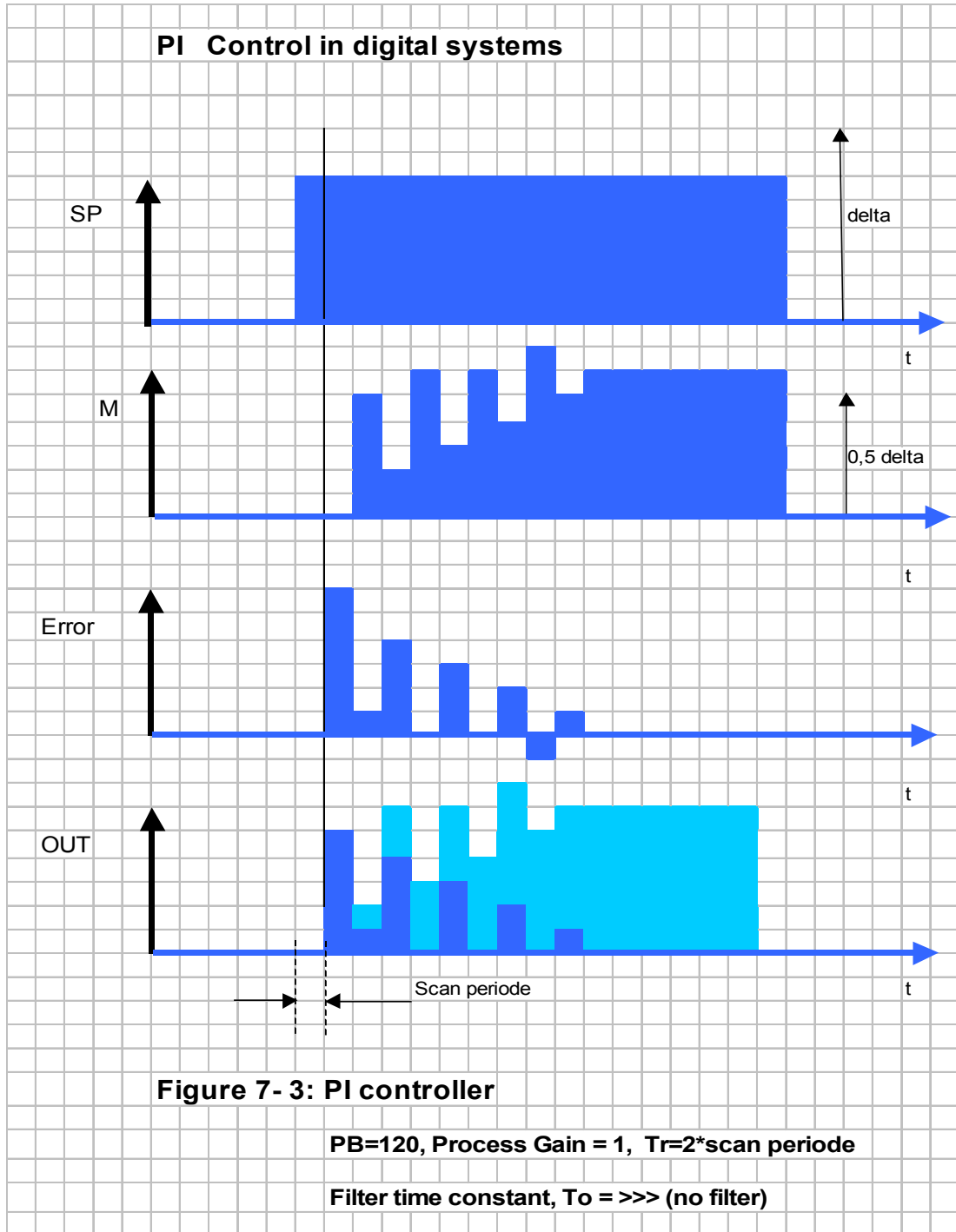
With process gain (Ko) = 1 and a PB = 100 ( than Ko*(100/PB) = 1), the offset created by the controller action will be 50% of the error. This offset can be reduced for PB smaller than 100.

If PB = 50 (or Ko*(100/PB) = 2 ) the resulting offset is 25 % of the error.

The above described behavior for P-only controller in control of very fast processes will be the same for PI controller: the process will react so fast, that the integral action will never become active. This is only true when Ko*(100/PB) = 1.

In case Ko*(100/PB) is smaller than 1 (that means the loop gain smaller than 1), the error will stay positive and the integral action will step by step reduce the error. The result is that after several cycles the measurement will reach the requested setpoint. For examples of described control behavior, see figures 7- 3 and 7- 4

The best control behavior for fast processes is reached when the integral time Tr of the controller is set as small as possible (Tr = scan period) and Ko*(100/PB) = 1

**PI Control in digital systems**

SP

delta

M

0,5 delta

Error

OUT

Scan periode

**Figure 7-3: PI controller**

**PB=120, Process Gain = 1, Tr=2*scan periode**

**Filter time constant, To = >>> (no filter)**

**PI Control in digital systems**



**Figure 7- 4: PI controller**

**PB=200, Process Gain = 1, Tr=2*scan periode**

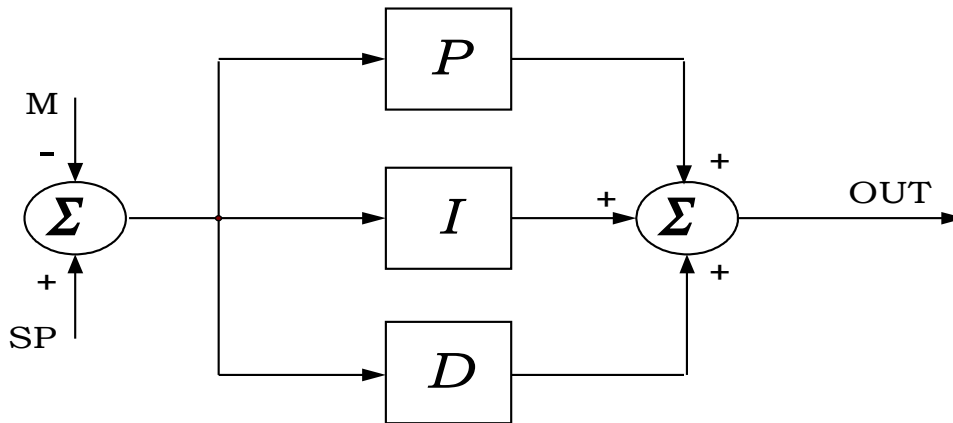**Filter time constant, To = >>> (no filter)**
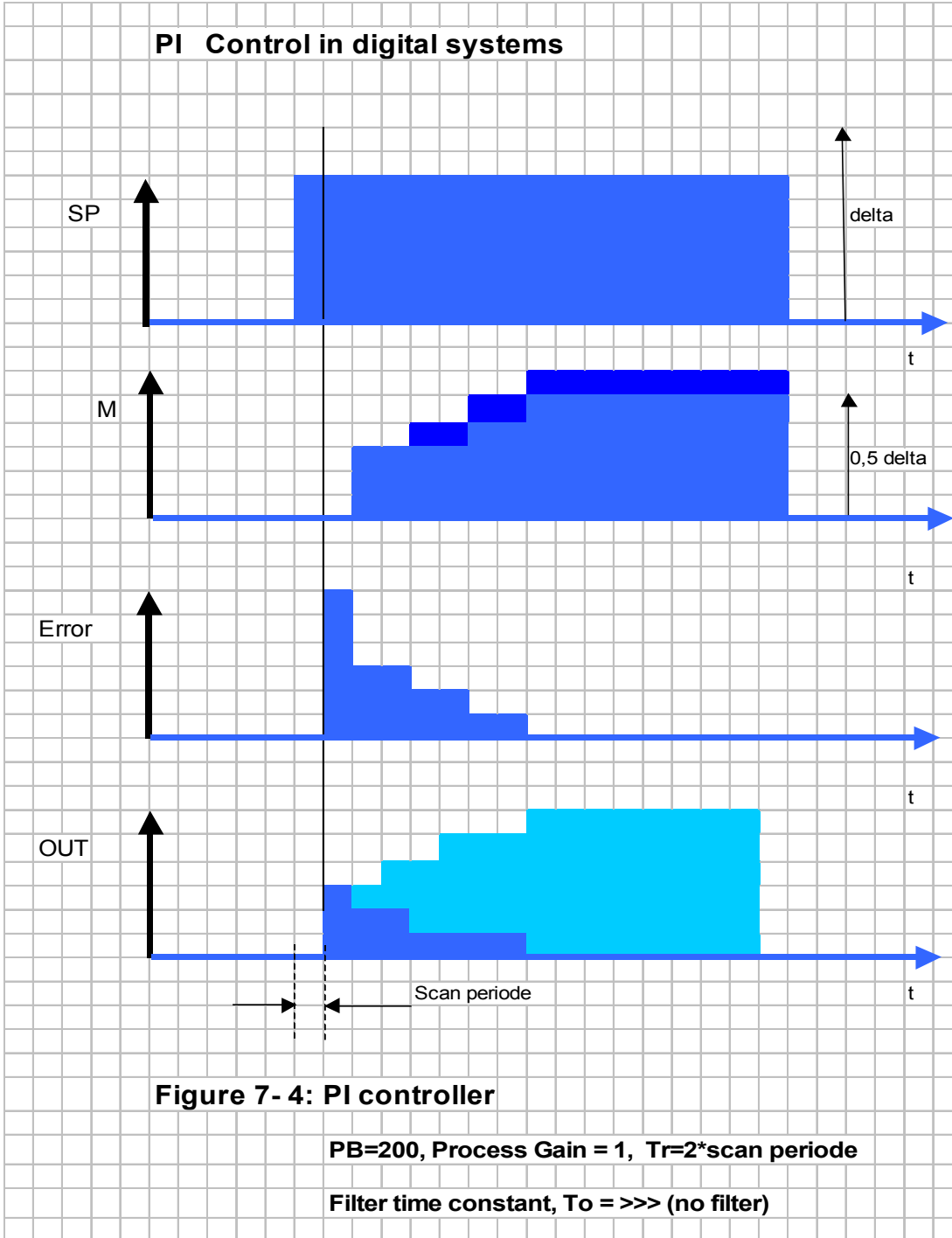
**8- Old and New structures for PI(D) controllers**

Figure 8- 1 shows the evolution that has taken place in the algorithms used for PI(D) controllers.

## Old PID Structure



$$OUT = (SP - M) * [k + \frac{1}{T_R s} + T_D s]$$

## New PID Structures

a)  $$OUT = (SP - M) * \frac{100}{PB} * [1 + \frac{1}{T_R s} + T_D s]$$

b)  $$OUT = SP * \frac{100}{PB} * [1 + \frac{1}{T_R s}] - M * \frac{100}{PB} * [1 + \frac{1}{T_R s} + T_D s]$$

**Figure 8- 1: Old and new PID controller structures**

In the Old PID structure the error (SP-M) was calculated and the controller proportional, integral and derivative actions were calculated without interaction:

P action was = (SP-M)*k,        where k = controller GAIN
I action was = (SP-M)*(1/Tr*s)  where Tr = the reset (or integral) time
D action was = (SP-M)*Td*s       where Td = the derivative time

**Foxboro  .  SimSci  .  Triconex  .  Wonderware**

invensys

Process Systems

ISO 9001 en VCA**

DNV

GECERTIFICEERD
BEDRIJF

In these controllers the proportional, integral and derivative actions were non-interactive; this means that when changing the controller gain, only the proportional action was influenced. So when controller gain k is increased with a factor 2, the proportional action would increase with factor 2, while the slope of the intergal action whould stay as it was before. Therefore, the term repeat time for integral action, can not be used for these non-interacting controllers.

Under new PID structures there are two new schemes:
The first scheme (a) shows that the proportional tuning parameter (PB) will have an effect on all three actions, the proportional, the integral and the derivative, as the error (SP_M) is multiplied by 100/PB; this result is being used to calculate the proportional, integral and derivative actions

P action was = $(SP-M)*(100/PB)$, where PB = controller Prop. Band
I action was = $(SP-M)*(100/PB)*(1/Tr*s)$ where Tr = the reset (or integral) time
D action was = $(SP-M)*(100/PB)*Td*s$ where Td = the derivative time

This structure can be found in old Foxboro controller, like SPEC 200

For these controllers the proportional, integral and derivative actions do interact. When PB is decreased with factor two, the integral action will be increased with factor 2 (slope will be twice as steep) and the differential action will also be twice as big. For the integral action it means that the repeat time (this is the time it takes for the integral action to become as big as the proportional action is constant = Tr.

The structure under new PID structure b shows yet another difference: here the derivative action (Td*s) is only active in the term with the Measurement (M). The Setpoint (SP) only has proportional and integral action. This means that there will be no derivative action when the setpoint of the controller is changed. Derivative action will only be seen on changes in the measurement. This has great advantages, as setpoint changes usually are step changes, giving a big "bump" to the derivative component of the output of the controller.

This structure can be found in Foxboro Spectrum and I/A controllers

**9- Special Setpoint Lead Lag function PI controller in PIDA (MODOPT = 4 or 5 or 6 )**

The PIDA block with MODOPT = 4 or 5 or 6 has a special algorithm, resulting in a different behavior of the controller output, depending in changes in the measurement or the setpoint.
In most standard PI controllers, the controller output will behave in a similar manner to a step change in the measurement or the setpoint. The algorithm of the PIDA block show a LEAD-LAG function for step changes in the setpoint. This is made clear through the calculations in figures 9- 1. The resulting block scheme for this controller is shown on figure 9- 2.
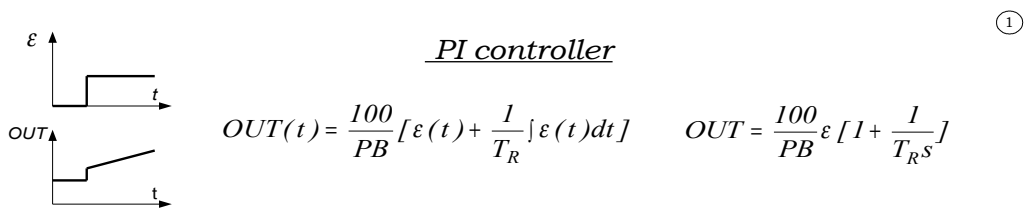


$$PI\ controller$$

$$OUT(t) = \frac{100}{PB}[\varepsilon(t) + \frac{1}{T_R}\int \varepsilon(t)dt] \qquad OUT = \frac{100}{PB}\varepsilon[1 + \frac{1}{T_R s}]$$

$$PIDA\ --\ MODOPT\text{-}4$$

$$OUT = [100/(PBAND * INTs)][(SPLLAG * INTs + 1)SPT - (INTs + 1)MEAS]$$

$$OUT = \frac{100}{PB * T_R s}[(LL * T_R s + 1)SP - (T_R s + 1)M]$$

$$OUT = \frac{100}{PB}[(LL + \frac{1}{T_R s})SP - (1 + \frac{1}{T_R s})M]$$

$$OUT = \frac{100}{PB}[(\frac{LL + \frac{1}{T_R s}}{1 + \frac{1}{T_R s}}) * (1 + \frac{1}{T_R s})SP - (1 + \frac{1}{T_R s})M]$$



$$OUT = \frac{100}{PB}[(\frac{1 + LL * T_R s}{1 + T_R s}) * (1 + \frac{1}{T_R s})SP - (1 + \frac{1}{T_R s})M] =$$

$$= \frac{100}{PB}(1 + \frac{1}{T_R s}) * [(\frac{1 + LL * T_R s}{1 + T_R s})SP - M]$$
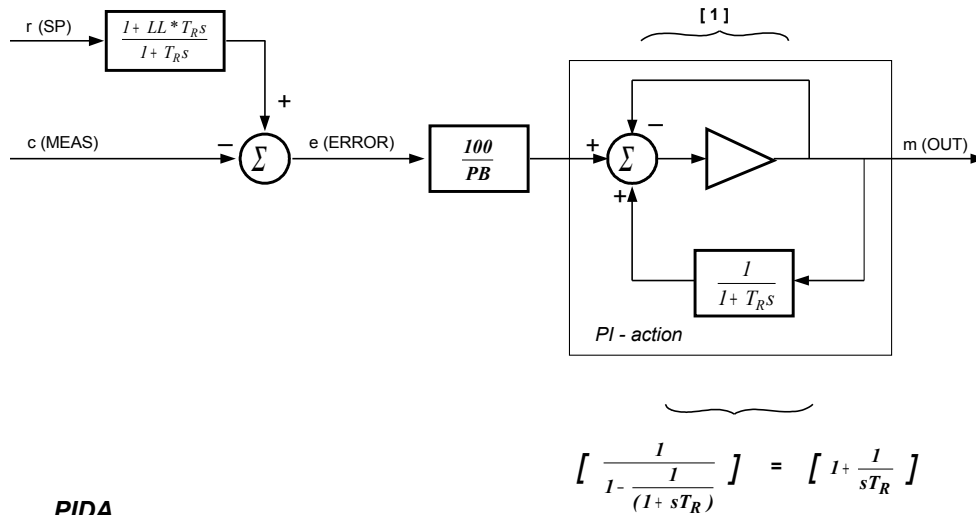
1)LL=1
$$OUT = \frac{100}{PB}(1 + \frac{1}{T_R s}) * [SP - M]$$

2)LL=0
$$OUT = \frac{100}{PB}(1 + \frac{1}{T_R s}) * [(\frac{1}{1 + T_R s})SP - M] =$$

$$= \frac{100}{PB}[(\frac{1}{T_R s})SP - (1 + \frac{1}{T_R s})M]$$

3)LL=0,5
$$OUT = \frac{100}{PB}[(0,5 + \frac{1}{T_R s})SP - (1 + \frac{1}{T_R s})M]$$

**Figure 9- 1: SPLLAG function in PIDA block**

③



**PIDA**
**Block Diagram**
**(MODOP-4)**

$$\left[ \cfrac{1}{1 - \cfrac{1}{(1 + sT_R)}} \right] \; = \; \left[ 1 + \cfrac{1}{sT_R} \right]$$

## Figure 9- 2 :   Block scheme of PIDA with SPLLAG function

This figure 9- 2 shows a Lead-Lag function in the setpoint signal to the controller

$$\frac{1 + SPLLAG * Tr * s}{1 + Tr * s}$$

(actually, this lead-lag is part of the controller). This means that a step change in a setpoint will not enter the controller algorithm as a step, but this step will go through a LEAD/LAG function (filter).

The parameter SPLLAG (Setpoint Lead-Lag) can have values between =0 and =1.

The behavior of a LEAD-LAG function block is shown in fig 9 -3

If SPLLAG = 1, than the LEAD-LAG  OUT signal behaves like a step change (no filter).
If the SPLLAG = 0, than the is filtered by a first order filter (see also fig. 9- 3)
If the SPLLAG is >0 and <1, than the is "filtered" by the Lead-Lag function.

④

### *LLAG  Action*
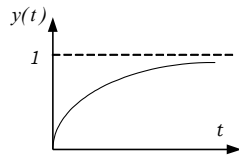


a)  LAG  action

$$y(t) = (1 - e^{-\frac{t}{T}})\qquad T\frac{dy}{dt} + y = x \qquad \frac{y}{x} = \frac{1}{1 + Ts}$$
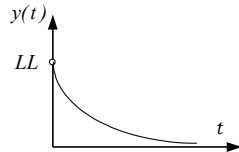
b)  LEAD  action

$$y(t) = LL * e^{-\frac{t}{T}}\qquad T\frac{dy}{dt} + y = LL * T\frac{dx}{dt}\qquad \frac{y}{x} = \frac{LL * Ts}{1 + Ts}$$

c)  LLAG  action

$$\frac{y}{x} = \frac{1}{1 + Ts} + \frac{LL * Ts}{1 + Ts} = \frac{1 + LL * Ts}{1 + Ts}$$

**Figure 9- 3: Signal response of SPLLAG function for different values of
SPLLAG**

**Foxboro   .   SimSci   .   Triconex   .   Wonderware**

In figure 9 – 4, the open loop behavior of the PIDA with MODOPT 4 is shown for different values of SPLLAG parameter. From these curves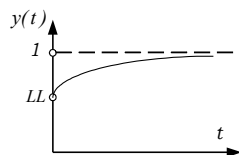 it is clear that the effect of SPLLAG **on the output** of the open loop controller is **different** as expected (not according above mentioned theory); the Setpoint does not seem to be filtered, however the SPLLAG parameter does only have an effect on the size of the proportional action due to changes in setpoint. (this becomes in particular clear for changes in setpoint with result that Error becomes = 0; in such situation there is no proportional action at all when SPLLAG = 0.)
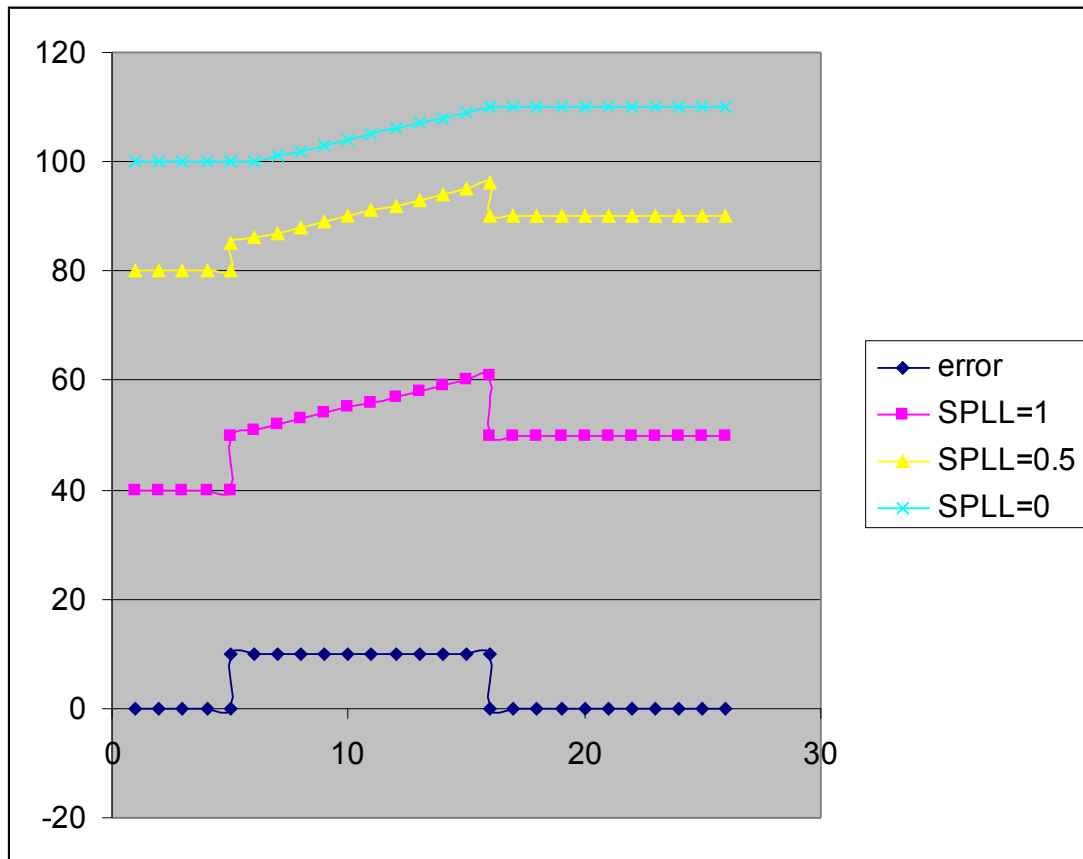


**Figure 9- 4 : Open Loop control behavior of PIDA, MODOPT 4 with SPLLAG**
                **(shown are open loop behavior controller outputs on change in**
                     setpoint )

If SPLLAG = 1,    than the controller gain is (100/PB)*1
If SPLLAG = 0.5, than the controller gain is (PB/100)*0.5
If SPLLAG = 0,    than the controller gain is (PB/100)*0 = 0

In general: **Controller GAIN** (for proportional part and for setpoint changes) **= SPLLAG * (PB/100)**

In other words, the SPLLAG parameter only changes the size of proportional action of the controller output for changes in setpoint (and only for changes in setpoint !) This is also clear from the formulas in Figure 9 – 1.

The above means that for values of SPLLAG <1, the effect on the process through a change in the controller output created by a setpoint change will be smaller, as the requested setpoint change will result in less proportional action.

In the PIDA with MODOPT=7 (PITAU) the parameter SPLLAG is no longer available. Still there is a filter on the setpoint, being:

$$\frac{1}{1 + Tr * s}$$

### 10- Controllers with Derivative Action

In controllers with Derivative action, the output signal of the controller will give a "spike" on each (step) change of the measurement (in older controller schemes also on stepchanges in setpoint, see paragraph 8). Measurements often do have some noise in the signal and in controllers with derivative action this can result in a very "nervous" controller output signal. Foxboro controllers have (and did have in the past) protection against this.

In older systems (SPEC 200, INTERSPEC,SPECTRUM) Foxboro implemented already extra protection against "nervous behavior" in the controller OUT signal. Fig 10- 1 shows the algorithm used in these systems. The setpoint changes will not have an effect on derivative action .



$$\left[ \cfrac{1}{1- \cfrac{1}{(1+ \frac{s}{T}R)}} \right] \ = \ \left[ 1+ \frac{1}{sT_R} \right]$$
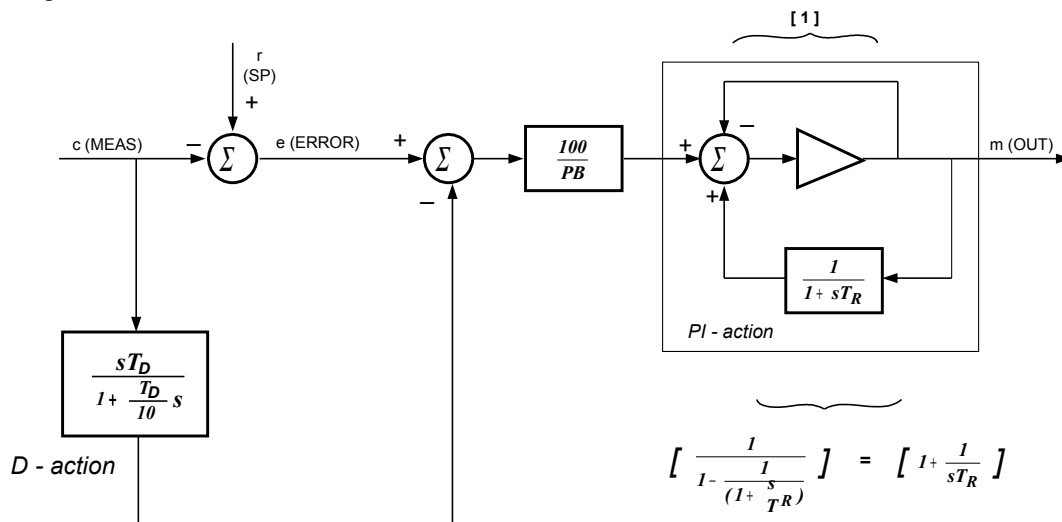
**PID**
**Block Diagram**
**(SPEC 200, INTERSPEC, SPECTRUM)**    The

**Figure 10- 1: PID structure in SPEC 200**

Measurement goes through the functionblock: $\dfrac{sT_D}{1+ \dfrac{T_D}{10} * s}$

This can also be rewritten by:

$$10 * \frac{1}{10+ T_D s} * T_D s$$

This represents the following signal conversions:

- for t=0, > s= ω and therefor the output will be = 10,

- for t=ω, > s=0 and therefor the output will be 0

- between t=0 and t=ω, the output will decrease with time constant Td/10

So, the (step)change in the measurement is at time t=0 multiplied by a factor 10 (for later controllers this factor is called the Derivative Gain (Kd)), than the signal reduces to 0 with a time constant equal to Td/10. This "filter" reduces the size of the derivative action (the peak), but it will at same time make the derivative peak wider.
The resulting signal through the derivative action is shown in figure 10- 2
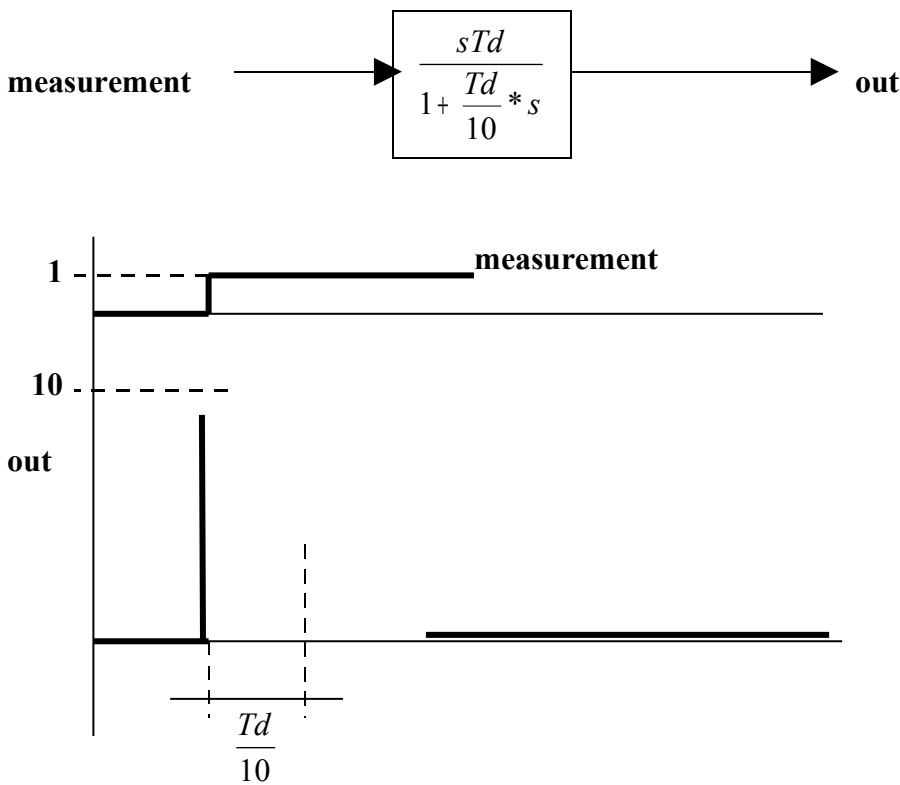




**Figure 10 – 2: Derivative action, open loop, with step change in measurement**
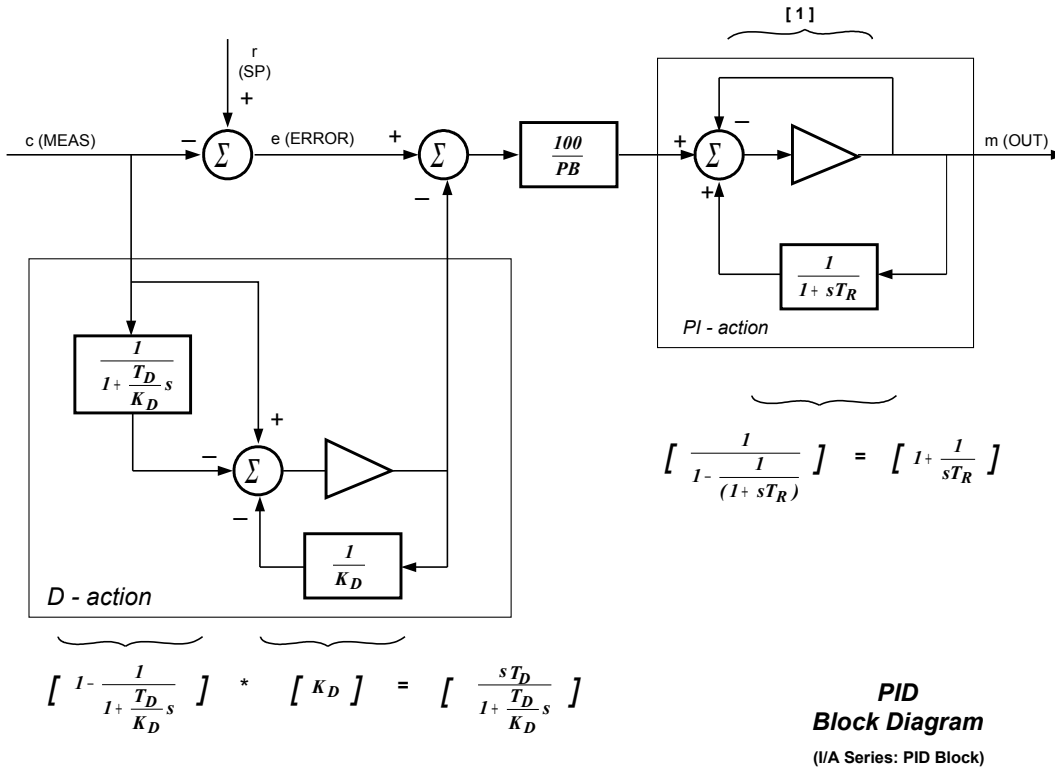
**Figure 10- 3: PID block structure in I/A**

In fig 10- 3 the block scheme for PID is shown and fig 10 – 4 (next page) shows the trend of an **open loop** PD controller (without any integral action) , working in AUTO. ( open loop = the output of the controller is not linked to the simulation process; therefore the process does not follow the controller action). Controller action for different settings of parameters Kd and DERIV are shown.
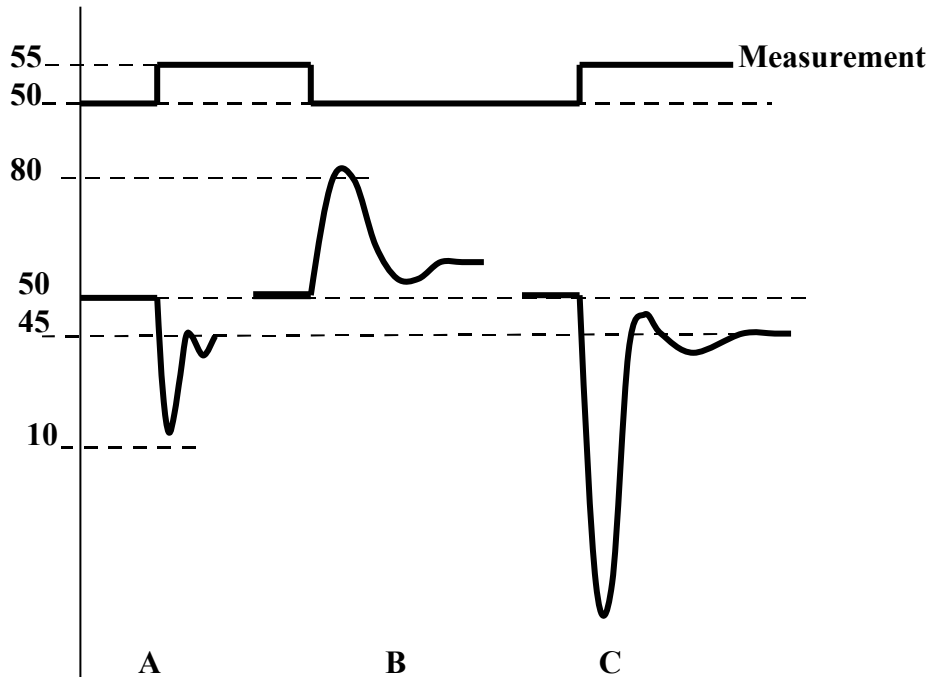
**Figure 10 – 4: open-loop behaviour of PID MODOPT 3 with:**
   **A: Kd=10, Deriv=1,**
   **B: Kd=10, Deriv=3,**
   **C: Kd=20, Deriv=**

In the **PIDA** controller the above mentioned multiplication factor (= 10) is being replaced by a configurable parameter, called the DERIVATIVE GAIN, Kd.

The parameter Kd needs some further explanation: Kd is the derivative multiplication factor. In the older PID block, this value was fixed on 10. This resulted in a derivative action equal to a peak of 10 for a step change in the error of 1.  In the PIDA block this Derivative Gain (Kd) can be set by the user.

In the PIDA MODOPT 3, 5, 6, 7 and 8 options, the MEASUREMENT signal first goes through a high frequency filter (so called BUTTERWORTH filter). The intention is to protect the PID controller against the effects of noise on the measurement. Noise will result in a very nervous behavior of the controller output (differential component in the output). The filter in the measurement will take away this nervous behavior.

The BUTTERWORTH filter is given the function $f(\tau_{au}s)$

$$f(\tau_{au}s) = \frac{1}{1 + \tau_{au} * s + 0.5 * (\tau_{au} * s)^2}$$  (second order butterworth filter)

were the value of $\tau_{au}$ is different for each MODOPT value:

MODOPT = 3:          $\tau_{au} = DERIV / Kd$

MODOPT = 5:          $\tau_{au} = \dfrac{Tr * DERIV}{(Tr + DERIV) * Kd}$

MODOPT = 6:          $\tau_{au} = \dfrac{DERIV}{Kd}$

MODOPT = 7 and 8          $\tau_{au} = 0.25 * \tau / Kd$

In which $\tau = deadtime$   and   Kd   = derivative multiplication factor (limit Kd >0,1)
        Tr = integral time and     DERIV = the Derivative tuning parameter.

A typical characteristic of a butterworth filter is that it gives a lot of filter action on high frequencies and very little filter action on low frequency signals. For MEAUREMENT signals with noise, the butterworth filter will suppress the high frequency noise and as a result this noise will not have an effect on the derivative action of the controller.

The butterworth filter is more effective for bigger values of $\tau_{au}$ .

Tuning a PIDA controller for Derivative actions is rather complex as Kd and DERIV parameters have effect not only on the derivative action but at same time on the effectiveness of the butterworth filter

For example: with **MODOPT = 5 and the integral parameter INT = 1000** (this means no integral action), the value for          $\tau_{au} = \dfrac{Tr * DERIV}{(Tr + DERIV) * Kd}$

equals          $\tau_{au} = \dfrac{DERIV}{Kd}$

For Kd=10 and DERIV=1  the value of $\tau_{au} = 1/10$: this results in a derivative peak in the output of about 10 seconds.

For Kd=10 and DERIV=3  the value of  $\tau_{au}$ =3/10: now the peak will be much wider (about 30 seconds)  and the top of the peak will be lower (about 40 % <<)

For Kd=20 and DERIV=1  the value of  $\tau_{au}$ =1/20: now the peak is slim (only about 3-4 seconds) and the top of the peak is higher (about 40% >>)

For very small values of  Kd, the filter constant  $\tau_{au}$   (for instance Kd=1 and DERIV = 1, thus  $\tau_{au}$ = 1)

the measurement is so strongly filtered that the change on controller output is slower as a result of a change in measurement than for a change in setpoint. (a change in setpoint will only show the proportional action; a step change in setpoint results in a step change in output) while a change in measurement will result in a slow movement of the output to an end value equal to the proportional action of the controller.

In case Kd is configured to be = 0, the controller will no longer change the OUT parameter on changes in measurement (no differential action and no proportional action).

When tuning a controller it is advised to ensure that the value of  $\tau_{au}$  will be between 1/20 and 1/3.

Values outside this range will result in a derivative action that is to fast (  $\tau_{au}$  < 1/20) or a derivative action which is to sluggish ( $\tau_{au}$ > 1/3 ).

It is also advised to keep Kd > 5.

## 11 Limit Options

The PIDA block has so called **limit options;** for the PIDA block the user must specify one out of 3 options (LIMOPT= 1, 2 or 3). The PID block does not have this feature.

However it is possible to configure a control loop with PID and AOUT block, with AOUT block having output clamping specified. First we will investigate the behavior of such standard PID controller which is connected to an AOUT block with output clamping.

The resulting control behavior (for PID + AOUT with clamp on AOUT) is shown in the trend curve on figure 11- 1.
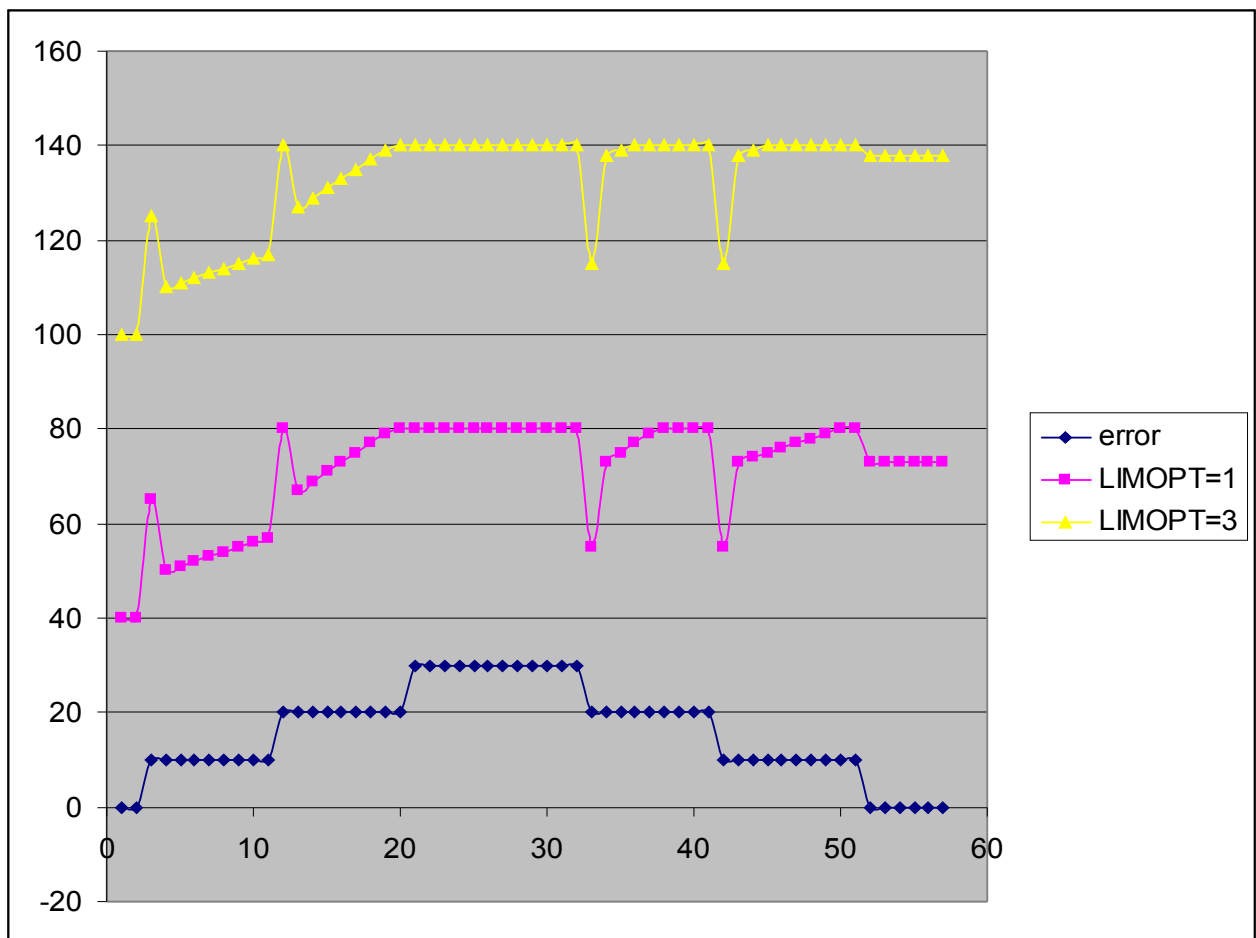


**Fig. 11 – 5: Open loop controller output for PIDA with differential action and LIMOPT 1 and 3.**
   **(for LIMOPT = 1: HILIM = 80**
    **for LIMOPT = 3: HILIM = 140)**

It can be seen from this trend, that in case the output clamp of the AOUT is set to 70, that the controller output will integrate in a normal manner to the 70% output. At that point, the feedback (FBK) signal to the controller, coming from the AOUT block parameter BCALCO, is clamped to the limit of 70%. This feedback signal goes through the positive feedback loop in the controller, via the first order lag, to the output.

So, while the output of the AOUT block is clamped to 70%, the output of the controller will continue to rise like a first order process to it own limit. The proportional action by the controller on variations in the Error (= Measurement – Setpoint) are still visible on the controller output, however this reaction becomes smaller with time (depending on time constant of the first order filter which is equal to the integral time of the controller).

The fact that the controller continues to integrate above the 70%, can be seen as a certain form of integral wind-up, however this windup is restricted to the controller proportional action output. When the output of the controller has been for a long time in the clamp position, the Error must be reduced to less that 0 (error must change sign) in order to get the output out of clamp position. The above will give a rather sluggish control behaviour.

The same open control loop configuration with a PI(D)A block with MODOPT=5 is tested and results are shown in figures 11- 2 for LIMOPT=1, figure 11- 3 for LIMOPT=2 and figure 11- 4 for LIMOPT=3. Figures 11 – 2/3/4 are all without any D action in the controller !.

**LIMOPT = 1:** From the trend shown in fig 11 – 2 it can be seen that the output signal of the controller, once it has reached the output clamp value, will react immediately when the Error signal becomes smaller than the value this Error had when reaching the output limit. In case the Error gets bigger while the output already has reached the output limit, the output signal will <u>not</u> react on decreasing Errors until this Error has reached the value it had when the output signal did go into clamp.

What actually happens is that the integral action stops immediately when the limit has been reached, while proportional action seems to continue in the background; this "background" signal (which is not being shown) seems to be increased and decreased with proportional action only, as long as the controller output is in limit.

**LIMOPT = 2:** From the trend (fig 11 – 2) it can be seen that for LIMOPT = 2 the controller output is clamped to the specified clamp. Any reduction in Error will immediately be reflected in the controller output. The advantage of this behavior is that the controller output will always react on reductions in Error (for LIMOPT = 1, this was not true).

What actually happens is that integral and positive proportional action are stopped when the limit is reached. The background signal is also clamped in the limit value. As soon as the proportional action makes a negative step, this will be seen in the actual controller output signal.

**LIMOPT = 3:** From the trends for this configuration it can be seen that the behavior is equal to the behavior of the PID + AOUT block configuration (for description, see above).

Identical tests with PIDA controller with Differential action and LIMOPT = 1/2/3 show the following:

IMPORTANT: LIMOPT = 2 in combination with differential action will give DANGEROUS controller actions: user should be warned never to use this combination.

The difference between LIMOPT 1 and 3 is being made visible in Fig. 11 – 5.; it can be seen that LIMOPT = 1 results in more action when the Error is getting smaller.
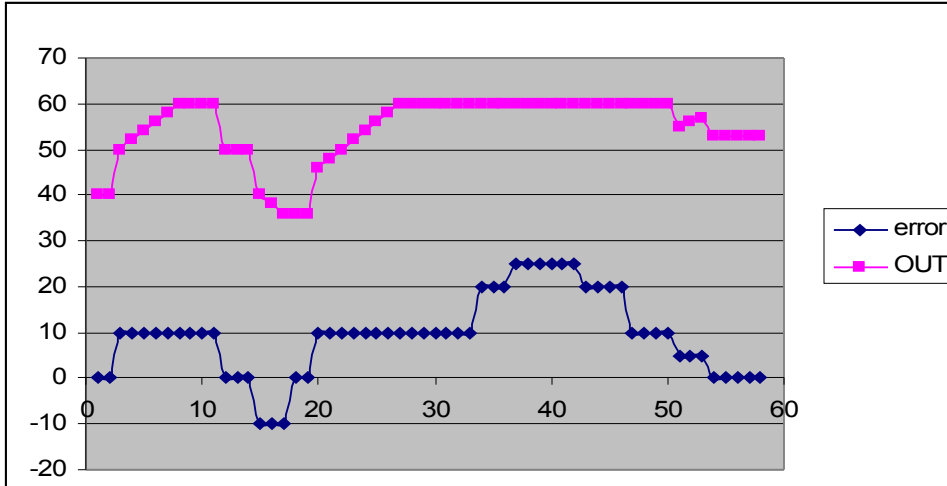
**Figures 11 – 2**

**LIMOPT = 1**
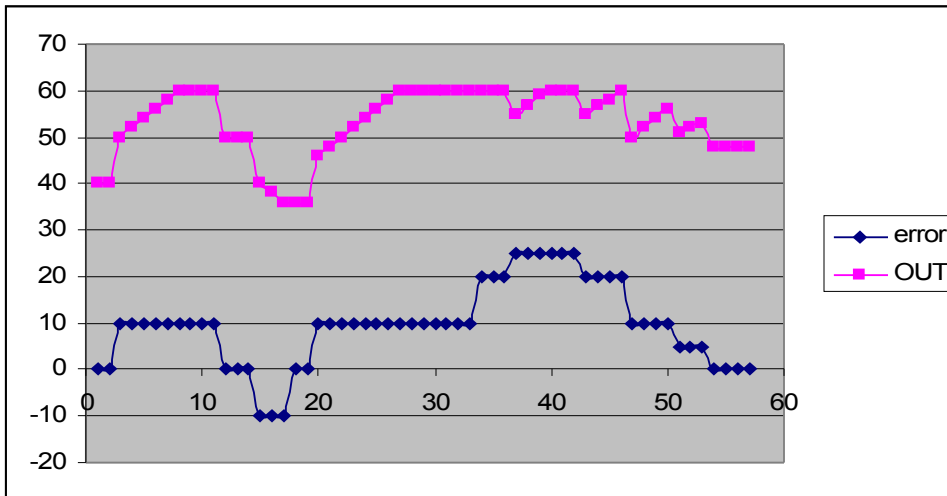
**Figures 11 – 3**

**LIMOPT = 2**
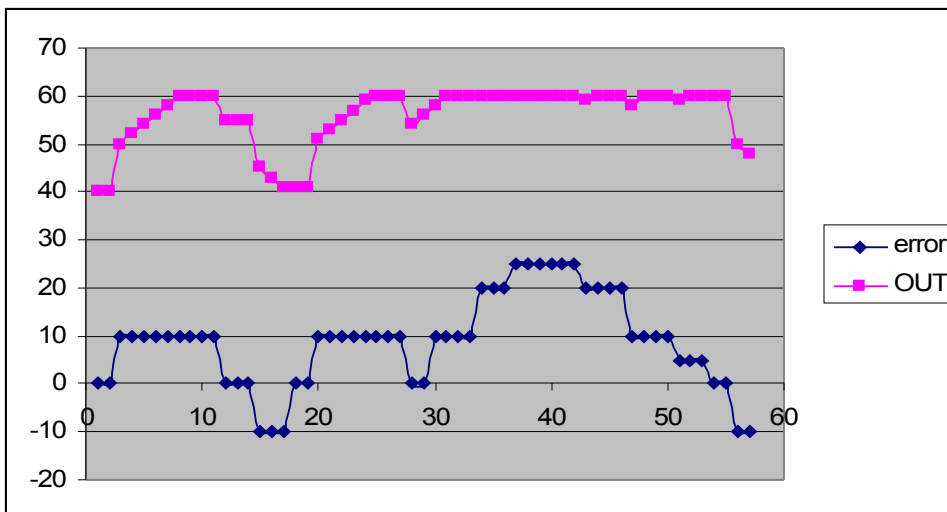
**Figures 11 – 4**

**LIMOPT = 3**

PI(D)A Controller
output behavior,
open loop,
with
OUTPUT LIM = 60
and
LIMOPT = 1 / 2 / 3
(for resp. fig. 2 / 3 / 4)

### 12  Process with DEAD time

The problem when using a PI or PID controller on a process with dead time is that during the dead time period of the process the integral action of the controller will be active. This is "working in the dark" as during the dead time period, there will be no reaction of the process visible. Therefor, with simple PID control, one must accept that the integral action must be slow (Tr  = >>). This will result in sluggisch control behavior.

The Foxboro PIDA controller with MODOPT 7 (PITAU) gives a better solution. The principle is simple: delay the integral action with the same amount of time as the dead time of the process, in other words, after a stepchange in setpoint or measurement, the proportional action will act directly, but the integral action will not start until the dead time of the process is passed.
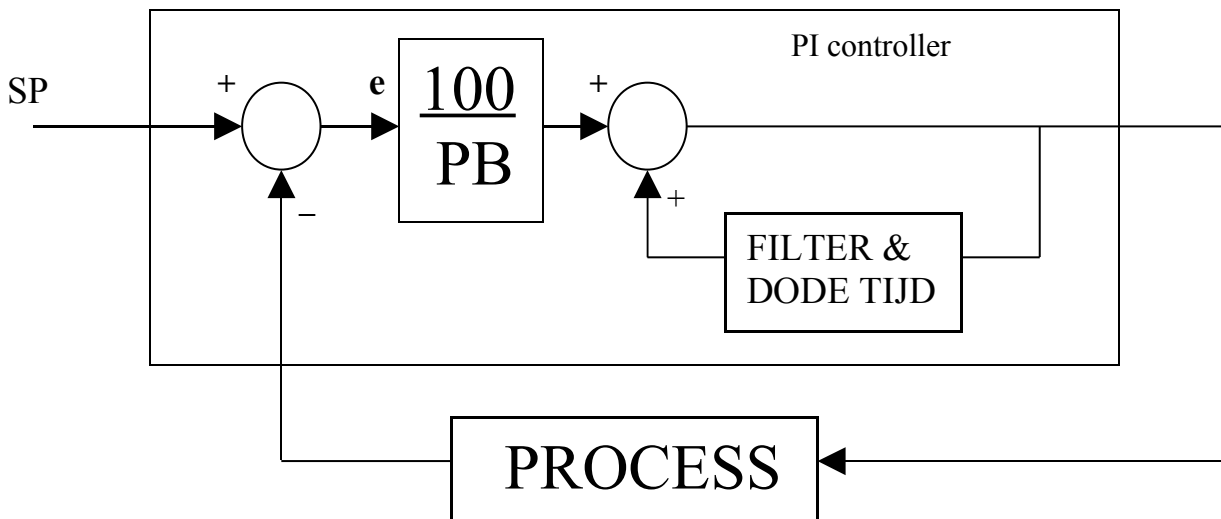


**Figure 12 – 1**: **block scheme of first order process  with dead time and PI control**
**with internally delayed Feedback signal for integral action.**

Figures 12- 1   shows the diagram of a process (first order with dead time = t) and a PITAU controller. The integral action of the controller is realized through the first order function in the feedback loop. When the (feedback) connection between OUT parameter and FBK parameter is broken, there is no integral action. Integral action is delayed through the introduction of the delay in the feedback signal . The open loop behavior of such controller is shown in Figure 12 – 2.  Here it can be seen that integral action only starts after the delay time that has been introduces in the feedback signal to the controller**; for optimal control this  deadtime in the feedback should be equal to the dead time of the process.**
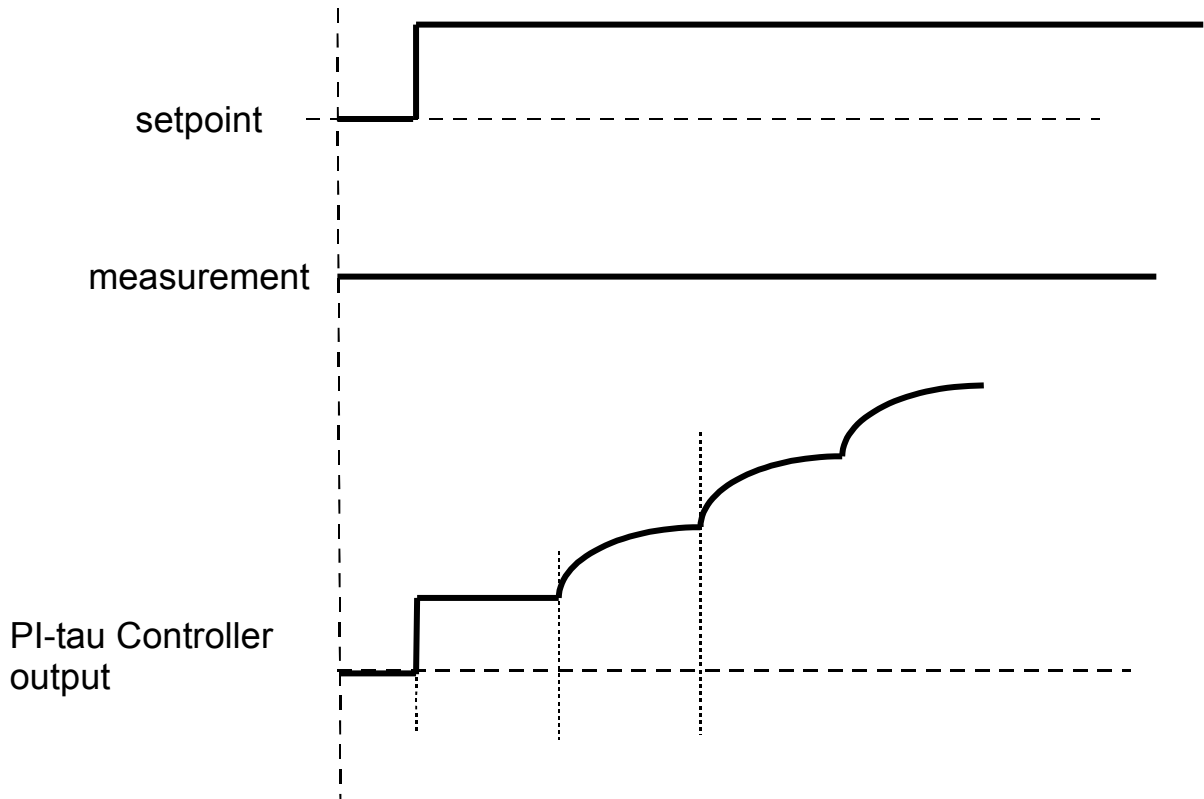
**Figure 12 – 2**: **Open-Loop behavior for PITAU controller**

In figure 12-3 the formula for the PIDA MODOPT 7 (with no derivative action; assumed Td=0) has been reworked in order to reach a layout that can be shown in the diagram of figure 12- 4

The following items are evident:
- The feedback signal in the integral term function (Hf) is delayed with  t
- There is a (first order ) filter in the setpoint (note:  there is no lead-lag function for the setpoint, like there was in the other PID functions)
- The measurement also goes through a filter (second order butterworth filter);
  for a value of Kd = >>, the filter constant "tau" is very small, and the "filter action" can be neglected.

The behavior of the controller is shown for open loop in the trend figure 12- 5.  These trends are valid for a filter constant (one extra parameter for PIDA block) equal to zero, meaning: no filter.

Measurement

**PID**TAU OUT

Setpoint

**PI**TAU OUT
(open loop)

**Figure 12 – 5:  Open-Loop controller behavior PIDTAU and**
**PITAU with DTIME= $\tau$**

The controller behavior for a closed loop is shown in Figure 12- 6
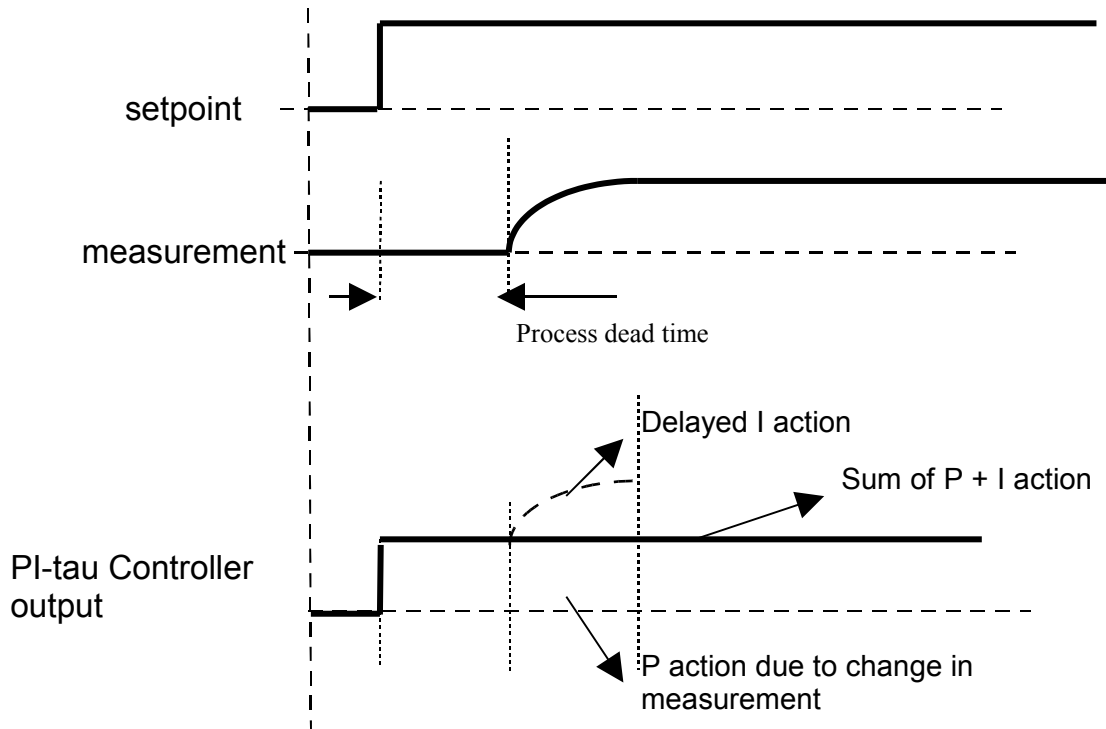


**Figure 12 – 6**:  **Closed loop behavior of PITAU controller**
**(DTIME PITAU = process deadtime)**

Here the most ideal controller tuning for PITAU is shown; the dead time of the process is set to 1 minute and therefore also the DTIM of the controller is set to 1 minute (dead ti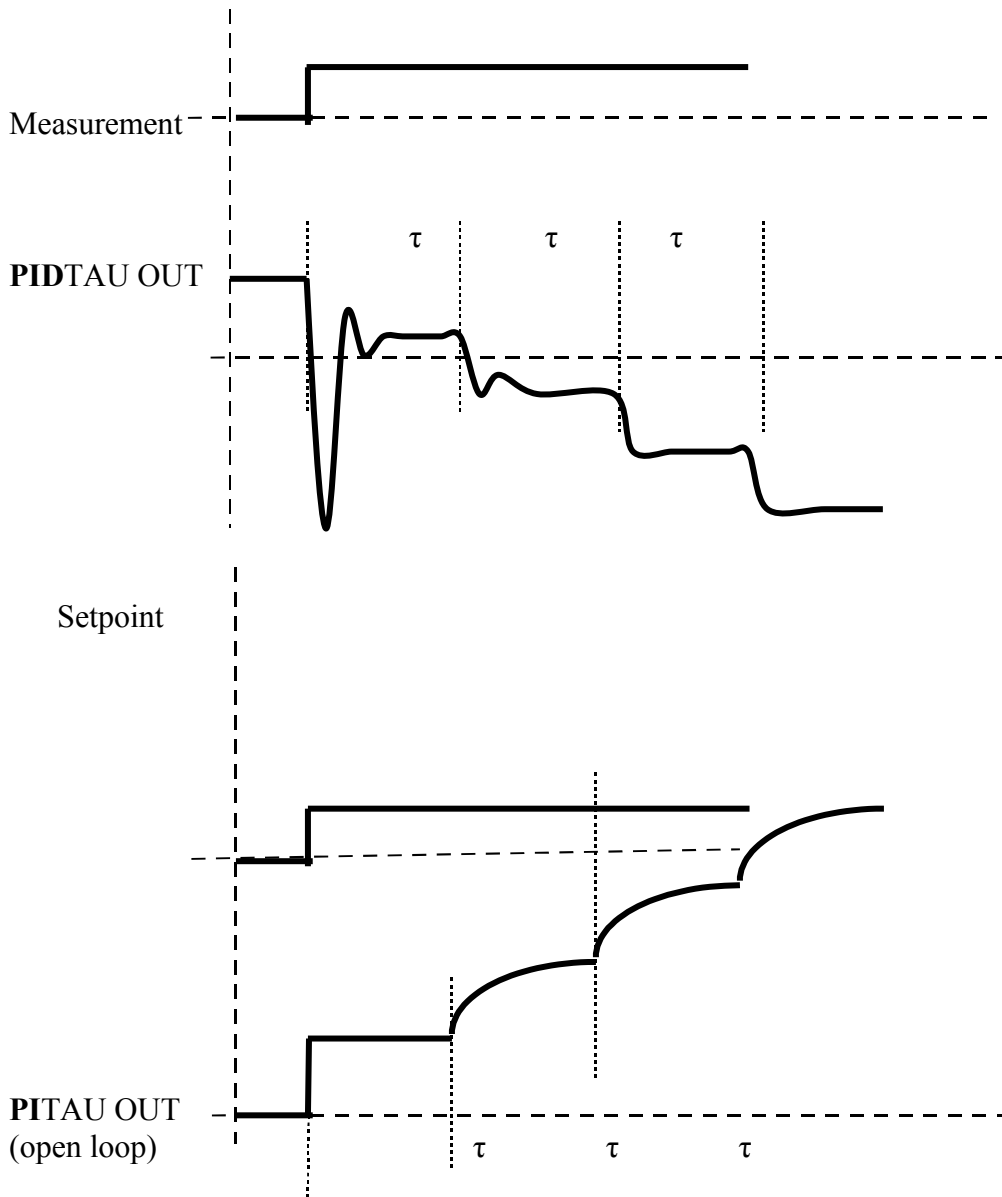me in feedback signal equal to dead time of process). The first order constant of the process was set to 0.2 minutes and the controller reset time [Tr] is made equal to the process first order time constant. In this manner the integral action that was seen in the open loop situation will exactly be reduced to zero as a result of the changes in the measurement in the closed loop situation (which results in the exact opposite action of the proportional factor of the controller).
For a stepchange in the setpoint the output signal does not change in a step ( as was the case for a change in measurement); here the effect of the filter in the setpoint becomes clear.

The function of the filter constant [ Tf] (which was set equal to 0 in above trends) is still to be discussed. Through experiments it has been found that the filter is active on setpoint- and on measurement-changes; this is evident from the function diagram in figure 12- 7
The filter [ 1/(1+sTf) ] is included in the scheme after the calculation of the ERROR (= STP-MEAS)

## 13  PIDA NON Linear option

The PIDA block offers the option to configure a Non-linear controller output behavior.
When this option is enabled **(NONLOP=1)**, it is possible to define a zone  around the ERROR=0
(thus Setpoint – Measurement =0) in which zone (between **LZONE** and **HZONE**)

the Gain  of the controller ( $\dfrac{100}{PB}$ ) is multiplied with a configurable factor (**KZONE**).

This is for instance applicable where it is a requirement to have very limited action on the control valve as long as
the Measurement is not far from the Setpoint and more action when the Measurement is far from Setpoint.

The upper zone and lower zone around the ERROR=0 point does not have to be centered. Upper zone and lower
zone can be configured individually and thus different.

The Non-linear action can also be seen as a controller for which two different proportional bands are defined,
usually a bigger PB close to ERROR =0 (less action) and a smaller PB when ERROR >> +/- 0.

In Figure 13 – 1 is shown how the controller output will be for a proportional only controller (open loop).



**CONTROLLER  PROPORTIONAL ACTION**

LZONE – **15** %

HZONE  + **8%**          **ERROR**
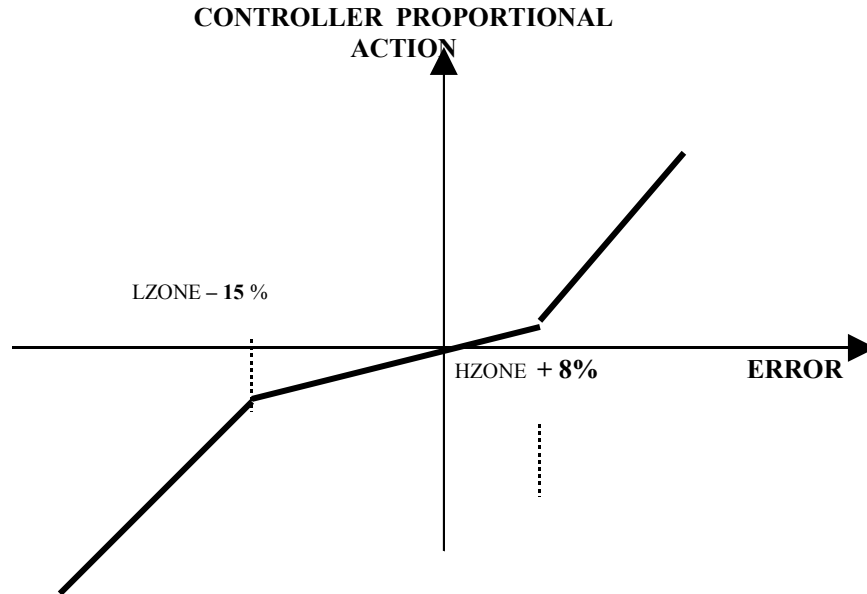
**Figure 13 – 1: NON LINEAIR controller output (proportional part only)**
**for P(ID)A controller with:**
Proportional Band = 100
Non-linear factor  = 6 (approximately)
Lower zone non-linear domain = - 15%
Upper zone non-linear domain = + 8 %

invensys
Process Systems

ISO 9001 en VCA**
DNV
GECERTIFICEERD
BEDRIJF

**14 Setpoint Ramp Option**

The PIDA block includes a SETPOINT RAMP Option. It allow the user to define or the rate (ie dgrC/min) or to
define the time to reach a predefined end setpoint.
The SETPOINT RAMP function only works when the controller is on LOCAL setpoint control.
The following parameters are in use:
SPROPT: Setpoint Ramp Option, can have values 1  to 4 (for explanation see below)
SPRATE: Setpoint rate parameter, defines or the rate of change (ie dgrC/min) of the time to reach the end value
SPTARG: Setpoint Target, is the end value that should be reached.

For SPROPT = 1 or 2 the SPRATE determines the ramp rate,
For SPROPT = 3 or 4 the SPRATE determines the time to reach the SPTARG value

For SPROPT = 2 or 4 the ramp action stops when Deviation Alarm (Error >>>); this does not happen when
SPROPT is = 1 or 3.

When the operator changes the setpoint during ramp action, the ramp functions is stopped and will not continue.

When SPTARG is changed during a ramp action, the block continue to ramp at the initial ramp rate (calculated on
the basis of the previous SPRATE time).

## 15 PIDE block

This PID controller includes an Exact calculation function which can automatically self-tune the controller tuning constants. The PIDE always functions as a PID controller (there are no MODOPT's), except when the parameter DFCT=0; in such case the derivative action is suppressed.

The PIDE block, when Block in Manual and the EXACT tune algorithm in Manual-Tune state (MNT), will manipulate the output of the controller with a step change of size = BMP (BMP parameter must be defined by user: it is the stepchange as a % of the total output signal range): this step change will come when operator makes a pre-tune request. Based on the resulting process response, the pre-tune calculations will make best estimates for the Reference values of Proportional Band, Integral time and Derivative time of the controller.

Before entering the self tune mode of the controller, the user must define the change limit parameter (CLM); this parameter will keep the tuning parameters calculated by the self-tune calculations within the specified band around the reference values for PB, Tr and DERV.

The results of the self-tune calculations is dependent on the value of the user defined parameters OVR (=overshoot limit) and DMP (= maximum allowed damping). These two parameters do interact with eachother. Test show the following results:
OVR = 1 and  DMP = 1:   PB = X, resulting in big oscillations in controller response.
OVR = 1 and  DMP = 0.1: PB = 4*X, less oscillations, better control, a little to slow.
OVR = 0.3 and DMP = 1:  PB = 2*X, less oscilations, better control

Test show that the self-tune function of the PIDE function well for first-order processes with (or without) deadtime. Results are not so good for third order processes (three lead/lag blocks in sequence).

The PIDE controller has difficulties to cope with disturbances that enter the process at the end (point B), as shown in below drawing.
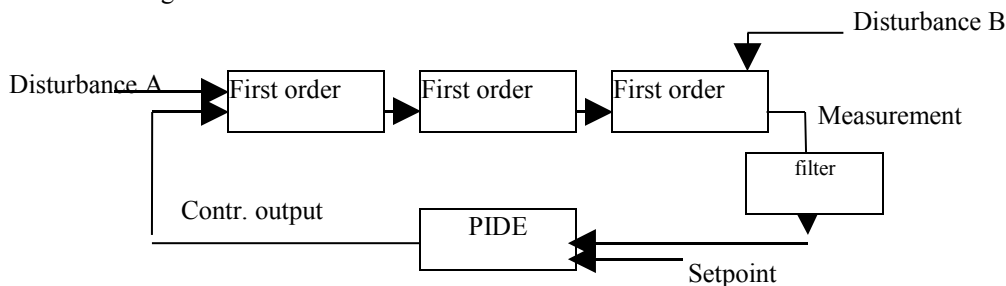


The major problem with the disturbances entering at point B are caused by the fast change in the error seen by the controller. This can be solved by adding a first or second order (butterworth) filter in the measurement before it enters the PIDE block (see dotted box).

## 16  PIDA with Feedback Tuning (FBTUNE)

The PIDA block can be linked to a Feedback Tuning block FBTUNE ( through linking the parameter PIDBLK of the FBTUNE block to the PIDA.BLKSTA parameter ). This link can be removed (or installed) without disturbing the function of  the PIDA block.

```
┌─────────────────────┐
│        PIDA         │
│       BLKSTA        │
└──────────┬──────────┘
           │
┌──────────┴──────────┐
│       PIDBLK        │
│       FBTUNE        │
└─────────────────────┘
```
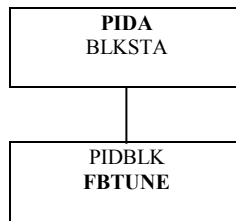
The function of the combination of PIDA and FBTUNE is equal to the function of the PIDE, however the combination PIDA/FBTUNE does have all extra functions offered by the PIDA block (including the different MODOPT options). A schematic representation is shown in figure 16 - 1



**Figure 16 – 1 :  Schematic of PIDA block in combination with FBTUNE**

The FBTUNE block allows for a so called Pre-tune function and for the so called self-tuning function.
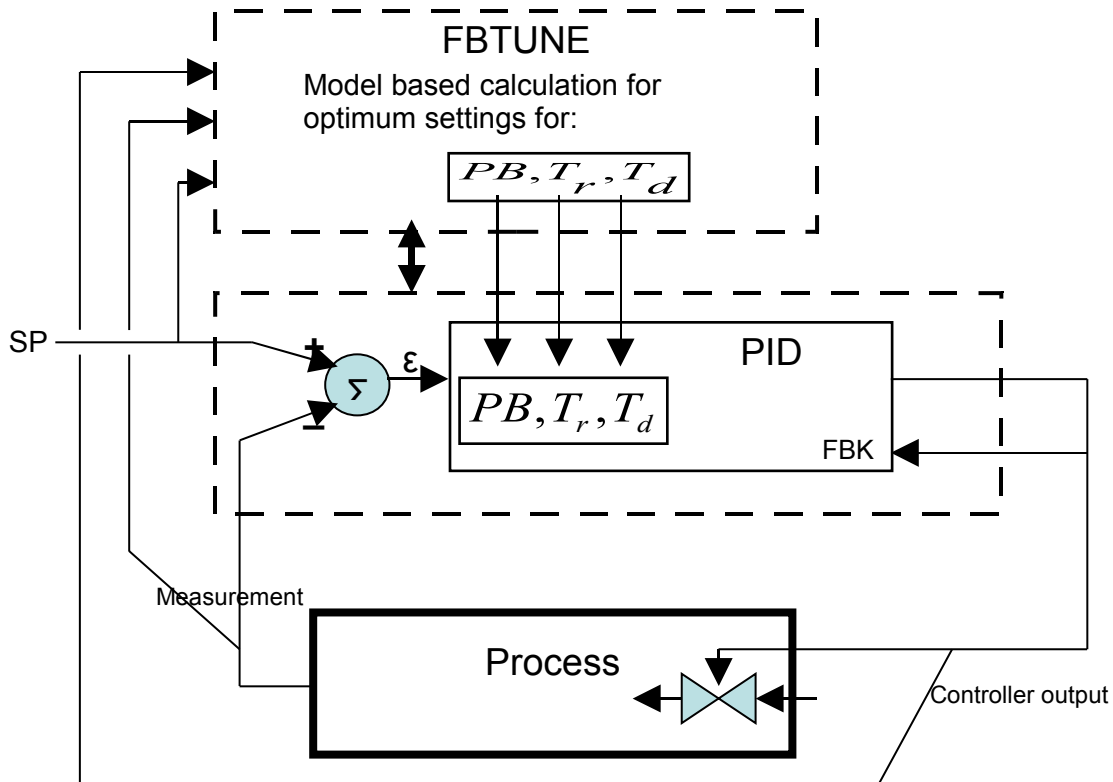The Pre-tune function works only when the PIDA block is in MANUAL. After the pre-tune request is set
(PRNREQ=1) the Pre-tune function will put a positive and a negative stepchange (stepchange size is determined by
the parameter FBTUNE.BMP{bump}) on the output of the controller and it will study the resulting response of the
measurement. Based on this study, the Pre-tune block will determine the required settings of Proportional Band
(PB), Integral time (INT), derivative time (DERIV), Kd (derivative gain) and SPLLAG. For PITAU and PIDTAU
also the parameters DTIME (delay time) and DEVTIME (delay time in deviation alarm) will be set.

After the pre-tuning is finished, the controller can be put in automatic. It is advised to let the controller in auto for
some period and inspect the behavior of the controller (tests done show that the pre-tune settings are not optimal for
all types of processes {see below}).

The self-tuning function can be activated (set parameter STNREQ=1: note: if STNREQ is already =1, the PIDA
block will automatically go from MAN to AUTO after successful Pré-tune operation). The self-tune functions
makes a study of the controller behavior and after 4 peaks, it will automatically optimize the settings of the
controller tuning parameters.

**More details on FBTUNE functions and related parameters.**

The following parameters have an important role in the manner in which a the FBTUNE block determines the
values for the related PIDA tuning parameters.

FBTUNE.STNREQ = Self Tune Request
FBTUNE.PTNREQ = Pré-Tune Request
FBTUNE.STHREQ = Self Tune HOLD Request

FBTUNE.BPM = BUMP or pulse height as a % of the PIDA output span, to be used during pré-tune function.
FBTUNE.THRESH = THRESHOLD as a % of the PIDA.MEAS signal,
      THRESHOLD determines the length of the pulse during pré-tune; when PIDA.MEAS has
      reached a value > THRESH, the pulse on PIDA.OUT is ended,
      THRESHOLD also determines if pré-tune can calculate the tuning parameters successfully:
      therefore PIDA.MEAS should become > 2.5* THRESH
Note: for very low gain process, it may be required to increase BPM and to decrease THRESH in order to get a
successful pré-tune.

FBTUNE.DFCT = Derivative Factor: this parameter determines how the pré-tune function will calculate the tuning
parameters for the PIDA block. Pré-tune will multiply the calculated DERIV parameter with DFCT before loading
the value to PIDA. This means:
For DFCT = 0: the derivative term will be eliminated, thus only PI control
For DFCT = 1, pré-tune will calculate optimal settings for tuning parameters, assuming a LAG-DELAY process
For DFCT > 1, pré-tune will assume a DELAY-LAG-LAG process; for process requiring a strong derivative
    action, the value of DFCT could be increased even to 4.

Pré-tune will calculate values for the following PIDA parameters:
   PIDA.PBAND
   PIDA.INT
   PIDA.DERIV

PIDA.SPLLAG
PIDA.KD
PIDA.DTIME (for PITAU and PIDTAU only)
PIDA.DEVTIEME (only for PITAU and PIDTAU; this is a delay time for deviation alarm)

Also pré-tune will calculate values for FBTUNE block:
FBTUNE.PBMAX
FBTUNE.PBMIN
FBTUNE.INTMAX
FBTUNE.INTMIN
FBTUNE.PR_TYP (see below)
FBTUNE.PR_FL (see below)
FBTUNE.DFCT (will overwrite this value)

FBTUNE.TYP: this parameter determines the expected process type;
- for a pure delay process, PR_TYP should be = -4
- for a process with delay = lag, PR_TYP should be = 0
- if there is a secondary lag (LAG-LAG process), PR_TYP should be between 0.3 and 1
- (PR_TYP > 1 only for process with negative lag (open loop instable))
FBTUNE.TYP is automatically updated by pré-tune and it will be adjusted by self-tune, provided PR_FL=0 and DFCT < 1.

FBTUNE.PR_FL: Process Factor, controls the type of self-tuning adaptation
- for PR_FL = 0 self-tune will use fussy interpolation method for PI and PID and will update the value of of PR_TYP
- PR_FL = 1: this will invoke algebraic tuning method, based on the given values of PR_TYP and DFACT
FBTUNE.PR_FL is automatically set to = 0 when DFCT > 1 and MODOPT >5 (PID)
- PR_FL = 2: this will suspend updating of tuning parameters, but allows the appropriate stored tuning set to be activated at the start of each isolated response, based on value of PROG parameter (see below) and response direction.

FBTUNE.OVR: overshoot parameter: this is a target value for the (absolute) ratio between second and first error peak; OVR can have values between 0 and 0.2. FBTUNE will use this parameter for its calculation of optimum tuning parameters for PIDA.

The FBTUNE block has another special function to cope with process (or control elements) non-linearity. FBTUNE can provide 6 different GAIN sets for non-linear behavior of the process; 3 sets are used for increasing signals and 3 sets are used for decreasing signals. The relevant signal must be linked to the FBTUNE.PROG parameter (in engineering units of connected variable). The range is sub-divided in 3 sub-ranges by the parameters FBTUNE.PROGLT and PROGUT (lower threshold and upper threshold).

The advantage of the PIDA with FBTUNE combination over the PIDE block is that PIDA solution offers the possibility to determine what control actions are being used by the PIDA; this is done by setting the MODOPT parameter (the PIDE block is always a PID controller).

A second advantage of the PIDA block is that in PIDA the value of Kd (derivative multiplication factor) can be made smaller than 10 (in PIDE the value is >10).

Finally, the PIDA block does have a filter in the measurement before the (controller) error is calculated: this filter will eliminate the problems described with the derivative action, as were seen in the PIDE block

**Results with FBTUNE:**

FBTUNE pré-tune function: Test with a simulated process show that the pré-tune function of the FBTUNE block do give reasonable results for first order processes with or without dead time.
However for second or third order processes (2 or 3 lead/lag blocks) the FBTUNE pre-tune function gives results that create an instable control behavior; the calculated PBAND value is in all cases (different MODOPT values were tested) far too low, resulting in far too much controller action.

FBTUNE self-tune function: Test show that the self-tune function calculates acceptable controller tuning parameters, in particular for first order processes with (or without) dead time. For second- or third-order processes, the self-tuning function will come to the conclusion that the calculated PBAND factor by the pre-tuner is too low and it will adjust this parameter after 4 peaks in the correct direction. Still the self-tuner will give PB factors that are higher than  would normally be used for such process (for a process with gain = 1 a PB would normally be between 80 and 120, the selftuner comes with PB factors of about 50).

The above test have been done with different settings of MODOPT (4=PI, 5=PID, 6=PID non-interactive and 7= PITAU). The above described results were more or less identical for all MODOPT's.

Conclusion is that for second- en third order processes the pre-tune function of the FBTUNE block gives a PBAND value which is too low. It is recommended to first determine what the process gain is and based on this gain, to estimate what PBAND value should be (process gain * PBAND/100 should be between 0.8 and 1.2.  Before putting a controller in AUTO after pre-tune, the PBAND may have to be changed to a greater value, in order to prevent the risk for an instable process.

Results of above mentioned test can be found in process graphic trends, figures 16- 1 until  16 -2
The two process simulations used were:
1-  first order process with dead time (first order constant 0.3 minutes and deadtime 0.3 minutes)
2-  third order process (with  three firstorder Lead/lag blocks in sequence, each with first order time constant of 0.2 minutes)

## 17 Feed forward Control

Very often, feed forward control is considered to be difficult, as process dynamics are not known exactly. However this is not completely true: more practical manners are available to determine what the required feed forward actions should be.

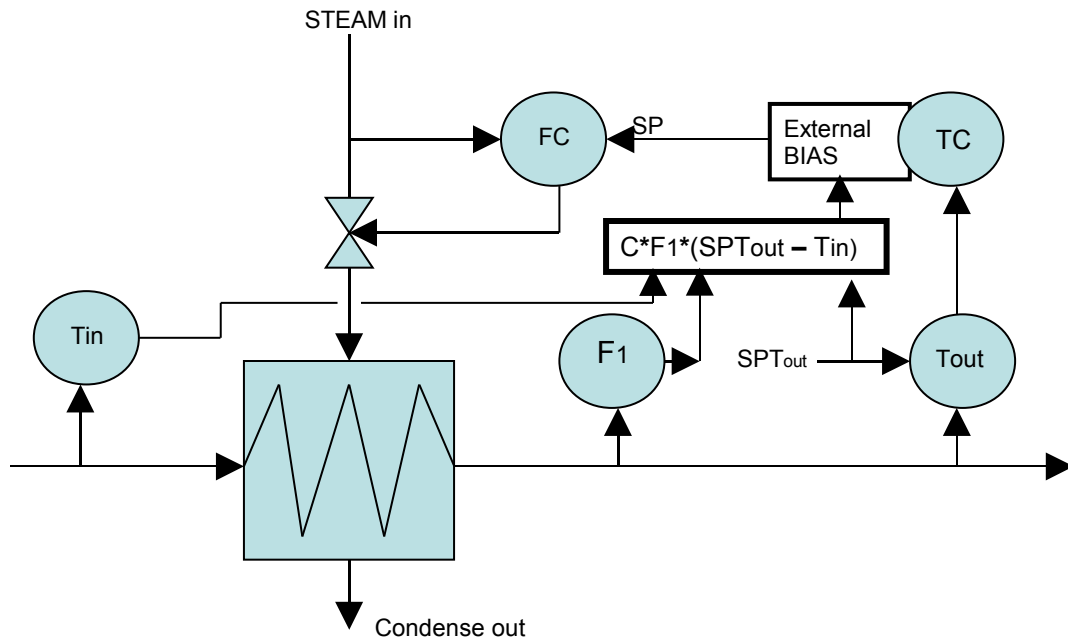We take a simple heat exchanger as an example (see figure 17 – 1)



**Fig. 17 – 1 : Example of feedforward control scheme**

For the process shown in figure 17-1 the following heat/mass balances are valid:

The medium to be heated will absorb the following energy per time unit:

$$F_1 * sh * (T_{out} - T_{in})$$

where sh = specific heat of the medium

The steam used to heat the medium is supposed saturated, which means that the steam will give all its condensation heat in the exchanger:

$$F_{St} * ch$$

where ch is condensation heat of the steam

This results in the following equation:

$$F_1 * sh * (T_{out} - T_{in}) = F_{St} * ch$$

or

$$F_{St} = \frac{sh}{ch} * F_1 * (T_{out} - T_{in})$$

In order to do accurate feed forward control, the value of the factor $\dfrac{sh}{ch}$ must be known.

The complete mathematical process description, including the determination of the factor $\dfrac{sh}{ch}$ is for this process very easy, still there is an even more simple manner to calculate the feed forward component.

Assume the same heat exchanger and consider it to be a black box..

It is fairly simple to run this exchanger and record the values for F-steam, F-1 and T-in and T-out.
Logical thinking will make clear that there will be a relationship (most likely a linear relationship) between F-1 and F-st. Also it may be clear that there will be a relation between required F-st and the requested delta T (T-out – T-in).
Based on this logical thinking one can formulate the following relationship:

$$F_{st} = C * F_1 * deltaT$$

where C is the unknown parameter

Assume the following running conditions:

|         | Run1 | Run2 | Run3 |
|---------|------|------|------|
| F-1     | 1000 | 2000 | 1000 |
| Delta T | 50   | 50   | 70   |
| F-st    | 20   | 40   | 28   |

From the values found during run 1 the value of C can be determined : $C = \dfrac{F_{st}}{F_1 * deltaT}$ ,

Thus    C = 0,0004.

The value found during run 2 and 3 can be used to check if the logically found formula for  the description of the process is correct.

After this Feed Forward loop has been installed, it is relatively simple to check if the estimated value for C (=0,004) is correct.  In case it is correct, than the output of the feed forward calculation (setpoint for steam flow controller) should on average be equal to the signal to the steam flow controller; the temperature controller function will be only a (feedback control) function to correct the error in the feed forward signal. So, as long as the average signal from the temperature controller is zero (and thus,  the setpoint to the steam flow controller is equal to the calculated feed forward signal) the value of C is correct. This method can be used to fine-tune the value of C.

In more complex processes, the above may not be so easy, however when an operator is asked what he will do when a certain disturbance is introduced, one usually will get a pretty good idea about the effect of a disturbance on the output of the process. For small process disturbances this relationship may be linearized.

In the above example, there may be an effect of ambient temperature on the amount of steam that is required, as ambient temperature will have an effect on heat loss.  Operators usually do know quite well what the effect is of a sudden change in ambient temperature. For instance they are able to tell you that when temperature changes suddenly, for instance 5 degrees, due to the start of a rain period, that  they can prevent changes in temperature of the temperature controlled flow, by increasing the steam flow by, let say 10 kg/hrs. This assumes a sensitivity of 2 kg/hr steam on 1 degree. The calculation of the feed forward signal can be adjusted in the following manner

$$F_{St} = 0,004 * F_1 * (T_{out} - T_{in}) + 2 * (40 - T_{amb})$$

where the 40 is an arbitrarily chosen value for the standard ambient temperature (in fact, this formulae assumes zero heatlosses at 40 degr).

Pls. note that in the above formulae the heatloss part ( $2 * (40 - T_{amb})$ ) is independent from $F_1$ and independent from Delta T. This seems a logical conclusion.

When this calculation is used for the first time, it may be a good idea to start with a value somewhat smaller that 2 and investigate how the controller behaves during a sudden ambient temperature dip. If the temperature of the temperature controlled flow reacts with also a dip, it is clear that the factor is too smal, however when a temperature dip is followed by an increase in controlled temperature the value is too high. In this manner it is possible to fine tune the value for sensitivity for ambient temperature.

## 18  PIDA with Feed Forward Tuning (FFTUNE)

The feed forward tuning block can be linked to a PIDA block in an identical manner as this was shown for BFTUNE. The link between FFTUNE and PIDA is made through linking the parameter PIDA.BLKSTA to the FFTUNE.PIDBLK parameter. This can be done (and undone) without disrupting the function of the PIDA block. For a schematic drawing of the PIDA in combination with FFTUNE, see figure 18 – 1.
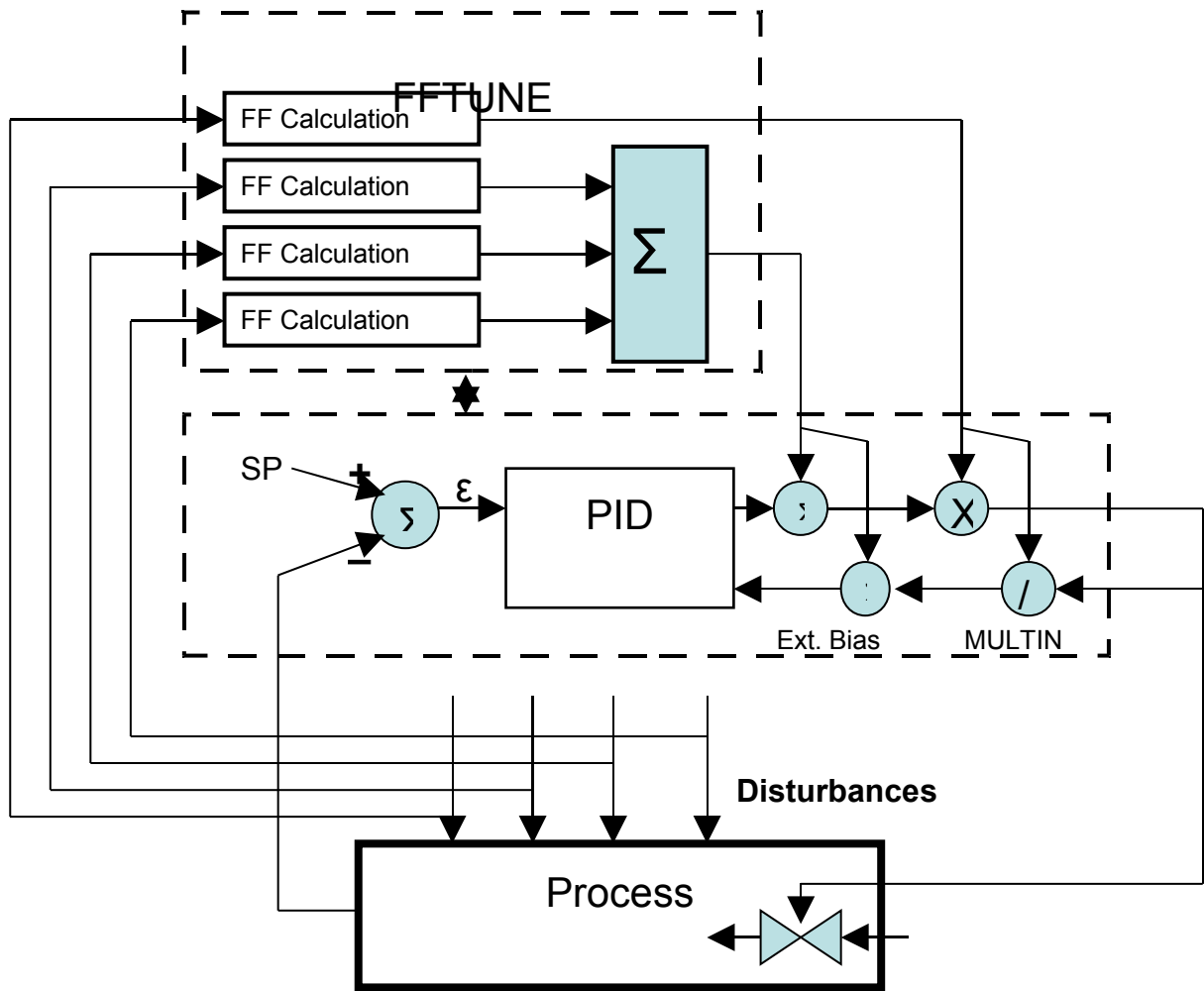


**Figure 18 – 1 : Schematic of PIDA controller with FFTUNE**

FFTUNE can monitor up to 4 loads ( determined by the values linked to the FFTUNE.LOAD1 to LOAD4 parameters) to the process and take corrective actions when a disturbance in this load is detected. Only one of these LOAD's can result in a multiplicative compensation (the others will have a BIAS function). If MULTIN or BIAS of the associated PIDA block is already linked to an other variable, the LOAD4 will be ignored.

FFTUNE has a Tune Request state (FTNREQ = 1); in this state the FFTUNE block monitors the connected load's and will determine the effect of the load on the controlled variable (this is PIDA.MEAS). Based on these test results, the FFTUNE will calculate the required Feed Forward Compensation for each individual LOAD disturbance.
When FFTUNE.FTNREQ is set to = 0 self tuning and feed forward compensation is suspended and the stored tuning parameters are erased.

With FTNREQ=0 the PID(A) controller will indicate that the status of the FFTUNE is    OFF
With FTNREQ=1 and a reasonable stable process, the FFTUNE status will start in state  QUIET
Now, when a bump is introduced in one of the disturbance signals, the status goes to      MEAS
If after the bump in the disturbance, the measurement moves more that THRESHOLD,
                                                                                the status goes to      SIGNIF
After the process is returning to setpoint, one can take the disturbance away and
                                                                        the status will stay in state      SIGNIF
Now one should wait until the FFTUNE state will go to                                                  QUIET

When status goes from SIGNIF to QUIET, the FFTUNE block will update the Feed Forward parameters and will act like a FF controller for a new disturbance.
If status goes from SIGNIF to UNMEAS, the FFTUNE block will **not** update the FF tuning parameters.

The FFTUNE.FTHREQ = Feed Forward Tune HOLD Request; when set to 1 the tuning is suspended, however the active tuning set remains active (so feed forward compensation is still active), using the stored tuning constants.

FFTUNE.THRESH = Threshold parameter; is the required absolute error (as a % of the full span) in order to discriminate a significant disturbance. If error > THRESH, the FFTUNE block will calculate new feed forward compensation factors.

Just like the FBTUNE block, the FFTUNE block has the ability to cope with non linear process behavior: sets of compensation factors can be stored by FFTUNE for 6 different sub ranges (3 up and 3 down) of the parameter that is linked to the FFTUNE.PROG parameter. The sub ranges are determined by the parameters FFTUNE.PROGUT and PROGLT (upper and lower threshold). FFTUNE.PROG must be linked to the parameter that best correlates with the non-linear behavior of the process (this is usually the setpoint of the PIDA or one of the LOAD parameters).