TECHNISCHE HOCHSCHULE LÜBECK

Control Systems II

Complete Paper V 3

For ISE/MSOE

by

Prof. Dr. Bayerlein

SS 2020

Complete notes of my lecture
Includes Workbook II
Capt7 + 8 + 9
V1.6 PFC for IT1- processes
V1.7 DB Bay + OR with any delay, DB bay n=3
V1.8 RegC# examples, PDT1 trapezoidal
V2.0 All Regdelph replaced with RegC#, 3+4PT1 HF(z)
V2.1 small change in PFC- Theory 9.7.5
V2.2 smith predictor + small debug +PFC as PI + Dist Comp
V2.3 only 9.13 Dist Comp 2PT1 corrected + IT1 added
V3.0 Version SS2020

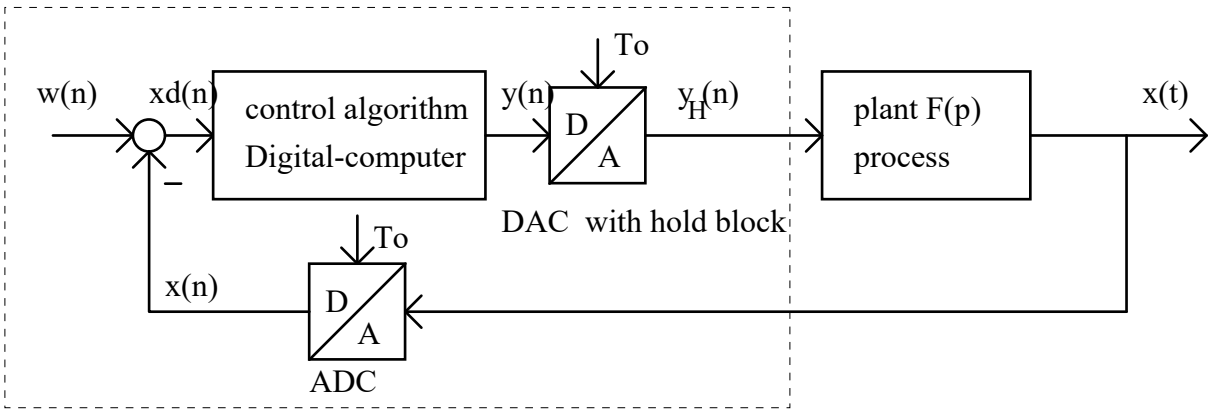
Content

6 Digital PID Controllers

6.1 Structure of System

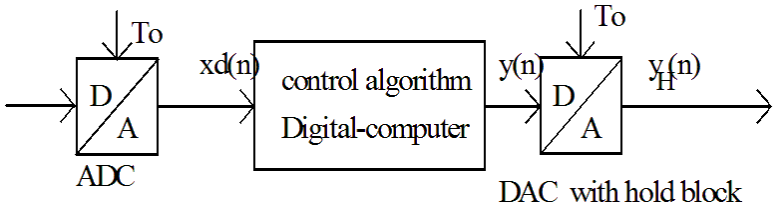
6.1.1 Overview

A digital Filter or a digital controller mainly contains an analogue to digital converter (ADC), a processor (e.g. a microprocessor, microcontroller, PC, PLC, DSP or similar) and a digital to analogue converter (DAC).



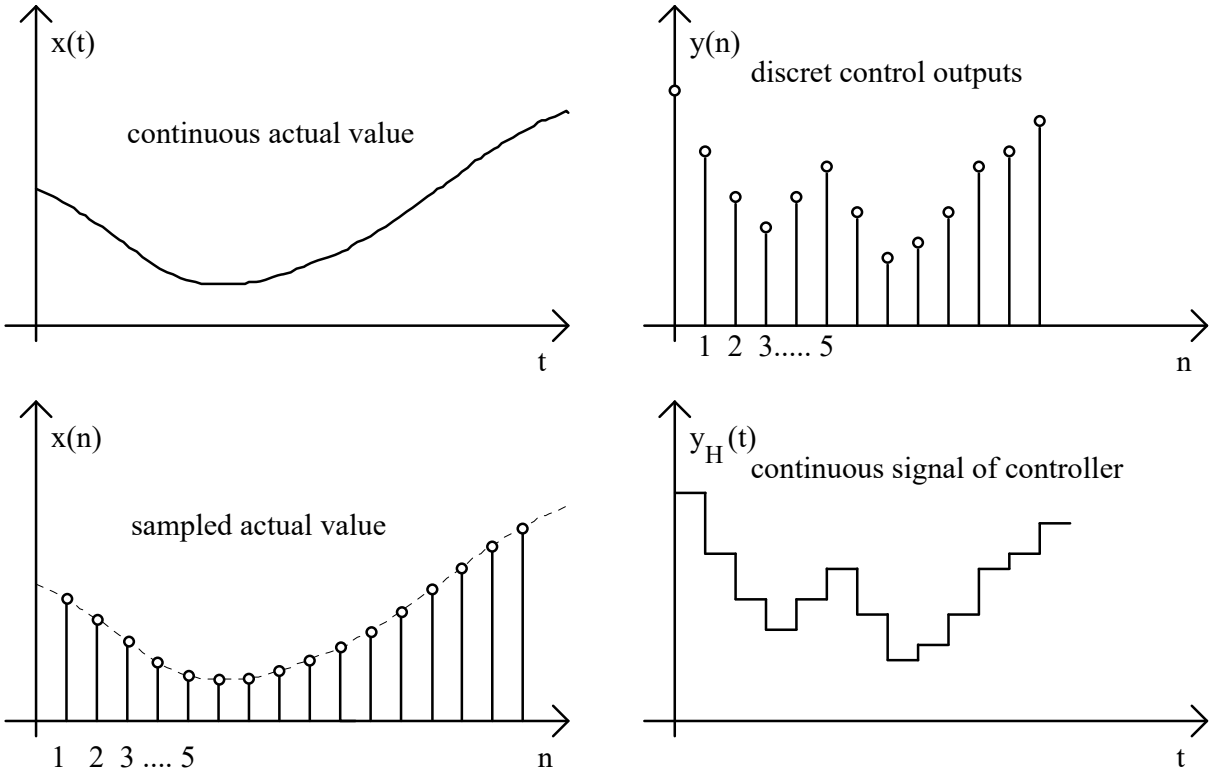
digital controller

There is no difference between a digital filter and a digital controller, only the use defines the name. A controller is nothing other than a filter used to control physical values. The structure of a digital controller containing negative feedback is displayed in the picture above.
A digital Filter samples the input voltage x(t), each new x(n) – value is sent to the filter algorithm, and the new output value y(n) is reconverted by DAC into a voltage $y_H(t)$.



Digital filter

We talk about sampling systems, which only samples an input signal each T_0 and calculates only then a new output value y(n). Between two samples there is no reaction on a change of the input signal. Typically, we can see that the output voltage contains steps and constant sections like stairs, see next diagram.



6.2 Introduction

First there will be a discussion about some advantages and disadvantages of digital controllers.

Advantages	Disadvantages
• Easy change of parameters, which are contents of variables • Good human interface possible • Multitasking possible (several controllers with the same hardware). • More complex controllers are possible like dead-beat-controllers, fuzzy controllers, adaptive controllers PFC etc.	• Development is more expensive • Additional delay time caused by calculation, conversion, S&H • Problems with time discretion, Shannon theorem • Problems with amplitude discretion, quantization errors

The choice of the sampling time T_o has to be done carefully. Some design hints are given below in a separate chapter. First the classical P, PI, PDT1 and PIDT1- Controller algorithms will be designed without Z- transformation using a method called "conversion by replacing differential equation with difference equation".

6.2.1 P- Control algorithm

An analogue P- controller does nothing other than multiplying input voltage $x_d(t)$ with the controller gain K_c. The (in this case trivial) differential equation is:

$y(t) = K_c * x_d(t)$.

Now the continuous time t is replaced by sampled time nT_o. n is an integer number of the sample. n is the actual sample, n-1 the previous, n-2 the sample before the previous and so on. The writing will be the following: y(t) will be replaced by $y(nT_o) = y(n)$. The new algorithm now can be written as:

$y(n) = K_c * x_d(n)$.

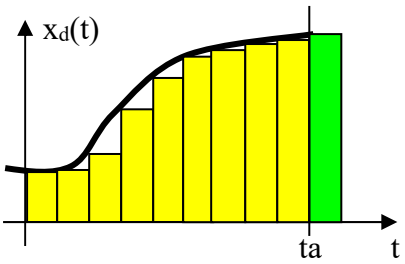
Read: The actual output value y(n) is calculated by multiplying the controller gain value K_c with the actual measured input value $x_d(n)$ each T_o.
The processor must be able to multiply floating point values (K_c is normally a floating point value), which normally is not supported with assembler language and some cheap C-compilers. Then the range of values must be controlled by your program. The algorithm must be called in equidistant time intervals with the distance T_o. This must also be supported by your processor either using a real time operating system (RTOS) like OS-9 or similar or realising the constant ticks via other methods (timer, interrupts etc).

6.2.2 I- control algorithm

Now we will try to convert an integrator into a digital I- algorithm. This will then be used in a PI or PIDT1- controller algorithm. We start with an analogue integrator with transfer function $F(p) = K_I/p$. The gain (in this case the unity gain radian frequency) K_I defines the integration time constant $T_I = 1/ K_I$. The differential equation of an integrator is

$$y(t) = K_I * \int_0^{ta} x_d(t)dt ,$$

where *ta* is the actual time. The integration starts at time t=0.
Now the replacement of the integration is done with the sum of rectangles. The integral describes the area under the curve x_d. The following picture describes this situation:



The curve ends at point $ta = nT_o$. The last green rectangle has the amplitude $x_d(n)$, the preceding rectangle $x_d(n-1)$ and so on. Each rectangle has the width T_o and the amplitude xd(i), if i is the time $i*T_0$. The area under the curve from 0 to *ta* can now be approximated with the sum

$\sum_{i=0}^{n-1} x_d(i) * T_o$. This sum ends with the last yellow rectangle; however, you can see that the actual area is larger than the yellow area. So another version adds the green rectangle to the sum, but then the area seems a little bit too large. This version has the approximation

$\sum_{i=0}^{n} x_d(i) * T_o$.

So we get the following two versions of I- algorithms:

Without green rec (Prof. Baumann)	With green rec (Prof. Bayerlein)
$y(n) = K_I * \sum_{i=0}^{n-1} x_d(i) * T_o$	$y(n) = K_I * \sum_{i=0}^{n} x_d(i) * T_o$

Both versions cannot be programmed because the sum of old $x_d(i)$ has to be calculated in each step each T_o. So any time an integrator is in the filter, the following step to get a **recursive form of algorithm** is necessary:
Trick to get recursive form of an algorithm.

$y(n) = K_I * \sum_{i=0}^{n-1} x_d(i) * T_o$ (equation 1)

$y(n-1) = K_I * \sum_{i=0}^{n-2} x_d(i) * T_o$ (equation 2, situation one step before, replaces n with n-1)

Now calculate the difference equ1 – equ2:

$$y(n) - y(n-1) = K_I * \sum_{i=0}^{n-1} x_d(i) * T_o - K_I * \sum_{i=0}^{n-2} x_d(i) * T_o .$$

The difference of the right sums is simply only the term with i=n-1. So the final result solved to y(n) is now without the unprogrammable large sum :

$y(n) = y(n-1) + x_d(n-1) * K_I * T_o$. (Baumann- version)

In the second version including the green rectangle we get the following result:

$y(n) = K_I * \sum_{i=0}^{n} x_d(i) * T_o$ (equation 1)

$y(n-1) = K_I * \sum_{i=0}^{n-1} x_d(i) * T_o$ (equation 2, situation one step before, replaces n with n-1)

Now calculate the difference equ1 – equ2:

$$y(n) - y(n-1) = K_I * \sum_{i=0}^{n} x_d(i) * T_o - K_I * \sum_{i=0}^{n-1} x_d(i) * T_o .$$

The difference of the right sums is simply only the term with i=n. So the final result solved for y(n) is:

$y(n) = y(n-1) + x_d(n) * K_I * T_o$. (Bayerlein- version).

Read this algorithm: Each T_o the new output value y(n) of the integrator is calculated as a sum of the previous output y(n-1) and the actual measured input value $x_d(n)$ is multiplied with Filter coefficient, here K_I*T_o. Compared with the P- algorithm you need an additional variable to store old or previous outputs and the addition.
Regarding the filter coefficients, now commonly the character q is used if you multiply them with input signals. In recursive algorithms output signals or previous output signals are used. If on the right side no output appears, we call the algorithm "nonrecursive". Sometimes the previous output signals are also multiplied with coefficients. Then for these we use coefficients the character p. So the general filter or control algorithm with these coefficients looks like:

$$y(n) = p_1 y(n-1) + p_2 y(n-2) + ... + q_0 x_d(n) + q_1 x_d(n-1) + q_2 x_d(n-2) + ...$$

Different filters are described with different p and q- values. The structure of algorithm is always the same. In I – case we have $p_1=1$ and $q_0=K_I*T_0$, all other p and q are zero.

6.2.3 PI- controller algorithm

Now we combine the P with the I algorithm. We start with the differential equation. The PI has the transfer function

$F(p) = \frac{K_c(1+pT_N)}{pT_N} = \frac{K_c}{pT_N} + K_c = \frac{Y(p)}{X_d(p)}$. This equation in the frequency domain can converted via inverse Laplace transform into the differential equation (formal way)

$$y(t) = K_c x_d(t) + \frac{K_c}{T_N}\int x_d(t)dt.$$

Now with discrete sampled times (t replaced by n) we get

$y(n) = K_c x_d(n) + \frac{K_c}{T_N}\sum_{i=0}^{n} x_d(i) * T_0$. You can see I start with the Bayerlein- version including green rectangle, the sum ends with i=n .

Because of the sum we have to go the recursive way to get an algorithm without the unprogrammable sum. The above equation one step before:

$y(n-1) = K_c x_d(n-1) + \frac{K_c}{T_N}\sum_{i=0}^{n-1} x_d(i) * T_0$. Difference of both equations and y(n-1) moved to right:

$y(n) = y(n-1) + K_c x_d(n) + \frac{K_c T_0}{T_N} x_d(n) - K_c x_d(n-1)$. The difference of the sums gives the only expression $K_c/T_N * x_d(n)$. Sorted and written with the coefficients q we get

$y(n) = y(n-1) + q_0 x_d(n) + q_1 x_d(n-1)$ with

$$q_0 = K_c(1+\frac{T_0}{T_N}) \quad and \quad q_1 = -K_c.$$

In the case of not using the green rectangle we get the following alternative:

$y(n) = K_c x_d(n) + \frac{K_c}{T_N}\sum_{i=0}^{n-1} x_d(i) * T_0$. Now you see I start with the Baumann- version without green rectangle, the sum ends with i=n-1 .

Again because of the sum we have to go the recursive way to get an algorithm without the unprogrammable sum. The above equation one step before:

$y(n-1) = K_c x_d(n-1) + \frac{K_c}{T_N}\sum_{i=0}^{n-2} x_d(i) * T_0$. Difference of both equations and y(n-1) moved to right:

$y(n) = y(n-1) + K_c x_d(n) + \frac{K_c T_0}{T_N} x_d(n-1) - K_c x_d(n-1)$. The difference of the sums gives the only expression $K_c/T_N * x_d(n-1)$. Sorted and written with the coefficients q we get

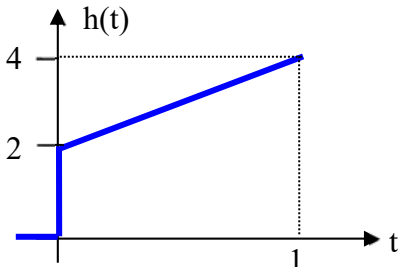
$y(n) = y(n-1) + q_0 x_d(n) + q_1 x_d(n-1)$ with

$$q_0 = K_c \quad and \quad q_1 = K_c(-1+\frac{T_0}{T_N}).$$

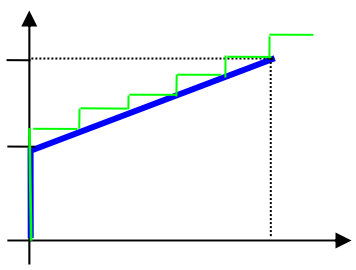
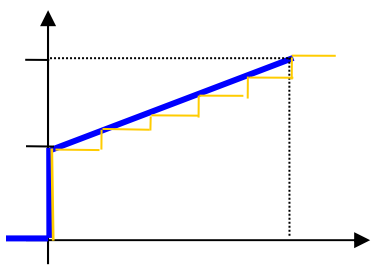
We can demonstrate this algorithm with a simple example. Let us convert the PI – controller with K_c=2 and T_N=1s working with the sampling time T_o=0.2 s.

First we compare the unit step responses and then the reference response in a loop.

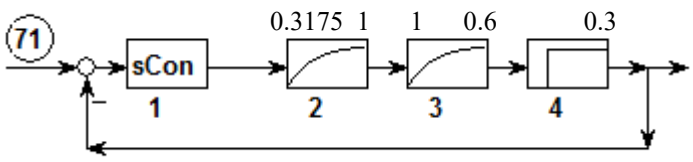
With p 16 of the workbook we easily can draw the unit step response:



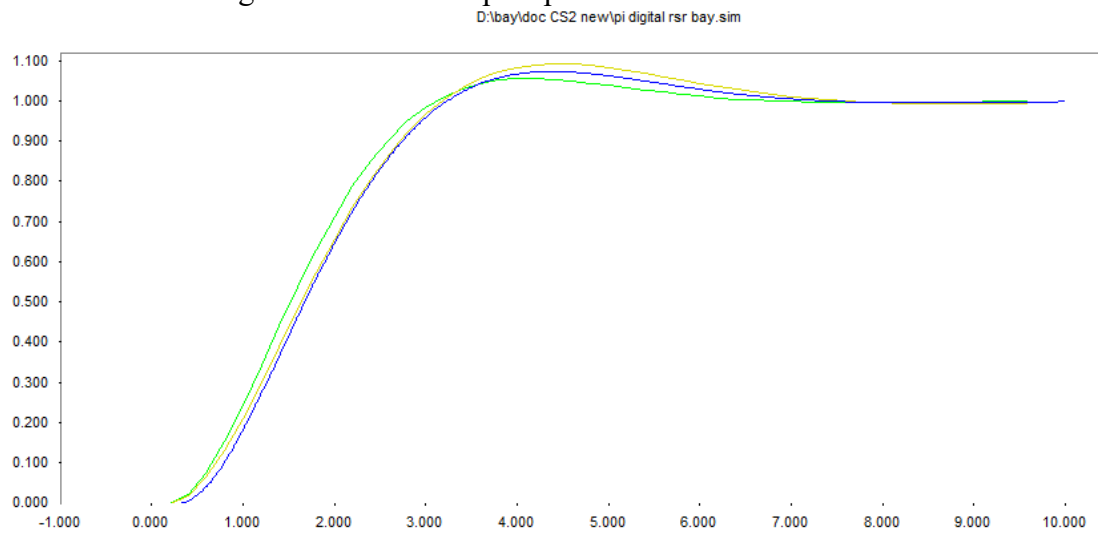
Now we calculate the unit step response of the digital PI- filter. First we have to define the sampled unit step function σ(t). For t<0, all values of σ(t) = 0 and for all t>0, the values of the σ(t)=1. But what happens at t=0? What is faster: the step or the sampler? We simply define that the step is faster than the sampler and σ(0)=1. In the following table we see the single values. The Bayerlein q- values are q_0=2.4, q_1=-2 and the Baumann – values are q_0=2 and q_1=-1.6, the algorithm is identical.

n	$x_d(n)$	$y(n)=y(n-1) + 2.4*x_d(n) – 2 * x_d(n-1)$	$y(n)=y(n-1) + 2*x_d(n) – 1.6 * x_d(n-1)$
-1	0	0	0
0	1	2.4	2
1	1	2.8	2.4
2	1	3.2	2.8
3	1	3.6	3.2
4	1	4.0	3.6
5	1	4.4	4.0
			

The left curve looks like a stair, which lies on top of the analogue blue step response and has in total too large amplitude. The right curve has a behaviour opposite of this in that it is too small, the stair touches the blue curve from the bottom side.
You can guess that a compromise could be a curve which is an average of both curves. This later is also a useful solution, in that chapter we talk about a *trapezoidal approximation* which is unrelated to both these *rectangular approximations*. There the area under the curve is replaced by the sum of trapezoidal small areas.
But the next point demonstrates that this really is not so important in a control loop application. We consider a control loop system with the following process:



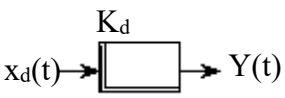
If you design a PI- controller with pole compensation and 60° phase margin, this results in exactly the previously used PI- controller with K_c=2 and T_N=1s. The following picture gives idea of the resulting unit reference step responses.



The blue curve is the analogue PI- response, the yellow curve is the response with the digital PI Baumann version, and the green curve is my preferred solution. Of course I have chosen an example, where my version is the best. But you see the differences are negligible. For all further discussions I will use the version including the green rectangle.
If T_0 changes to smaller values, the differences also become smaller. If T_0 for example is changed to T_0=0.02 s, then q_0=2.04, q_1=-2. Then one step of the ramp- stair is replaced by 10 steps with amplitude 0.04. Then the difference is so small that you can see no difference in the diagram.

6.2.4 D-algorithm

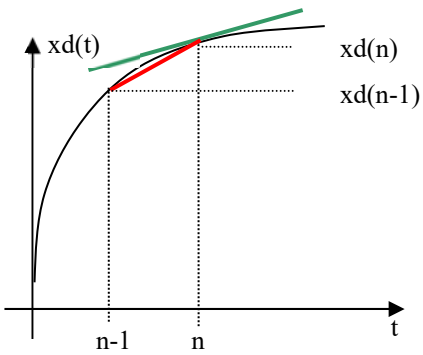
The next step is to include a differentiator. So, I will start by discussing a pure differentiator, after that the total PD, and finally the PID- algorithm.



The differential equation of a differentiator is

$$y(t) = K_d * \frac{d}{dt} x_d(t)$$

so y(t) is proportional to the gradient or slope of the x_d – curve. The next picture illustrates this:



The analog differentiator has an output proportional to the slope of the green line. If we sample the xd- curve, the ADC can only measure the values, not the changes in values. So a

good approximation is the gradient of the secant, the slope of the red line. With T_0 as the distance of two samples and the change of $\Delta x_d = x_d(n) - x_d(n-1)$ we get the approximation

$$\frac{d}{dt}x_d(t) \approx \frac{\Delta x_d}{\Delta t} = \frac{x_d(n) - x_d(n-1)}{T_0}.$$

So we finally get the algorithm of a pure differentiator:

$$y(n) = \frac{K_d}{T_0} * x_d(n) - \frac{K_d}{T_0} * x_d(n-1) = q_0 * x_d + q_1 * x_d(n-1) \text{ with } q_0 = -q_1 = \frac{K_d}{T_0}.$$

6.2.5 PD- algorithm

Now we combine a P with a D to get a PD- algorithm. We start with analog PD-block: $F(p) = K_d*(1+pT_v)$. The differential equation looks like

$$y(t) = K_d * x_d(t) + K_d T_v \frac{d}{dt} x_d(t)$$

Now the difference equation (you see the name comes from replacement of *differential* quotient by *difference* quotient):

$$y(n) = K_d * x_d(n) + \frac{K_d T_v}{T_0} * x_d(n) - \frac{K_d T_v}{T_0} * x_d(n-1) = q_0 * x_d + q_1 * x_d(n-1) \text{ with}$$

$q_0 = K_d * \left(1 + \frac{T_v}{T_0}\right)$ *and* $q_1 = -\frac{K_d T_v}{T_0}$. This is called the nonrecursive form of the PD-algorithm. The recursive form is also possible (see below).

6.2.6 PID- algorithm

Now put PD- algorithm- and the I- algorithm together to get the basic PID – algorithm: Start with transfer function of PID with stepdepth st=∞:

$$F(p) = K * \left(1 + 1/pT_I + pT_D\right) = K_R \frac{(1+pT_N)(1+pT_V)}{pT_N}$$

Left expression is the summing form; the right expression is the bode form. In Workbook p.21 you can find the conversion equations. The summing form is the best to convert into the differential equation:

$$y(t) = K * \left(x_d(t) + \frac{1}{T_I}\int x_d(t)dt + T_D \frac{d}{dt} x_d(t) \right).$$

Discrete difference equation with rectangular approach including green area and secant approximation as we have done before:

$$y(n) = K * \left(x_d(n) + \frac{T_0}{T_I}\sum_{i=0}^{n} x_d(i) + \frac{T_D}{T_0}\left(x_d(n) - x_d(n-1)\right) \right).$$

The sum forces the step into the recursive form. Above equation in the previous step:

$$y(n-1) = K * \left(x_d(n-1) + \frac{T_0}{T_I}\sum_{i=0}^{n-1} x_d(i) + \frac{T_D}{T_0}\left(x_d(n-1) - x_d(n-2)\right) \right).$$

Difference sorted to the $x_d(i)$:

$$y(n) = y(n-1) + K * \left(\left(1 + \frac{T_0}{T_I} + \frac{T_D}{T_0}\right) x_d(n) + \left(-1 - 2\frac{T_D}{T_0}\right) x_d(n-1) + \frac{T_D}{T_0} x_d(n-2) \right).$$

Written with coefficients q_i :

$$y(n) = y(n-1) + q_0 x_d(n) + q_1 x_d(n-1) + q_2 x_d(n-2).$$

The coefficients q_i then can be written as

$$q_0 = K*\left(1+\frac{T_0}{T_I}+\frac{T_D}{T_0}\right), \quad q_1 = K*\left(-1-2\frac{T_D}{T_0}\right), \quad q_2 = K*\frac{T_D}{T_0}$$

This is the general, famous recursive PID- algorithm used in many digital controller applications. In each step T_0 you need 3 multiplications, 3 additions and you have to store 3 floating point values y(n-1), x_d(n-1) and x_d(n-2). In the following table you can find a summary of all this different algorithms including a recursive form of P and PD.

Type	F(p)	q_i, p_i, all missing coefficients q_i, p_i=0	Form
P	K	q_0=K	non recursive
P	K	q_0=K, q_1=-K, p_1=1	recursive
D	Kp	q_0=-q_1=K/T_0	non recursive
I	K_I /p	p_1=1, q_0=K_I *T_0	recursive
PI	K_R(1+pT_N)/pT_N	q_0=K_R(1+T_0/T_N), q_1=-K_R, p_1=1	recursive
PD	K(1+pT_V)	q_0=K(1+T_V/T_0), q_1=-KT_V/T_0, st≈1+T_V/T_0	non recursive
PD	K(1+pT_V)	q_0=K(1+T_V/T_0), q_1=-K(2T_V/T_0+1), q_2=KT_V/T_0, p_1=1, st≈1+T_V/T_0	recursive
PID	K(1+pT_D+1/pT_I)= $\frac{K_R(1+pT_N)(1+pT_V)}{pT_N}$	q_0=K(1+T_D/T_0+T_0/T_I), q_1=-K(2T_D/T_0+1), q_2=KT_D/T_0, p_1=1, st≈1+T_V/T_0	recursive design I

6.3 Exercise Example PID

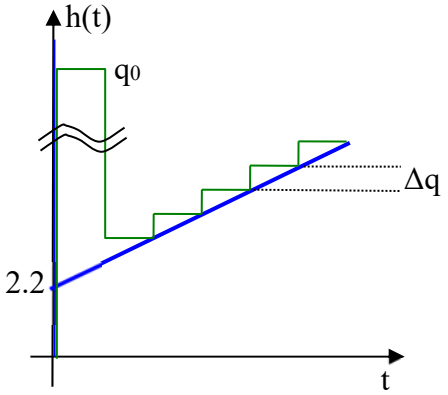
Task: Convert a PID with the parameters K_R=2, T_N=1, T_v=0.1 into a digital PID with sampling time T_0=0.01. Compare the unit step responses.
First convert given bode form parameters into summing form parameters with WB p 21:

$$K = K_R\frac{T_N+T_V}{T_N} \qquad T_I = T_N + T_V \qquad T_D = \frac{T_N T_V}{T_N+T_V}$$

calculated: K = 2.2, T_I= 1.1 and T_D=0.09090…

q_0=22.22, q_1=-42.20, q_2 =20.00. Note that it is very important to calculate with an accuracy of at least 4 significant digits! Some more details relating calculation errors will follow.
Step response calculation table:

n	$x_d(n)$	y(n)
-1	0	0 initialized!!
0	1	$=q_0=22.22$
1	1	$=y(0)+q_0+q_1=2.24$
2	1	=2.26
3	1	=2.28
4	1	=2.30



Note that starting with n=2 the output signal was calculated by previous value + sum of the three q with the value $\Delta q= q_0+q_1+q_2= 0.02$. This caused the ramp increase. The value of the fourth digit of q_0 is extremely relevant for this effect!

Note that the upper diagram contains a break in the axis scale.
The blue curve is the unit step response of the analogue PID. It starts with a Dirac impulse. The digital response starts with an impulse with width T_0 and height q_0. If T_0 becomes smaller, steps become smaller and the amplitude of the first pulse will be higher. In q_0 in one term T_0 is in the denominator.

6.4 Programming a digital filter / controller

As mentioned above to run a digital filter, you can use any processor and an ADC- Interface with at least one input and one output. Furthermore a constant calling function with the predictable time T_0 is necessary.
Algorithm expressed mathematically:

$$y(n) = y(n-1) + q_0 x_d(n) + q_1 x_d(n-1) + q_2 x_d(n-2)$$

With valid C- variable names:
*yn=yn1 + q0 * xdn + q1 * xdn1 + q2 * xdn2;*

Then the structure of program looks like this:

Preparation before start of the continuous calling of control algorithm:
Input of PID- parameters and T_0. Calculation of the values q0, q1, q2. Initialization of variables of old values: yn1, xdn1,xdn2=0.

Call control algorithm each T_0:

Input by ADC of new actual value x
Calculation of control difference xdn=w-x;
*yn=yn1 + q0 * xdn + q1 * xdn1 + q2 * xdn2;*
Output of yn on DAC
Actualizing: yn1=yn; xdn2=xdn1;xdn1=xdn;

You can see how simple the programming can be realized. The algorithm itself is only one line of C- Code. Any filter can be realized in this way, you need just some few lines of C, Java, C# or any other high level language code.

6.5 Choice of T_0

You have to be careful with the choice of T_0. If it is too large, changes in x can not respond quickly. If T_0 is too small, other problems could occur. Here you can find some recommendations:

1. You know the Shannon- limit. The sampling frequency must be larger than 2 times the highest signal frequency. Otherwise the signal will be destroyed by aliasing. This limit can not be used directly in control systems because we know nothing about the signal frequency. A good recommendation is derived from a look into a simple PT1- System. The output step response is the well known function h(t)=K{1-exp(-t/T)}. The change of output is mainly influenced by the time constant T. 63% full range change is done in one time constant T. If a control system should react on a change of the output, then the sampling time should be small enough. A good control system has a sampling time, which is smaller than 10% of the largest process time constant. So measure the dominant time constant T of your system and set ***upper limit*** of $T_0 \leq 0.1$ T.

2. Another estimation of ***upper limit*** for T_0 is possible, if PID- design is made by FRA-method. Then the crossover frequency w_d is known. A digital controller has a delay time between $T_0/2$ and $1.5T_0$ which will be explained later. The worst case is $T_d = 1.5^* T_0$. A delay time block has a negative phase shift of $\varphi = - w*T_d*180°/\pi$, where w is the radian frequency. This reduces the phase margin. If this reduction is as small as an acceptable value $\Delta\varphi_{max}$ (e.g. $\Delta\varphi_{max}$ =-5°), then this gives an upper limit of T_0.

   $\Delta\varphi_{\max} \geq \omega_d * 1.5 * T_0 * 180° / \pi$, solved to T_0:

   $$T_0 \leq \frac{\Delta\varphi_{\max} * \pi}{1.5 * 180° \omega_d} = 0.01164 * \Delta\varphi_{\max} / \omega_d = 0.05818 / \omega_d .$$

3. Now some lower limits. First, ***lower limit*** is simply defined by calculation time. Of course the sampling time must be larger than calculation time T_c including conversion time of ADC and DAC, so $T_0 > T_c$. The way to estimate the calculation time is described later.

4. Second, ***lower limit*** is a special estimation in PID- controllers. The smaller the sampling time, the higher the first pulse after a reference step function. If this amplitude should not be limited, T_0 should not be too small. With the following values you can calculate a lower limit: Maximum controller output y_{max}, input reference step amplitude x_0 and the PID- parameters K, T_I, T_D and T_0.

   $$y_{\max} \geq x_0 q_0 = x_0 K \left(1 + \frac{T_D}{T_0} + \cancelto{0}{\frac{T_0}{T_I}} \right) .$$

   If you neglect the very small term T_0/T_I this can be solved to T_0:

   $$T_0 \geq \frac{T_D}{\frac{y_{\max}}{K x_0} - 1} .$$

   In our above example with the values T_D=0.0909 s, y_{max} = 10V, x_0=1V and K=2.2 we get the limit $T_0 \geq 25.6$ ms. Remember with T_0=10 ms we got the amplitude of 22.22V which is too large when compared with the 10 V maximum.

5. Third, ***lower limit*** is caused by rounding errors in the calculation of the algorithm. Especially in the PID- algorithm this leads to a clear limit of T_0. Look first again on the q_0- value. Take the values of the first PID- example K=2.2, T_I=1.1s, T_D=0.09090s. Now compare the terms in the calculation of q_0:

$$q_0 = K\left(1 + \frac{T_D}{T_0} + \frac{T_0}{T_I}\right).$$

T_0	T_D/T_0	T_0/T_I	ratio
10 ms	9.0909	0.009090	1000
1 ms	90.9090	0.0009090	100000
0.1 ms	909.090	0.00009090	1e7

The ratio of the big term and the small term increases with the square of the decrease of T_0. Both terms are added to get q_0. The accuracy of the calculation dictates whether or not the small term can be represented in the sum. It is important to mention that the small term represents the ramp gradient. It contains the complete I- part of the controller. If this small term is destroyed by a rounding or truncation the I- part is destroyed. Now here is some information about internal representation of numbers in a computer. There are integer numbers (int, short int) and floating point numbers (float, double, extended). These variables have one part of bits used for a fixed-point-part (mantissa) and a second part used for an exponent of basic 2. Additionally 1 bit for the sign of mantissa and if there is an exponent a bit for the sign of exponent. The accuracy is only defined by the bits of the mantissa.

Type	Digits accuracy n_{10}	Bits for exponent	Bits for mantissa n_2	Total byte length
float	≈7	7	23	4
double	≈15	10	52	8
extended	≈19	15	63	10
short int	≈4	-	15	2
int	≈9	-	31	4
int in DSP	≈7	-	23	3

The maximum error can be simply calculated by the size of 1 Bit compared with the largest number of the mantissa, if it is assumed that the range of mantissa is optimized and lays between 0.5 and 1* maximum number. Example: in a float number this error can be $1 / 2^{23} = 0.0000001192 \approx 1.2 * 10^{-7}$. The double type is more accurate. The error is $1 / 2^{52} = 2.2 * 10^{-16}$. I have used this information to get an expression for T_0. To have an acceptable I- part of the PID- controller the term representing this I-part T_0/T_I should be 10 times larger than the error of the term T_D/T_0. Then the I- error is less than 10%. :

$$\frac{T_0}{T_I} \geq 10 * \frac{T_D}{T_0} * \frac{1}{2^{n_2}}.$$

Solved for T_0 this gives the ***lower limit*** for T_0 : $T_0 \geq \sqrt{\frac{10 * T_I T_D}{2^{n_2}}}$.

Some results in our example (T_I= 1.1 s, T_D= 0.0909 s):

.

Type	Bits for mantissa n_2	T_0 larger than
float	23	0.35 ms
double	52	0.000015 ms
extended	63	0.00000033 ms
short int	15	5.52 ms
int	31	0.022 ms
int in DSP	23	0.35 ms

You can see you will have problems if you want to use the very fast short - int- algorithms of assembler language. Then the lower limit of T_0 is about 5 ms. This problem is magnified with larger PID time constants. If processes are slower (e.g. temperature controls are very slow), then T_I could reach 1000 s, T_D 100 s. With both these times the estimation with float – variables (Standard C- compiler on embedded systems) gives a $T_0 \geq 0.345$ s !!!

Before you design a digital control system, check these 5 limits in order to get a good system.

6.6 Accuracy of q_i in PID- algorithm

The last point of last chapter is not only important in the choice of T_0, but the accuracy of the q_i – coefficients must carefully be chosen. As mentioned, the term T_0/T_I carries the I- information. To demonstrate the danger of inaccuracy we conduct the following experiment: we take the previous example (q_0=22.22, q_1= - 42.2 and q_2=20.00) and increase the q_1 – value with a very small 0.5% change. Caused by this error- propagation changes the effective T_I – time constant. This T_I can be recalculated with the following method:
With given q- values you can solve the three q- equations for K, T_D and T_I:

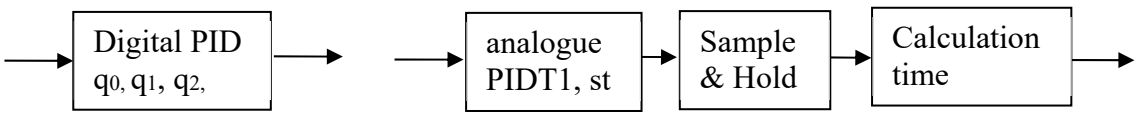
$$q_0 = K*\left(1+\frac{T_0}{T_I}+\frac{T_D}{T_0}\right), \quad q_1 = K*\left(-1-2\frac{T_D}{T_0}\right), \quad q_2 = K*\frac{T_D}{T_0}$$

$$K = -q_1 - 2q_2 \text{ and } T_D = \frac{T_0 q_2}{K} \text{ and finally } T_I = \frac{KT_0}{q_0+q_1+q_2}.$$

With the above q_i you get back the original K=2.2, T_I=1.1 and T_D=0.09090909. With the 0.5% - change (q_1= - 42.411) you get K=2.411, T_D=0.083 and T_I= - 0.126. The K and T_D change does not matter, but the effective T_I now is negative! This causes an unstable loop. With a 0.5% - change of q the T_I value changes over 100% !!! Be careful!

6.7 Improved FRA – design of digital PID: Dirt effects

Next step is to develop a more realistic substitute of a digital PID for use in standard simulation Programs like Regdelph or for use in a standard simple FRA- design.
The substitution is simple and very possible and consists of analog blocks. I found the following analog replacement:



This covers the most important negative effects of a digital PID. Details:

6.7.1 Step- depth- estimation

The behaviour of the digital PID, which is developed starting with a pure PID (with st=∞) is more like a PIDT1. But the digital PID behaves more like a PIDT1 with a starting impulse in the step response with a definite width and height. The starting impulse of the digital PID has amplitude q_0, not a Dirac impulse like a PID.
Now it follows the choice of a stepdepth st. This is chosen such, that the impulse amplitude of the first impulse of digital PID and its analogue substitute circuit are equal. The first impulse amplitude of the digital PID is already known: $y(0) = q_0 = K*(1+T_D/T_0 +T_0/T_I)$. The impulse amplitude of the PIDT1 is, as you know, equal to the value h(t=0) of the PIDT1 - step response, and it is $h(t=0) =K*T_D/T_1 = K_c *T_V/T_1$, if T_1 is the PT1 - time constant of the PIDT1. With $st=T_V/T_1$ follows $h(t=0) = st*K_R$ (see p. **Fehler! Textmarke nicht definiert.**).
If you now equate both impulse amplitudes, you will get the relationship y(0) = h(t=0) or assembled:

$$st * K_c = K\left(1+\frac{T_D}{T_0}+\frac{T_0}{T_I}\right) = K_c\frac{T_N+T_V}{T_N}\left(1+\frac{T_N T_V}{T_N+T_V}\frac{1}{T_0}+\Delta\right)$$

Now Δ will be neglected (small steps) and one receives the relationship

$$\boxed{st = \frac{T_N+T_V}{T_N}+\frac{T_V}{T_0} \approx 1+\frac{T_V}{T_0}}$$

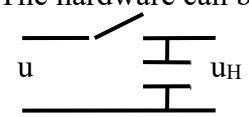
The term $(T_N+T_V)/T_N$ is about equal 1, if $T_N >> T_V$, so that the stepdepth is determined essentially through the relation of T_V to T_0. With it you get excellent consistency between theory (controller design) and practice (controller behaviour).
So there is a good stepdepth estimation of

$$st \approx 1+\frac{T_v}{T_0}$$

The other parameters K, T_D, T_I, T_N and T_V remain unchanged.

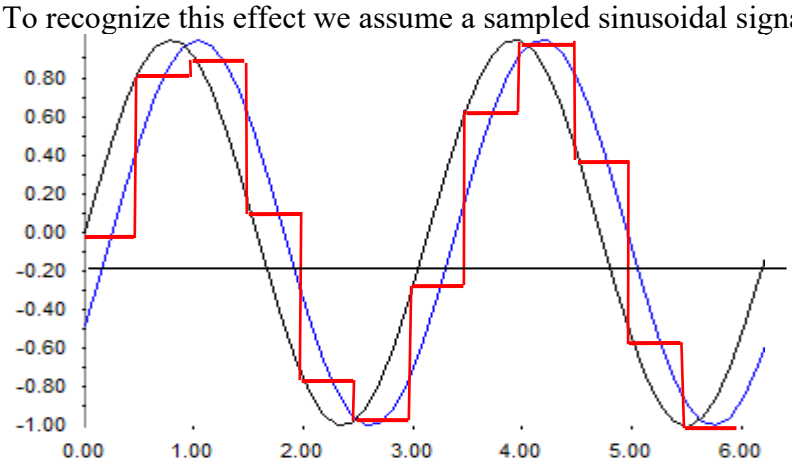
6.7.2 Sample & Hold

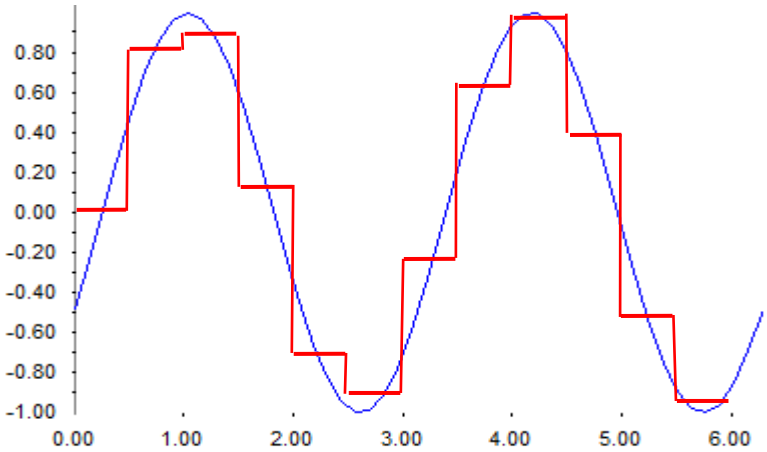
The Sample & Hold- block replaces the effect of the DAC. An output voltage is held constant for a complete T_0 cycle. Then the output voltage is updated and held for the next and so on. The hardware can be simply simulated with a capacitor and a very fast switch.



Each T_0 the switch is closed a very short time; the capacitor is loaded with the actual input voltage. Then the switch is opened and the voltage u_H is held constant by the capacitor.

To recognize this effect we assume a sampled sinusoidal signal.





The black curve is the input signal u(t). T_0 is 0.5 s. The red curve is u_H, resulting voltage after sampling. If you now would filter the red curve with an ideal low pass filter, which removes all harmonics of the red curve you get the blue curve. If you now compare black and blue curve, you see the same curve but shifted by half of the sampling time. The Sample & Hold behaves like a delay time block with a delay time of $T_0/2$! This is mathematically proven in the next step.

It can be shown, that a Sample & Hold- block has the complex transfer function

$F_H(p) = \dfrac{1-e^{-pT_0}}{pT_0}$ with the complex frequency response $F_H(j\omega) = \dfrac{1-e^{-j\omega T_0}}{j\omega T_0}$

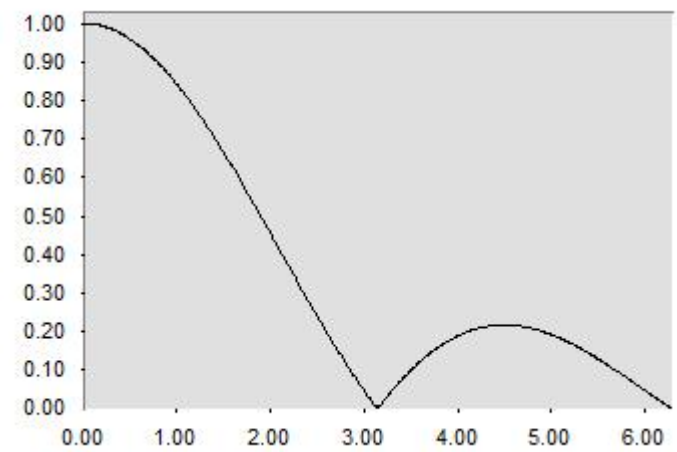
To separate the magnitude and the phase- function we use a trick. With Leonhard Euler the e-function e^{jx} can be replaced with cos x + j sin x.

$$F_H(j\omega) = \frac{1-\cos\omega T_0 + j\sin\omega T_0}{j\omega T_0}$$

After some further trigonometrical conversions we get the result

$$F_H(j\omega) = \frac{\sin\left(\dfrac{\omega T_0}{2}\right)}{\dfrac{\omega T_0}{2}} e^{-j\frac{\omega T_0}{2}}$$

The expression in front of the e- function is real and defines the amplitude function of the Sample & Hold block. The e-function itself has a magnitude 1 and defines the phase. The amplitude function is the well-known SI- function, which is depicted in the following diagram:



Drawn is the magnitude function

|SI(x)|=|sin(x)/x|.

You see zeroes at $x_0=\pi$, $2*\pi$, …
x in our Sample & Hold is

$$x = \frac{\omega T_0}{2}.$$

So we have a first zero at the frequency $\omega_0=2\pi/T_0$. This is the sampling frequency! If we sample a sinusoidal signal with frequency f_x

with the sampling frequency $f_0=f_x$, then we get only one point per period. These sampled points always have the same amplitude, so the output is a DC-signal.

Now consider the phase expression. A delay time block has the transfer function $F(p)=e^{-pT}$, if T is the delay time. The frequency response is $F(j\omega)=e^{-j\omega T}$. This fits exactly to the phase function of the Sample & Hold- block and you can see as shown before:

Comparison:

Phase of Sample & Hold- block	Phase of delay
$F_H(j\omega) = \lvert F_H \rvert e^{-j\frac{\omega T_0}{2}}$	$F(j\omega) = e^{-j\omega T}$
$\varphi_H = -\omega T_0/2$ (rad)	$\varphi = -\omega T$ (rad)
$\varphi_H = -\omega T_0\, 180°/(2\pi)$ (grad)	$\varphi = -\omega T\, 180°/\pi$ (grad)

A Sample & Hold block behaves in the phase shift exactly like a delay time block with $T=T_0/2$! This is proven.
Now we look at the values at Shannon limit: $\omega=\omega_0/2 = 2\pi f_0/2 = \pi/T_0$.
Phase value: $\varphi_H(\omega_0/2) = -90°$.
Magnitude value: $\sin(\pi/2)/(\pi/2) = 0.6266$, this is -3.9 dB.

In control loops now, the influence of the phase is much worse than the influence of the magnitude. Negative phase makes the loop less damped and more unstable, the magnitude effect is decreasing gain, this stabilizes the loop. In the following steps this is the reason to neglect the magnitude function and replace the Sample & Hold- block simply with a delay time block with the delay of $T_0/2$.

6.7.3 Calculation time

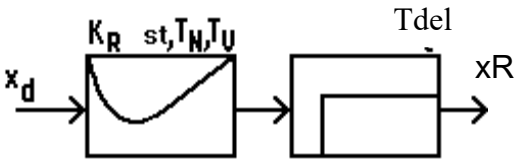
The processor needs some time to calculate the algorithm result. A PID algorithm needs 3 floating point multiplications. Furthermore, you have to add the time for the ADC-conversions. This conversion time depends on the method. A dual slope conversion is much more time consuming than a successive approximation method. This takes more time than a flash- conversion. This time is a delay and should be known at design.
There are 4 methods to get information about the value of this calculation time T_c, which behaves like a simple delay time block.

1. Use data sheet values to put all information together (clock cycles of microprocessor, ADC- times, multiplication times and so on. This is the most costly method.
2. Measure this time for example with a digital scope.
3. If you estimate the calculation time as a very small value compared with the delay time caused by Sample & Hold, neglect it. (if $T_c \ll T_0/2$, neglect T_c).
4. Use the "***real algorithm***" instead of the "***ideal algorithm***". These names are not used by other authors, these are my special names. The real algorithm artificially delays the output of the DAC until the next call of the control algorithm. This results in a calculation time of $T_c= T_0$ and you are completely independent of hardware. The disadvantage is of course the finally larger delay, which makes the control loop worse. So in industrial applications you should avoid this method. But in Lab- experiments this serves as a way to be independent of hardware. Realization is simple: Do nothing other than changing the order of command for DAC in the code:

Ideal algorithm	**Real algorithm**
Input by ADC of new actual value x Calculation of control difference xdn=w-x; *yn=yn1 + q0 * xdn + q1 * xdn1 + q2 * xdn2;* **Output of yn on DAC** Actualizing: yn1=yn; xdn2=xdn1;xdn1=xdn;	Input by ADC of new actual value x **Output of yn on DAC** Calculation of control difference xdn=w-x; *yn=yn1 + q0 * xdn + q1 * xdn1 + q2 * xdn2;* Actualizing: yn1=yn; xdn2=xdn1;xdn1=xdn;

6.7.4 Conclusion

If you want to design a digital PID, use the replacement



Tdel=3/2To with real, Tdel=1/2To+Tc with ideal A.
second analogous substitute of a digital PID

Delay time: Sum of Sample & Hold delay $T_0/2$ and calculation time T_c.

Step depth: st = 1 + T_v/T_0.

The complete steps for an FRA design of a digital PID See the next Box:

Procedure of the FRA-design of digital PID - controller:

1. Decide for an appropriate sampling time T_0, if possible T_0<0.1* largest process time constant.
2. In your FRA (Frequency Response Approach) - design you have to add to the process a delay time term with T_{del} =1/2T_0+T_c, whereby T_c represents the calculation time. With the real algorithm T_c=T_0. Delay phase is φ_{del}= - ωT_{del} *180°/π.
3. If st is not 1 (PI – case) add the phase of the PDT1- part of the controller. Choose T_V (e.g. with pole compensation or -30 dB method) and st = 1 + T_V/T_0 and the PDT1- phase is φ_{PDT1}=arctan(ωT_v)-arctan(ω*T_v/st).
4. This gives $\varphi_{new} = \varphi_S + \varphi_{del} + \varphi_{PDT1}$.
5. Now continue with standard FRA with the point 3 to 5 depending on the PI method symmetrical optimum or pole compensation of Workbook CS I page 37.
6. With the following equation set determine the parameters q_i of the algorithm with
$q_0 = K(1 + \frac{T_0}{T_I} + \frac{T_D}{T_0})$, $q_1 = -K(1 + 2\frac{T_D}{T_0})$, $q_2 = K\frac{T_D}{T_0}$ and
$K = K_R(T_N + T_V)/T_N$, $T_I = T_N + T_V$ and $T_D = T_N T_V/(T_N + T_V)$.

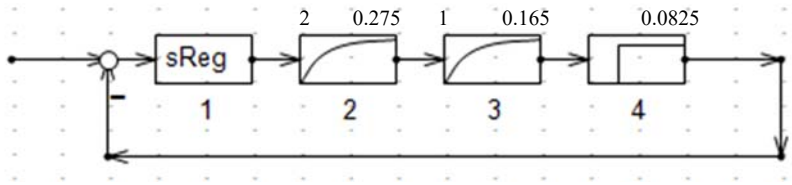
Now an exercise with a 2PT1- process:

6.7.5 Example

We will now design a digital controller with a sampling time of T_0=55 ms. The process transfer function reads
$F_S = K_S/(1 + pT_1)(1 + pT_2)$ with K_S=2, T_1=5T_0 =0.275 s and T_2=3T_0=0.165s.

It should be a real PIDT1 - controller, the phase margin being φ_R=60°.

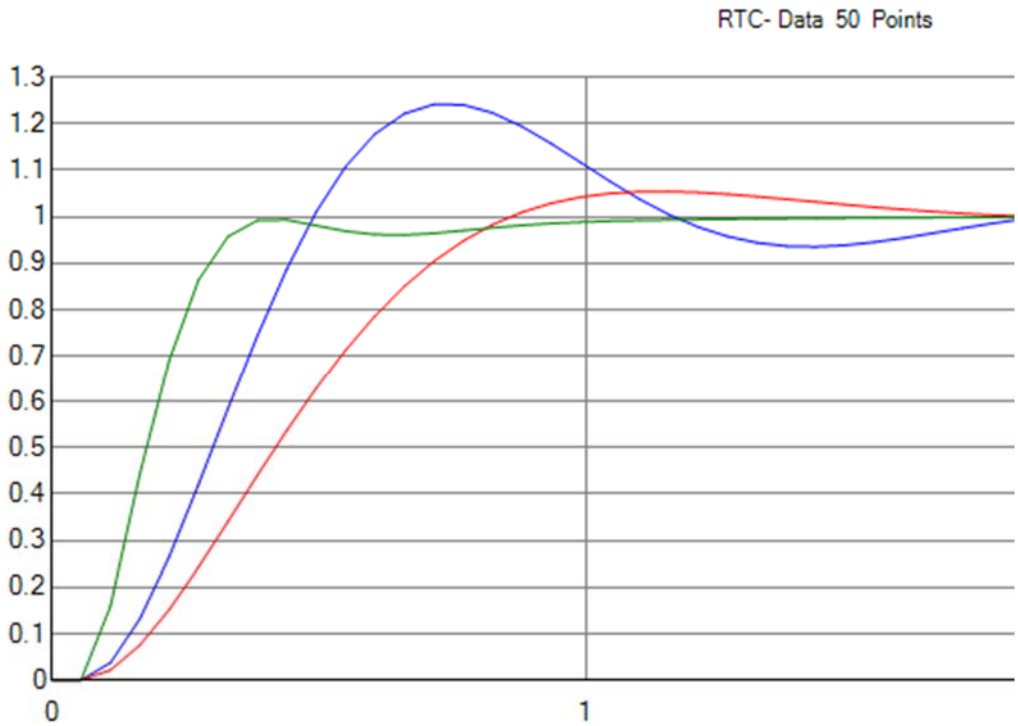
<u>Solution:</u> The block diagram for *RegC#* appears as follows:



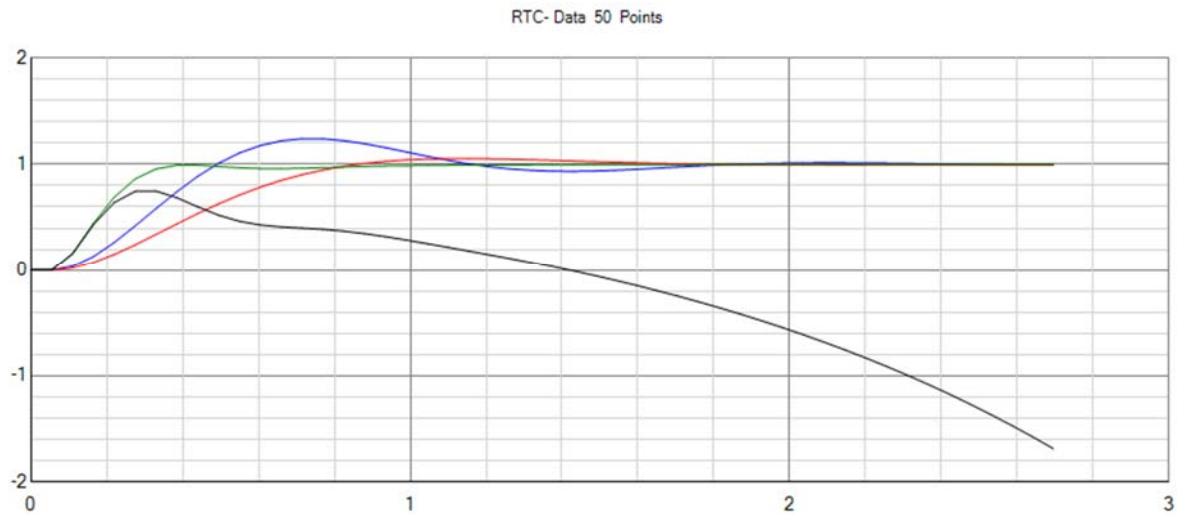
The D-part of the controller is determined, here with pole compensation (compensation of the 2nd largest process time constants), therefore $T_V=T_2=0.165$s. The stepdepth st can be calculated then to $st=1+3T_0/T_0=4$, i.e. the PT1 -time constant of the controller is $T_3=T_V/4=41.25$ ms. (In *RegC#* the st can directly be changed). The result of the controller design is at a fixed choice of $T_N=0.275$s now $K_R=0.5925$.
You can find complete description of RegC# design and WindfC# - controller in the paper *Tutorial Windfc# with CSII lecture examples.pdf* in moodle-course.

With *WindfC#* the digital parameters q_i can be calculated to $q_0=2.8447$, $q_1=-4.5041$ and $q_2=1.7779$, $p_1 = 1$ (green curve).

If you ignore the delay time block (redesigning the controller without delay) results in a controller gain of $K_R=2.222$. If you convert this controller into a digital one, you get the values $q_0=10.666$, $q_1=-16.888$ and $q_2=6.666$, $p_1 = 1$ and a closed loop will be unstable (no curve)!
A PI design with and without delay is added with the resulting reference step responses on next page:
PI without delay: $q_0=0.666$, $q_1=-0.5555$, $p_1 = 1$ (blue curve).
PI with delay $q_0=0.3809$, $q_1=-0.3174$, $p_1 = 1$ (red curve).



To demonstrate the problems with inaccuracies add to the described experiments the PID where the second q_1- parameter is changed from -4.5021 into -4.7021. Then you can see that loop will be unstable (black curve in bottom display). A simple 4.4% change causes a disaster! If you change all K or T- values in the same system with 4.4%, you can see only neglectable effects. So be careful with the qs.



Black curve: instability caused by q1- change of less than 5% !!!

```
double y,yold, xdold1, xdold2; // global var

void control_algorithm()
  {
  double xd0;
  input(xrtc);   // read actual x-Value from ADC
  output(y);     // store new y - value to the DAC

  xd0=w-xrtc;;   //error signal

  y=yold + 2.8436*xd0 - 4.5021*xdold1 + 1.7771*xdold2;
  yold=y; xdold2= xdold1; xdold1=xd0;
  }

void init()
  {
  y=0; yold=0; xdold1=0; xdold2=0;
  }
```

Last final information: See code example in MS Studio C# - syntax.
The control alg. must be called each T_0. Before start call one time the init()- function.
The input() and output() are the interface functions to the ADC / DAC.

7 Introduction into Z- transformation

The design method used in the previous chapter converts a differential equation into a difference equation. For simple filters this is useful, but for more complex filters there are other better methods using the mathematical tools of the Z-transformation. What is this?
To get a very simple introduction I take a general filter algorithm and convert this into the frequency domain using the Laplace transformation:

$$y(n) = p_1 y(n-1) + p_2 y(n-2) + ... + q_0 x(n) + q_1 x(n-1) + q_2 x(n-2) + ...$$

Considering that y(n) is converted into the new function Y(p) and x(n) into X(p) and applying the shift theorem of Laplace (y(n-1) is the signal y(n) delayed with one T_0), so y(n-1) is converted into $Y(p)*e^{-pT_0}$. Finally:

$$Y(p) = p_1 Y(p) e^{-pTo} + p_2 Y(p) e^{-2pTo} + ... + q_0 X(p) + q_1 X(p) e^{-pTo} + q_2 X(p) e^{-2pTo} + ...$$

You see that the shift operator e^{-pTo} appears several times so people introduced the replacement

$$z = e^{pT0}$$

The z- transformation is nothing other than a Laplace transformation adapted to digital systems. Why the first mathematician used the replacement with positive exponent is unknown. For me this seems crazy because all shift terms have negative exponent. But this is now absolutely unchangeable.

We write with the replacement Y(p) → Y(z) and X(p) → X(z) the following equation:

$$Y(z) = p_1 Y(z) * z^{-1} + p_2 Y(z) * z^{-2} + ... + q_0 X(z) + q_1 X(z) * z^{-1} + q_2 X(z) * z^{-2} + ...$$

Now put all Y(z) terms to the left, extract Y(z) and X(z) and you get:

$$Y(z) * \left(1 - p_1 z^{-1} - p_2 z^{-2} - .....\right) = X(z) * \left(q_0 + q_1 z^{-1} + q_2 z^{-2} + ...\right).$$

Equivalent to the complex transfer function F(p) we now can define a Z- transfer function F(z) of a general digital filter:

$$F(z) = \frac{Y(z)}{X(z)} = \frac{q_0 + q_1 z^{-1} + q_2 z^{-2} + ...}{1 - p_1 z^{-1} - p_2 z^{-2} - .....}$$

Like in continuous filters the Diff. equation ○→—● F(p) in digital filters the algorithm ○→—● F(z). The symbol ○→—● can be read as "corresponds with".

The coefficients p and q describe the function. The biggest difference to continuous filters is the huge advantage that you can realize and program a digital filter in one line of C-code! If the coefficients are known, you write the algorithm

$$y(n) = p_1 y(n-1) + p_2 y(n-2) + ... + q_0 x(n) + q_1 x(n-1) + q_2 x(n-2) + ...$$

in the C- program line

```
y=p1*yn1 + p2*yn2 + q0*x + q1*xn1 + q2*xn2
```

with the same principles and additional step as described in the previous PID- chapters. Send y to the DAC, get x from the ADC and actualise the global variables each step with yn2=yn1;yn1=y; xn2=xn1;xn1=x;

Example: The F(z) of the PID from the last exercise can be written as

$$F(z) = \frac{2.8436 - 4.5021 z^{-1} + 1.7771 z^{-2}}{1 - z^{-1}}.$$

7.1 Properties of F(z)

If all p_i=0 we talk about non recursive filters. Another name of this filter is FIR or *finite impulse response* filter. If there is only one $p_i \neq 0$ this is a recursive filter. Another name of this filter is IIR or *infinite impulse response* filter.

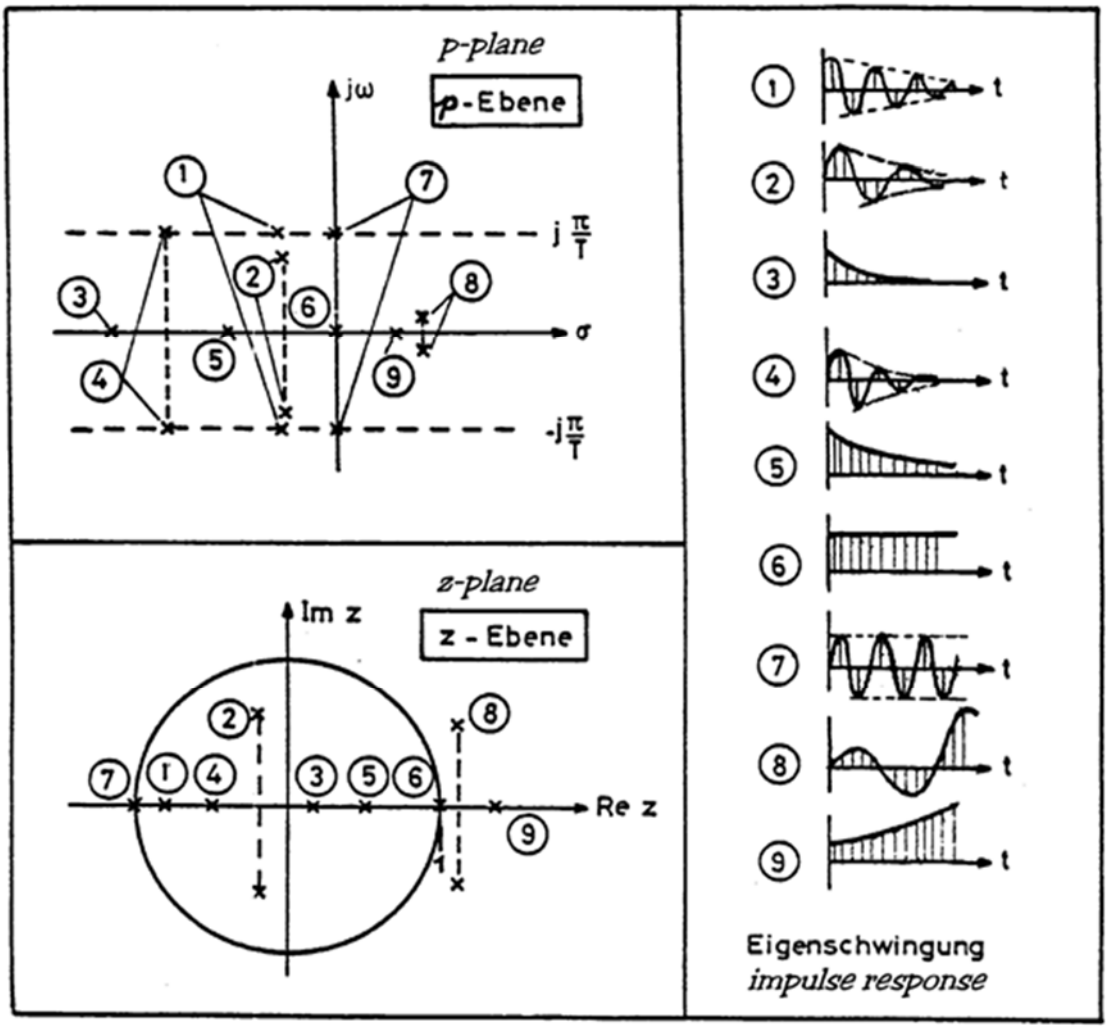
The final value of the unit step response of a digital filter, which is in other words means that the DC- gain of that filter can be calculated with p = 0, which is equivalent to z = 1. So replace all z with 1 ($z^{-x} = 1$, if z = 1 independent of x). So DC- gain of the above PID:

$$F(z=1) = \frac{2.8436 - 4.5021 + 1.7771}{1 - 1} = \frac{0.1186}{0} = \infty$$

The infinite high DC- gain is caused by the integrator in the filter.

Stability of a filter is similar to the pole- criterion. A digital filter is stable if and only if all poles of F(z) are laying in the unit circle or in other words $|z_{\infty i}| < 1$ if $z_{\infty I}$ are all poles. If the pole $z_{\infty I} = 1$ you have an infinite DC – gain like the previous example. All non-recursive filters are stable, because they have no poles, only recursive filters can be unstable.

In the following picture you can find some examples of poles in the p- plane with the corresponding position in the z- plane and the corresponding impulse responses.



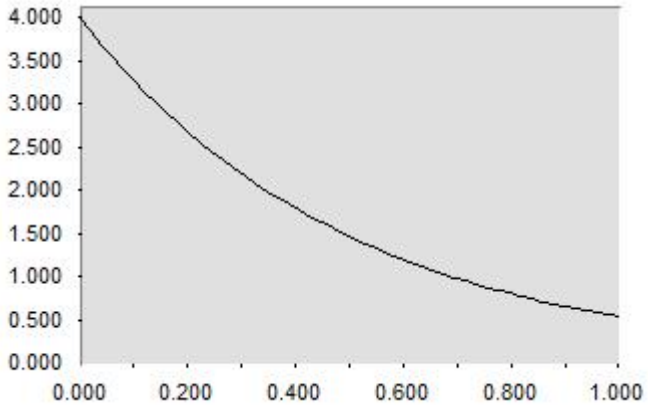
7.2 Conversion of F(p) into F(z)

With the next four methods you can convert any continuous, analogue filter into a digital one. The last two methods are available as software tools in WindfC#. First, we will start with two analytical methods. We will demonstrate each method with a very simple PT1- example. The parameters are K=2, T = 0.5s and T_0 = 0.1s.

7.2.1 Impulse response invariant method

A very simple technique to get this filter is described by the following: Take the desired impulse response and sample with T_0. Than take the sample amplitudes as q- values. Our example:

The unit impulse response of this PT1 is (see WB p.9) $g(t)=(K/T) * e^{-t/T}$. With our numbers we get $g(t)=4/s * e^{-2t/s}$. This is the response after a unit Dirac impulse with area 1. We get the following values:

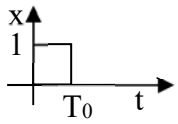


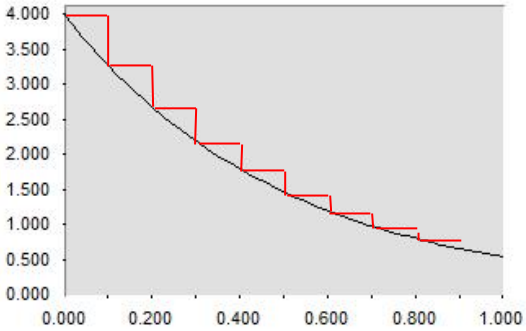
t	y(t)	q
0	4	q_0
0.1	3.27492301	q_1
0.2	2.68128018	q_2
0.3	2.19524654	q_3
0.4	1.79731586	q_4
0.5	1.47151776	q_5
0.6	1.20477685	q_6
…	…	…

The filter F(z) is

$$F(z) = q_0 + q_1 z^{-1} + q_2 z^{-2} + ...$$

If you apply a digital unit impulse with bottom diagram to the input of this filter, you get exactly the same values at the output y, but now with the red steps.





The same result you get with a method using F(z) – conversion tables like the table on the following page.

$f(t)$	$F(p)$	$\mathcal{Z}\{F(p)\} = F(z)$
$\delta(t)$	1	nicht existent
1	$\frac{1}{p}$	$\frac{z}{z-1}$
t	$\frac{1}{p^2}$	$\frac{T_0 \cdot z}{(z-1)^2}$
e^{-at}	$\frac{1}{p+a}$	$\frac{z}{z-e^{-aT_0}}$
$\frac{1}{T} \cdot e^{-\frac{t}{T}}$	$\frac{1}{1+pT}$	$\frac{1}{T} \cdot \frac{1}{1-e^{-\frac{T_0}{T}} \cdot z^{-1}}$
$1-e^{-at}$	$\frac{a}{p(p+a)}$	$\frac{(1-e^{-aT_0}) \cdot z}{(z-1)(z-e^{-aT_0})}$
$e^{-at} \cdot \sin \omega t$	$\frac{\omega}{(p+a)^2+\omega^2}$	$\frac{z^{-1} \cdot e^{-aT_0} \cdot \sin \omega T_0}{e^{-2aT_0} - 2z^{-1} e^{-aT_0} \cos \omega T_0 + z^{-2}}$
t^2	$\frac{2}{p^3}$	$\frac{T_0^2 z(z+1)}{(z-1)^3}$
$t \cdot e^{-at}$	$\frac{1}{(p+a)^2}$	$\frac{T_0 \cdot z \cdot e^{-aT_0}}{(z-e^{-aT_0})^2}$
$e^{-at} - e^{-bt}$	$\frac{b-a}{(p+a)(p+b)}$	$\frac{z(e^{-aT_0} - e^{-bT_0})}{(z-e^{-aT_0})(z-e^{-bT_0})}$
$\frac{at-(1-e^{-at})}{a}$	$\frac{a}{p^2(p+a)}$	$\frac{T_0 \cdot z}{(z-1)^2} - \frac{(1-e^{-aT_0}) \cdot z}{a(z-1)(z-e^{-aT_0})}$
$(c-a)e^{-at} + (b-c)e^{-bt}$	$\frac{(b-a) \cdot (p+c)}{(p+a) \cdot (p+b)}$	$\frac{(c-a) \cdot z}{z-e^{-aT_0}} + \frac{(b-c) \cdot z}{z-e^{-bT_0}}$

Let's try the simple conversion:

$$F(p) = \frac{K}{1+pT} \Rightarrow F(z) = \frac{K}{T} \frac{1}{1-e^{-To/T} z^{-1}}.$$

With numbers:

$$F(p) = \frac{2}{1+p*0.5} \Rightarrow F(z) = \frac{2}{0.5} \frac{1}{1-e^{-1/5} z^{-1}} = \frac{4}{1-0.8187307531 z^{-1}}.$$

n	x(n)	y(n)
0	1	4
1	0	3.27492301
2	0	2.68128018
3	0	2.19524654
4	0	1.79731586
5	0	1.47151776
6	0	1.20477685

This can be converted into the algorithm

$$y(n) = 0.8187 \;\; y(n-1) + 4x(n).$$

This algorithm has the same impulse response values, see above tables! But this is the recursive and the above is the non-recursive form: Less calculations in the algorithm.

The problem with a filter of this type is the wrong DC- value. Especially in control systems we need exact DC- values to avoid wrong results in control errors. The DC gain of analog filter is K=2, the digital filter has a DC- gain of

$F(z=1)=\dfrac{4}{1-0.8187307531}=22.0666$, this is far away from 4.

7.2.2 Step response invariant filter

This type of filter is used for process simulations and to calculate dead-beat – controllers. The procedure is finally described by

$$H_0F(z)=\frac{z-1}{z}Z\left\{\frac{1}{p}F(p)\right\} .$$

The notation $Z\{\ \}$ can be read as "Z- transformed function of" and can be realized with the Z-transform table see above. So take your F(p), divide by p, find equivalent F(z) in table and multiply this result with (z-1)/z. In our example we get with row number 6 and a=2

$$Z\left\{\frac{2}{p(1+p*0.5)}\right\}=Z\left\{2*\frac{2}{p(2+p)}\right\}\Rightarrow F(z)=2*\frac{\left(1-e^{-2To}\right)z}{(z-1)\left(z-e^{-2To}\right)}.$$

For the final filter function this has to be multiplied with (z-1)/z and we get with T_0=0.1

$$H_0F(z)=2*\frac{1-e^{-2To}}{z-e^{-2To}}=\frac{0.3625384938}{z-0.8187307531}.$$

This cannot directly be converted into an algorithm because the F(z) must be normalized. This means only z with negative exponents is allowed and the coefficient in the denominator without a z must be 1. So we normalize this result by dividing by z. We get

$$H_0F(z)=\frac{0.3625384938z^{-1}}{1-0.8187307531z^{-1}}.$$

This corresponds with the algorithm

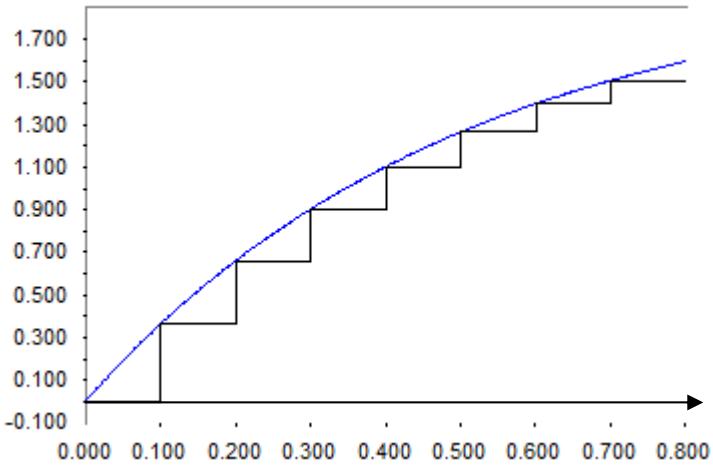
$$y(n)=0.8187307531*y(n-1)+0.3625384938*x(n-1).$$

The unit step response of this filter is easily calculated with our table step by step and can directly be compared with the values of the PT1-step response function $h(t)=2(1-e^{-t/0.5})$:

n	t	x(n)	y(n)	$h(t)=2(1-e^{-t/0.5})$
0	0	1	0	0
1	0.1	1	0.3625384938	0.3625384938
2	0.2	1	0.6593599078	0.6593599078
3	0.3	1	0.9023767277	0.9023767277
4	0.4	1	1.1013422072	1.1013422072
5	0.5	1	1.264241118	1.264241118
…	….	…	….	….
∞	∞	1	2	2

You can see that the values of the digital algorithm and the analogue filter are absolutely identical. That is the reason for the name "step response invariant filter". The following

diagram shows both curves, the blue is the analogue filter step response, the black with steps the digital one.



This filter type is, on the one hand, used to calculate Dead-beat controllers and furthermore useful in simulation programs of analogue filters. The output values not only of step responses but of responses of all kind of stepwise curves are identical. I use this filter type in my WindfC#- program in the menu "ADC-Cards →Hardware simulation" to simulate processes.

7.2.3 Filter with rectangular approach

The idea of this filter type is to replace all p in F(p) with an approximation via Taylor- series. If $z = e^{pTo}$, than

$$p = \frac{1}{T_0} \ln z$$

This replacement cannot be directly used because then the resulting F(z) contains ln z – functions. These F(z) cannot be converted into an algorithm.
The first simple Taylor- series of this expression is

$$p \approx \frac{1}{T_0}(1 - z^{-1}) \ .$$

If all p in a F(p) expression are replaced with this you get a F(z) expression. Let's do this first with a pure integrator with $F(p) = K_I/p$. We get

$$F(z) = \frac{K_I T_0}{1 - z^{-1}}$$

The corresponding algorithm is

$$y(n) = y(n-1) + K_I T_0 x(n) \, .$$

If you compare with page 6 of this paper you recognize this as exactly the green coloured Bayerlein- version of Integrator. Because the area was approximated with sum of rectangles this method is the "rectangular approach". So if we replace all p with this approach we get exactly all algorithms described in pages 4-10 of this paper.

When applied to our low pass filter we get

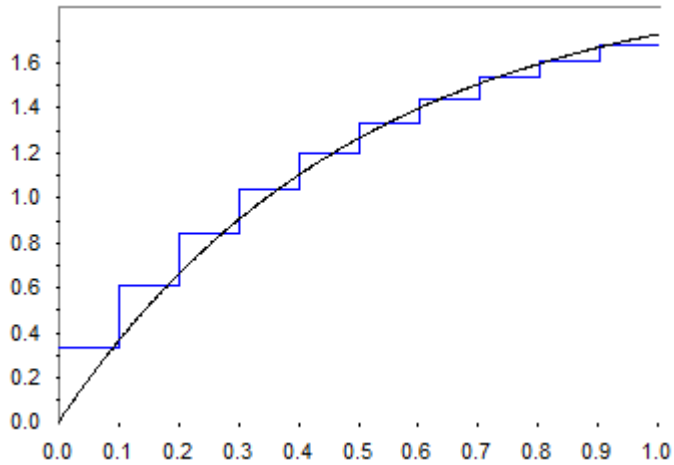
$$F(z) = \frac{K}{1 + pT} = \frac{K}{1 + \frac{1}{T_0}(1 - z^{-1})T} = \frac{2}{1 + 5(1 - z^{-1})} = \frac{2}{6 - 5z^{-1}} = \frac{1}{3}\frac{1}{1 - \frac{5}{6}z^{-1}} \, .$$

Written as an algorithm yields

$$y(n) = \frac{5}{6} y(n-1) + \frac{1}{3} x(n) = 0.833333333333\, y(n-1) + 0.33333333333\, x(n) .$$

As a result we get the step response in the following table and diagram:

n	x(n)	y(n)
0	1	0.333333333333333
1	1	0.611111111111111
2	1	0.842592592592592
3	1	1.035493827160490
4	1	1.196244855967080
5	1	1.330204046639230
6	1	1.441836705532690



In the beginning, the steps are a little bit too high, in the end too low, but the DC- gain is correct. F(z=1) is

$$F(1) = \frac{1}{3} \frac{1}{1-\frac{5}{6}} = \frac{1}{3} \frac{1}{1/6} = 2 \quad .$$

7.2.4 Filter with trapezoidal approach

A much better Taylor series of ln(z) leads to the trapezoidal approach, in which p is replaced by

$$p \approx \frac{2}{T_0} \frac{(1-z^{-1})}{(1+z^{-1})} .$$

This is also known as “Boxer- Thaler transformation” or “Bilinear transformation”.

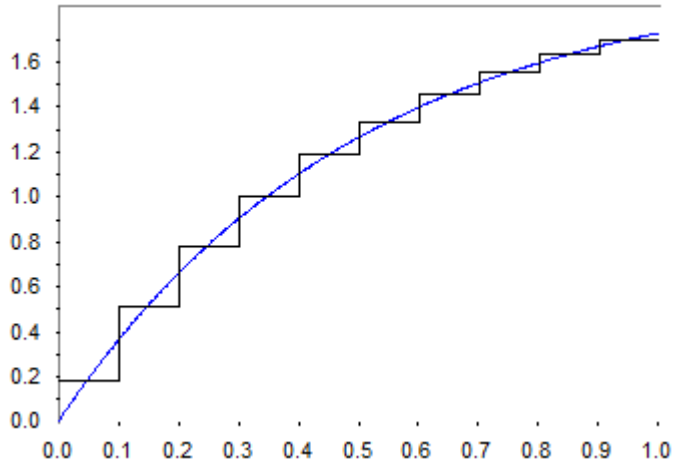
If we would do this replacement with the pure integrator, we get a result identical to a replacement of the area by a sum of trapezoids. This is the reason for the name. Now apply this to our PT1:

$$F(z) = \frac{K}{1+pT} = \frac{K}{1+\frac{2}{T_0}\frac{1-z^{-1}}{1+z^{-1}}T} = \frac{2(1+z^{-1})}{1+z^{-1}+10(1-z^{-1})} = \frac{2(1+z^{-1})}{11-9z^{-1}} = \frac{2}{11}\frac{1+z^{-1}}{1-\frac{9}{11}z^{-1}} .$$

Corresponding algorithm:

$$y(n) = \frac{9}{11} y(n-1) + \frac{2}{11} x(n) + \frac{2}{11} x(n-1) = 0.\overline{81}\, y(n-1) + 0.\overline{18}\, x(n) + 0.\overline{18}\, x(n-1)\,.$$

n	x(n)	y(n)
0	1	0.181818181818182
1	1	0.512396694214876
2	1	0.782870022539444
3	1	1.004166382077730
4	1	1.185227039881780
5	1	1.333367578085090
6	1	1.454573472978710



Also this filter type has a correct DC- value of 2.

7.3 Digital filter design with software tool WindfC#

Either the rectangular or the trapezoidal approach can be executed with the tool program WindfC#. It can be found on moodle course “Toolprogramme Bayerlein” in FHL It runs with all windows versions higher than XP.

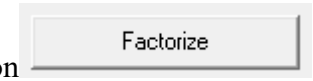
The analogue complex transfer function must then be typed in or imported as file *.ufk or file *zk. This is possible in both forms. The general form (file *.ufk) can be described with

$$F(p) = \frac{d_m p^m + \ldots + d_1 p + d_0}{c_n p^n + \ldots + c_1 p + c_0} e^{-pT0}.$$

You can type in the coefficients with the module behind the button 

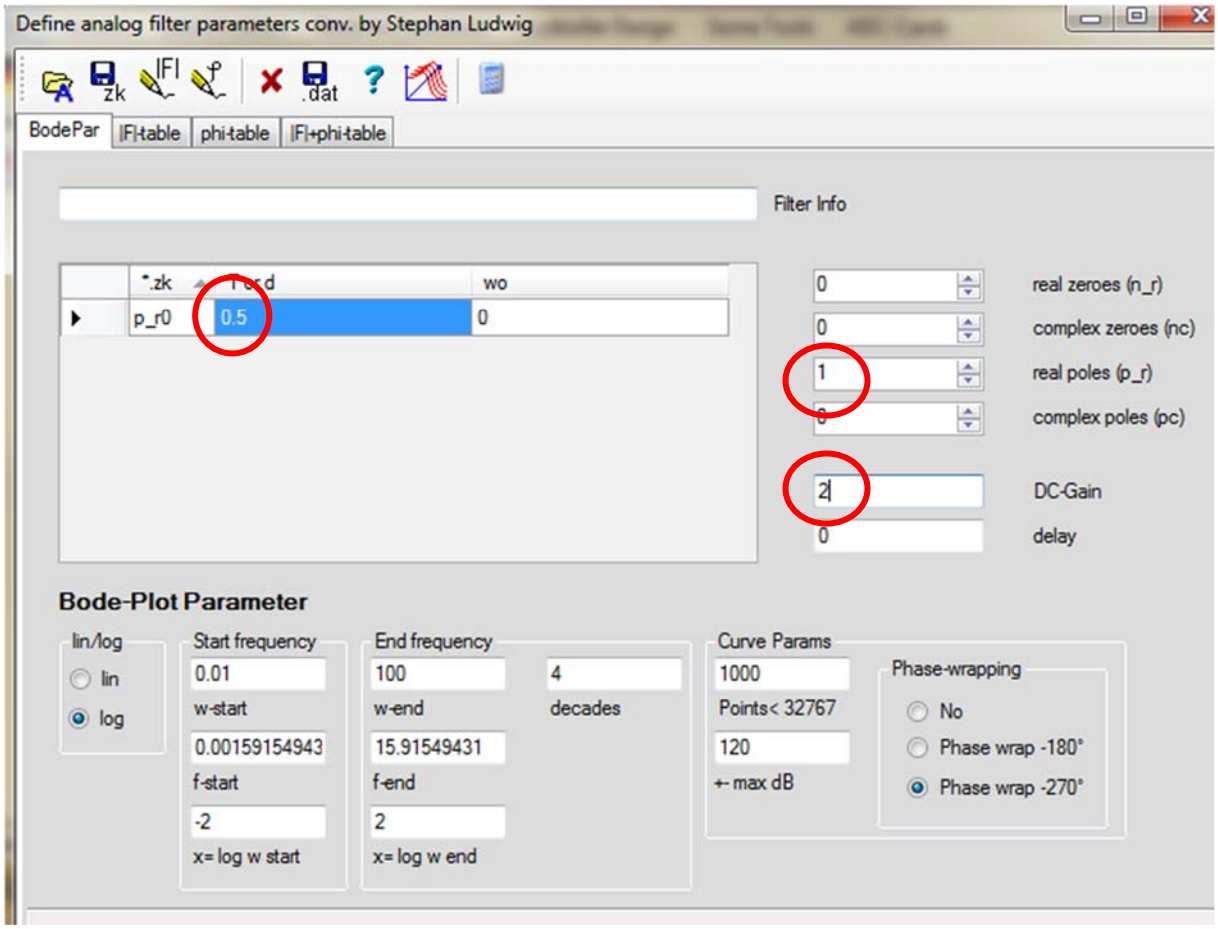
. The conversion into a digital filter is only possible in the time-constant form described by:

$$F(p) = K \frac{(1+pT_{01})..(1+pT_{0nr})\left(1+2d_{01}p/\omega_{01}+p^2/\omega_{01}^2\right)..\left(1+2d_{0nc}p/\omega_{0nc}+p^2/\omega_{0nc}^2\right)e^{-pT0}}{(1+pT_{\infty 1})..(1+pT_{\infty pr})\left(1+2d_{\infty 1}p/\omega_{\infty 1}+p^2/\omega_{\infty 1}^2\right)..\left(1+2d_{\infty pc}p/\omega_{\infty pc}+p^2/\omega_{\infty pc}^2\right)}.$$

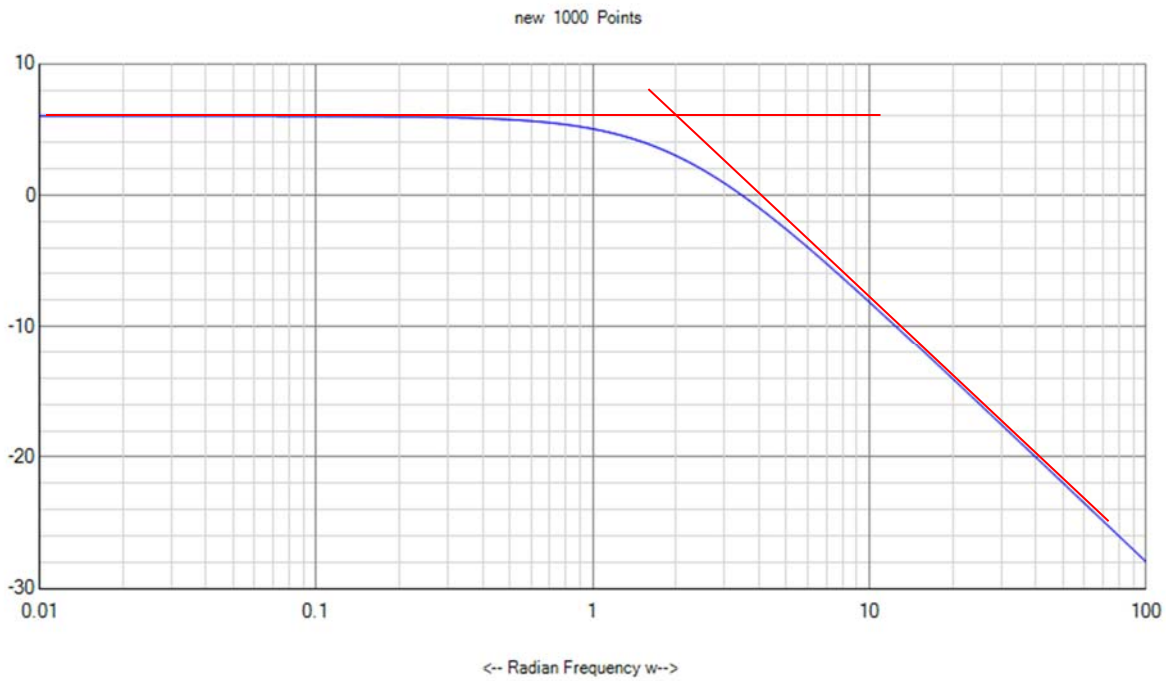
With the button 

in the ufk module you can convert the general form into this time-constant form. After factorizing, leave the module with the  - button.

With the button  you can type in T or d / wo – values or load a *.zk- file or edit a converted file from factorizing. Let's do this with our simple low pass filter used in the previous chapters to compare the results and have a look on the bode plots. If you open the window behind the button  , you get this window. Note the both values K=2 and T=0.5.

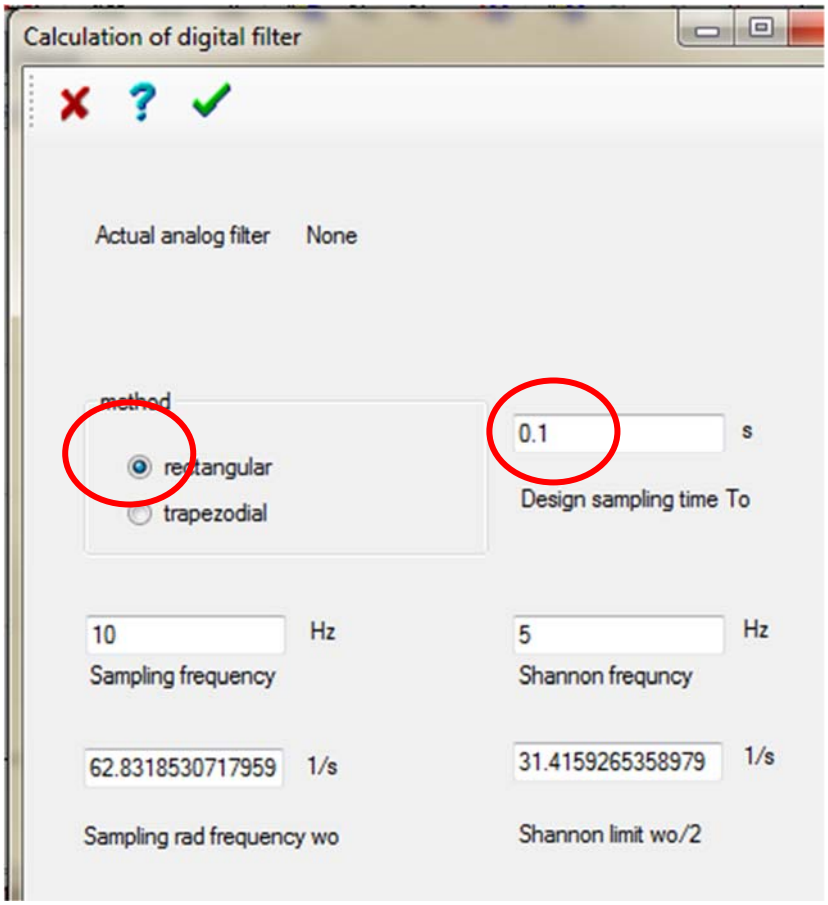


Now you can close this window with the **|F|** - button and the magnitude curve of the bode plot appears in the main window.

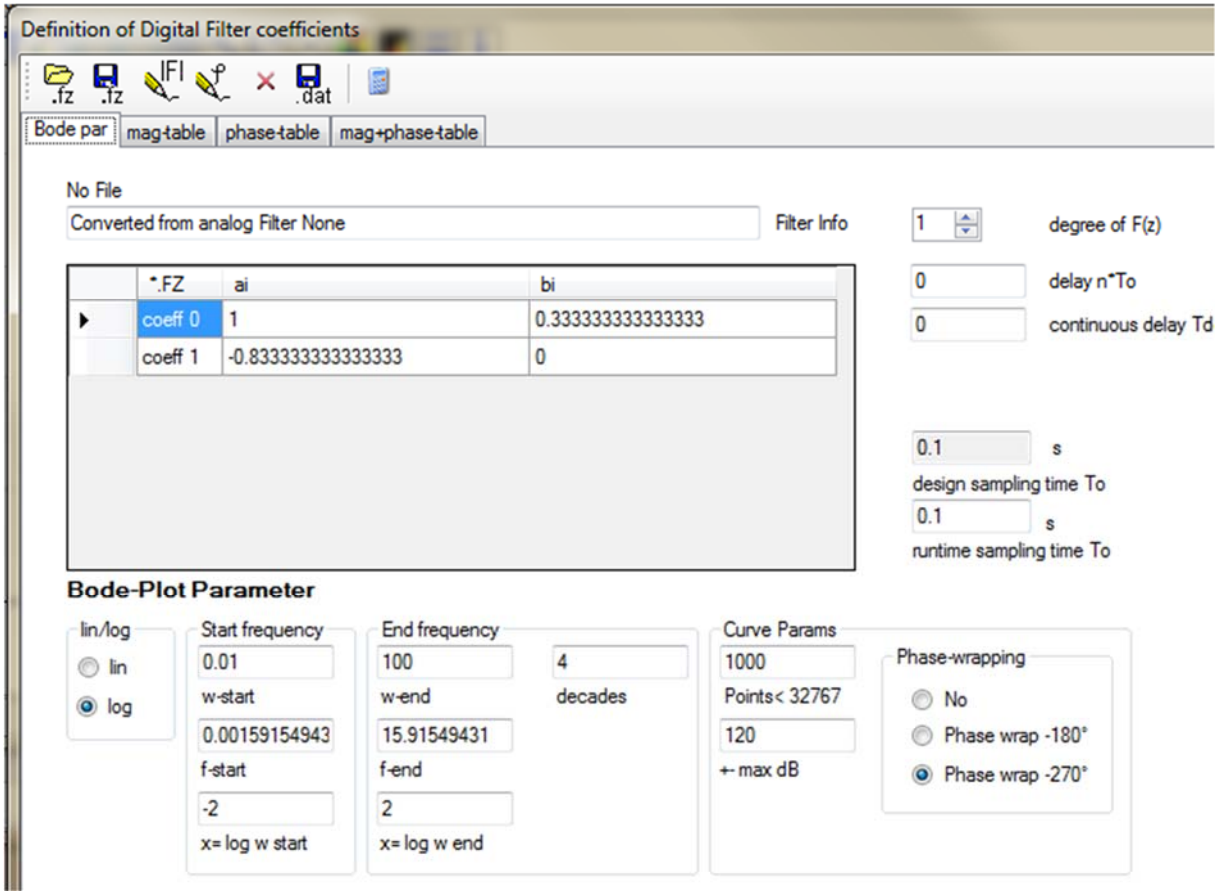


The red curves are manually drawn asymptotical curves. Note that horizontal axis carries the value of ω. The crossing defines the corner frequency at $\omega_c = 2 = 1 / T$. This gives the T-value of 0.5. OK?

Now we open the conversion box for digital filters with the  - button. You have to select the method and the sampling time T_0.



By closing this box with - button, the digital filter is calculated and the values are available in the digital filter window behind the button . We get this window:



Compare the coefficients with the values in this paper. In this box I have used the form

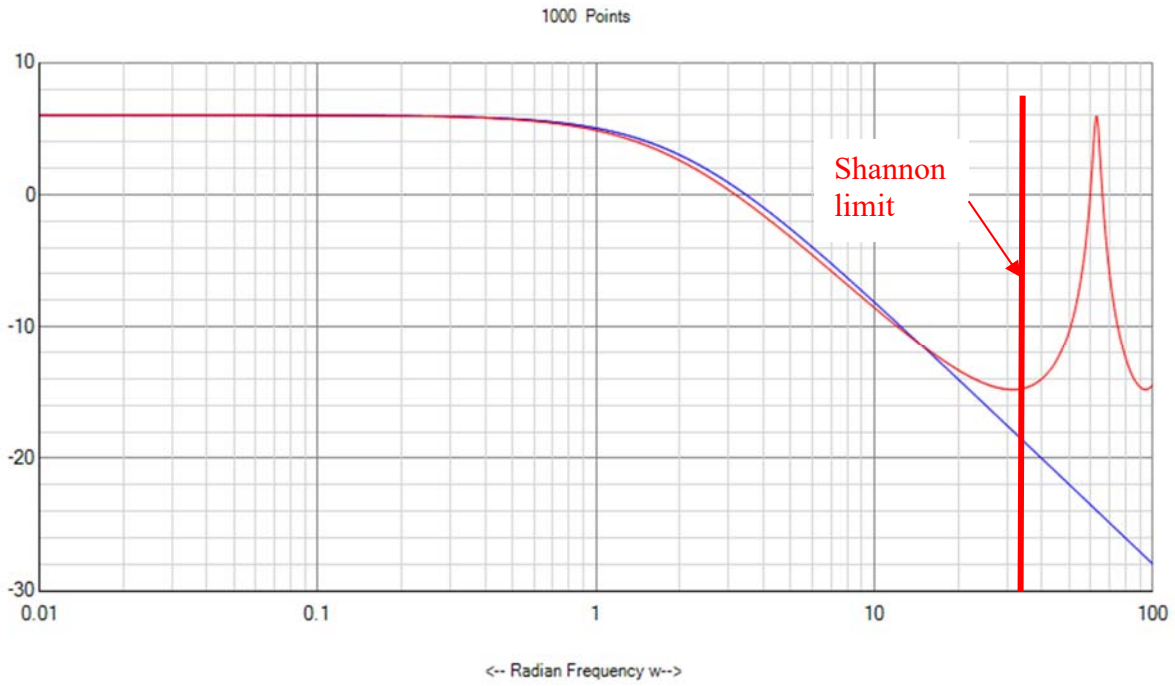
$$F(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_m z^{-m}}{1 + a_1 z^{-1} + \ldots + a_m z^{-m}} z^{-d}$$

which is more common for digital filters, but in control systems the form

$$F(z) = \frac{q_0 + q_1 z^{-1} + \ldots + q_m z^{-m}}{1 - p_1 z^{-1} - \ldots - p_m z^{-m}} z^{-d}$$

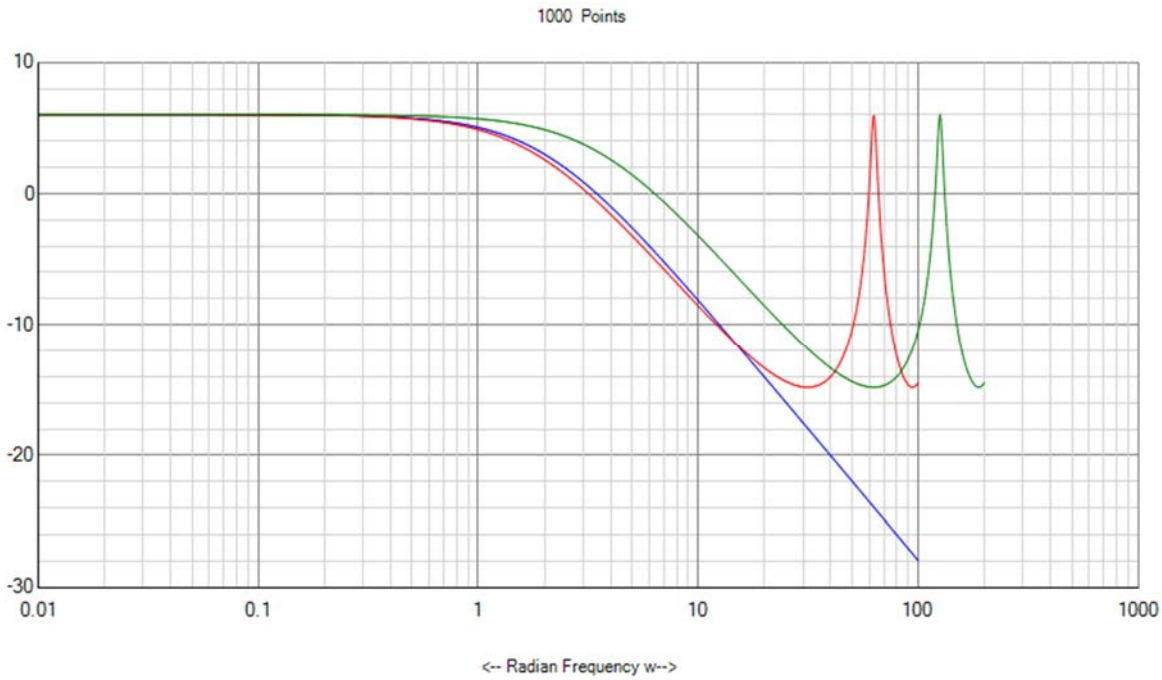
is more common for controllers. The difference is the sign of the coefficients in the denominator.

If we now leave with the |F|- button, the program draws the magnitude curve of this digital filter drawn now in red colour.

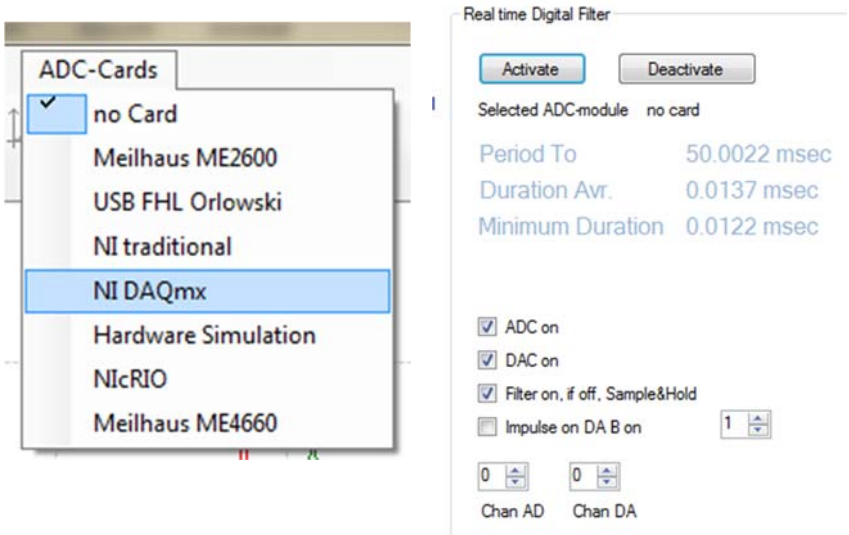


You see, the low frequency part is nearly identical to the continuous filter, but in the right high frequency region there are growing differences. At ω = 63 1/s you get a 6 dB- gain –peak instead of the -23 dB – value of the original filter. This is the sampling frequency $f_0=\omega_0/ 2\pi$ = 10 Hz, we have calculated the filter with T_0=0.1 s. Because we know the Shannon theorem, this digital filter should never be used with signal frequencies larger than 5 Hz, otherwise you get aliasing problems and the signal will be destroyed. I have marked the Shannon limit 5 Hz with red line. (ω = 31.4 1/s).

Now to address the very important difference between „**design sampling time**“ and „**runtime sampling time**“…
The **design sampling time** is that time you have selected in the calculation of the filter behind this button  . This time defines the coefficients a and b.
The **runtime sampling time** is the T_0, with which the filter actually runs. That is the time of the running algorithm. This could be different. This is one of the strongest advantages of digital filters. With the change of the runtime sampling time you can shift the magnitude curve to any position in the bode plot. The shape of bode plot, the filter properties except the corner frequencies remains unchanged. Have both T_0 the same value, analog and digital filter are congruent nearly until the Shannon limit.
If we make the runtime T_0 = 0.05, we double the sampling frequency and get following curves:



The green curve is the new magnitude curve. It is the red curve shifted one octave right.



In the tool program WindfC# you can test this filter, if an ADC- card is available. First select your card in the main menu, then open the digital filter window again and start the running with click on the button “Activate”. Then connect signal generator with the selected AD input and measure the output at DAC channel 0. If it is running, you see blue text which contains actual time measurements of the T_0 and the calculation time, here actual about 12 µs.
You can additionally switch on a DA- 1 V -impulse at a second DA – channel 1. With a scope, a real time measurement is possible. The period of pulse is T_0, the duration the calculation time T_c.

7.4 Improved PIDT1 - controller design with selectable stepdepth

7.4.1 Method

The above design described (design I) has the disadvantage, that the active stepdepth st results firmly from T_V and T_0 with st=1+T_V/T_0. If T_0 is very small, st becomes very large and, joined with that, the starting impulse q_0 is also very large. This caused the fact, that the control value gets very fast into the limitation and the nice correspondence between theory and practice is destroyed.

$$q_0 = K(1+\frac{T_0}{T_I}+\frac{T_D}{T_0}),\quad q_1 = -K(1+2\frac{T_D}{T_0}),\quad q_2 = K\frac{T_D}{T_0} \quad \text{Design I}$$

With the following way using z- transform you can chose the stepdepth freely between 1 and the maximum possible value $st_{\max} = \frac{K}{K_R}\left(1+\frac{T_d}{T_0}\right) = \frac{T_N+T_V}{T_N}+\frac{T_V}{T_0} \approx 1+\frac{T_V}{T_0}$. The change of the differential equation into a difference equation described above can also be accomplished by the Z-transform with the same result. The integration was replaced e.g. through a sum of trapezoidal areas of the width T_0. In the Z-domain this is equal if you replace p thru the bottom expression.

$$p = \frac{2(1-z^{-1})}{T_0(1+z^{-1})} \text{ (Trapezoidal approximation)}$$

But now this transformation is applied not on PID, but on a PIDT1 - controller with

$$F_R(p) = K_R\frac{(1+pT_N)(1+pT_V)}{pT_N(1+pT_1)} = K(1+pT_D+\frac{1}{pT_i})\left(\frac{1}{1+pT_1}\right).$$

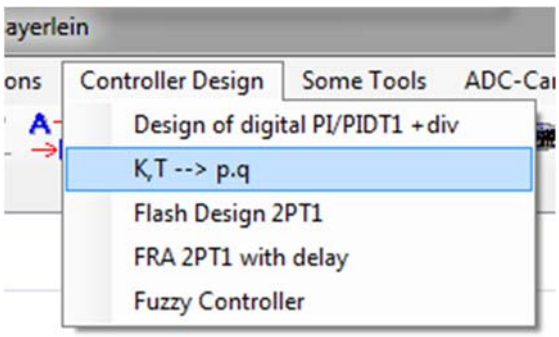
You get the transfer function of a digital PIDT1 - controller with $F_R = \frac{q_0+q_1z^{-1}+q_2z^{-2}}{1-p_1z^{-1}-p_2z^{-2}}$

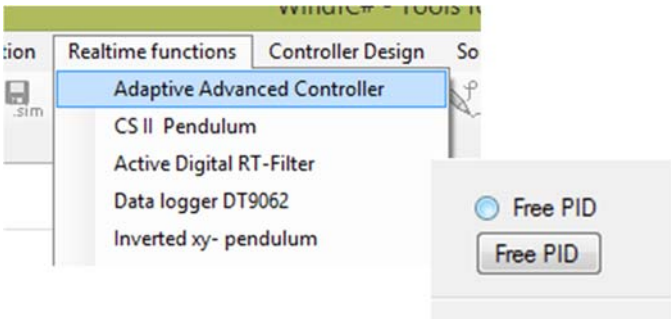
with these coefficients:

$$q_0 = \frac{K_RT_0}{2T_N}\frac{(1+2T_N/T_0)(1+2T_V/T_0)}{(1+2T_1/T_0)},\quad q_1 = \frac{K_RT_0}{2T_N}\left(\frac{(1+2T_V/T_0)(1-2T_N/T_0)+(1+2T_N/T_0)(1-2T_V/T_0)}{(1+2T_1/T_0)}\right)$$

$$q_2 = \frac{K_RT_0}{2T_N}\frac{(1-2T_N/T_0)(1-2T_V/T_0)}{(1+2T_1/T_0)},\quad p_1 = \frac{4T_1/T_0}{(1+2T_1/T_0)} \quad and \quad p_2 = 1-p_1 \qquad \text{Design I\#}$$

The program *WindfC#* uses these calculation formulas in menu "Controller Design → K,T - -> p,q" :





You can play with these controllers in *WindfC#* in the "Advanced control Box" with "Free PID" See the following example:

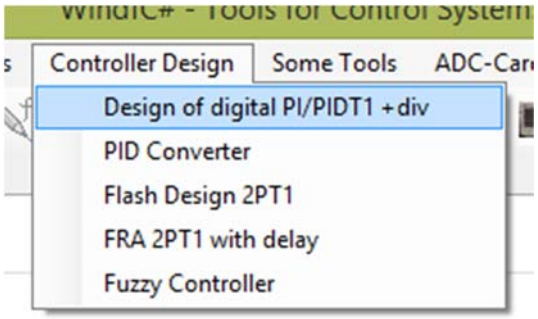
Example: For given 2PT1- process the PID- design now is changed to To=5ms instead of 55 ms. This leads to two new designs (method I and method IV):

Method I: K_c=5.984, st=34, q0= 207.17, q1= -404.54, q2= 197.48, p1=1.

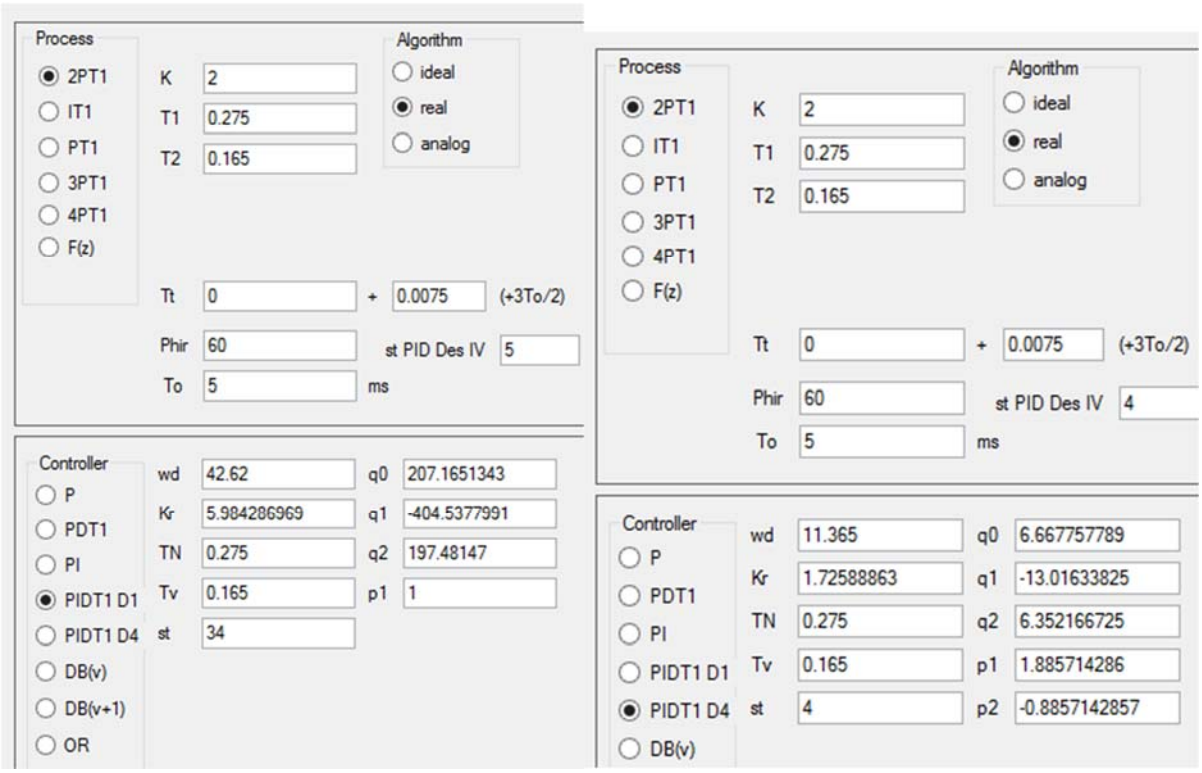
Method IV: K_c = 1.726, st= 4, q0= 6.667, q1= -13.02, q2= 6.352, p1= 1.8857, p2= -0.8857

This design could be done with RegC# or directly with WindfC#, because process is simple 2PT1.

Open this module in WindfC#



See following page:

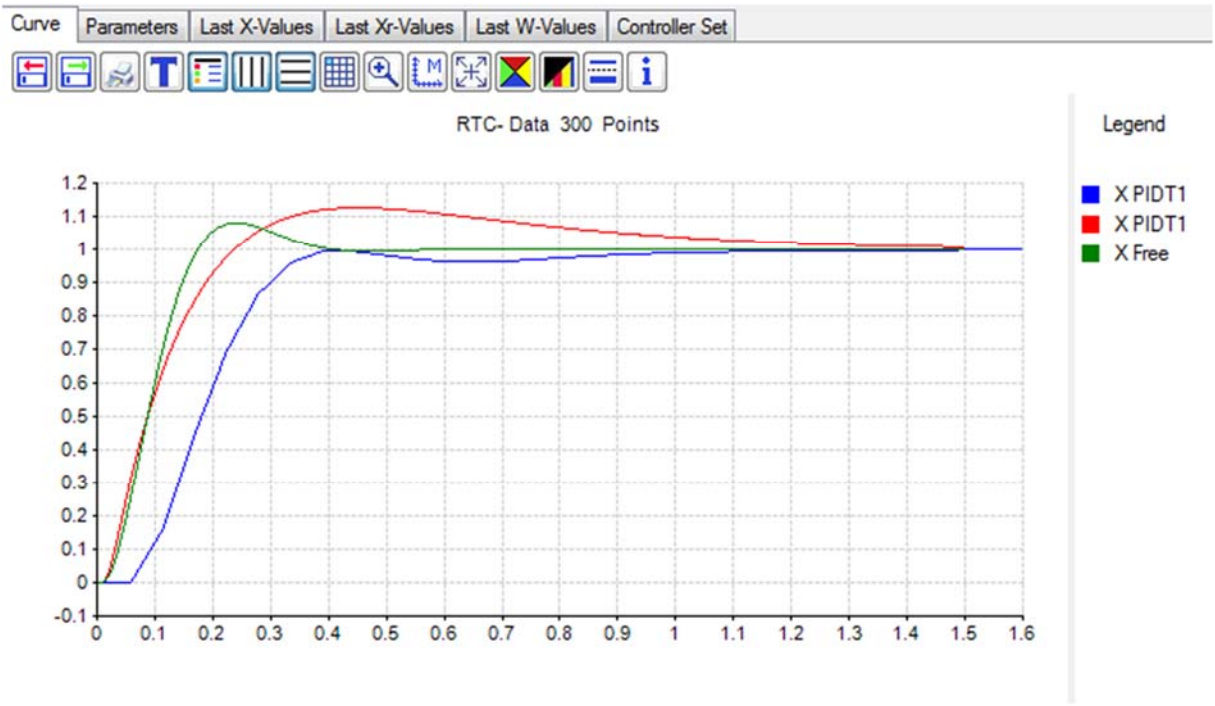


An analogue PIDT1 has a starting impulse at controller output after reference step of K_c*st = 4*2.222 = 8.888.

With the Design I – stepdepth 34, this gives K_c=5.9842 with resulting q of q_0=207.16, q_1=-404.53, q_2=197.48, p_1=1 and p_2=0. This relates to a stepdepth of st_I=st_{max}=34. The starting impulse at controller output after reference step has not the amplitude 8.888, but 207.16. This large impulse caused a nonlinear wind-up – effect.

With the trapezoidal-*Design I#* you get the result: q_0=6.668, q_1-13.017, q_2=6.3526, p_1=1.8857and p_2= 0.8857. Now the starting impulse is only 6.668 instead of 8.888, but very close to this value.

The resulting reference step responses with old PID and both new PIDT1 compared with above curves including control output limitation at 10:



The blue is previous PID Design I with 55 ms. The red is the PIDT1 with 5 ms with design I- you see large overshoot caused by st=34- wind up effect. The green PIDT1- response has a very good behaviour very close to analogue PIDT1-response.
So Design I# gives very good results.

7.4.2 Final Overview of digital PID control algorithms

Ideal algorithm: delay time $T_{del} = T_0/2 + T_c$

$$y(n) = p_1 y(n-1) + p_2 y(n-2) + q_0 x_d(n) + q_1 x_d(n-1) + q_2 x_d(n-2)$$

Real algorithm: delay time $T_{del} = 3T_0/2$

$$y(n) = p_1 y(n-1) + p_2 y(n-2) + q_0 x_d(n-1) + q_1 x_d(n-2) + q_2 x_d(n-3)$$

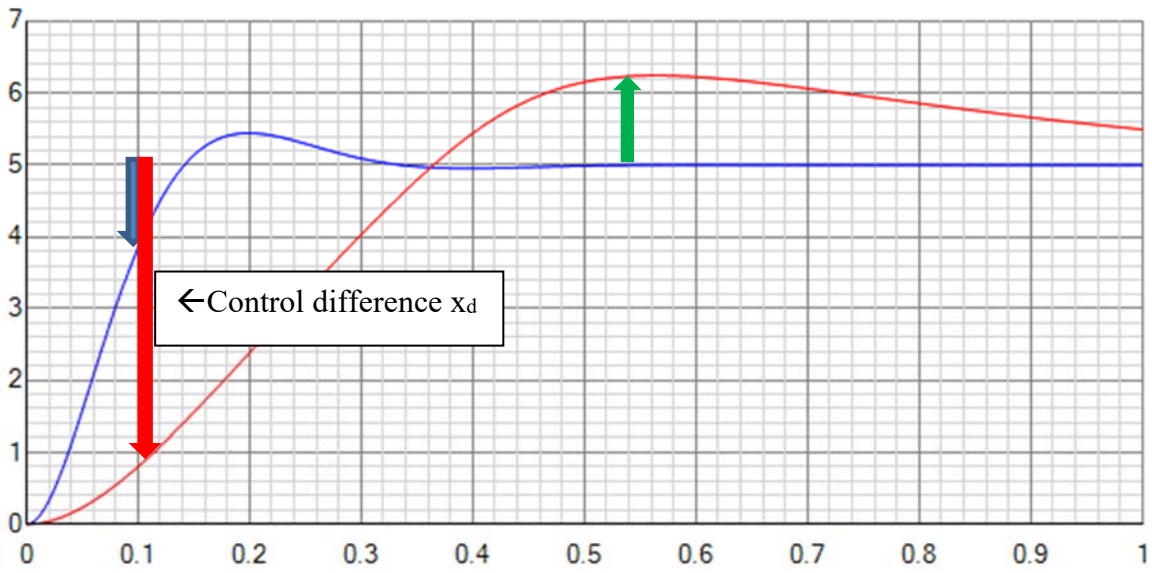
Type	F(p)	q_i, p_i, all missing q_i,p_i=0	remarks
P	K	q_0=K	non recurs.
P	K	q_0=K, q_1=-K, p_1=1	recursive
PI	$K_R(1+pT_N)/pT_N$	$q_0=K_R(1+T_0/T_N)$, $q_1=-K_R$, p_1=1	recursive
PD	$K(1+pT_V)$	$q_0=K(1+T_V/T_0)$, $q_1=-KT_V/T_0$, st≈$1+T_V/T_0$	non recursive

PD	$K(1+pT_V)$	$q_0=K(1+T_V/T_0)$, $q_1=-K(2T_V/T_0+1)$, $q_2=KT_V/T_0$, $p_1=1$, $st\approx 1+T_V/T_0$	recursive
PID	$K(1+pT_D+1/pT_i)=$ $\frac{K_R(1+pT_N)(1+pT_V)}{pT_N}$	$q_0=K(1+T_D/T_0+T_0/T_i)$, $q_1=-K(2T_D/T_0+1)$, $q_2=KT_D/T_0$, $p_1=1$, $st\approx 1+T_V/T_0$	recursive Design I
PIDT1	$K(1+pT_D+1/pT_i)/(1+pT_1)$, $\frac{K_R(1+pT_N)(1+pT_V)}{pT_N(1+pT_1)}$ $st_{max}=1+T_V/T_0$,	$q_0=b(1+2\frac{T_N}{T_0})(1+2\frac{T_V}{T_0}),\ b=\frac{K_R T_0}{2T_N(1+2T_1/T_0)}$ $q_1=b\left((1-2\frac{T_N}{T_0})(1+2\frac{T_V}{T_0})+(1+2\frac{T_N}{T_0})(1-2\frac{T_V}{T_0})\right)$ $q_2=b(1-2\frac{T_N}{T_0})(1-2\frac{T_V}{T_0}),$ $p_1=\frac{4T_1/T_0}{(1+2T_1/T_0)}\ and\ \ p_2=1-p_1$	recursive Design I# trapezoids,
PDT1	$\frac{K_R(1+pT_V)}{(1+pT_1)}$ $st_{max}=1+T_V/T_0$,	$q_0=b(1+2\frac{T_V}{T_0}), b=\frac{K_R}{(1+2T_1/T_0)}$ $q_1=b(-4\frac{T_V}{T_0}), q_2=b(-1+2\frac{T_V}{T_0}),$ $p_1=\frac{4T_1/T_0}{(1+2T_1/T_0)}\ and\ p_2=1-p_1$	recursive Design I# trapezoids,

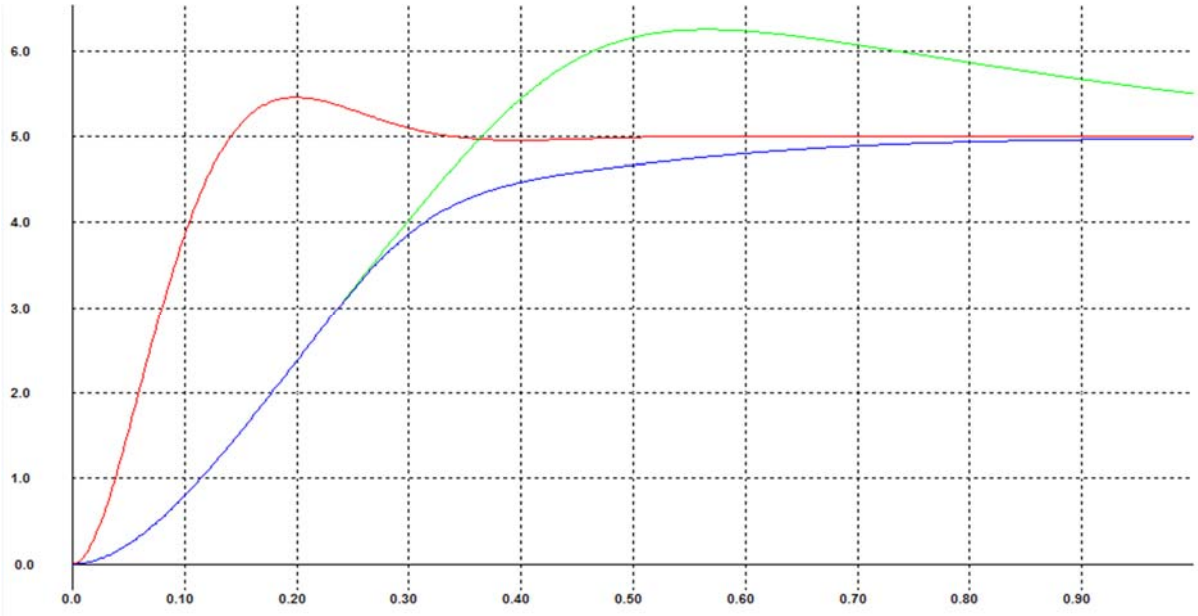
7.4.3 Anti-Wind-Up- Mechanism

If controller has I- part and the controller output during controlling reaches its limitation, then wind- up – effect could occur.

Example: we use above introduced 2PT1-process now with $K_S=1$, $T_1=0.275$ s and $T_2=0.165$s. Step amplitude 5. $K_c=4.444$ with st=4 and polecompensation. Resulting reference step response without (blue) and with limitation (red) to 10:



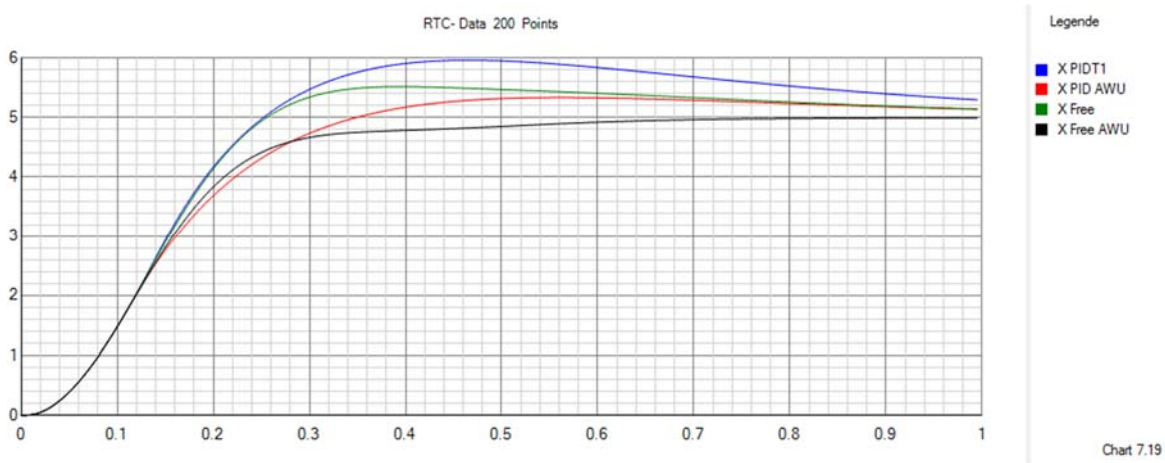
You see a large overshoot. This comes from the fact, that the integrator of PID gets longer a control difference x_d and integrating effect has a larger output. This wrong –I- output must be removed with a longer negative x_d (see green arrow). This overshoot can be avoided with an "Anti-Wind-Up- Mechanism" = AWU. The idea is simple: switch off integrator if PID-output reaches the limitation.
Result (sorry, but Regc# has a bug, so this curve comes with Regdelph)



Other colors: red unlimited, green limited and blue limited with AWU, the overshoot is removed.
In software this can be done in the following way with PID design I:

```
yout = p1i * yold + q0i * xd0 + q1i * xdold1 + q2i * xdold2;
if ((yout < -limit_low) || (yout > limitation))
{
     yout = yout - (q0i + q1i + q2i) * xd0;  // Subtract I- part
}
```

The I- part is hidden in q0i. The sum of the q results in just this I- part and you can simply subtract it without changing the yold - value. In digital PID this looks like next screen:



With AWU overshoot is smaller or completely removed.

8 Identification of processes

8.1 Identification with characteristic values

This method uses the step response of the process / system and looks for characteristic points / properties of it. Here you can find for 4 typical simple systems (PT1, IT1, PT2 and PTn) some methods. It is assumed, that an offset voltage is subtracted before using these equations. So y(t<0) = 0. Input step amplitude is U_0.

8.1.1 PT1:

The step response of the PT1- block follows the equation $y(t) = K(1-\exp(-t/T))$, if the transfer function is $F(p) = K/(1+pT)$. The both unknown parameters K and T should now be determined through the step response. If the final value fv is reached (measuring time >5*T), then K=fv/U_0. The time constant can be derived with three methods: Either with the tangent method: The tangent drawn at t=0 cuts the final value at T. Or secondly with the 63% - point: The value of y(T)=0.6321*fv, with other words the value of the step response at time T is 63% of the final value. Or use the 10%-90%- rise time. Some scopes automatically can measure the rise time t_{1090} of pulses, which is the time between the 10% and the 90% value of the final value. T can then be calculated with T=t_{1090}/ln9= 0.4551*t_{1090}.
If the final value is not reached, then you can use the two –point method. Look for the following two points $P_1(t_1,y_1)$ and $P_2(t_2,y_2)$ of the step response: P_2 has a y_2 which is a value about 10% below the maximum value to avoid saturation effects. Now P_1 is chosen with $t_1=t_2/2$. Now the desired parameters are calculated with

$KU_0 = \dfrac{y_1}{2-(y_2/y_1)}$ and $T = -t_2/\ln(1-y_2/KU_0)$. ***See Tool in WindfC#: Menu "Identification" → "PT1 with 2 points".***

8.1.2 PT2:

A PT2 has the transfer function $F(p) = \dfrac{K}{1+2dp/\omega_0 + p^2/\omega_0^{\,2}}$, K, d and ω_0 are to be identified. For this, it is necessary that the final value of the step response is reached closely enough. Then with the first relative overshoot ü you can immediately take advantage of the relations $d = (-\ln ü)/\sqrt{\pi^2 + (\ln ü)^2}$ and $\omega_0 = \pi/t_{\max}\sqrt{(1-d^2)}$, which are described in the workbook p. 17 . The amplification K results in K= fv / U_0. ***See tool in WindfC# : Menu "Some Tools" → PT2-equations.***

8.1.3 IT1:

An IT1 - block has the transfer function $F(p) = K/p(1+pT)$. For this, it is necessary that the step response is measured already in the linear increase region (t>>T). If you now take the last part (e.g. last half) of the step response and extrapolate this part linearly to the beginning, then this tangent will cross the t-axis exactly at T. K can immediately be computed to K=A/U_0, if A=Δy/Δt is the gradient of the tangent and U_0 is the input step amplitude. No WindfC#-tool.

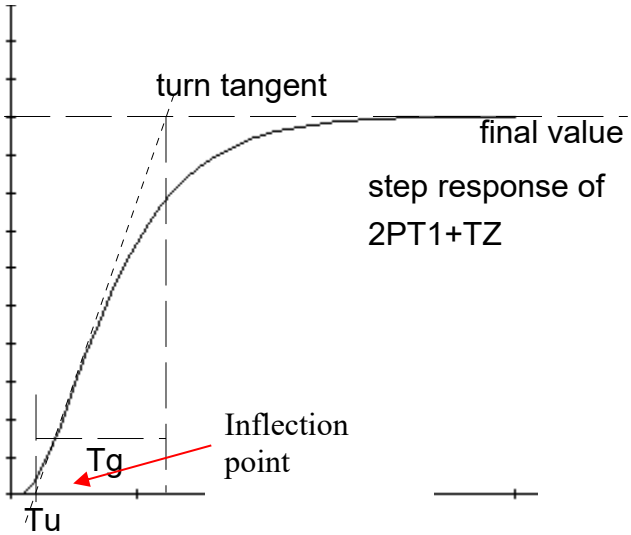
8.1.4 PTn :

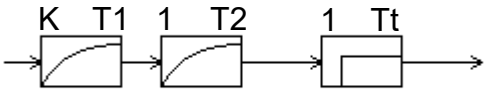
If the process contains no Integrator plus a series of two or more PT1, then the process can be approximated with 2PT1 and an additional delay time (2PT1+TZ). A process identification can easily be implemented with the following inflection tangent (tangent at the point of

inflection) construction, which also can be done automatically with a computer if the noise is not too large.

Procedure:

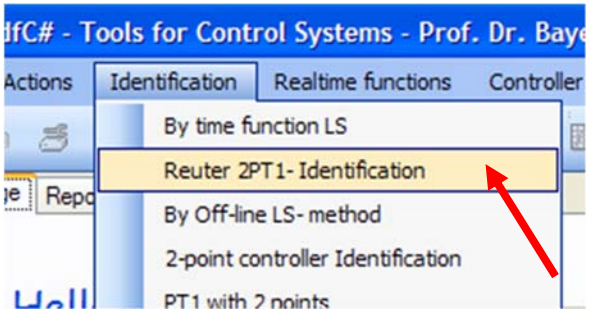
1. If the process can be approximated with 2PT1 + delay time block, the step response should look like this:



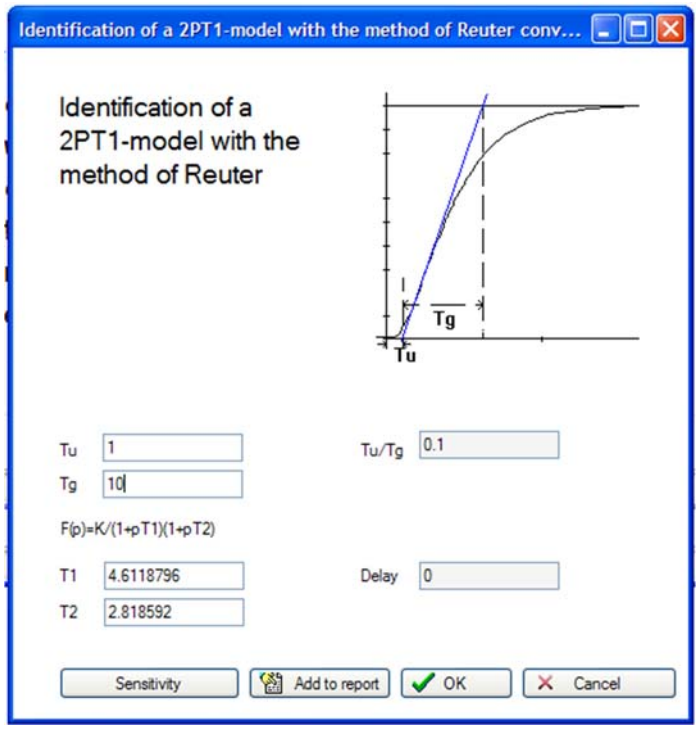


$K=fv / U_0$.

2. Construction of the turn tangent. Determination of Tu and Tg. ***Using tool in WindfC#: Menu "Identification" → "Reuter 2PT1-Identification"***



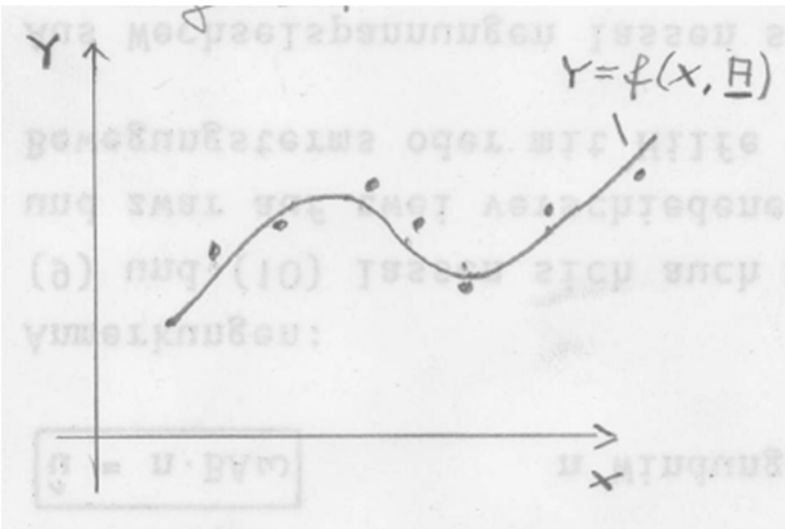
Type in Tu and Tg and get the resulting times T1, T2 and sometimes delay.



8.2 Identification with least square optimization

8.2.1 Method

If step response is available as a file or data array, then you can use this next method. In general this method can find the best set of parameters A of a given equation y=f(x,A). A is a series of unknown parameters A = (A_1, A_2, ….). You should have some measuring points with the coordinates (x_1,y_1), (x_2,y_2) … The number must be larger than number of parameters.



An example is the step response of a PT1: $y(x) = K(1-\exp(-x/T))$, where the x- axis is the time t and A= (K,T).
To get a good set of A, the criterion

$$(Gl.8.1) \qquad s = \sum_{i=1}^{n} (y_i - f(\underline{A}, x_i))^2$$

indicates the quality of the model and must be minimized. This leads to the name: find the least squares, so Least Square Method (LS). The best set of A has the smallest s.
The minimum is defined at that set of parameters A where all derivatives ds/dA_i are zero. Normally the equation f(x) is nonlinear, so this could not be solved directly. So in our PT1-example T is in the denominator of the exponential function: nonlinear!
Solution with the Gauss- Newton- method:
In this case the nonlinear equation f(x) is developed around a starting point $A^{(l)}$. The approximation looks like this:

$$(Gl.8.2) \qquad y = f(x,\underline{A}) = f(x,\underline{A}^{(l)}) + \frac{\partial f(x,\underline{A})}{\partial A_1}\Big|_{A^l} (A_1 - A_1^{(l)}) + \frac{\partial f(x,\underline{A})}{\partial A_2}\Big|_{A^l} (A_2 - A_2^{(l)}) + ...$$

In our example A_1=K an A_2=T. The first derivatives to A_1 and A_2 are:

$$\frac{\partial f(x,\underline{A})}{\partial A_1} = 1 - \exp(-x/T) \text{ and } \frac{\partial f(x,\underline{A})}{\partial A_2} = -\frac{Kx}{T^2}\exp(-x/T)$$

Now put 8.2 into 8.1 and you get:

$$S = \sum_{i=1}^{n} \left\{ Y_i - f(x_i, \underline{A}^{(l)}) - \frac{\partial f(x,\underline{A})}{\partial A_1}\Big|_{\underline{A}^{(l)}, x_i} (A_1 - A_1^{(l)}) - \dots - \frac{\partial f(x,\underline{A})}{\partial A_M}\Big|_{\underline{A}^{(l)}, x_i} (A_M - A_M^{(l)}) \right\}^2$$

Now derive s to the single parameters:

$$\frac{\partial S}{\partial A_1} = 0 = \sum_{i=1}^{n} 2\{\dots\} \cdot \left(- \frac{\partial f(x,\underline{A})}{\partial A_1}\Big|_{\underline{A}^{(l)}, x_i} \right)$$

$$\vdots$$

$$\frac{\partial S}{\partial A_M} = 0 = \sum_{i=1}^{n} 2\{\dots\} \left(- \frac{\partial f(x,\underline{A})}{\partial A_M}\Big|_{\underline{A}^{(l)}, x_i} \right) .$$

With the abbreviations

$$q_j^{(l)} = \sum_{i=1}^{n} \left\{ \left[Y_i - f(x_i, \underline{A}^{(l)}) \right] \cdot \frac{\partial f(x,\underline{A})}{\partial A_j}\Big|_{\underline{A}^{(l)}, x_i} \right)$$

and

$$A_{jk} = \sum_{i=1}^{n} \left\{ \frac{\partial f(x,\underline{A})}{\partial A_j}\Big|_{\underline{A}^{(l)}, x_i} \cdot \frac{\partial f(x,\underline{A})}{\partial A_k}\Big|_{\underline{A}^{(l)}, x_i} \right\}$$

You can write the M equations with matrices:

$$\begin{pmatrix} q_1^l \\ q_2^l \\ \\ q_M^l \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & & A_{1M} \\ A_{21} & A_{22} & & A_{2M} \\ & & & \\ A_{M1} & & & A_{MM} \end{pmatrix} \begin{pmatrix} A_1 - A_1^l \\ A_2 - A_2^l \\ \\ A_M - A_M^l \end{pmatrix}$$

The q_i and the A_{ij} – values can be calculated with the starting values of $\underline{A}$ and the measured points. The A_i are the new estimated parameters and the A_i^l are the starting values or old parameters. The above matric equation can be solved:

$$\begin{pmatrix} A_1 \\ A_2 \\ \\ A_M \end{pmatrix} = \begin{pmatrix} A_1^l \\ A_2^l \\ \\ A_M^l \end{pmatrix} + \begin{pmatrix} A_{11} & A_{12} & & A_{1M} \\ A_{21} & A_{22} & & A_{2M} \\ & & & \\ A_{M1} & & & A_{MM} \end{pmatrix}^{-1} \begin{pmatrix} q_1^l \\ q_2^l \\ \\ q_M^l \end{pmatrix}$$

So a matric – Inversion is necessary, but not a problem, because several algorithms are available since years.
Convergence is possible (the new sum of squares with the new parameters is smaller than the previous one), if starting values are near to the valley (minimum).
If the function f(x) is a parabolic function like $f(x) = A_1+A_2x+A_3x^2+A_4x^3+ \dots$ then convergence is guaranteed in one step.
Example:
A PT1- step response is measured in the file SRtestidLS.sim. The content of this file is with a blank separator:

0 0
1 0.78694
2 1.2642
3 1.5537
4 1.7293

So we have 5 points, sampling time is 1 sec. These points are generated with the PT1 with the original parameters K=2 and T=2. The starting parameter set should be K=1.5 and T=1.5 with the square sum s=0.20586.

In the first step we get the following numbers:

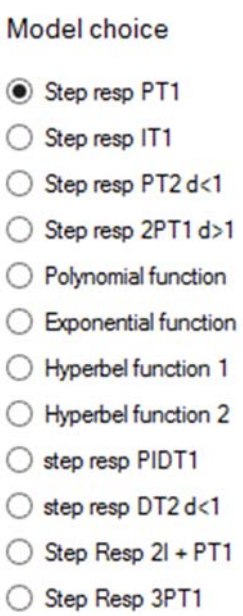
$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix} + \begin{pmatrix} 2.39256 & -0.83182 \\ -0.83182 & 0.34827 \end{pmatrix}^{-1} \begin{pmatrix} 0.67761 \\ -0.20690 \end{pmatrix}$$

$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix} + \begin{pmatrix} 2.463994 & 5.88499 \\ 5.88499 & 16.927 \end{pmatrix} \begin{pmatrix} 0.67761 \\ -0.20690 \end{pmatrix}$$

$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix} + \begin{pmatrix} 0.45198 \\ 0.48543 \end{pmatrix} = \begin{pmatrix} 1.95198 \\ 1.98543 \end{pmatrix}$$

So after one step the new parameters are K= 1.95198 and T=1.98543. The square sum is now s=0.003313, a really good improvement.

In the program WindfC# there are 12 prepared functions including 8 step responses and one parabolic function up to the order of 6. The list can be seen in the next picture.



If you want to add an additional function load the source code of Windfc# and use the MS Studio to compile. The module is called "FRegression.cs".
Add a *RadioButton* in the group "*model Choice*", in the *CheckedChange*- Event set the relevant information (e.g. max number of Parameters) and the text- labels and give a name to *regrtype*. Then add in the function "`funk_ableit`" the if – statement for your function. You need to program function and derivatives in C, that's all.
Behind the "Init"- Button you can find a module which can find good starting values. I have implemented the functions described in Chapter 8.1 for the step responses and add two methods for the step responses of PIDT1 and DT2. The other functions have no init- function, you have to do this manually.

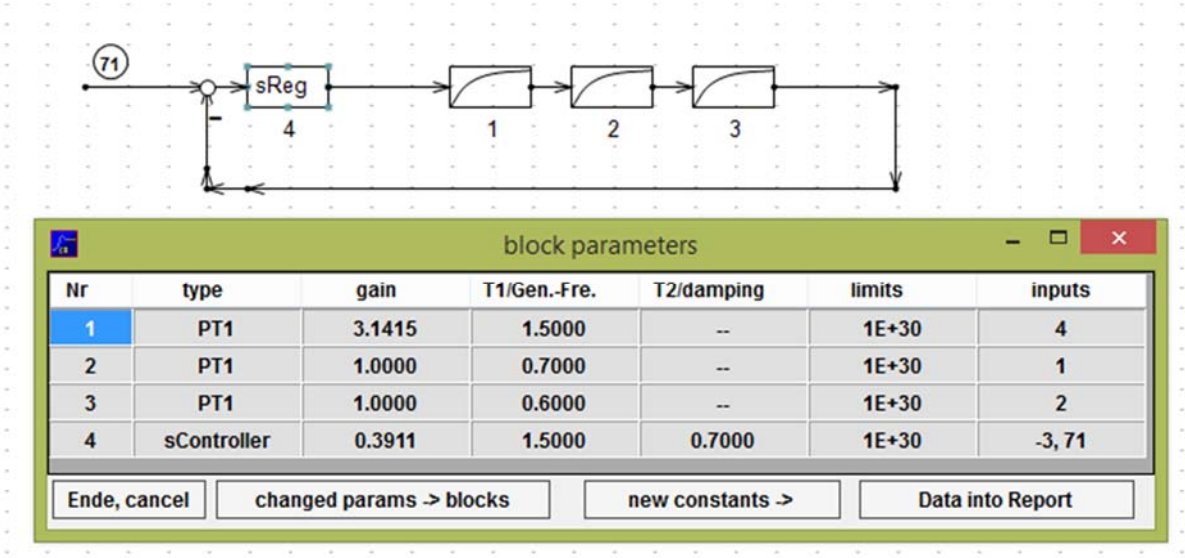
Name	F(p)*exp(-pT$_{del}$)	f(t)+U$_{off}$
PT1	$F(p)=\frac{K}{1+pT}$	$y=K\left[1-\exp(-t/T)\right]$
IT1	$K/p(1+Tp)$	$KT\left(e^{-t/T}+\frac{t}{T}-1\right)$
PT2,d<1	$\frac{K}{1+2dTp+T^2p^2}$, $\omega_0=1/T$	$K-\frac{K\exp(-dt/T)}{\sqrt{1-d^2}}\cdot\sin\left(\frac{\sqrt{1-d^2}}{T}t+\Phi\right)$, $\tan\Phi=\frac{\sqrt{1-d^2}}{d}$,
2PT1	$\frac{K}{(1+T_1p)(1+T_2p)}$	$1+\frac{1}{T_2-T_1}\left(T_1e^{-t/T_1}-T_2e^{-t/T_2}\right)$
Poly	./.	$y=a+bx+cx^2+dx^3+ex^4+fx^5+gx^6$
Exp	./.	$y=K_1\exp(K_2x)+K_3$
Hyperbel 1	./.	$y=\frac{a+c*x+d*x^2}{1+b*x}$
Hyperbel 2	./.	$y=\frac{a+d*x+e*x^2}{1+b*x+c*x^2}$
PIDT1	$F(p)=K\frac{1+1/pT_I+pT_D}{1+pT_1}$	$h(t)=K\left[\frac{t-T_1}{T_I}+1+\left(\frac{T_D}{T_1}-1+\frac{T_1}{T_I}\right)\exp(-t/T_1)\right]$
DT2	$F(p)=\frac{U_2}{U_1}=\frac{Kp}{1+\frac{2d}{\omega_0}p+\frac{1}{\omega_0^2}p^2}$	$h(t)=\frac{K\omega_0}{\sqrt{1-d^2}}e^{-d\omega_0 t}\sin\omega_E t$ $\omega_E=\omega_0\sqrt{1-d^2}$

2I+PT1 and 3PT1 – equations are not displayed in table.

8.2.2 Example for Controller design purpose

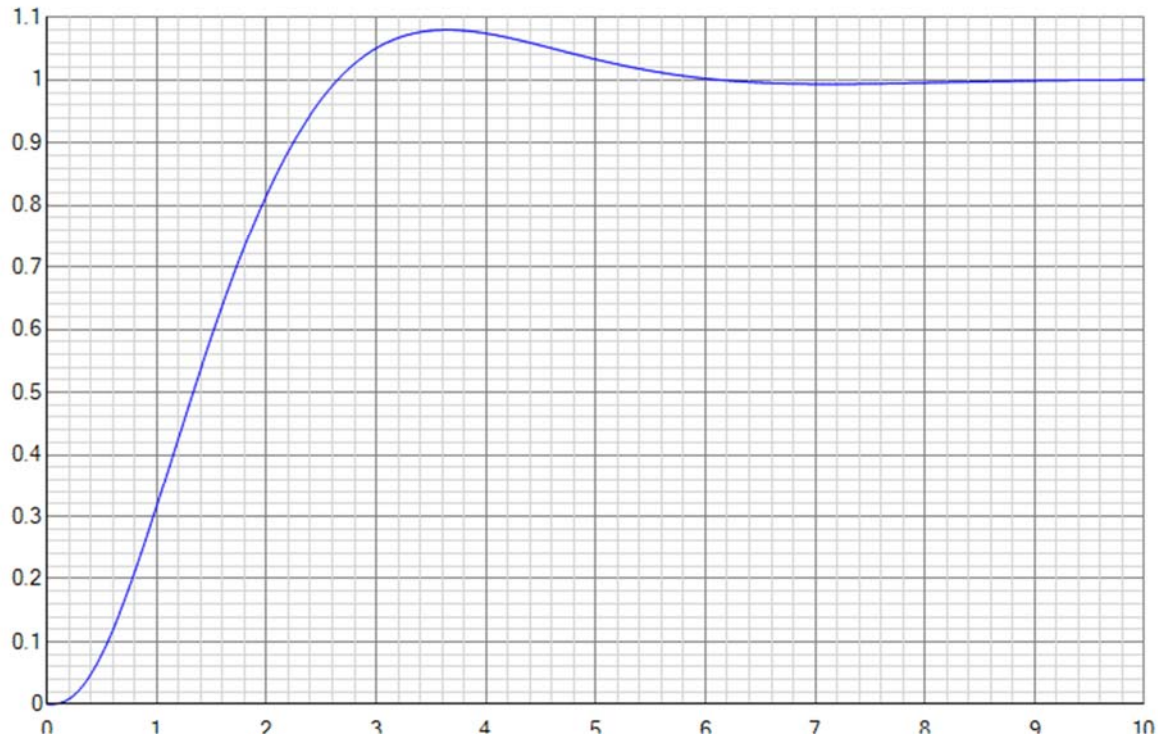
Now a 3PT1- process should be identified via step response and used to design a controller, which is tested at the original 3PT1- process. The data of the 3 PT1 are K=3.1415, T1=1.5, T2=0.6 and T3=0.7. The step response is created with program RegC# and is stored in the file *3pt1SA_10sec.sim.* This file can be loaded with the identification module in WindfC# via menu "Identification →time function LS". The identification should be set to 2PT1, try Init-Button, set "number of parameters" to 5 and then you should get the following result:

First design of PIDT1 at original 3PT1- process with st=5, phase margin =60° and polecompensation T_N=1.5 and T_V=0.7.

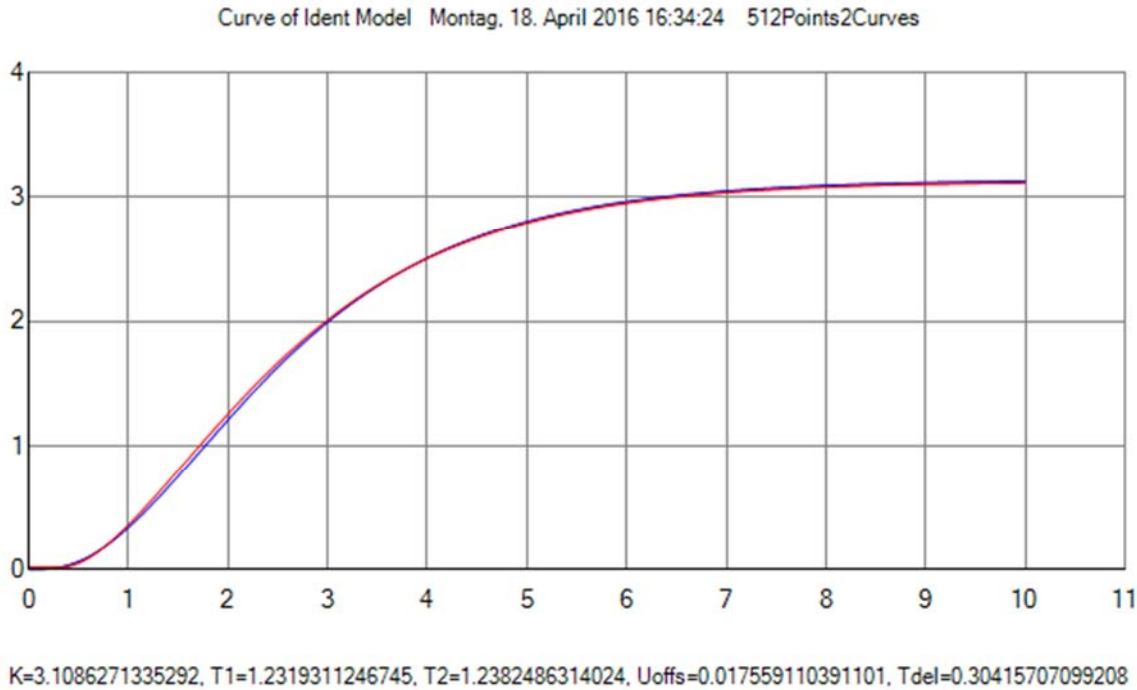


Nr	type	gain	T1/Gen.-Fre.	T2/damping	limits	inputs
1	PT1	3.1415	1.5000	--	1E+30	4
2	PT1	1.0000	0.7000	--	1E+30	1
3	PT1	1.0000	0.6000	--	1E+30	2
4	sController	0.3911	1.5000	0.7000	1E+30	-3, 71

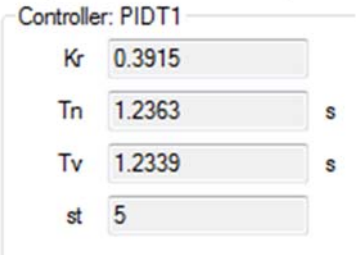
The reference step response of a controller designed with this process and st=5 and phase margin=60° has the above data. The response looks like the curve in the next picture:

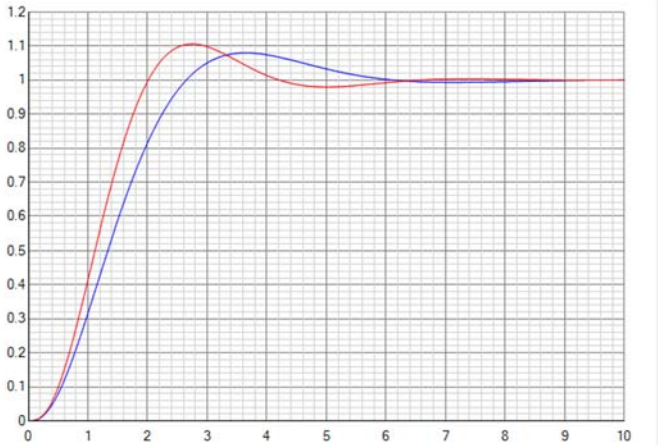


Overshoot = ü= 8%, settling time with 5% tolerance range is about 4.5 sec.
Now identification with the above process step response:



With this design a second PIDT1 using tool in menu "Controller design → FRA 2PT1 with delay" has the resulting values

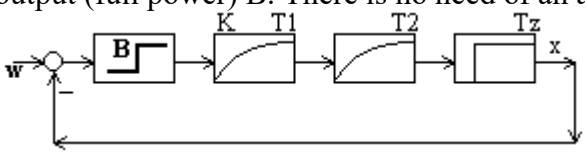




Red: with this model
Blue: Previous original controller
Result: overshoot a little bit larger, but faster.

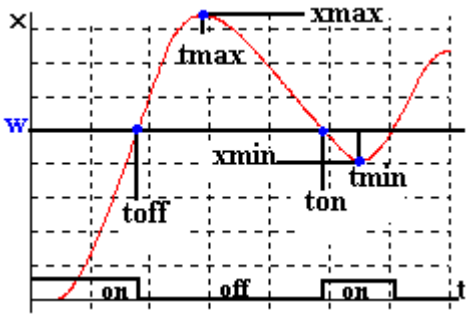
8.3 Identification using Two-Point-Controller response

This method can identify only processes with the type 2PT1with additional delay time block. It is well suited for temperature control systems and is developed at Fachhochschule Lübeck together with a Lübeck company (Gabler) in a Diploma thesis. It uses the response of a two point controller reference step response. The idea is to avoid a step response measurement where the selection of the input step amplitude could be time consuming in the case of very slow temperature processes (time constants of hours!).
The unknown process (no integrator is allowed) is closed with a Two-Point-Controller. The transfer line of this controller has for negative inputs zero output, for positive inputs constant output (full power) B. There is no need of an actuator, only a power switch can do the job.



You have to select a desired value w, with the optimal value B*K/2. This could be the final desired temperature of the system. Then you have to wait for the first maximum and the following minimum. Then the identification can start. You need the coordinates of the four points at the times t_{off}, t_{max}, t_{on} and t_{min}. The x- values of t_{off} and t_{on} are known as w, but the x_{max} and x_{min} values has to be measured, see the left picture.

Theory:



8.3.1 The algorithm

The function for the signal x(t) can be expressed with following three equations:

For $T_z \leq t \leq t_{off} + T_z$ is valid: (for $t<T_z$ is x(t)=0)

$$x_1(t) = B*K*\left\{1+\frac{1}{T_2-T_1}\left[T_1\exp\left(-\frac{t-T_z}{T_1}\right)-T_2\exp\left(-\frac{t-T_z}{T_2}\right)\right]\right\},$$

for $t_{off}+T_z \leq t \leq t_{on}+T_z$ you can write :

$$x_2(t) = \frac{B*K}{T_2-T_1}\left\{T_1\exp\left(-\frac{t-T_z}{T_1}\right)\left[1-\exp\left(\frac{t_{off}}{T_1}\right)\right]-T_2\exp\left(-\frac{t-T_z}{T_2}\right)\left[1-\exp\left(\frac{t_{off}}{T_2}\right)\right]\right\}$$

and for $t \geq t_{on}+T_z$ until next switch off:

$$x_3(t) = BK\left(1+\frac{1}{T_2-T_1}\left\{T_1\exp\left(-\frac{t-T_z}{T_1}\right)\left[1-\exp\left(\frac{t_{off}}{T_1}\right)+\exp\left(\frac{t_{on}}{T_1}\right)\right]+A\right\}\right) with$$

$$A = -T_2\exp\left(-\frac{t-T_z}{T_2}\right)\left[1-\exp\left(\frac{t_{off}}{T_2}\right)+\exp\left(\frac{t_{on}}{T_2}\right)\right]$$

The function $x_2(t)$ has a maximum at t_{max} with the value

$$x_{max} = BK\exp\left(-\frac{t_{max}-T_z}{T_1}\right)\left[\exp\left(\frac{t_{off}}{T_1}\right)-1\right] \quad .$$

The function $x_3(t)$ has a minimum at t_{min} with the value

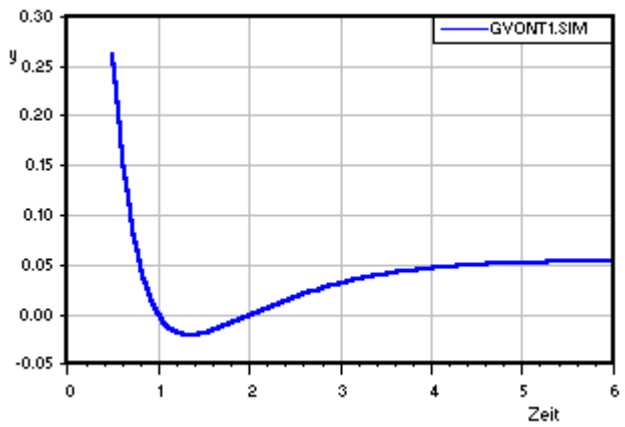
$$x_{min} = BK\left\{1-\exp\left(-\frac{t_{min}-T_z}{T_1}\right)\left[1-\exp\left(\frac{t_{off}}{T_1}\right)+\exp\left(\frac{t_{on}}{T_1}\right)\right]\right\}.$$

Here you have two equations with the three unknowns BK, T_1 und T_z. Now divide x_{max} by x_{min} and BK can be reduced and you have one equation with two unknowns T_1 and T_z.

Start with T_z =0 (no delay) and look for a solution without delay (2PT1-process). Solve the equation $g(T_1)=0$ e.g. with nested intervals. If there is no solution add a delay.

$$g(T_1) = \frac{x_{\max}}{x_{\min}}\left[\exp\left(-\frac{t_{\min}-t_{off}}{T_1}\right)-\exp\left(-\frac{t_{\min}}{T_1}\right)-\exp\left(-\frac{t_{\min}-t_{on}}{T_1}\right)+1\right]$$
$$-\exp\left(-\frac{t_{\max}-t_{off}}{T_1}\right)+\exp\left(-\frac{t_{\max}}{T_1}\right)=0$$

See following numerical example: toff = 1.3863, tmax = 2.1972, ton = 3.5835, tmin = 3.7741, xmax = 1.3333 , xmin = 0.9697. Then the function $g(T_1)$ hast the displayed curve:



You see two zeros at T_1=1 and T_2=2. Because both time constants have absolut the same importance, this gives the solution for both time constants. The solution is difficult, if both time constants have nearly the same value. If the original process has more than two time constants, sometimes no solution is possible, $g(T_1)$ lays completely over the zero-axis. Then you have to add a delay. To look for the zeroes use nested intervals. This always converges, if the starting values are on the left and right side of a zero. It is helpful to look first for the minimum of $g(T_1)$. If this value is negative: OK, if not add a delay. Derive the equation g(T1). With the short expressions $a=t_{min}-t_{off}$, $b=t_{min}$, $c=t_{min}-t_{on}$, $d=t_{max}-t_{off}$ and $e=t_{max}$ and $ea=\exp(-a/T_1)$, $eb=\exp(-b/T_1)$ etc. and $\Lambda=X_{max}/x_{min}$

$g(T_1)= \Lambda(ea-eb-ec+1)-ed+ee$ und

$g'(T_1)= (-1/T_1^2)* \Lambda(a*ea-b*eb-c*ec)-d*ed+e*ee,$

The factor $(-1/T_1^2)$ has no influence to the zero of g'.
Use now following order:

1. Look for a starting Ta, so that g'(Ta)<0
2. Look for an ending Te, so that g'(Te)>0
3. Look for Tm with g'(Tm)=0 using nested intervals.
4. Look for a new Ta>Tm, so that g(Ta)>0
5. Look for a new Te<Tm, so that g(Te)>0
6. Look for T_1 between Te and Tm so, that $g(T_1)$=0 using nested intervals.
7. Look for T_2 between Tm and Ta so, that $g(T_2)$=0 using nested intervals.

Now calculate KB with following equation for the maximum

$$KB = x_{\max} / \exp\left(-\frac{t_{\max}}{T_1}\right)\left[\exp\left(\frac{t_{off}}{T_1}\right)-1\right].$$

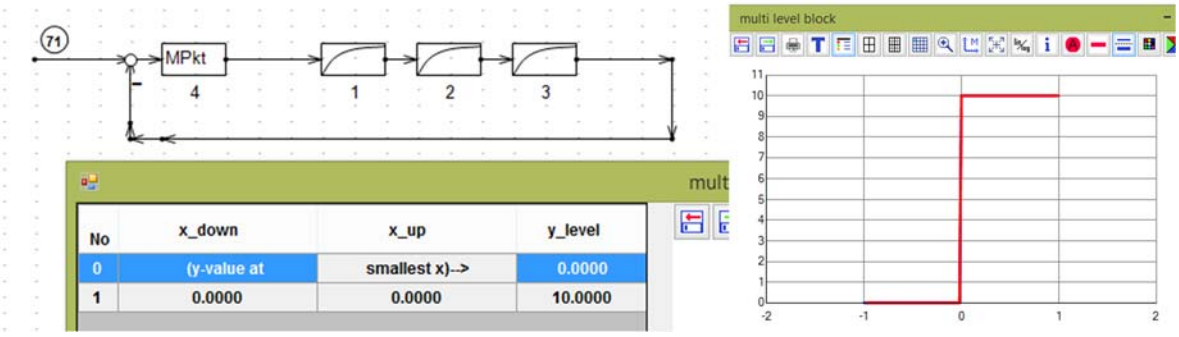
For control purpose you can additionally check the known desired value w with $w=x_1(t_{off})$.

If after point 3 the value g(Tm) >0 there are no zeros. Then add a delay time with iterative trying. It must not be greater than $t_{min}-t_{on}$. Good experience I have made with a starting value of the delay with T_z = dz = ($t_{min}-t_{on}$)/10 and stepwise increase of dz as long as you get success with the search of zeroes. In the equation of x_{max} and x_{min} you can see that the delay

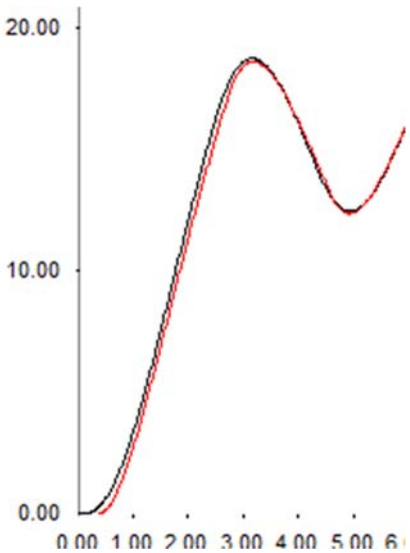
appears only as subtractions of t_{min} and t_{max}. So simply subtract the value dz from these times and repeat the search.

You can find a module in the program WindC# behind the menu “Identification → 2-point-Controller Identification”.

Example: Back to the 3PT1- model. In RegC# add a “Multi- level-block” with the parameters “y-level for small x-values”=0, x_down=0, x_up=0 and y-level =B=10.



Now calculate a reference step response with w=15, nearly half of the value K*B=31.4. Store resulting curve as sim-file with the name *zpr3PT1out.sim.*
Now open the tool in WindfC#. You can find out the four points either in a printout, looking in the file or you can use a “point-finder” in the tool. Loading this file shows the graph with these values:



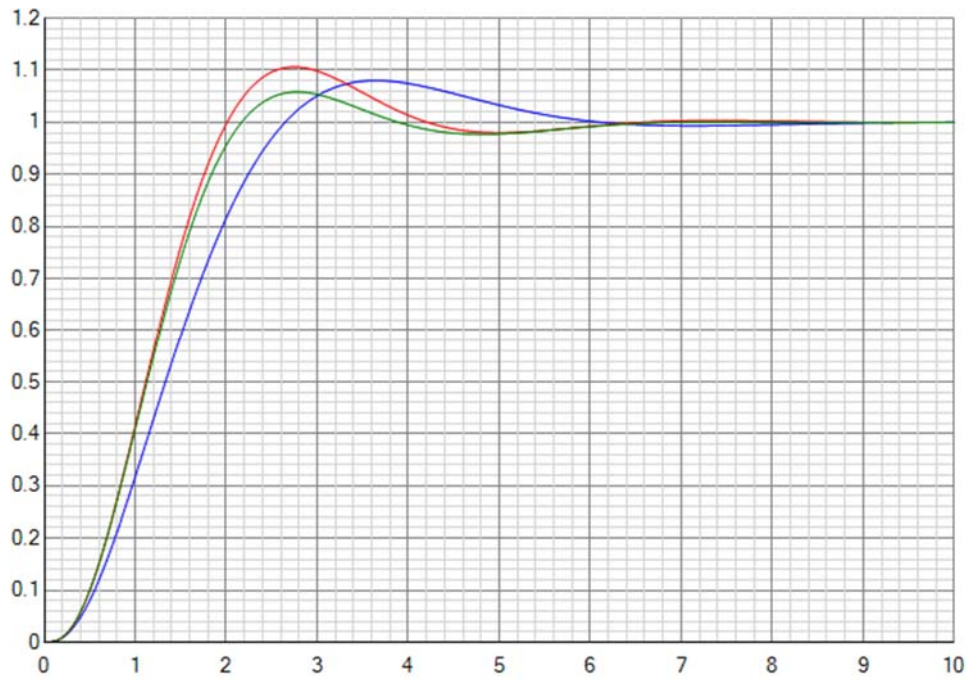
toff	2.34834
tmax	3.15
ton	4.20744
tmin	4.93151
xmax	18.7815
xmin	12.4853

K	3.16664
T1	1.42728
T2	1.18681
Tz	0.362035
w	14.2435

A controller design with these values gives the following numbers and RSR:

Controller: PIDT1

Kr	0.40439	
Tn	1.4273	s
Tv	1.1868	s
st	5	

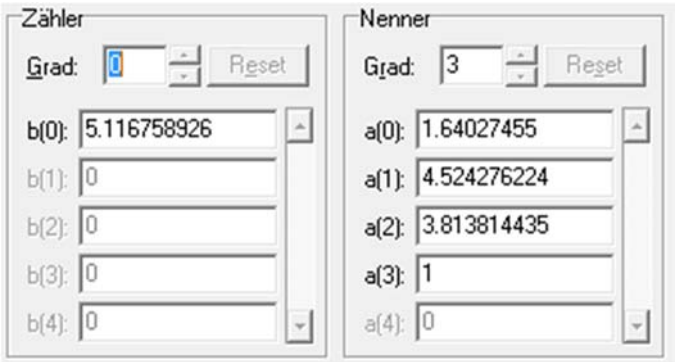


blue: Original controller
red: Step response identification
green: Two-Point-Controller identification.

8.4 Identification with program IDA.exe

This method has the advantage to identify a free transfer function F(p) with any input and the responding output. Input and output signals must start from constant signals (zero initial conditions). Disadvantage: The source code is not available, the program is a commercial one from a German Engineering office Kahlert (www.kahlert.com). The official version is Winfact8, FH-Lübeck version is 6.
I have prepared two versions of signals around the 3PT1- process. First with a reference step response together with the Two-Point-controller the input signal of the process (this is the output of the controller) and output signal of the process are stored in the both files *zpr3PT1in.sim* and *zpr3PT1out.sim*. A second version uses one of the PIDT1- controller, a reference step response has produced the both signals stored in the files *idaPIDprocessin.sim* and *idaPIDprocessOut.sim.*
Now start IDA.exe. The menu language is German. Load input and output files with menu "Datei → Eingangssignal x(t)" and "Datei → Ausgangssignal y(t)". The resulting window with the files *zpr3PT1….* looks like this:

Now in menu "Datei → Steuerparameter" set the "Nennergrad " = Denominator-degree" to 4 and checkBox "n anpassen". Close this window and start

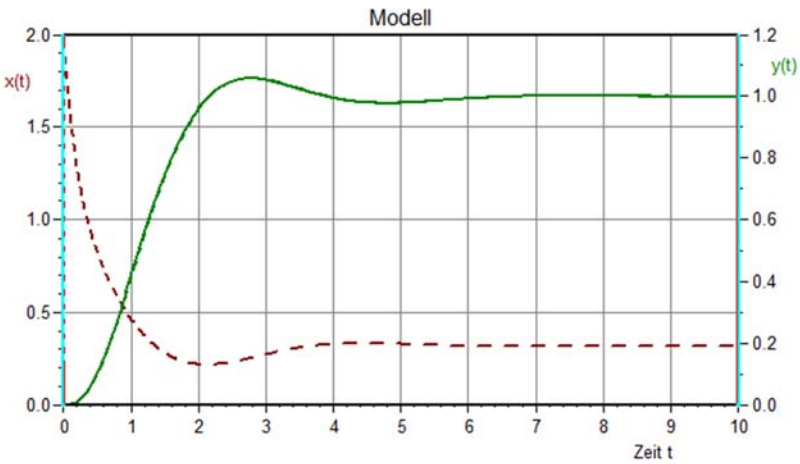


"approximation". Each click on "Weiter" increases the n-value. With degree of 3 you get the

best coincidence between model- response and measured response, with degree 4 there is no success. The resulting transfer function can be displayed and stored in the window with menu "Ausgabe → Übertragungsfunktion anzeigen". You see this result:
Store it with button "UFK- Datei speichern" and you can find a file with the content of the next box:

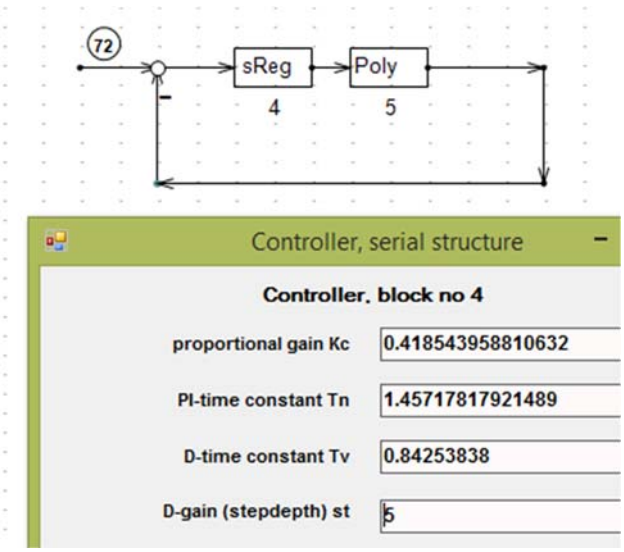
```
!
0
3
 5.11675892600000E+0000
 1.64027455000000E+0000
 4.52427622400000E+0000
 3.81381443500000E+0000
 1.00000000000000E+0000
 0.00000000000000E+0000
```

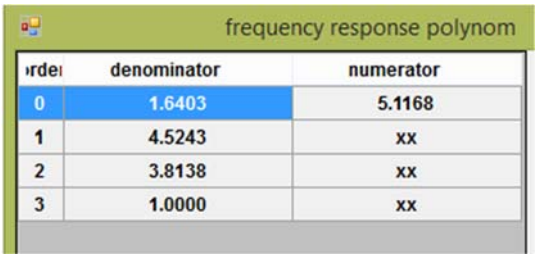
If you do the same job with the files *idaPIDprocess*…. you get similar results.



To check the quality of identification you can import this transfer functions in RegC#, design a controller with these models and check together with the original process.
The import of ufk- files into RegC# is easy with block "Poly".

With the zpr- files I have got this (file *zprresultApril2011.UFK*):





rder	denominator	numerator
0	1.6403	5.1168
1	4.5243	xx
2	3.8138	xx
3	1.0000	xx

With the files *idaPIDprocess* the following data can be calculated using the IDA resulting file "*PIDresultApril2011.UFK*":

frequency response polynom

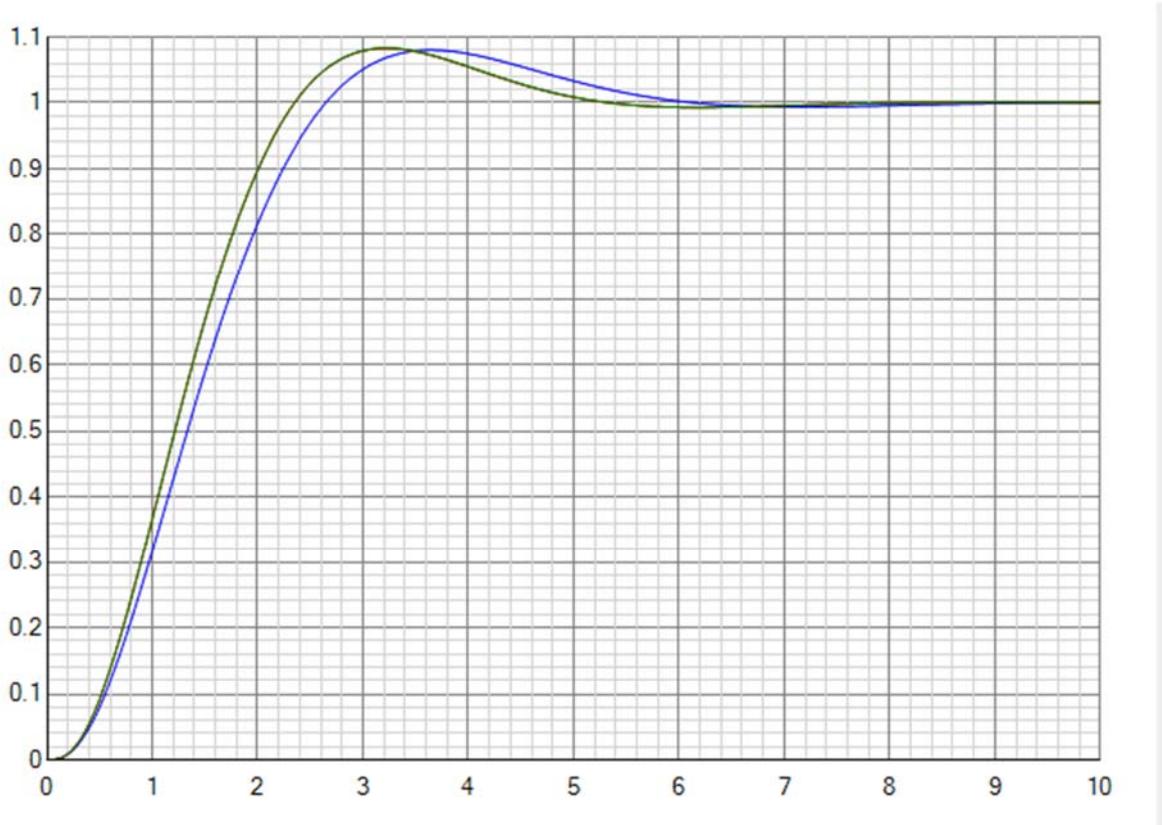
ırder	denominator	numerator
0	1.696655268	5.327023155
1	4.640034543	xx
2	3.918776204	xx
3	1	xx

Controller, serial structure

Controller, block no 4

proportional gain Kc	0.418444898326867
PI-time constant Tn	1.45851799149457
D-time constant Tv	0.8447881
D-gain (stepdepth) st	5

You can see only very small differences. The three reference step responses (Original, zpr-data and PID data) are displayed in the following picture (there are really three curves!). Blue original, red+green IDA-identification.



Last step is the comparison of the three time constants. For this purpose the transfer function F(p) has to be converted into a time-constant form using tool Windfc#- You can load the ufk-files with the ufk- button, then click on button "Factorise" and you get the following results, compared with the original time constants T1=1,5, T2=0.7 and T3=0.6.

T or d
1.4315258
0.78238697
0.54433049

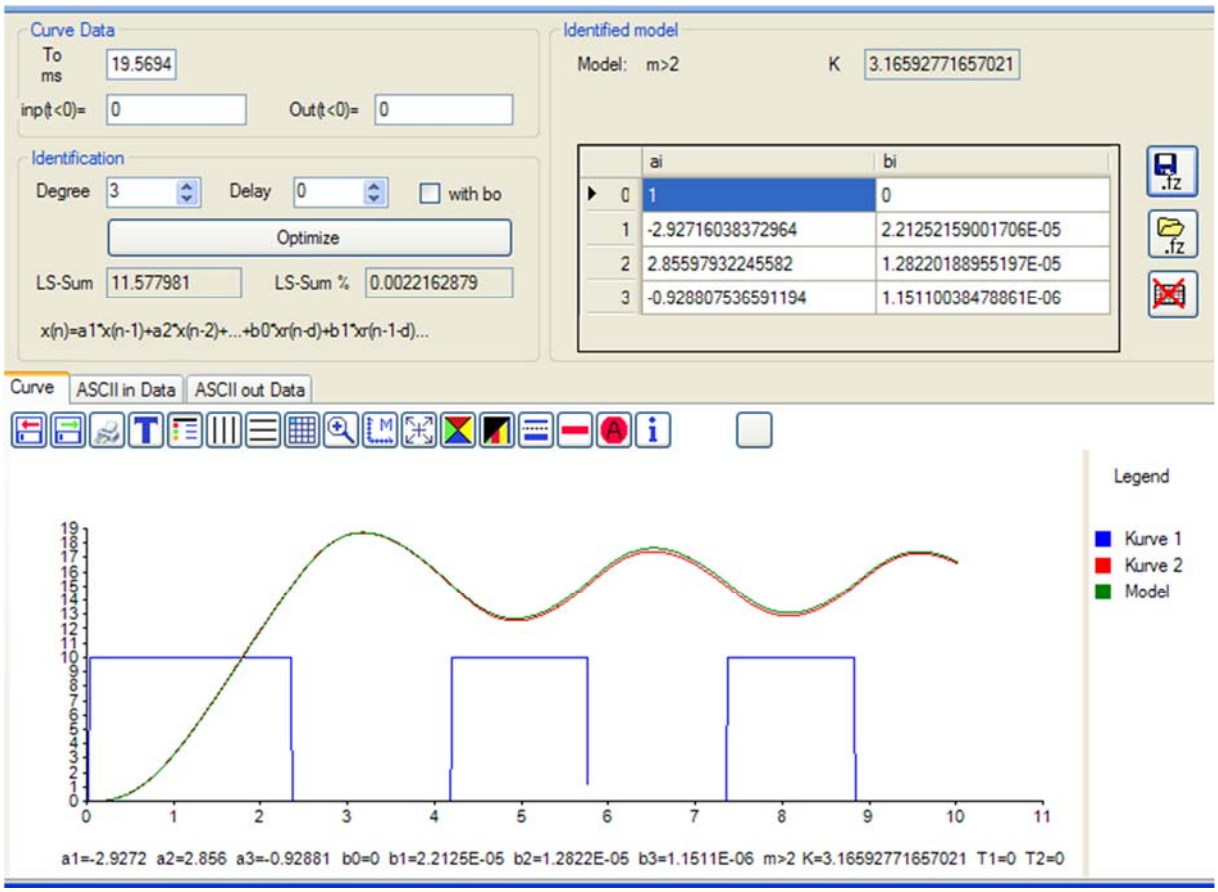
T or d
1.3370105
0.91714963
0.4806527

You can see, that difference in control system behavior is small in spite of the differences in time constants are big, more than 20%!

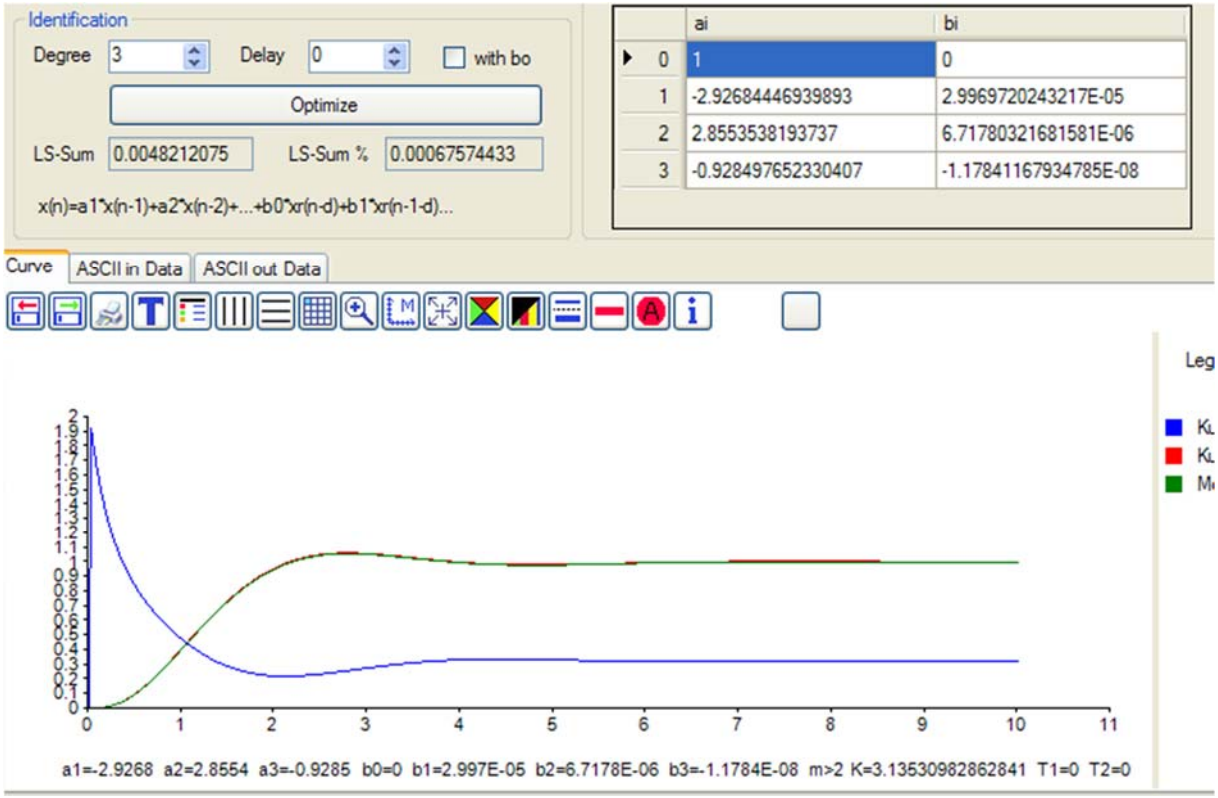
8.5 Identification with LS-Offline

This method is available as tool in WindfC# (menu identification → by Offline LS – method) and - equal to IDA - expects an input –signal of any form and the relating response of the output. These signals can be loaded as file or directly imported from the adaptive control Box. In opposite to IDA this tool calculates the digital filter transfer function F(z).
The input and output signals must be started with zero initial condition, only a constant offset is allowed and can be cleaned.

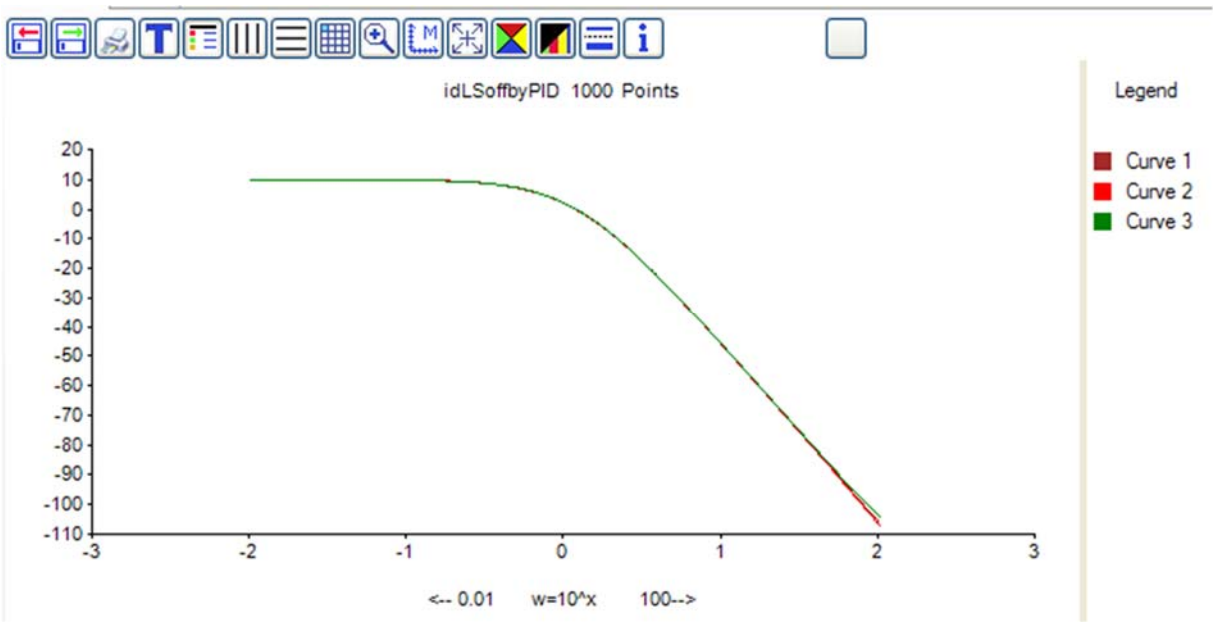
The same files as in IDA can be loaded: the next example starts with the ZPR- files *zpr3PT1in.sim* and *zpr3PT1out.sim*.
Result:



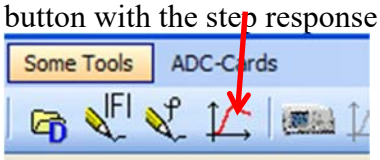
The resulting F(z) is stored as *idLSoffbyzpr.fz*.
Similar result can be achieved with the PID-files *idaPIDprocessin.sim* and *idaPIDprocessOut.sim*.



These both results can be compared as bode plot with the original 3PT1- bodeplot: This results in three nearby identic curves:



This identification should be tested now by a new F(z) with free ai and bi and an arbitrary signal. The output generator of an F(z) with any input is a module in Windfc# behind the button with the step response icon.

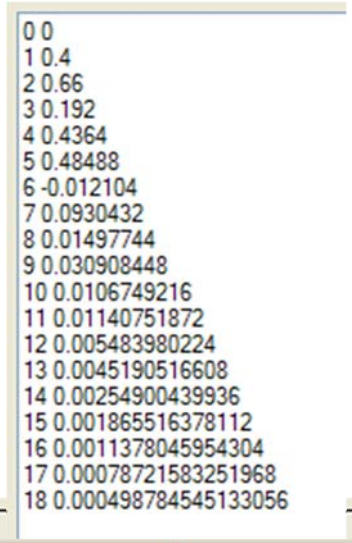


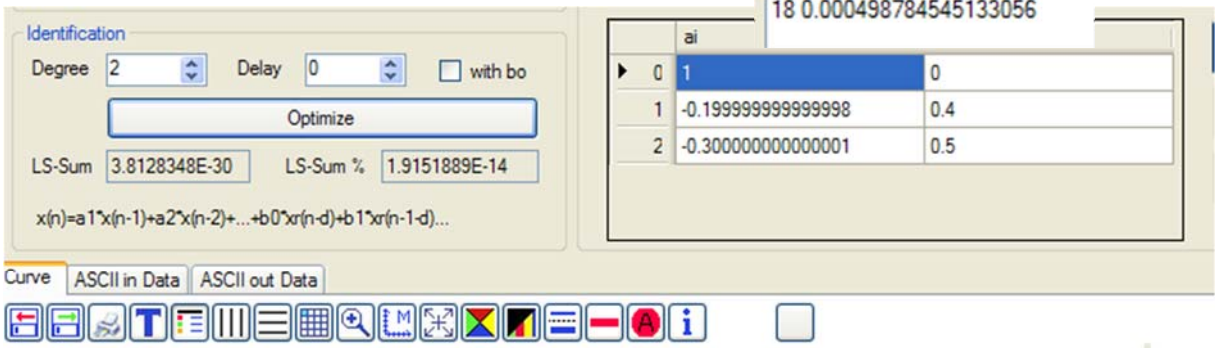
Example: $$F(z) = \frac{0.4z^{-1} + 0.5z^{-2}}{1 - 0.2z^{-1} - 0.3z^{-2}}$$ (file *testls.fz*)

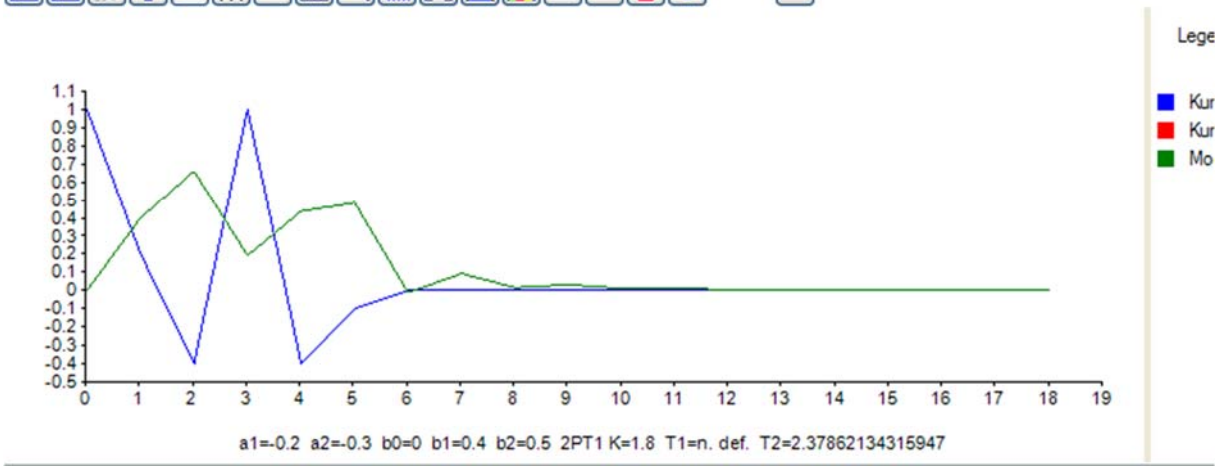
Input signal (file *newImpuls.sim*)

0	1
1	0.2
2	-0.4
3	1
4	-0.4
5	-0.1
6	0

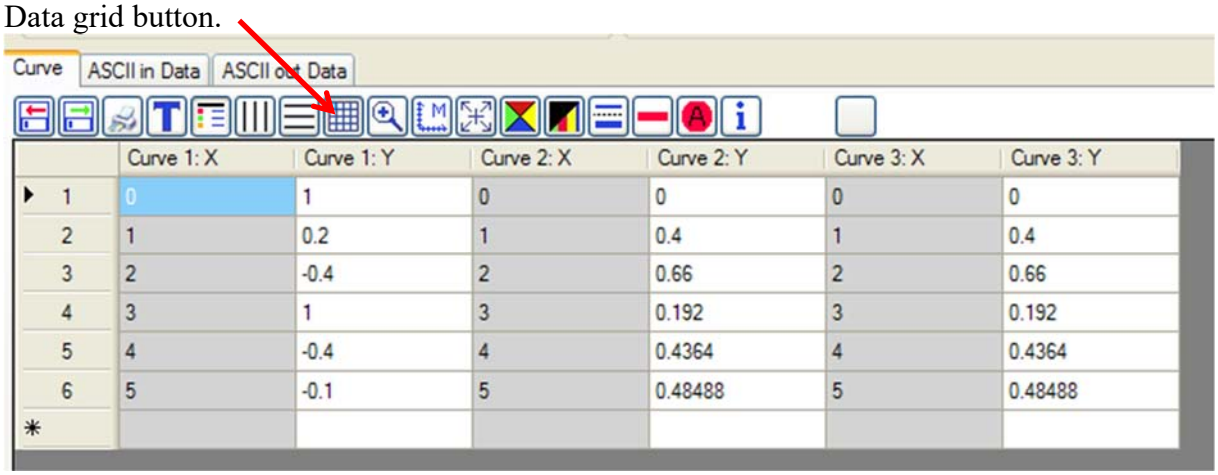
Result: (note, that inp(t<0) has to be set to 0!)





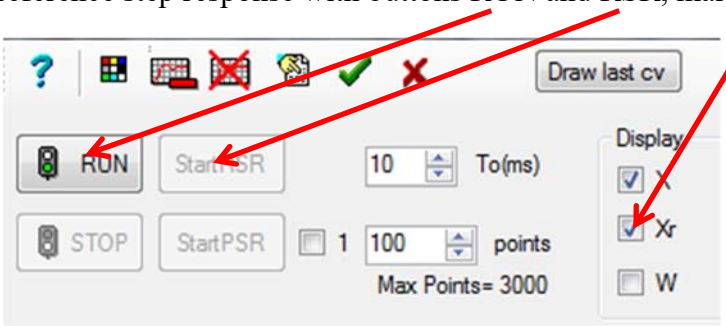


The first 5 points are enough to identify exact coefficients of F(z), see the data after click on Data grid button.



	Curve 1: X	Curve 1: Y	Curve 2: X	Curve 2: Y	Curve 3: X	Curve 3: Y
1	0	1	0	0	0	0
2	1	0.2	1	0.4	1	0.4
3	2	-0.4	2	0.66	2	0.66
4	3	1	3	0.192	3	0.192
5	4	-0.4	4	0.4364	4	0.4364
6	5	-0.1	5	0.48488	5	0.48488
*						

This module is prepared to use this method in a realtime- measurement with the controller-module. Open *"Real-Time- Controller → Adaptive Advanced controller"*.
Example: First go to menu "ADC-Cards" and select "→Hardware simulation", then select any model and close this window. Open *"Real-Time- Controller → Student control Box"*. Start a reference step response with buttons RUN and RSR, mark Xr- Checkbox and leave with green arrow. ✔



After this open menu "identification → by Offline LS – method" and push "Get RTC- Data" – Button. Then both curves (input and output of the realtime- process) can be seen in the display. Rest is already described. Result should be identical with the selected model in the hardware simulation.

Theory:
A general process has the transfer function

$$F(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_m z^{-m}}{1 + a_1 z^{-1} + \ldots + a_m z^{-m}} z^{-d}$$

With m is the degree of the function and d is a delay. The unknowns are the a_i and b_i. A PTn-process has b_0=0. This gives the algorithm (output y and input u)

$$y(k) = -a_1 y(k-1) - \ldots - a_m y(k-m) + b_0 u(k-d) + b_1 u(k-d-1) - \ldots - b_m u(k-d-m).$$

The same equation one step before:

$$y(k-1) = -a_1 y(k-2) - \ldots - a_m y(k-m-1) + b_0 u(k-d-1) + b_1 u(k-d-2) - \ldots - b_m u(k-d-m-1).$$

Now do this as long there are as many equations as unknowns. This equation system looks like this:

$$\underbrace{\begin{bmatrix} y(k) \\ y(k-1) \\ \ldots \\ \ldots \\ \ldots \\ \ldots \\ \ldots \\ y(k-N) \end{bmatrix}}_{Y} = \underbrace{\begin{bmatrix} -y(k-1) & -y(k-2) & \ldots & -y(k-m) & u(k) & u(k-1) & \ldots & u(k-m) \\ -y(k-2) & -y(k-3) & \ldots & -y(k-m-1) & u(k-1) & u(k-2) & \ldots & u(k-m-1) \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ -y(k-N-1) & -y(k-N-2) & \ldots & -y(k-N-m) & u(k-n) & u(k-N-1) & \ldots & u(k-N-m) \end{bmatrix}}_{\Psi_K} * \underbrace{\begin{bmatrix} a_1 \\ a_2 \\ \ldots \\ a_m \\ b_0 \\ b_1 \\ \ldots \\ b_m \end{bmatrix}}_{\Theta}$$

Written with matrices:

$$\underline{y}_k = \underline{\underline{\Psi}}_k * \underline{\Theta}$$ and solution $$\underline{\Theta} = \underline{\underline{\Psi}}_k^{-1} * \underline{y}_k$$

Are there more points, then a least square method can be used:
The LS- Sum can be calculated with

$$V = \sum_{k=1}^{N} e^2(k) = \underline{e}^T \underline{e}$$

The error e(k) is difference between measured y(k) and the model estimation

$$\underline{e}(k) = \underline{y}_k - \underline{\underline{\Psi}}_k * \underline{\Theta}$$

Differentiating of V and setting to zero (Minimum) gives the solution

$$\hat{\underline{\Theta}} = \underline{\underline{P}}_k * \underline{\underline{\Psi}}_k^T * \underline{y}_k$$

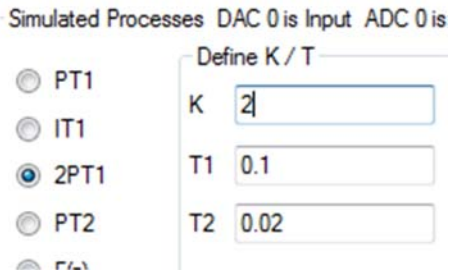
with

$$\underline{\underline{P}} = [\underline{\underline{\Psi}}^T * \underline{\underline{\Psi}}]^{-1}$$

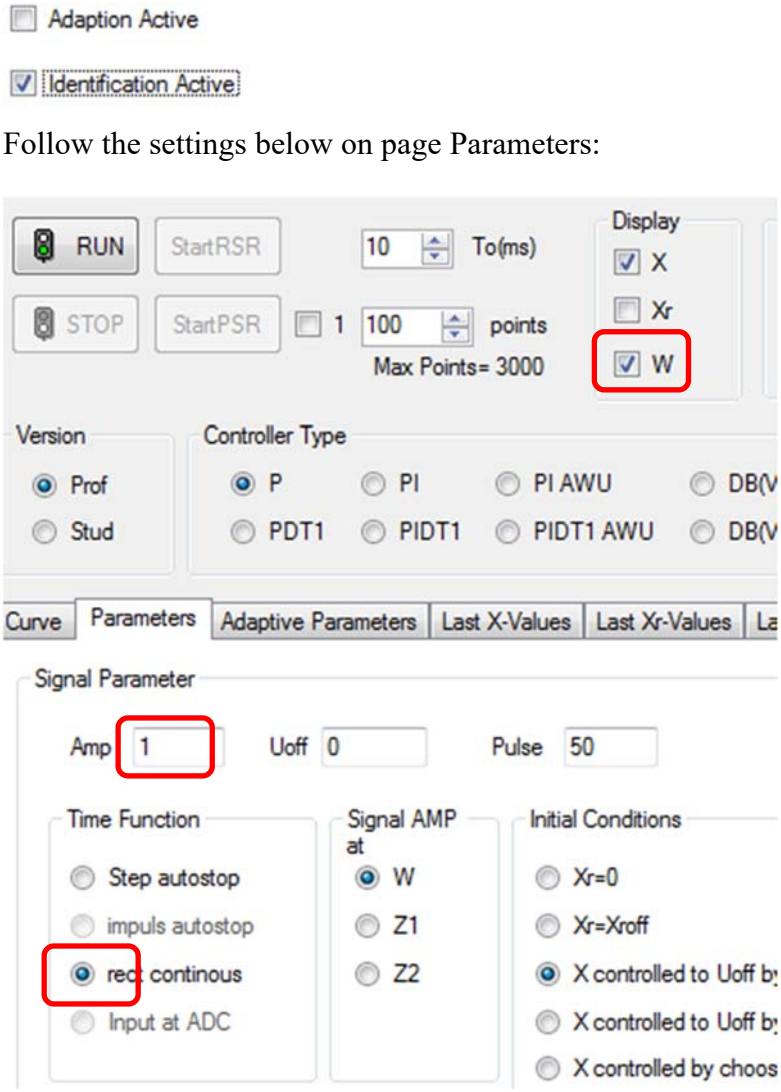
P is a symmetrical matrix with (2m+1) * (2m+1) elements, which needs an inversion. This is a non-iterative solution, which needs no starting value and always gives a solution.

8.6 Recursive online-Identification with LS-Online

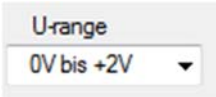
This method is able to identify a process during a running control system. Each new input / output point pair gives an improved estimation of process parameters a_i and b_i . It is available in the real time controller box with menu "*Real-Time- Controller → Adaptive Advanced controller*". On the page "Adaptive parameters" you can activate this identification.
Example settings: Select a Hardware simulation as described before in the last subchapter 8.5.
Select this model:



Then design controllers with menu "*Controller Design → Design of digital PIDT1+div*", select the same process with same parameters and leave with green arrow button.
Activate identification on "*Real-Time- Controller → Adaptive Advanced controller*" on the page "Adaptive parameters":



Activate Realtime Curve with



and set the amplitude scale U- range to 0 – 2V.

Now play with the different controllers and change K and T –values in the simulation with the buttons to double or half the values and see the reaction of the identification.

Theory:
The non - recursive estimation equation from the last chapter was:

$$\hat{\underline{\theta}}_k = \underline{P}_k * \underline{\Psi}_k^T * \underline{y}_k$$

The same equation but now at time k+1:

$$\hat{\underline{\theta}}_{k+1} = \underline{P}_{k+1} * \underline{\Psi}_{k+1}^T * \underline{y}_{k+1}$$

With this together I found in a German book from Isermann as a result the recursive estimation equation:

$$\underline{\hat{\Theta}}_{k+1} = \underline{\hat{\Theta}}_k + \underline{\gamma}_k * [y_{k+1} - \underline{\Psi}^T_{k+1} * \underline{\hat{\Theta}}_k]$$

Θ_{k+1} is a new set of parameters, Θ_k the old set. y_k is vector of the old outputs at k and earlier, y_{k+1} the output at k+1. $\underline{\Psi}^T$ is described above.

Now the part

$$\underline{\Psi}^T_{k+1} * \underline{\hat{\Theta}}_k = \hat{y}(k|k+1)$$

could be understood as a one- step prediction of the model: with old parameters and the newest value of the input the output value y_{k+1} is predicted. New writing:

$$\underline{\hat{\Theta}}_{k+1} = \underline{\hat{\Theta}}_k + \underline{\gamma}_k [y_{k+1} - y(k|k+1)]$$

This is a recursive estimation equation.
See the source code in Windfc#- project in file “FormAdaptiveControl.cs”.
After actual measurement of process output in variable val:

```
schaetzvec(val, ref tetaid, mgradIdent, gamma);
teta2tetab0();
```

After calculation of new output value yout:

```
korrekvec(yout, val, mgradIdent, kovmat, ref gamma, wichtung);
kovmatrix(mgradIdent, ref kovmat, gamma, wichtung);
```

That‘s all.

9 Special Digital Controllers

9.1 Preparation step response invariant function

To design a dead-beat controller with known F(p) of the process first you have to calculate the step response invariant filter function $H_0F(z)$ of the process. This can be done with the mechanism described in chap 7.2.2:

$$H_0F(z) = \frac{z-1}{z} Z\left\{\frac{1}{p}F(p)\right\}$$

So in other words: Take the process F(p), multiply with 1/p, go to the Z- transform – look up table (see page 27), take the F(z) and multiply this function with (z-1)/z. Finally normalize the result, so that the coefficient in the denominator without a z is one.

9.2 Calculation of $H_0F(z)$ from F(p) with simple standard blocks

For the simple transfer blocks PT1, IT1, PT2, 2PT1, 3PT1 and 4PT1 the coefficients of the z - transfer function with hold block can be derived. $H_0F(z)$ has the following form:

$H_0F(z) = \frac{y(z)}{x_R(z)} = \frac{b_1 z^{-1} + ... + b_m z^{-m}}{1 + a_1 z^{-1} + ... + a_m z^{-m}}$, m is the model degree, therefore at the PT1 m=1, at the other three blocks m=2, 3 or 4. For this transfer function, the linear difference equation can easily be written as $y(k) = -a_1 y(k-1) - ... - a_m y(k-m) + b_1 x_R(k-1) + ... + b_m x_R(k-m)$, which has the same known coefficients a_i, b_i and which can easily be programmed as a digital filter. This is realized in the program *Windfc#*

The coefficient b_0, which does not show up in the above formula, is, in all that processes, zero. Otherwise the process has at high frequencies ($p \to \infty$, that means $z \to 0$) still a final amplification, which is not attainable.

PT1:

With $F(p) = K/(1+pT)$ and m=1 we get $a_1 = -\exp(-T_0/T)$, $b_1 = K(1+a_1)$.

IT1:

With $F(p) = K/p(1+pT)$ and m=2 there follows

$a_1 = -1 - \exp(-T_0/T)$, $a_2 = \exp(-T_0/T)$, $b_1 = KT(T_0/T - 1 + \exp(-T_0/T))$ and

$b_2 = -KT(T_0/T \exp(-T_0/T) - 1 + \exp(-T_0/T))$

PT2:

With $F(p) = \frac{K}{1 + 2dp/\omega_0 + p^2/\omega_0^2}$ and m=2 there follows with

$\omega = \omega_0\sqrt{1-d^2}$, $z_0 = \exp(-d\omega_0 T_0)$, $\delta = d\omega_0$

$a_1 = -2z_0\cos(\omega T_0)$, $a_2 = z_0^2$, $b_1 = K(1 - z_0(\cos(\omega T_0) + \frac{\delta}{\omega}\sin(\omega T_0)))$ und

$b_2 = Kz_0(z_0 - \cos(\omega T_0) + \frac{\delta}{\omega}\sin(\omega T_0))$.

2PT1:

With $F(p) = \dfrac{K}{(1+pT_1)(1+pT_2)}$ and m=2 and with $c_2 = \exp(-T_0/T_2)$, $c_3 = \exp(-T_0/T_1)$,
$d_1 = K\dfrac{T_1T_2}{T_2-T_1}$, $d_2 = \dfrac{1}{T_1} - \dfrac{1}{T_2} - \dfrac{1}{T_1}c_2 + \dfrac{1}{T_2}c_3$ and $d_3 = (\dfrac{1}{T_1} - \dfrac{1}{T_2})c_2c_3 - \dfrac{1}{T_1}c_3 + \dfrac{1}{T_2}c_2$ there follows $a_1 = -c_2 - c_3$, $a_2 = c_2c_3$, $b_1 = d_1d_2$ *and* $b_2 = d_1d_3$

3PT1:

$F(p) = \frac{K}{(1+pT_1)(1+pT_2)(1+pT_3)}$ and m=3 with
$c_1 = \exp(-T_o/T_1), c_2 = \exp(-T_o/T_2), c_3 = \exp(-T_o/T_3),$
$a_1 = -c_1 - c_2 - c_3,\ \ a_2 = c_1c_2 + c_1c_3 + c_2c_3,\ \ a_3 = -c_1c_2c_3,$
$B_1 = \frac{-K}{(1-T_2/T_1)(1-T_3/T_1)}$, $B_2 = \frac{-K}{(1-T_1/T_2)(1-T_3/T_2)}$, $B_3 = \frac{-K}{(1-T_1/T_3)(1-T_2/T_3)}$,

$$b_1 = Ka_1 + B_1(-1 - c_2 - c_3) + B_2(-1 - c_1 - c_3) + B_3(-1 - c_1 - c_2)$$

$$b_2 = Ka_2 + B_1(+c_2 + c_3 + c_3c_2) + B_2(+c_1 + c_3 + c_1c_3) + B_3(+c_1 + c_2 + c_1c_2)$$

$$b_3 = Ka_3 - B_1(c_3c_2) - B_2(c_1c_3) - B_3(c_1c_2).$$

4PT1 (similar to 3PT1):

$F(p) = \frac{K}{(1+pT_1)(1+pT_2)(1+pT_3)(1+pT_4)}$ and m=4 with
$c_1 = \exp(-T_o/T_1), c_2 = \exp(-T_o/T_2), c_3 = \exp(-T_o/T_3), c_4 = \exp(-T_o/T_4)$
$a_1 = -c_1 - c_2 - c_3 - c_4,\ \ a_2 = c_1c_2 + c_3c_4 + c_1c_3 + c_1c_4 + c_2c_3 + c_2c_4,$
$a_3 = -c_1c_3c_4 - c_2c_3c_4 - c_1c_2c_3 - c_1c_2c_4,\qquad a_4 = c_1c_2c_3c_4,$
$B_1 = \frac{-K}{(1-T_2/T_1)(1-T_3/T_1)(1-T_4/T_1)}$, … , $B_4 = \frac{-K}{(1-T_1/T_4)(1-T_2/T_4)(1-T_3/T_4)}$,

$$b_1 = Ka_1 - B_1(1 + c_2 + c_3 + c_4) - \cdots - B_4(1 + c_1 + c_2 + c_3)$$

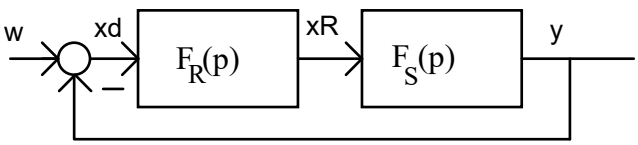
$$b_2 = Ka_2 + B_1(+c_2 + c_3 + c_4 + c_2c_3 + c_2c_4 + c_3c_4) + \cdots \\ + B_4(+c_1 + c_2 + c_3 + c_1c_2 + c_1c_3 + c_2c_3)$$

$$b_3 = Ka_3 - B_1(c_2c_3c_4 + c_2c_3 + c_2c_4 + c_3c_4) - \cdots - B_4(c_1c_2c_3 + c_1c_2 + c_1c_3 + c_2c_3).$$

$$b_4 = Ka_4 + B_1(c_2c_3c_4) + B_2(c_1c_3c_4) + B_3(c_1c_2c_4) + B_4(c_1c_2c_3)$$

9.3 The Dead Beat Algorithm

Theory -description in German: File *Theorie Dead beat Regler Baumann - German.pdf*
A Dead-Beat Controller is theoretically and practically the fastest possible controller of the world, if no controller output limitation appears. Theory predicts a settling time of m*T_0 with the ideal and (m+1)T_0 with the real algorithm, if m is the denominator degree of the process. So e.g. a 2PT1- process has a settling time of exactly 2 T_0 in the ideal algorithm.



Is the general $H_0F(z)$ written as

$$\mathrm{H}_0 F(z) = \frac{b_0 + b_1 z^{-1} + ... + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + ... + a_m z^{-m}} z^{-d} = \frac{y}{x_R}$$

and the resulting DBC described with the following function:
(**ideal** version d=0):

$$F_R(z) = \frac{q_0 + q_1 z^{-1} + ... + q_m z^{-m}}{1 - p_1 z^{-1} - p_2 z^{-2} - ... - p_m z^{-m}} = \frac{x_R}{x_d} \text{ with the algorithm}$$

$$x_R(n) = p_1 x_R(n-1) + p_2 x_R(n-2) + ... + q_0 x_d(n) + q_1 x_d(n-1) + q_2 x_d(n-2) + ...$$

or the **real** version (d=1):

$$F_R(z) = \frac{q_0 z^{-1} + q_1 z^{-2} + ... + q_m z^{-m-1}}{1 - p_1 z^{-2} - p_2 z^{-3} - ... - p_m z^{-m-1}} \text{ with the algorithm}$$

$$x_R(n) = p_1 x_R(n-2) + p_2 x_R(n-3) + ... + q_0 x_d(n-1) + q_1 x_d(n-2) + q_2 x_d(n-3) + ...$$

then you can calculate the q and p coefficients simply with the following equations:

$q_0 = 1/(b_0 + b_1 + ... + b_m)$ and

$q_1 = q_0 * a_1$, $q_2 = q_0 * a_2$, $q_3 = q_0 * a_3$, …..and
$p_1 = q_0 * b_1$, $p_2 = q_0 * b_2$, $p_3 = q_0 * b_3$, …..

In the above example the coefficients become to
b_1=0.3935 and a_1=-0.6065 and then

$q_0 = 1/b_1$=2.541, q_1= - 1.541 and p_1=1.

The ideal algorithm is

$$x_R(n) = x_R(n-1) + 2.541 x_d(n) - 1.541 x_d(n-1)$$

and the real:

$$x_R(n) = x_R(n-2) + 2.541 x_d(n-1) - 1.541 x_d(n-2)$$

9.3.1 The Dead Beat version DB(v+1)

A second version of Dead beat controller is the DB(v+1). This controller has the advantage of a smaller starting impulse. The large q_o is separated on two smaller q_{onew}, but this algorithm takes one step more. It has the settling time of (m+1)*T_0 with the ideal and (m+2)T_0 with the real algorithm.
The calculation of the p_i and q_i looks like this:

$$q_0 = u_0 \quad (\textit{free choice}) \;\; \textit{with} \;\; A = 1/\sum b_i$$

$$q_1 = q_0 * (a_1 - 1) + A \;\; \textit{and} \;\; p_1 = q_0 * b_1$$

$$q_2 = q_0 * (a_2 - a_1) + a_1 * A \;\; \textit{and} \;\; p_2 = q_0 * (b_2 - b_1) + b_1 * A$$

…..

$$q_m = q_0 * (a_m - a_{m-1}) + a_{m-1} * A \;\; \textit{and} \;\; p_m = q_0 * (b_m - b_{m-1}) + b_{m-1} * A$$

$$q_{m+1} = q_0 * (-a_m) + a_m * A \;\; \textit{and} \;\; p_{m+1} = q_0 * (-b_m) + b_m * A$$

q_o could be a value between the maximum value of the standard algorithm (r=0) and a minimal possible value (r=1).

$$q_0 = q_{0min} + (1-r) * (q_{0max} - q_{0min})$$

r is a value between 0 and 1

$$q_{0max} = \frac{1}{\sum b_i}$$

$$q_{0min} = \frac{1}{(1-a_1) * \sum b_i}$$

Disadvantages of DBCs: If controller output is limited, you get bad behaviour, worse than PID. If m>1 large negative controller outputs appear: Some processes could not realize negative inputs, e.g. a heater could not cool or the cruise control has to brake. Not applicable for unstable processes.

9.4 Direct design of a Dead-Beat - controller from F(p)

With the following formulas the fitting parameters p and q of a Dead-Beat -controller can be simply derived from given process parameters K and T without any detour using H_0 F(z). This has been explicitly calculated for the process types PT1, 2PT1 and IT1. With it, an adaptive control can be easily set up.

9.4.1 Dead-Beat - controller for a simple PT1 - process

For a PT1 - process with the transfer function

$$F_S(p) = \frac{K}{1+pT}$$

you can find a Dead-Beat-controller with the 'real' algorithm

$$x_R(n) = x_R(n-2) + q_0 x_d(n-1) + q_1 x_d(n-2)$$

which has a delay time of T_0 that is added immediately to the process for the controller-design. The values for q results in:

$$q_0 = \frac{1}{K\left(1-\exp(-T_0/T)\right)} \quad and \quad q_1 = \frac{-\exp(-T_0/T)}{K\left(1-\exp(-T_0/T)\right)} = \frac{1}{K} - q_0 \quad .$$

Numerical example: With K=1 and T=5T_0 follows q_0=5.5161 and q_1=-4.5161.

9.4.2 Dead-Beat - controller for a 2PT1 - process

For a 2PT1 - process with the transfer function

$$F_S(p) = \frac{K}{(1+pT_1)(1+pT_2)}$$

you can find a Dead-Beat-controller with the 'real' algorithm

$$x_R(n) = p_1 x_R(n-2) + p_2 x_R(n-3) + q_0 x_d(n-1) + q_1 x_d(n-2) + q_2 x_d(n-3)$$

which again has a delay time of T_0 that is added immediately to the process for the controller-design. The values for q come out to:

$$q_0 = \frac{\exp(T_0/T_1)\exp(T_0/T_2)}{K\left(1-\exp(T_0/T_1)\right)\left(1-\exp(T_0/T_2)\right)} \quad ,$$

$$q_1 = -\frac{\exp(T_0/T_1)+\exp(T_0/T_2)}{K\left(1-\exp(T_0/T_1)\right)\left(1-\exp(T_0/T_2)\right)} \quad ,$$

$$q_2 = \frac{1}{K\left(1-\exp(T_0/T_1)\right)\left(1-\exp(T_0/T_2)\right)} \quad ,$$

$$p_2 = -\frac{T_1\left(1-\exp(T_0/T_1)\right)-T_2\left(1-\exp(T_0/T_2)\right)}{(T_1-T_2)\left(1-\exp(T_0/T_1)\right)\left(1-\exp(T_0/T_2)\right)} \; and \; p_1 + p_2 = 1.$$

Numerical example: With K=2 and T_1=5T_0 and T_2=3T_0 follows

q_0=9.7306, q_1=-14.9391, q_2=5.7084, p_1=0.5443 and p_2=0.4557.

9.4.3 Dead-Beat - controller for a IT1 - process

For an IT1 - process with the transfer function

$$F_S(p) = \frac{K}{p(1+pT)}$$

you can find a Dead-Beat-controller with the 'real' algorithm

$$x_R(n) = p_1 x_R(n-2) + p_2 x_R(n-3) + q_0 x_d(n-1) + q_1 x_d(n-2) + q_2 x_d(n-3)$$

which again has a delay time of T_0 . The values for q now come out to:

$$q_0 = \frac{1}{KT_0\left(1-\exp(-T_0/T)\right)} \quad ,$$

$$q_1 = -\frac{1+\exp(-T_0/T)}{KT_0\left(1-\exp(-T_0/T)\right)} \quad ,$$

$$q_2 = \frac{\exp(-T_0/T)}{KT_0\left(1-\exp(-T_0/T)\right)} \; , (with \; q_0 + q_1 + q_2 = 0!!)$$

$$p_1 = T\frac{T_0/T - 1 + \exp(-T_0/T)}{T_0\left(1-\exp(-T_0/T)\right)} \; and \; p_1 + p_2 = 1.$$

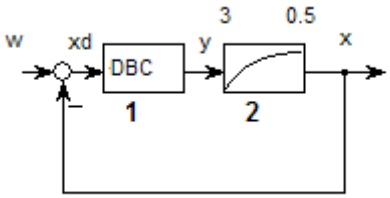
Numerical example: With K=1/T=3.643/s and T=5T_0 follows

q_0=27.5833, q_1=-50.1666, q_2=22.5833, p_1=0.5166 and p_2=0.4834.

9.5 Examples of two dead beat controllers

Let's work thru an old examination task:

9.5.1 EBay53 Task4



First design the invariant step response filter function $H_0F(z)$ of the process and then a Dead-Beat-controller for the left system, which should run with real algorithm at T_0=0.1s. Draw the diagrams of the signals x and y as a response to a desired step of amplitude w= 2 in the time range 0 to 0.4s.

Solution, first ***process***:

$$F(p)=\frac{3}{1+0.5p}. \quad F(z)=Z\left\{\frac{3}{p(1+0.5p)}\right\}=Z\left\{\frac{6}{p(2+p)}\right\}=3\frac{\left(1-e^{-2To}\right)z}{(z-1)\left(z-e^{-2To}\right)}$$

$$H_0F(z)=3\frac{1-e^{-0.2}}{z-e^{-0.2}}=\frac{0.54380774z^{-1}}{1-0.81873075z^{-1}}=\frac{X(z)}{Y(z)}=\frac{output}{input}$$

with algorithm $x(n)=0.81873075\,x(n-1)+0.54380774\,y(n-1)$.

a_1=-0.81873075 and b_0=0 and b_1= 0.54380774. Note that this is the process function with output x and input y.

Now ***Dead- beat controller*** with standard design (s.a.) real algorithm, degree m=1:

$$F_R(z)=\frac{Y(z)}{X_d(z)}=\frac{q_0z^{-1}+q_1z^{-2}+\ldots+q_mz^{-m-1}}{1-p_1z^{-2}-p_2z^{-3}-\ldots-p_mz^{-m-1}}=\frac{q_0z^{-1}+q_1z^{-2}}{1-p_1z^{-2}}$$

The p_i and q_i :

$$q_0=1/(b_0+b_1+\ldots+b_m)=1/b_1=1.8388852 \quad \text{and}$$

$q_1=q_0*a_1$=- 1.50555185 and $p_1=q_0*b_1$=1. We get the transfer function of the DB-controller

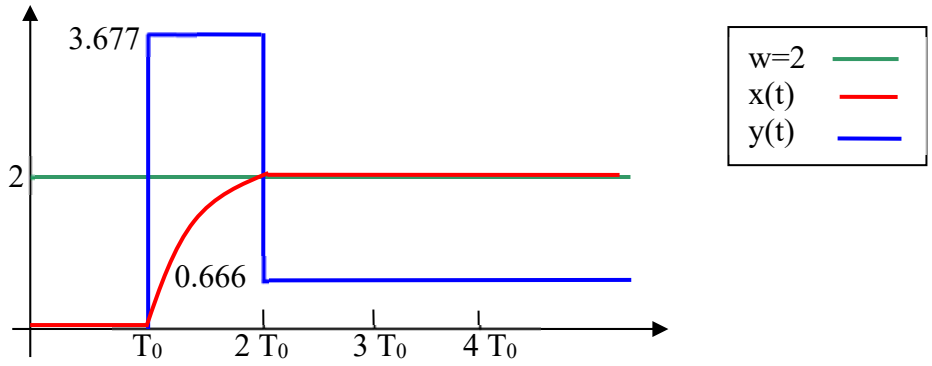
$$F_R(z)=\frac{Y(z)}{X_d(z)}=\frac{1.8388852z^{-1}-1.505551852z^{-2}}{1-z^{-2}}$$ with the corresponding algorithm

$$y(n)=y(n-2)+1.8388852\,x_d(n-1)-1.505551852x_d(n-2).$$

Now the ***drawings*** with reference step w=2:

n	**t in s**	**w(n)**	**x(n)**=0.8187..x(n-1)+0.5438..y(n-1)	**x_d(n)**	**y(n)**=see above
0	0	2	0	2	0
1	0.1	2	0	2	3.67777

2	0.2	2	0.5438*3.6777=2.0000000	0	1.839*2-1.5056*2=0.66666
3	0.3	2	0.8187*2+0.5438*0.66666=2.0000	0	3.6777-1.506*2=0.66666
4	0.4	2	Dito =2	0	0.66666666



Description of the function in words: Because it is a real function the first cycle is empty, nothing happens. Then the dead beat controller throws out a first pulse with an amplitude of 3.677. This value is exactly the amout which is necessary to move the PT1- output in one T_0 to the desired value 2. Here's the proof: The PT1- step response has the function

$$x(t) = 3.6777 * 3 * (1 - e^{-t/T}) = 11.0331 * (1 - e^{-0.1/0.5}) = 11.0331 * 0.181269 = 2.0000$$

at time t=T_0 this is exactly 2. After this "pull- up-step" the controller switches to value 0.66666, which is necessary to hold the output of the PT1 at 2 similar to trickle charging or maintenance charging of accumulators because 3*0.6666=2. So as predicted after 2 steps in a first order process the reference step reaches the desired value.

This behaviour can be compared with kind to boil potatoes with experienced users. First, switch to full power and in the right moment switch back to the power which maintains the boiling temperature.

Now a second exercise with order m=2 and an ideal algorithm, similar to exam EBAY56/6, there I changes from real into ideal. You can compare the exam results with these.

9.5.2 EBay56 Task6

Design an ideal dead beat controller of a 2 PT1- process with K=2, T1=0.1 s and T2=0.2 s by using the HoF(z)- funktion of the process. Sampling time is To=20 ms. How large are the a and b - coefficients of the process and the q and p - coefficients of the controller? Draw the diagrams of the signals w (desired value), xd (error signal), x (process output) and y (controller output) of the first 40 ms.

Solution, first ***process***:

$$F(p) = \frac{2}{(1+0.1p)(1+0.2p)}.$$

$$F(z) = Z\left\{\frac{2}{p(1+0.1p)(1+0.2p)}\right\} = not\ available\ in\ \ Z-Transform-table.$$

Direct result from chapter 9.4.2.:

$$H_0F(z) = \frac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} = \frac{X(z)}{Y(z)} = \frac{output}{input}$$

with algorithm $x(n) = -a_1 x(n-1) - a_2 x(n-2) + b_1 y(n-1) + b_2 y(n-2)$.

From from chapter 9.4.2 we get

$$c_2 = \exp(-T_0/T_2) = 0.904837,\ c_3 = \exp(-T_0/T_1) = 0.818731,$$

$$d_1 = K\frac{T_1 T_2}{T_2 - T_1} = 2\frac{0.2*0.1}{0.2-0.1} = 0.4,$$

$$d_2 = \frac{1}{T_1} - \frac{1}{T_2} - \frac{1}{T_1}c_2 + \frac{1}{T_2}c_3 = 10 - 5 - 10*0.904837 + 5*0.818731 = 0.045285 \text{ and}$$

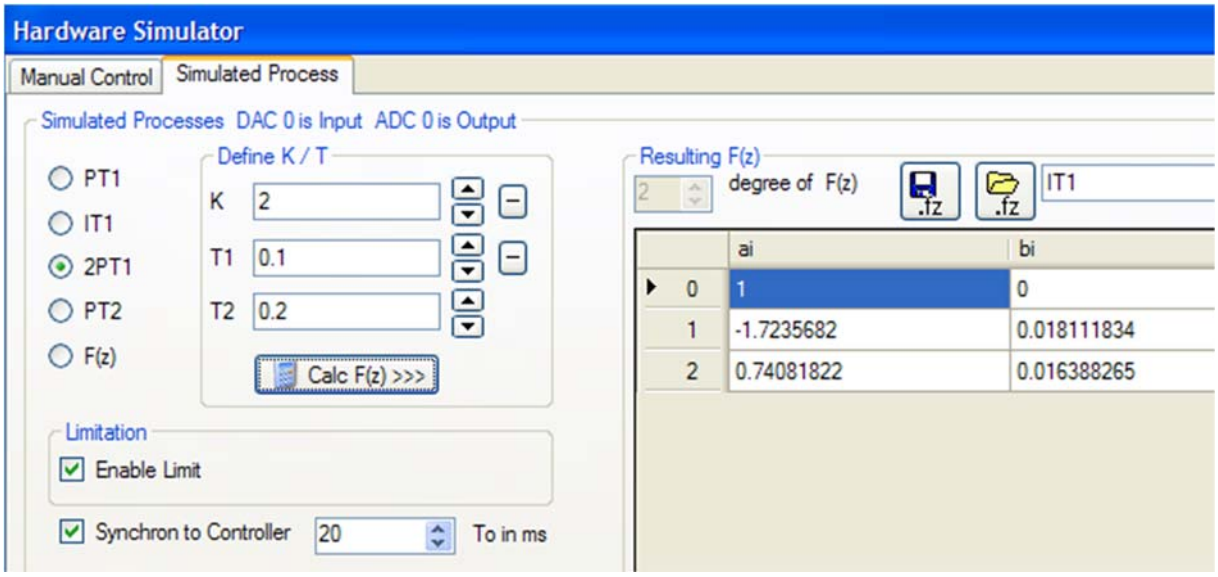
$$d_3 = (\frac{1}{T_1} - \frac{1}{T_2})c_2 c_3 - \frac{1}{T_1}c_3 + \frac{1}{T_2}c_2 = 5*0.904837*0.818731 - 10*0.818731 + 5*0.904837$$

$d_3 = 0.0409655$;

With this we calculate

$$a_1 = -c_2 - c_3 = -1.723568,\ a_2 = c_2 c_3 = 0.740818,\ b_1 = d_1 d_2 = 0.018114\ and\ b_2 = d_1 d_3 = 0.0163862$$

Compare with WindfC#, main menu "AC-Cards" → "Hardware simulation" : The b – values differ in the 5th digit because above accuracy was 6 significant digits, in WindfC# I have 16 digits.



Note, that this is the process function with output x and input y.

Now ***Dead- beat controller*** with standard design, ideal algorithm and degree m=2:

$$F_R(z) = \frac{Y(z)}{X_d(z)} = \frac{q_0 z + q_1 z^{-1} + \ldots + q_m z^{-m}}{1 - p_1 z^{-1} - p_2 z^{-2} - \ldots - p_m z^{-m}} = \frac{q_0 + q_1 z^{-1} + q_2 z^{-2}}{1 - p_1 z^{-1} - p_2 z^{-2}}$$

The p_i and q_i :

$$q_0 = 1/(b_0 + b_1 + \ldots + b_m) = 1/(b_1 + b_2) = 28.9855 \quad \text{and}$$

$q_1 = q_0 * a_1 = -49.9524$ and $p_1 = q_0 * b_1 = 0.524986$.

$q_2 = q_0 * a_2 = 21.4729$ and $p_2 = q_0 * b_2 = 0.475014$.

Note that $p_1+p_2=1$! Finally we get the transfer function of the DB-controller

$$F_R(z)=\frac{Y(z)}{X_d(z)}=\frac{28.9855-49.9524z^{-1}+21.4729z^{-2}}{1-0.524986z^{-1}-0.475014z^{-2}}$$ with the corresponding algorithm

$y(n)=0.5250y(n-1)+0.4750y(n-2)+28.99x_d(n)-49.95x_d(n-1)+21.47x_d(n-2)$. With 4 significant digits. Compare the WindfC#- results, in the main menu "Controller-Design" the item "Design of digital PI/PIDT1+div" has following results (part of the window):

	qi	pi
1	28.985424	1
2	-49.958354	0.52497919
	21.47293	0.47502081

Now the ***drawings*** with reference step w=1:

$$x(n)=1.724x(n-1)-0.7408x(n-2)+0.01811y(n-1)+0.01639y(n-2)$$

$$y(n)=0.5250y(n-1)+0.4750y(n-2)+28.99x_d(n)-49.95x_d(n-1)+21.47x_d(n-2)$$

n	**t in s**	**w(n)**	**x(n)**	**x_d(n)**	**y(n)**=see above
0	0	1	0	1	28.99
1	0.02	1	0.01811*28.99=0.5250	0.4750	See below y(1)=-20.96
2	0.04	1	See below x(2)=1.000	0	See below y(2)=0.500
3	0.06	1	See below x(3)=1.000	0	See below y(3)=0.5
4	0.08	1	Constant=1	0	Constant= 0.5

$$y(1)=0.5250y(0)+0.4750y(-1)+28.99x_d(1)-49.95x_d(0)+21.47x_d(-1)=$$

$$y(1)=0.5250*28.99+28.99*0.4750-49.95=-20.96$$

$$x(2)=1.724x(1)-0.7408x(0)+0.01811y(1)+0.01639y(0)=$$
$$x(2)=1.724*0.5250+0.01811*(-20.96)+0.01639*28.99=1.00066$$

$$y(2)=0.5250y(1)+0.4750y(0)+28.99x_d(2)-49.95x_d(1)+21.47x_d(0)$$
$$y(2)=0.5250*(-20.96)+0.4750*28.99+28.99*0-49.95*0.475+21.47*1=0.51$$

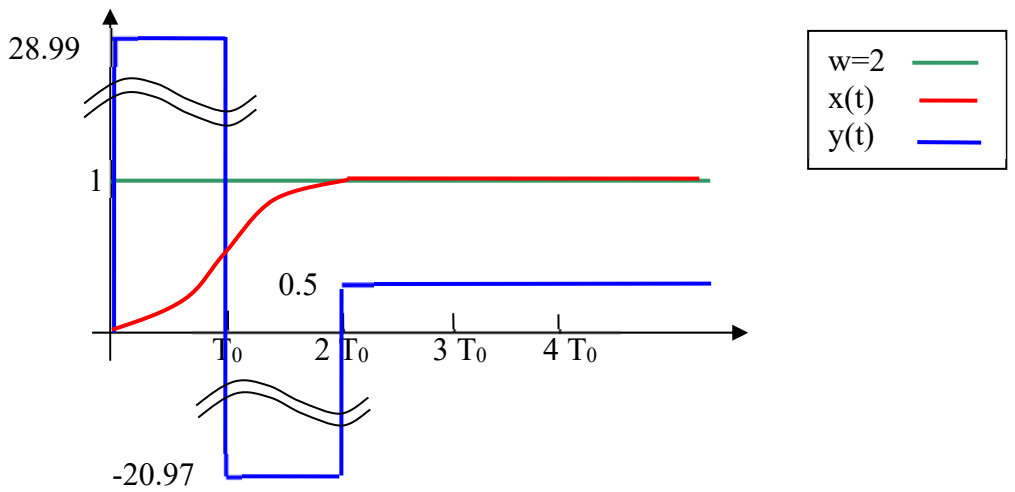
$$x(3)=1.724x(2)-0.7408x(1)+0.01811y(2)+0.01639y(1)=$$
$$x(3)=1.724*1-0.7408*0.525+0.01811*0.5+0.01639*(-20.96)=1$$

$$y(3)=0.5250y(2)+0.4750y(1)+28.99x_d(3)-49.95x_d(2)+21.47x_d(1)=$$
$$y(3)=0.5250*0.5+0.4750*(-20.96)+28.99*0-49.95*0+21.47*0.475=0.5$$

$$x(4)=1.724x(3)-0.7408x(2)+0.01811y(3)+0.01639y(2)=$$
$$x(4)=1.724-0.7408+0.01811*0.5+0.01639*0.5=1$$

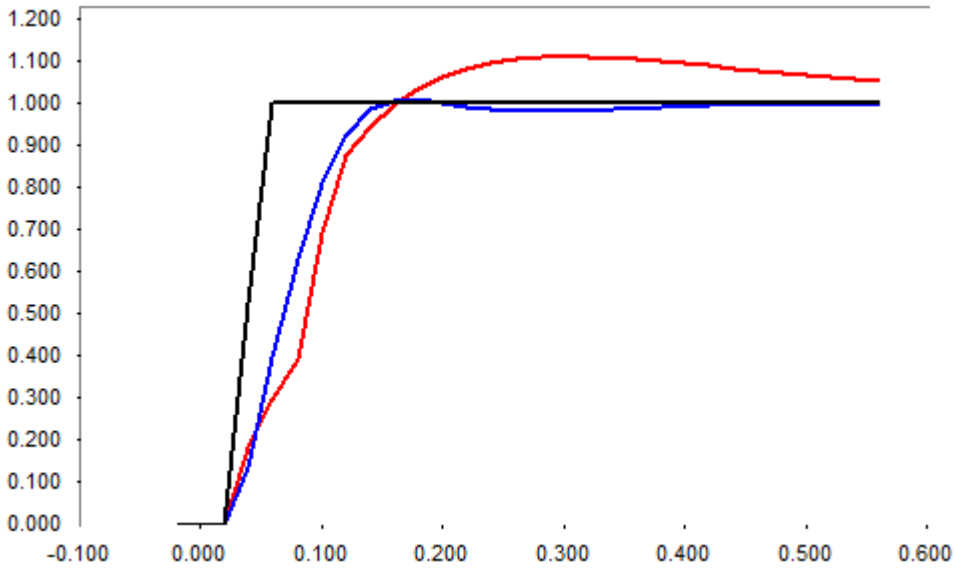
$$y(4)=0.5250y(3)+0.4750y(2)+28.99x_d(4)-49.95x_d(3)+21.47x_d(2)=$$
$$y(4)=0.5250*0.5+0.4750*0.5=0.5$$

Starting with n=4 there will be no change.



The controller starts first with a strong pull up impulse, then it has to brake with a strong negative controller output. Not all actuators can work with negative output signals. A motor – actuator must be able to brake, a temperature controller must be able to cool!

See next page simulation of this example with WindfC#:



Black curve DB- controller here with real algorithm. After 3 * T_0 desired value is reached. Blue curve is RSR of a PIDT1 with 60° phase margin, pole compensation. Red curve is the RSR of same Dead-beat but now with limitation to + and – 10 V. Result is worse than PID.

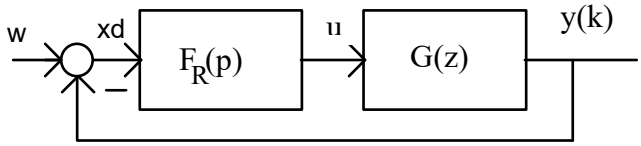
9.6 Orientation Controller

This type is developed in 1991 at TU Chemnitz (Ehrlicher).
File : [9.1] *Orientierungsregler Ehrlich TU Chemnitz1.pdf*
The advantages compared with Dead beat controller are:

1. No problems with limited controller outputs
2. No problems with unstable processes
3. Simple design from $H_oF(z)$, so online adaptive mode possible

The algorithm:

Note: now different letters for the signals:



$F_R(z) = \dfrac{g_0 z^{-1} + g_1 z^{-2} + \ldots + g_m z^{-m-1}}{1 - h_1 z^{-2} - h_2 z^{-3} - \ldots - h_m z^{-m-1}}$ is the controller.

Process:

$$G(z) = \frac{B(z)}{A(z)} = \frac{b_1 z^{-1} + b_2 z^{-2} \ldots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} \ldots + a_m z^{-m}}$$

1st: Calculation of a correction factor depending of old inputs of process u and outputs y:

$$c(k) = \frac{y(k) + \sum [(a_r * y(k-r) - b_r * u(k-d-r)]}{\sum b_r}$$

r is index from 1 to m

Now a prediction of a process output depending of model parameters:

$$y(k+1) = \sum_{r=1}^{m} [b_r * (u(k-r) - c(k)) - a_r * y(k+1-r)]$$

Calculation of new controller output u(k):

$$u(k) = f_3 * w(k) - c(k) + \sum (g_r * y(k+1-r-d)) - \sum (h_r * (u(k+1-r) + c(k))$$

With

$$h_1 = b_1 + b_2 + b_3 - a_1 * (b_1 + b_2 - a_1 * b_1) - a_2 * b_1$$

$$f_3 = \frac{1}{h_1}$$

$$g_r = \frac{a_{r+2} - a_1 * a_{r+1} + (a_1 * a_1 - a_2) * a_r}{h_1} = q_r$$

$$h_r = \frac{b_{r+2} - a_1 * b_{r+1} + (a_1 * a_1 - a_2) * b_r}{h_1} = p_r$$

Program code in WindfC#:

Design to do one time before run or each time after change of a_i and b_i :

```
hvkl = a[1] * a[1] - a[2];
h1 = b[1] + b[2] + b[3] - a[1] * (b[1] + b[2] - a[1] * b[1]) - a[2] * b[1];
if (h1 == 0) f3 = 1000; else f3 = 1.0 / h1;
for (i = 1; i <= m; i++) q[i] = (a[i + 2] - a[1] * a[i + 1] + hvkl * a[i]) * f3;
for (i = 2; i <= m; i++) p[i] = (b[i + 2] - a[1] * b[i + 1] + hvkl * b[i]) * f3;
```

and run each T_0:

```
void xr_or(ref double uk, double[] hr, double[] gr, double[] u, ref
double[] y, int m, double[] teta, double wk, double f3l, double yk)
{   /*Orientierungsregler  only for real mode*/
    double ck, SumB, hv1, hv2, xprae;
    int i;
    y[0] = yk;
    SumB = 0; hv2 = 0;
    for (i = 1; i <= m; i++) SumB = SumB + teta[i + m + 1];
    for (i = 1; i <= m; i++) hv2 = hv2+teta[i]*y[i]-teta[i+m+1] * u[i + 1];
    if (SumB == 0) ck = 10 * limitation; else ck = (yk + hv2) / SumB;
    if (ck > 10 * limitation) ck = 10 * limitation;
    if (ck < -10 * limit_low) ck = -10 * limit_low;
    xprae = 0;
    for (i=1;i<=m;i++) xprae=xprae+teta[i+m+1]*(u[i]-ck)-teta[i] * y[i-1];
    hv1 = gr[1] * xprae;
    for (i=2;i<=m;i++) hv1=hv1-hr[i] * (u[i - 1] - ck) + gr[i] * y[i - 2];
    uk = f3l * wk - ck + hv1;
    for (i = m; i > 0; i--) y[i] = y[i - 1];
}
```

uk is output of new controller value
hr=pr and gr=qr are the controller coefficients
u[] are the old controller outputs
y[] are the old process outputs
m degree
teta the a_i and b_i in one vector
wk the actual desired value
f31 the f_3 value
yk the actual process output.

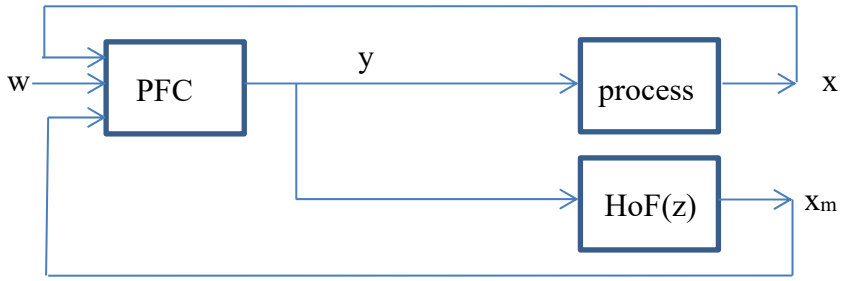
9.7 PFC- Predictive Functional Control

9.7.1 Introduction

See Book: [9.2] *Predictive Functional Control - Principles and Industrial Applications - Richalet, O'Donovan.*
See [9.3] atp – paper of Prof. Haber FH Köln

The idea is 20 years old, but has not resettled in daily control system design. The "father" of this idea is the French scientist Richalet. I was on a two days presentation in FH Köln and was impressed by this idea. This method has great success in chemical industry, if processes are nonlinear and complicated.
The application engineers give this method more future than state space design, which is normally used in similar cases.

The idea: Because computers and processors become more powerful it should be possible to use the knowledge of the model in each step.
PID and Dead- Beat controllers use the knowledge only in the design phase, after design in the runtime phase model is not used, controller parameters are constant.
The PFC calculates in each step the optimal controller output with the model function F(z) including influences of limitations to reach the desired value. There are several versions depending on the optimization method.



First with this idea an optimized Dead- Beat- Controller is developed, which has the advantage, that the limitation does not destroy the good behavior.
This should be demonstrated with a simple PT1- example.
The model function is

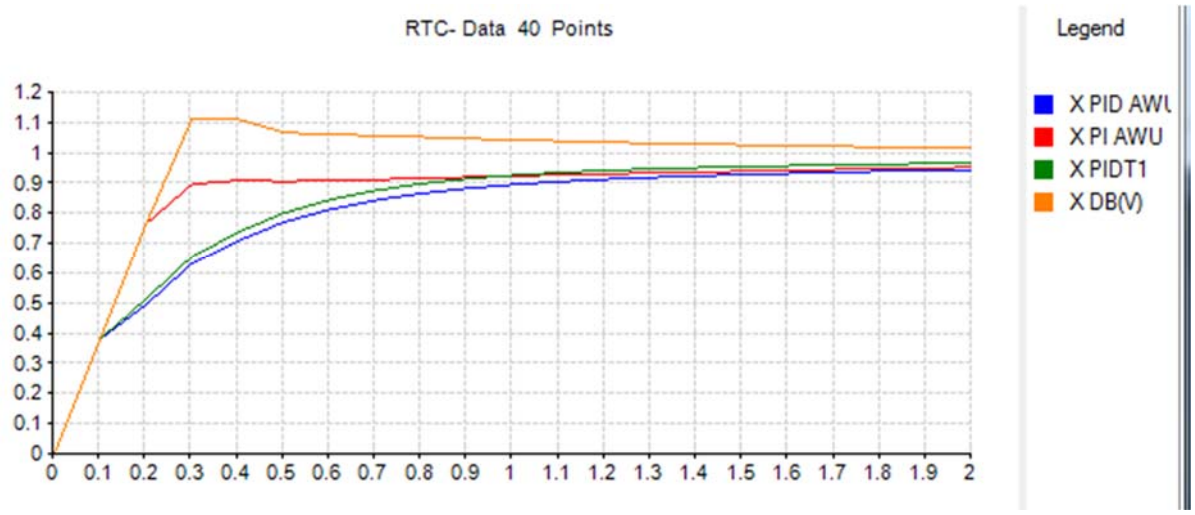
$H_0F(z) = \frac{x_m(z)}{y(z)} = \frac{b_1 z^{-1}}{1 + a_1 z^{-1}}$, the model degree is one. For this transfer function, the linear difference equation can easily be written as $x_m(k) = b_1 y(k-1) - a_1 x_m(k-1)$.
With $F(p) = K/(1+pT)$ it follows $a_1 = -\exp(-T_0/T)$, $b_1 = K(1+a_1)$.
We want to use the special example K=0.4, T=1, T_o=0.1 and a limitation of controller output of 10. Reference step amplitude 1. The values of F(z) are:

	ai	bi
1	-0.904837418	0.03806503279

A comparable DBC has the following parameters:

	qi	pi
1	26.27083	1
	-23.77083	1

The RSRs of standard controller with WindfC#:



9.7.2 Theory of version 1st order DB with limited controller output (any delay)

Let the starting point be x(n)=x_m(n)= 0. Desired value is one. With model function a prediction can be made:

$$x_m(n+1) = b_1 y(n) - a_1 x(n) = 0.038065\, y(n) + 0.90484 x(n)$$

This can be used to calculate the necessary controller output y(n) to get the desired value w=1 from any starting point x(n):

$$x_m(n+1) = b_1 y(n) - a_1 x(n) = w$$

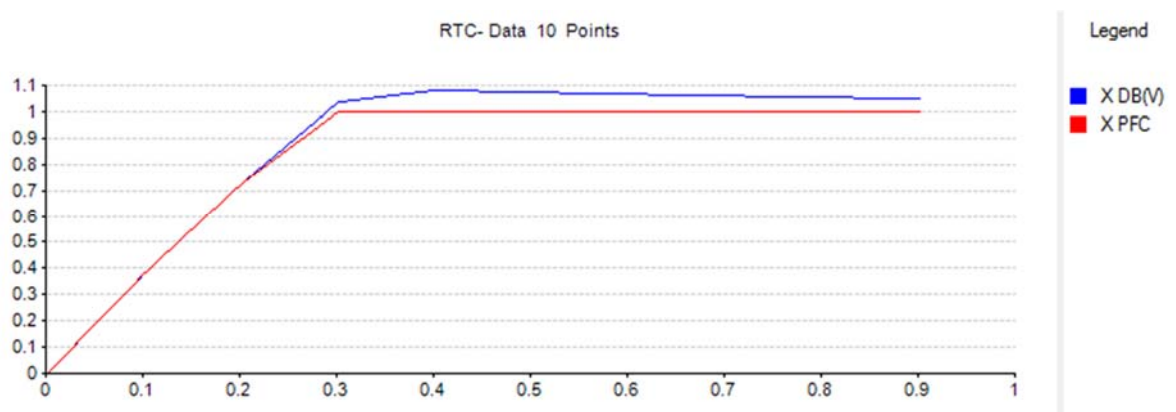
In simple PT1- case the solution is simple:

$$y(n) = \frac{w + a_1 x(n)}{b_1}$$

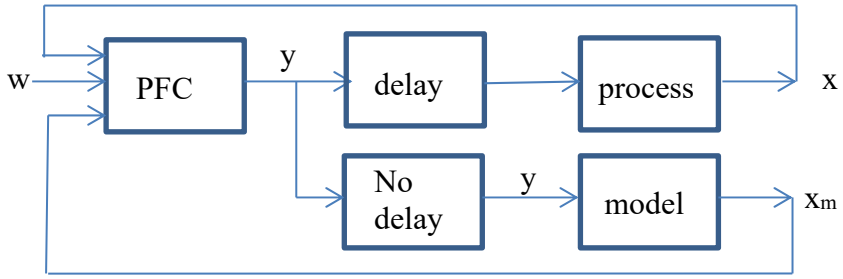
The first controller output value and following steps are:

n	y(n)	y(n) limited	x_m(n)
0	26.27	10	0
1	17.22	10	0.38065
2	9.035	9.035	0.72508
3	2.5	2.5	1
4	2.5	2.5	1

If there is no limitation, this controller behaves like a DB, the first impulse is the q_o of the DBC. But with limitation this PFC has no overshoot, in opposite to the DBC:



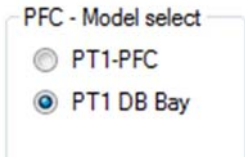
If a real algorithm is used to be independent of hardware delays, the algorithm has to be changed in the following version. The process has one additional delay To:



The undelayed model output is:

$$x_m(k) = b_1 y'(k-1) - a_1 x(k-1)$$

This previous equation is modified into this version:



$$y(k) = \frac{w + a_1\left(x_m(k) + x(k) - x_m(k-d)\right)}{b_1}$$

This is called Smith-Predictor (see more in Chap 9.11).
So all model output values have to be stored into an array, size depends on maximum d-value. y(k) is calculated with the undelayed model output $x_m(k)$. The feedback is realized with the delayed real measured output x(k). If model and process are identical, this process output is compensated with the delayed model output $x_m(k-d)$.
Code in WindfC#:

```
fXm[0] = fXm[1] * fa1 + fb1 * fYr;      // Modeloutput now
valfilt = val + fXm[0] - fXm[delay];    // delay- comp. like PFC
fYr = (w + fa1 * valfilt) / fb1;        //controller output

for (hzp = MaxDelay-1; hzp >= 1; hzp--)
    fXm[hzp] = fXm[hzp - 1];
```

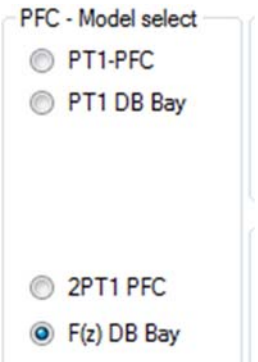
with the prepared variables in design:

```
fa1 = -Math.Exp(-fT0 / fT1);
fb1 = fKm*(1 + fa1);

in fT1 Time constant T of model PT1
in fKm Gain of model PT1
fT0 is sampling time

w desired and val actual x value
```

9.7.3 Theory of version 2nd order DB with limited controller output (any delay)



Now I want to introduce this version with 2nd order processes. The idea is similar to Dead Beat. In a second order process one step is not sufficient to reach the final value, a Dead Beat controller – the fastest possible one – needs two steps. So following equations are valid:
Process equation is used to predict future values (k+1, k+2 …):
x(k+2) should reach the desired value w and all further x(k+x) too.

$$H_0F(z) = \frac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} = \frac{X(z)}{Y(z)} = \frac{output}{input}$$

$$Equ(1):\quad x(k) = -a_1 x(k-1) - a_2 x(k-2) + b_1 y(k-1) + b_2 y(k-2)$$

$$Equ(2):\quad x_m(k+1) = -a_1 x(k) - a_2 x(k-1) + b_1 y(k) + b_2 y(k-1)$$

$$Equ(3):\quad x(k+2) = -a_1 x_m(k+1) - a_2 x(k) + b_1 y(k+1) + b_2 y(k) = w$$

$$Equ(4):\quad x(k+3) = -a_1 x(k+2) - a_2 x(k+1) + b_1 y(k+2) + b_2 y(k+1) = w$$

$$Equ(5):\quad x(k+4) = -a_1 x(k+3) - a_2 x(k+2) + b_1 y(k+3) + b_2 y(k+2) = w$$

In Equ(1) all values are known and measured. In Equ(2) the value $x_m(k+1)$ is a predicted output of the process and unknown. y(k) is the unknown new PFC- controller output. In Equ(3) x(k+2) should reach the desired value w. A third unknown y(k+1) appears in this equation. In Equ(4) y(k+2) is the value which holds the final value and can be called y(k+2) = y(∞). x(k+2) =w. So Equ(4) and Equ(5) can be rewritten as

$$Equ(4):\quad w = -a_1 w - a_2 x_m(k+1) + b_1 y(\infty) + b_2 y(k+1)$$

$$Equ(5):\quad x(k+4) = w = -a_1 w - a_2 w + b_1 y(\infty) + b_2 y(\infty)$$

The solution of Equ(5) gives the final value of controller output to hold the desired value at process output, which is w/K, if K is the process DC – gain.

$$Equ(6):\quad y(\infty) = w\frac{1 + a_1 + a_2}{b_1 + b_2}$$

Now we have with Equ(2), Equ(3) and Equ(4) three equations with three unknowns, which can be solved:

$$Equ(2):\quad \boxed{x_m(k+1)} = -a_1 x(k) - a_2 x(k-1) + b_1 \boxed{y(k)} + b_2 y(k-1)$$

$$Equ(3):\quad w = -a_1 \boxed{x_m(k+1)} - a_2 x(k) + b_1 \boxed{y(k+1)} + b_2 \boxed{y(k)}$$

$$Equ(4):\quad w = -a_1 w - a_2 \boxed{x_m(k+1)} + b_1 y(\infty) + b_2 \boxed{y(k+1)}$$

With the starting values z_1 und z_2

$$Equ(7):\quad z_1 = -a_1 x(k) - a_2 x(k-1) + b_2 y(k-1)$$

$$Equ(8):\ z_2 = -a_2 x(k)$$

I got the following solutions:

$$Equ(9):\ x_m(k+1) = \frac{b_1\left[b_2\left(w - z_2 + b_2 z_1/b_1\right) - b_1\left(w + a_1 w - b_1 y(\infty)\right)\right]}{b_2^2 - a_1 b_1 b_2 + a_2 b_1^2}$$

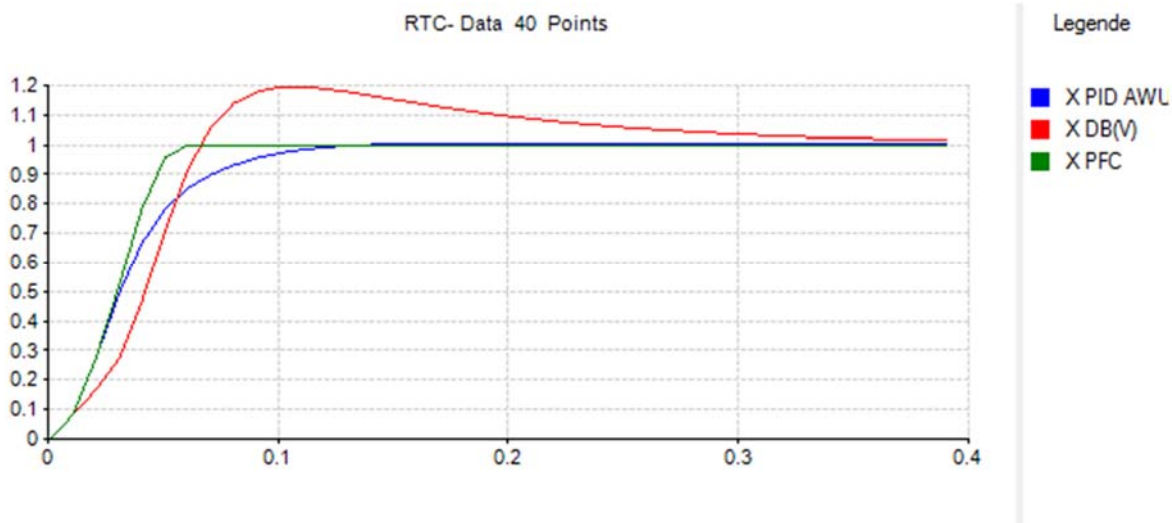
$$Equ(10):\ y(k) = \left(x_m(k+1) - z_1\right)/b_1$$

$$Equ(11):\ y(k+1) = \left(w + a_1 x_m(k+1) + z_2 - b_2 y(k)\right)/b_1$$

If process has a delay d with T_{del}=d*To, again replace x(k) with x_m (k) +x(k) – x_m(k-d) as described before as Smith Predictor (9.11). Undelayed model output x_m(k) can be calculated with

$$x_m(k) = -a_1 x(k-1) - a_2 x(k-2) + b_1 y(k-1) + b_2 y(k-2)$$

Results with above process 2PT1 model with K=0.4, T_1=0.1s T_2=0.02s and T_0=10ms. With actuator limit +-10V.



Green curve shows this PFC with absolute ideal curve. If no limit of actuator occurs this controller is identical with the traditional dead beat one!

9.7.4 Theory of version 3rd order DB bay with any delay To

Now finally the solution for a process with a degree of 3 should be described. With same ideas as above we get following equations. A 3rd order process need at least three steps, so x(k+3) should reach the desired value w and all further x(k+x) too.

$$H_0F(z) = \frac{b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}} = \frac{X(z)}{Y(z)} = \frac{output}{input}$$

$$Equ(1):\ \boxed{x_m(k+1)} = -a_1 x(k) - a_2 x(k-1) - a_3 x(k-2) + b_1 \boxed{y(k)} + b_2 y(k-1) + b_3 y(k-2)$$

$$Equ(2):\ \boxed{x_m(k+2)} = -a_1 \boxed{x_m(k+1)} - a_2 x(k) - a_3 x(k-1) + b_1 \boxed{y(k+1)} + b_2 \boxed{y(k)} + b_3 y(k-1)$$

$$Equ(3):\ x_m(k+3) = -a_1 \boxed{x_m(k+2)} - a_2 \boxed{x_m(k+1)} - a_3 x(k) + b_1 \boxed{y(k+2)} + b_2 \boxed{y(k+1)} + b_3 \boxed{y(k)} = w$$

$$Equ(4):\ x_m(k+4) = -a_1 w - a_2 \boxed{x_m(k+2)} - a_3 \boxed{x_m(k+1)} + b_1 y_{DC} + b_2 \boxed{y(k+2)} + b_3 \boxed{y(k+1)} = w$$

$$Equ(5):\ x_m(k+5) = -a_1 w - a_2 w - a_3 \boxed{x_m(k+2)} + b_1 y_{DC} + b_2 y_{DC} + b_3 \boxed{y(k+2)} = w$$

Starting with k+3 the y should be constant and is again the y(∞) = y_{DC}. All unknowns are indicated with red boxes. We have 5 unknowns with 5 equations, so solvable.
The solution is easy described in matrix- form. Sorted to unknowns left and knowns right:

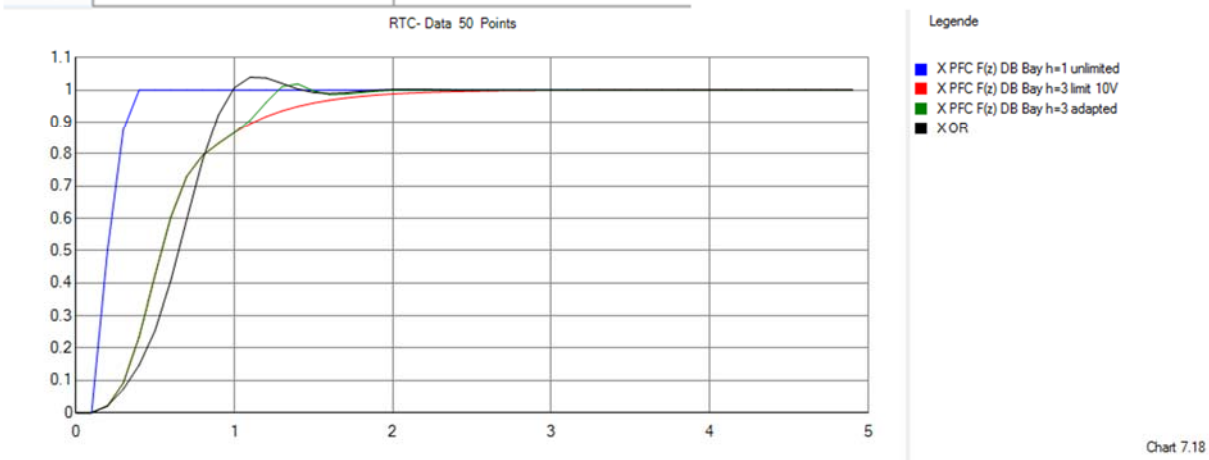
$$\begin{pmatrix} -a_1x(k)-a_2x(k-1)-a_3x(k-2)+b_2y(k-1)+b_3y(k-2) \\ -a_2x(k)-a_3x(k-1)+b_3y(k-1) \\ -w-a_3x(k) \\ -w-a_1w+b_1y_{DC} \\ -w-a_1w-a_2w+b_1y_{DC}+b_2y_{DC} \end{pmatrix} = \begin{pmatrix} -b_1 & 1 & 0 & 0 & 0 \\ -b_2 & a_1 & 1 & -b_1 & 0 \\ -b_3 & a_2 & a_1 & -b_2 & -b_1 \\ 0 & a_3 & a_2 & -b_3 & -b_2 \\ 0 & 0 & a_3 & 0 & -b_3 \end{pmatrix} \begin{pmatrix} y(k) \\ x_m(k+1) \\ x_m(k+2) \\ y(k+1) \\ y(k+2) \end{pmatrix}$$

Z=A * Y
With the following solution:
$Y=A^{-1}*Z$
The controller output y(k) can then be calculated. A matrix inversion is necessary, but no problem, because A contains only a and b coefficients. So the inversion can be done onr time before start of controlling. If online adaptive method is used, inversion is necessary in each To- step.
Delays can be added with same idea than before.
If process has a delay d with Tdel=d*To, again replace x(k) with xm (k) +x(k) – xm(k-d) as described before. Undelayed model output xm(k) can be calculated with

$$x_m(k) = -a_1x(k-1) - a_2x(k-2) - a_3x(k-3) + b_1y(k-1) + b_2y(k-2) + b_3y(k-3)$$

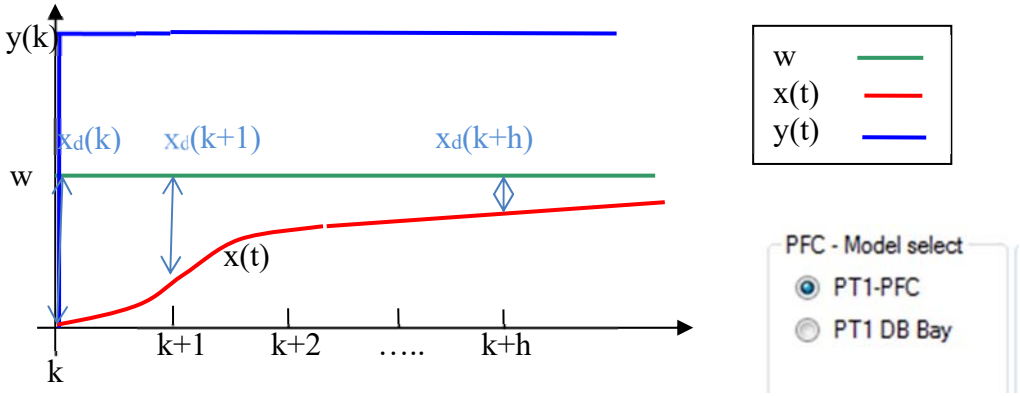
Example: 3PT1-process F(z) given as screenshot WindfCsp: (file "*Fz with trapezoidal 100ms -bo0.fz*")

	ai	bi
0	1	0
1	-2.64830438378825	0.00207873
2	2.33564929693962	0.0015590570719603
3	-0.686021505376344	0.000519685690653433



Blue without limit, ready as fast as DB traditional after 4 steps (m+1)To
Red: h=3 (h=1 and h=2 become unstable behavior)
Green: h=3, h- adapted in that way that it is switched to 1 if no limit occurs
Comparable OR- controller
Since V1.8 a 4- degree controller is available, see WindfC# V7.6.22 +. Theory and results see end of this paper in Chap 9.10.

9.7.5 Theory of Richalet - PFC with PT1- process with any delay d*To



See [1]. The control error (difference) $x_d(k) = w - x(k)$ now should be decreased by each step with a constant factor λ_r (reduction ratio), y(k) is assumed to be constant (will be actualized in each step).
$x_d(k+1) = \lambda_r * x_d(k)$,
$x_d(k+2) = \lambda_r * x_d(k+1) = \lambda_r^2 * x_d(k)$,
$x_d(k+h) = \lambda_r^h * x_d(k)$,

h: number of steps to reach the final value with about 5%, called horizon.

A PT1 step response reaches the final value within 5% in about three times the time constant of this PT1: $(h(t) = (1 - \exp(-t/T)) = 0.95 \rightarrow t = 3 * T = T_c$. Here T_c is the desired 5% - settling time of the closed loop.
So with $T=T_c/3$ this gives the control difference for a unit step (w=1) $xd(t) = 1-h(t) = \exp(-t/T) = \exp(-3t/T_c)$.
In one step: $t=T_0$: $\lambda_r = \exp(-3T_o/T_c)$.
In h steps: $t=h*T_0$: $\lambda_r^h = \exp(-3T_o*h/T_c)$.

Example : T_o=1ms , T_c=50ms : $\lambda_r = \exp(-3*1ms/50ms)=0.9417$ (one step).
h=50: (50 steps, now $\lambda_r^{50}=0.0498$ = about 5%). Typical values : $T_c=h*T_o$.

This is the definition of the goal of the PFC- control. Now which controller output y(k) is necessary to reach this goal? See this:

Now with the PT1- process model a prediction is possible:
$x(k) = -a_1 * x(k-1) + b_1 * y(k-1)$.
$a_1=-\exp(-T_0/T_m)$ and $b_1 = K_m*(1+a_1)$ with K_m and T_m as model parameters.
So:
$x_m(k) = -a_1 * x(k-1) + K_m*(1+a_1) * y(k-1)$ (note: not = actual measured process output x(k))
With assumed constant controller output y(k) we get the next steps:
$x_m(k+1) = -a_1 * x_m(k) + K_m*(1+a_1) * y(k)$.
$x_m(k+2) = -a_1 * x_m(k+1) + K_m*(1+a_1) * y(k)$.
…
$x_m(k+h) = -a_1 * x_m(k+h-1) + K_m*(1+a_1) * y(k)$.

With successive setback:
$x_m(k+2) = -a_1 *(-a_1 * x_m(k) + K_m*(1+a_1) * y(k)) + K_m*(1+a_1) * y(k) =$

$=(-a_1)^2\, x_m(k) + (1+(-a_1))K_m(1-(-a_1)) * y(k)$
$=(-a_1)^2\, x_m(k) + (1-(-a_1)^2)K_m * y(k)$
….
$x_m(k+h) =(-a_1)^h\, x_m(k) + (1-(-a_1)^h)K_m * y(k)$
On both sides subtract $x_m(k)$, this gives the change of model output Δx_m from now to predicted h steps:
$\Delta x_m = x_m(k+h) - x_m(k) = -(1-(-a_1)^h)x_m(k) + (1-(-a_1)^h)K_m * y(k)$
Now we calculate the desired change of the process output $\Delta x = x(k+h)-x(k) = w-x_d(k+h)-x(k)$ with $x_d(k+h) = \lambda_r^h * x_d(k)$ we get :
$\Delta x = w - \lambda_r^h * x_d(k) - x(k) = w - \lambda_r^h * (w - x(k)) - x(k) = (w - x(k))(1-\lambda_r^h)$.
Now both differences Δx_m and Δx are set equal:
$-(1-(-a_1)^h)x_m(k) + (1-(-a_1)^h)K_m * y(k) = (w - x(k))(1-\lambda_r^h)$.

Solved to the final controller output value y(k):

$$y(k) = \frac{(1-\lambda_v^h)(w-x(k)) + x_m(k)(1-(-a_1)^h)}{K_m(1-(-a_1)^h)} = \frac{(1-\lambda_v^h)(w-x(k))}{K_m(1-(-a_1)^h)} + \frac{x_m(k)}{K_m}$$

We can prepare the coefficients of (w-x(k)) and $x_m(k)$ before start of the loop, because they does not change from step to step. So the final steps to calculate the PFC- output are very simple:

$$y(k) = A_0(w-x(k)) + A_1 x_m(k) \quad with \quad A_0 = \frac{(1-\lambda_v^h)}{K_m(1-(-a_1)^h)} \quad and \quad A_1 = \frac{1}{K_m}$$

In PT1- case the h could be simply to 1, then h >1 is useful in the 2PT1- case, see next chapter. A_o is like a P-controller gain multiplied with actual control difference. The second expression is responsible for the zero control error of the PFC- controller. If h is one and Tc is very small = T_o then A_o is near the q_o – value of a dead beat controller.
But now again the trick with Smith predictor (see 9.11) can be used: This gives a very important advantage to standard PID control, because delays cause large negative phases and make loops with PID very slow.
The changed PFC- equation with a process with a delay of $T_d=d*T_0$ (d is integer).

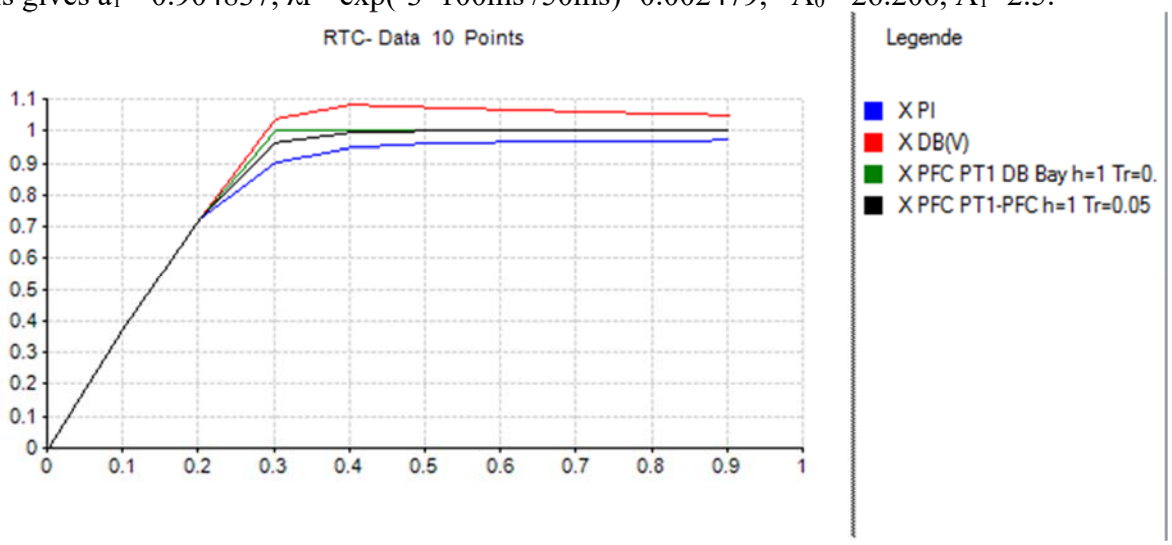
$$y(k) = A_0\left[w - x(k) - x_m(k) + x_m(k-d)\right] + A_1 x_m(k)$$

The delayed real process output x(k) is subtracted by the delayed model output $x_m(k-d)$ and with this replaced by the not delayed predicted model output $x_m(k)$.

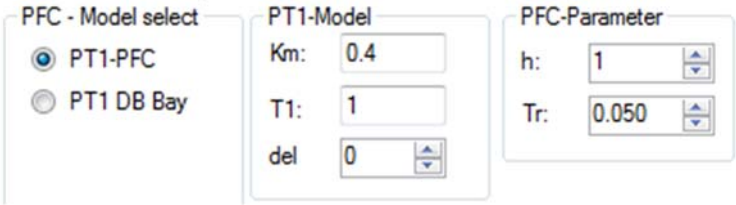
Example: Above PT1 with K=0.4, T=1, To=0.1 and a limitation of controller output of 10.

Reference step amplitude 1, h=1, T_c=50 ms.

This gives a_1=-0.904837, λr= exp(-3*100ms /50ms)=0.002479, A_0= 26.206, A_1=2.5.

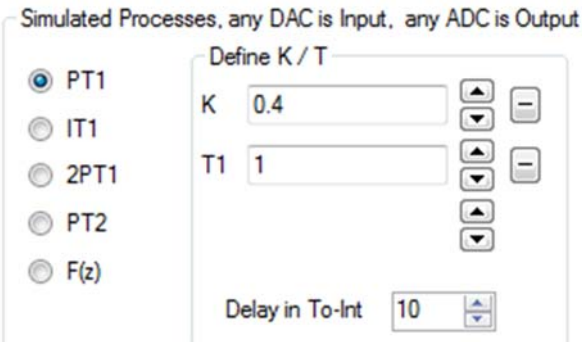


The PFC- settings were

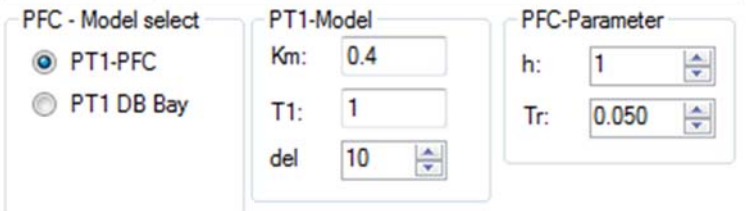


Now with a delay of 1 sec, this is d=10. No DB is available with this delay, also no DB in my version. I compare PI and PFC:

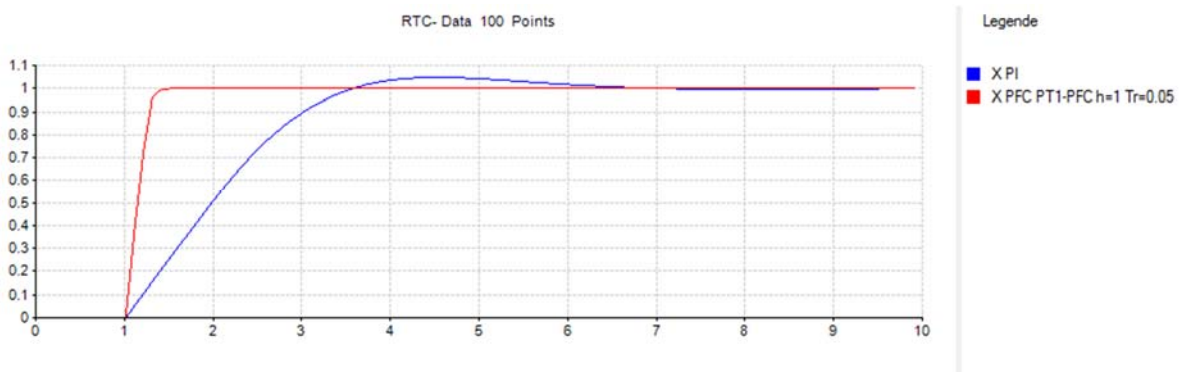
Hardware simulator: PI- design (Kr without delay is 26.18)



PFC- Settings



Result:



Impressive! Or not?

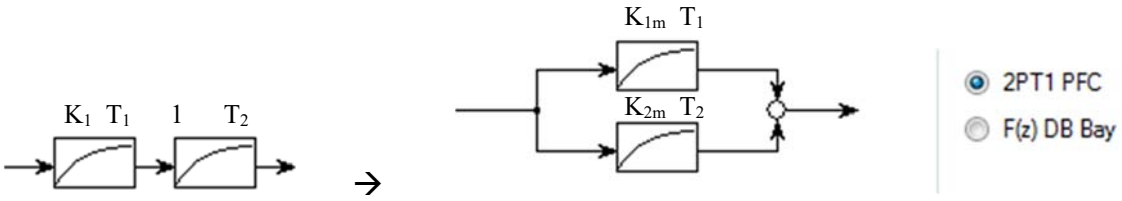
9.7.6 Theory of PFC with 2PT1- process with any delay d*To

If process is a 2PT1, then the design has to be changed, because with one positive impulse the desired value cannot be reached, there must be a breaking step. So the time for the inflection point t_w is calculated of the step response of the model:

$$t_w = \frac{T_1 T_2}{T_1 - T_2} \ln(T_1 / T_2)$$

Now the number of the steps to this inflection point is calculated in the "horizon":

h= t_w/T_0 with truncated decimals (h is integer), this is a recommendation.
To have not to great changes to the PT1- solution, this 2PT1- solution uses not the serial combination of two PT1, but the parallel combination. This is according to publication [9.3].
First we have to convert 2 PT1 serial into two PT1 parallel:



With the conversions:

$$K_{1m} = \frac{K_1 T_1}{T_1 - T_2} \text{ and } K_{2m} = \frac{K_1 T_2}{T_2 - T_1}.$$

Without presented calculation the new PFC- controller output is:

$$y(k) = A_0 \left[w - x(k) - x_m(k) + x_m(k-d) \right] + A_1 x_{m1}(k) + A_2 x_{m2}(k)$$

With

$x_{m1}(k) = -a_{1m} * x_{m1}(k-1) + K_{1m}*(1+a_{1m}) * y(k-1)$
$x_{m2}(k) = -a_{2m} * x_{m2}(k-1) + K_{2m}*(1+a_{2m}) * y(k-1)$

and

$a_{1m}=-\exp(-T_0/T_1)$, $a_{2m}=-\exp(-T_0/T_2)$, $x_m=x_{m1}+x_{m2}$,

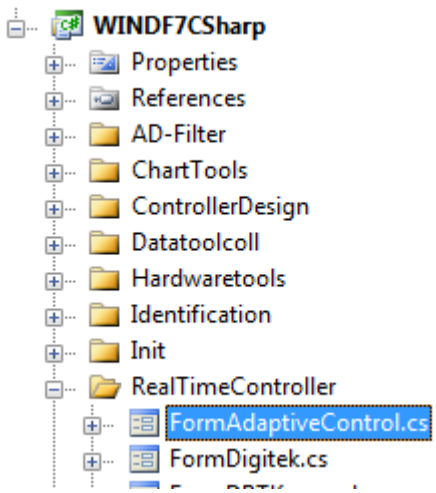
$$A_0 = \frac{(1-\lambda_v^h)}{K_{1m}(1-(-a_{1m})^h) + K_{2m}(1-(-a_{2m})^h)}$$

$$A_1 = \frac{(1-(-a_{1m})^h)}{K_{1m}(1-(-a_{1m})^h) + K_{2m}(1-(-a_{2m})^h)}$$

$$A_2 = \frac{(1-(-a_{2m})^h)}{K_{1m}(1-(-a_{1m})^h) + K_{2m}(1-(-a_{2m})^h)}$$

9.8 PFC- Predictive Functional Control in tool program Windfc#

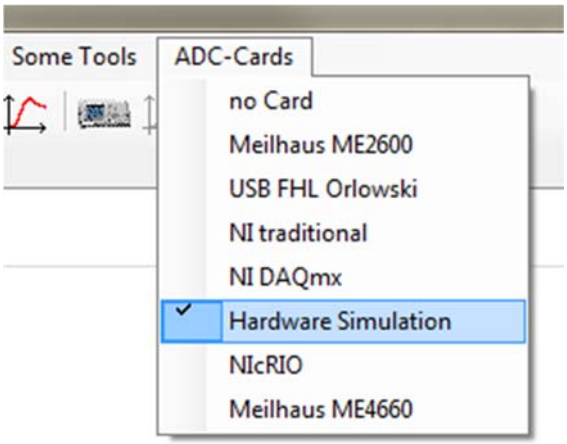
Here you can find some hints to use and test these PFC- controllers in my tool program Windfc# starting with version nb. 7.4.30. The source code is also published, so it should be easy to implement these controllers in other hardware combinations.



You can find the PFC- source in the file "FormAdaptiveControl.cs". The source runs under MS Studio 2010.

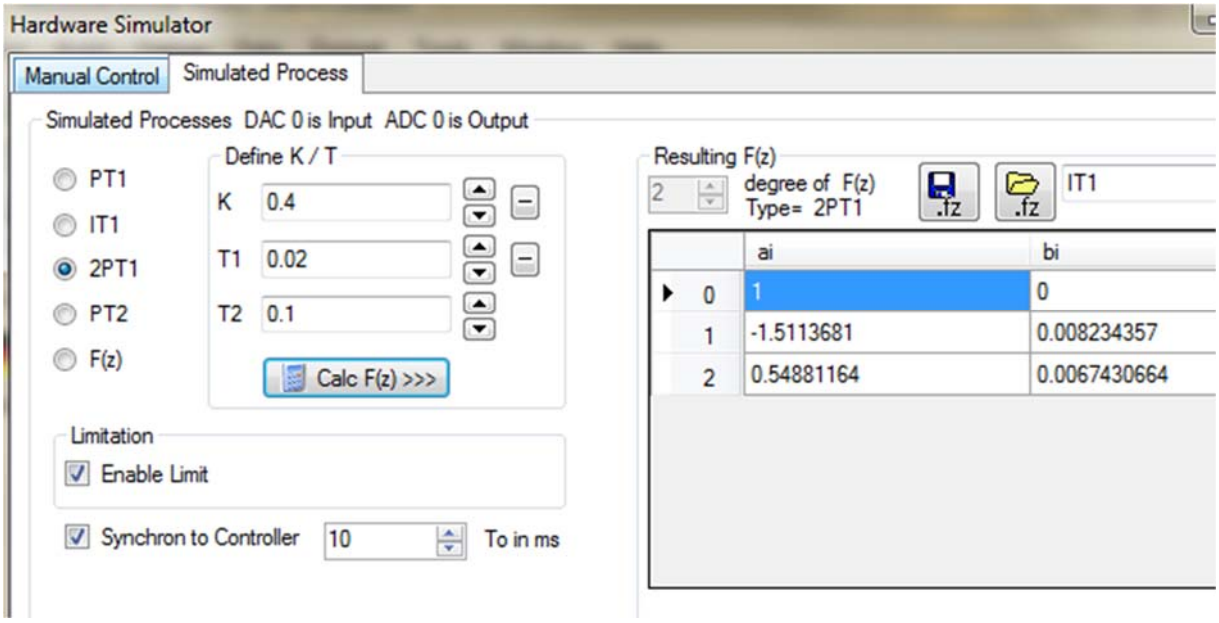
With some screenshots the following pages give instructions to work with a PFC—Controller together

with a simulated 2PT1- process. In Lab we have used this controller together with a hardware of a speed control system with same positive results.
To initialize system following steps are necessary:

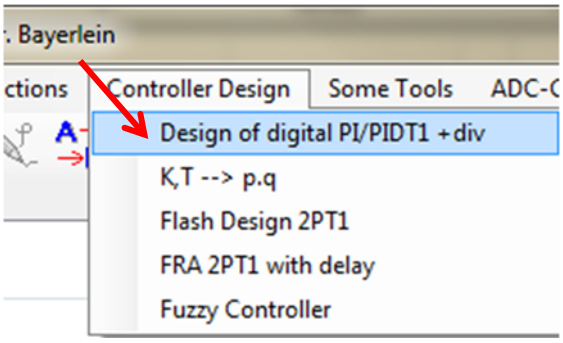


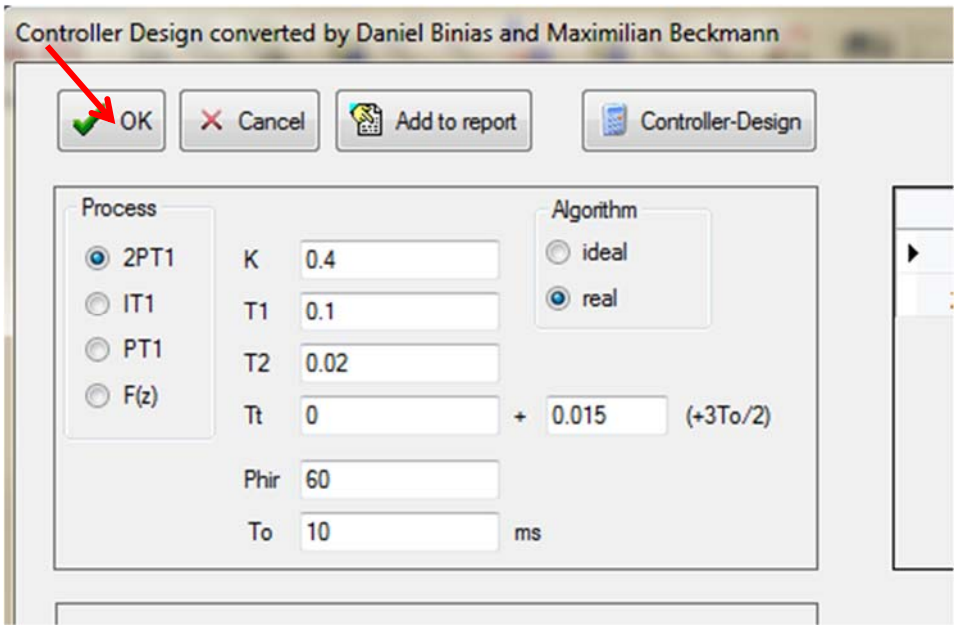
After start Windfc# first select simulated model with Menu "ADC-Cards" → "Hardware Simulation".

In the next window select a 2PT1- process with the above parameters K=0.4, T1=0.02, T2=0.1 and sampling time 0.01 s. This box converts the 2PT1- process into a F(z) digital filter, which is used in the simulation.

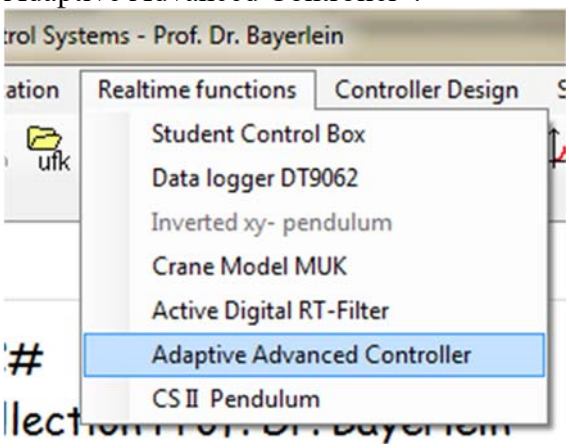


Now it could be a good idea to design some other classical controllers relating to this process. So you can compare PFC- Controller with standard PID or Dead-Beat – controllers. This design is possible in menu "Controller Design" → Design of digital PI/PIDT1…".
In the next Window type in the same process parameters, click on button "Controller Design" and leave the window with the "green arrow button".

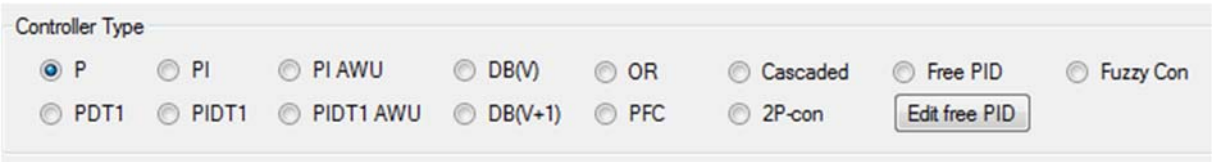




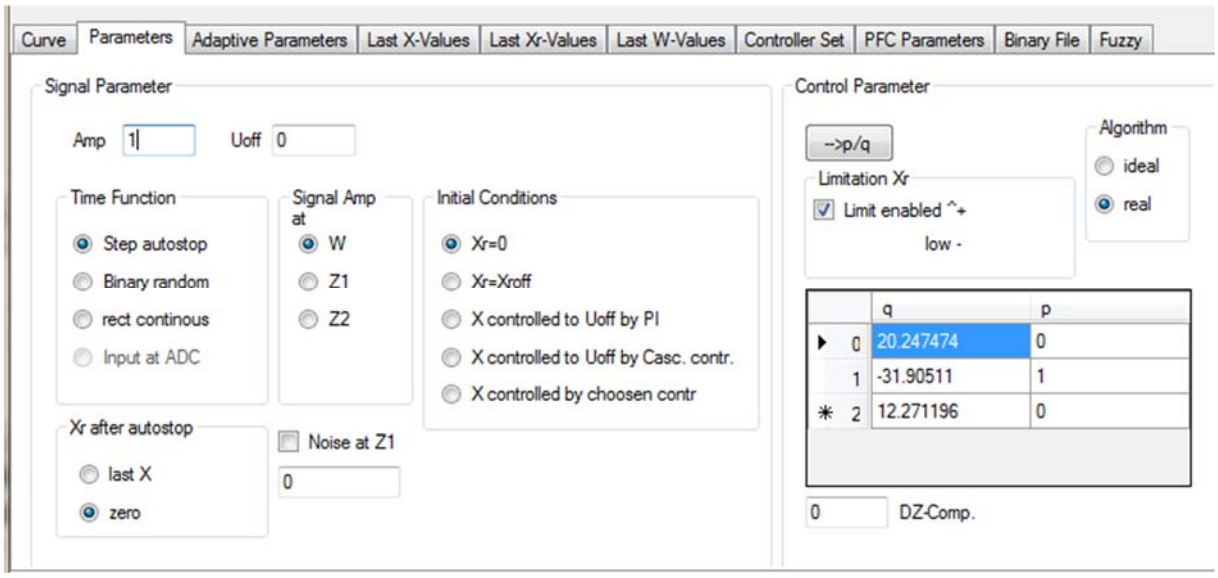
Now open the window with the realtime- controllers with menu "Realtime functions → Adaptive Advanced Controller".



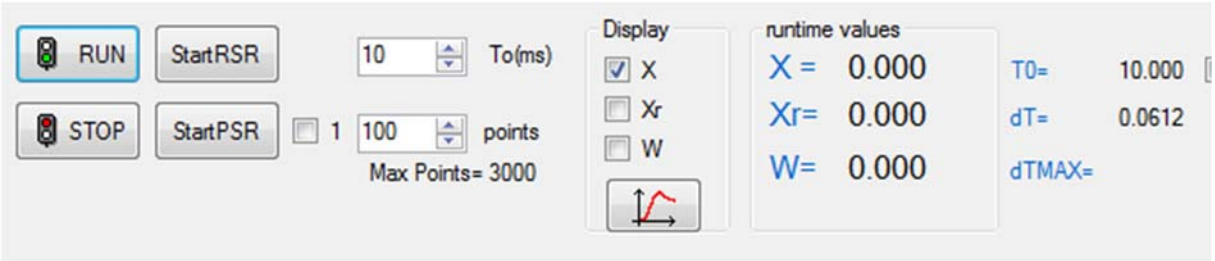
Here you can select different types of controllers.



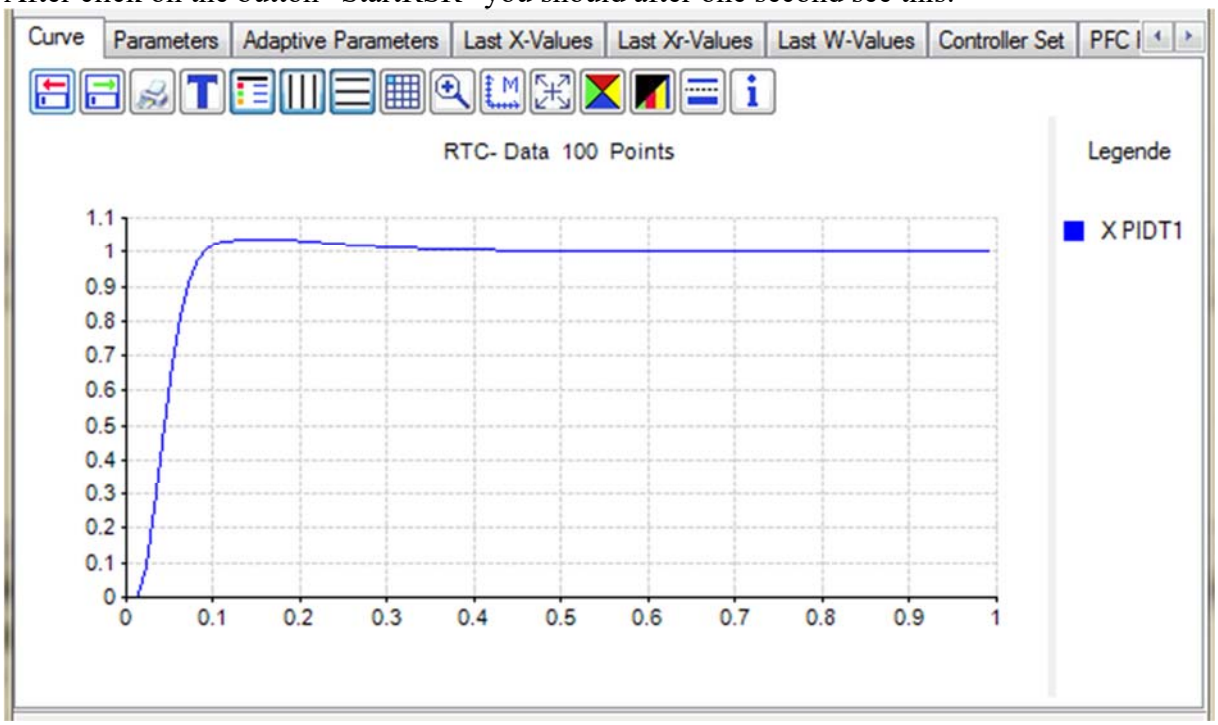
This module can now calculate a single reference step response (RSR) or a continuous response on a square reference signal jumping up and down. The single RSRs are displayed in the folder "Curve", the continuous signal is displayed in real time in a new window.
Lets start with the single RSRs. In folder "Parameters" you can set some settings. Choose the following settings for the first RSR with a PIDT1- controller with this process:



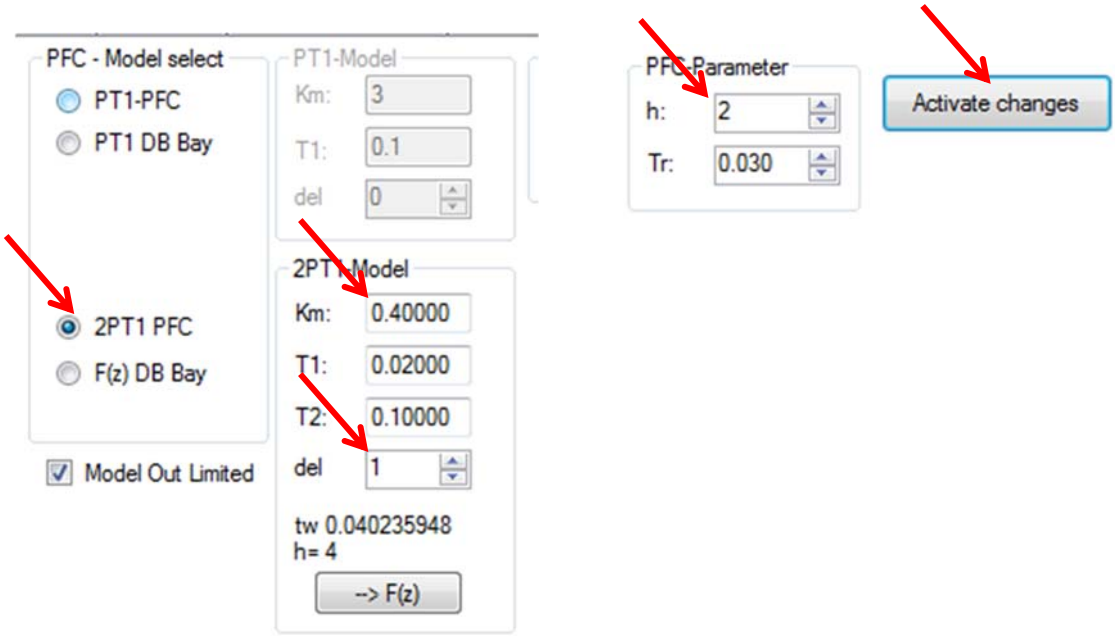
Amp= 1V, Controller type: PIDT1, Uoff=0V, rest default.
After click on RUN – button you can see the system working. You should wait until the output of the process is settled to zero volt before start of the reference step with button "StartRSR".



After click on the button "StartRSR" you should after one second see this:

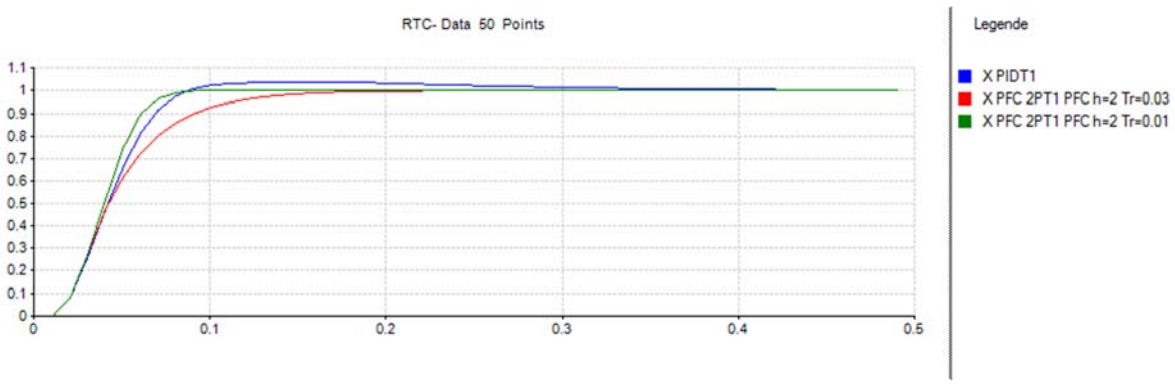


This is the reference step response of our process with a digital PIDT1 in an “real algorithm”, where the delay of the controller is artificially enlarged to one To to avoid problems with hardware dependent delays like calculation time and AD – conversion time.
Now PFC- Controller. The setting are made in the “PFC Parameters”- folder:

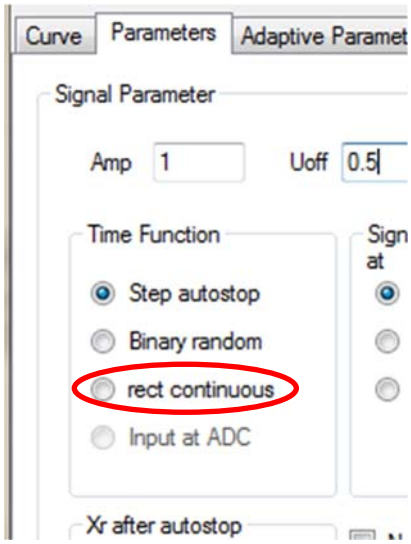


Start first with first second order PFC, change the 2PT1- parameters of the PFC to our process values, select h=2 (two step horizon), click on button “→F(z)” to get the F(z) of the model and click on “Activate changes” – button.

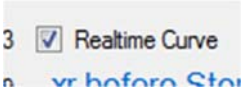
Now select controller type “PFC” and start RSR with RUN and StartRSR – button. Result:



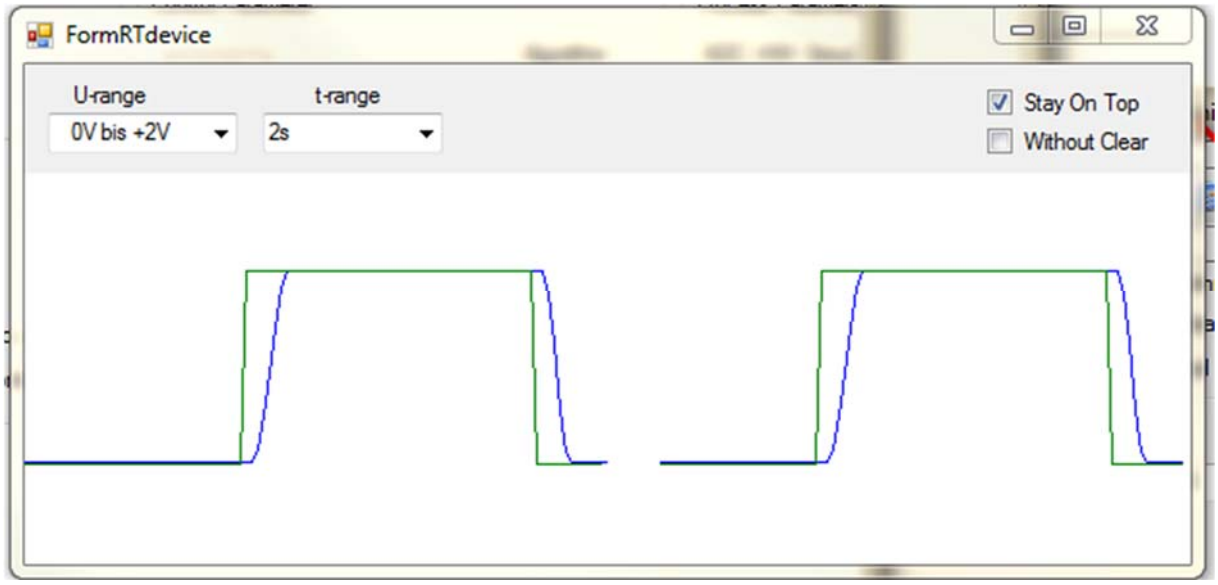
Red and green curves are PFC- results.



Now we try the continuous mode. For this click in folder Parameters on radio button "rect continuous", change the offset voltage to 0.5 V (to get a desired value jumping from 0.5 to 1.5 and back) and activate the real time curve



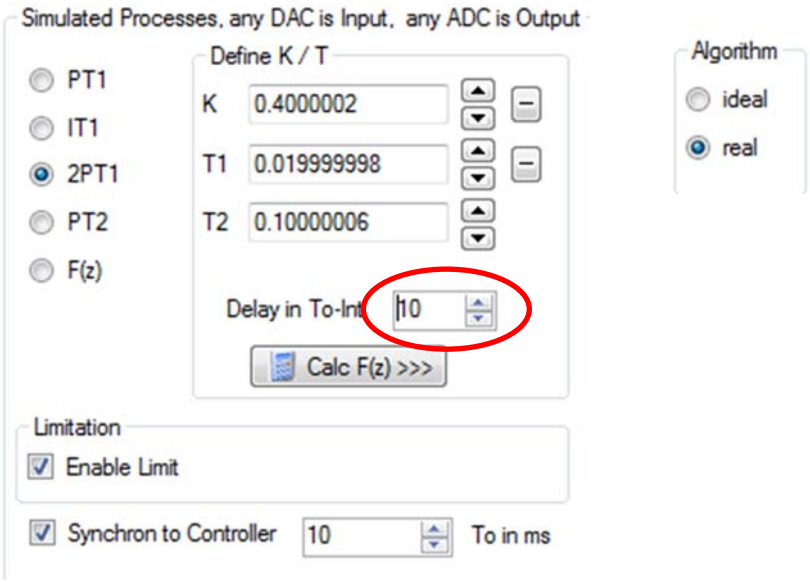
Now you can see the new real – time curve window. Select a vertical range of "0V bis +2V" and click on checkbox "Stay on Top". After Run – button you see this picture, I have activated the desired value curve with checkbox "w":



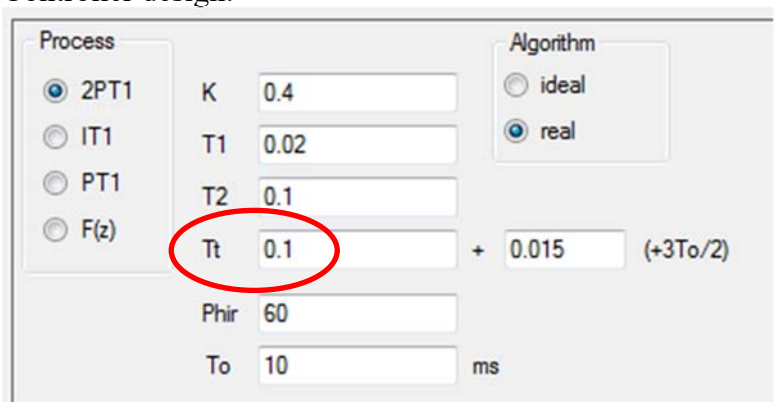
Green curve is desired value w and blue curve is process output x. Now you can play with all parameters and see reaction on the controlled process.

Finally we add a delay of 100 ms to the process to demonstrate the big advantage of PFC with this property. DB / OR / DB with Controller limit are not prepared to work with this delay, but PFC is.
New PID design is necessary and in the PFC parameters make following marked changes.

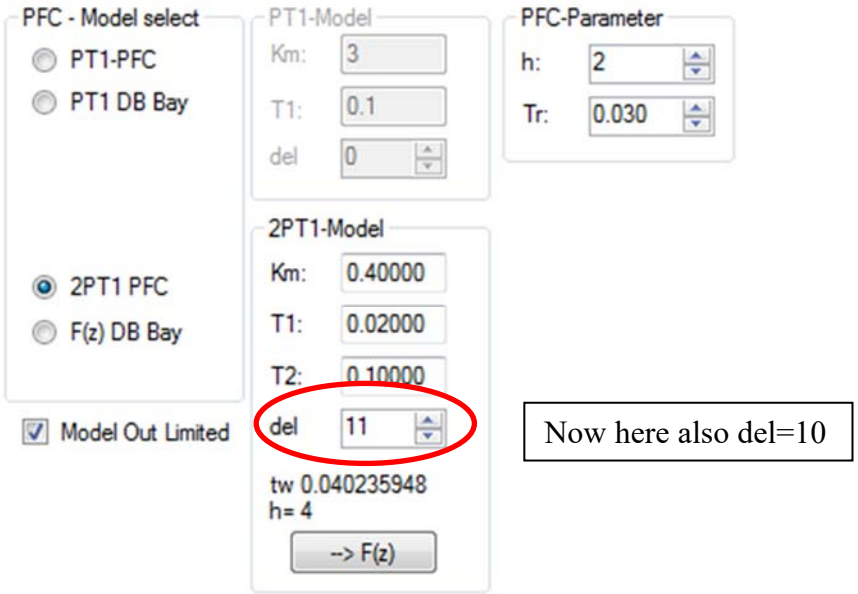
Hardware simulation:



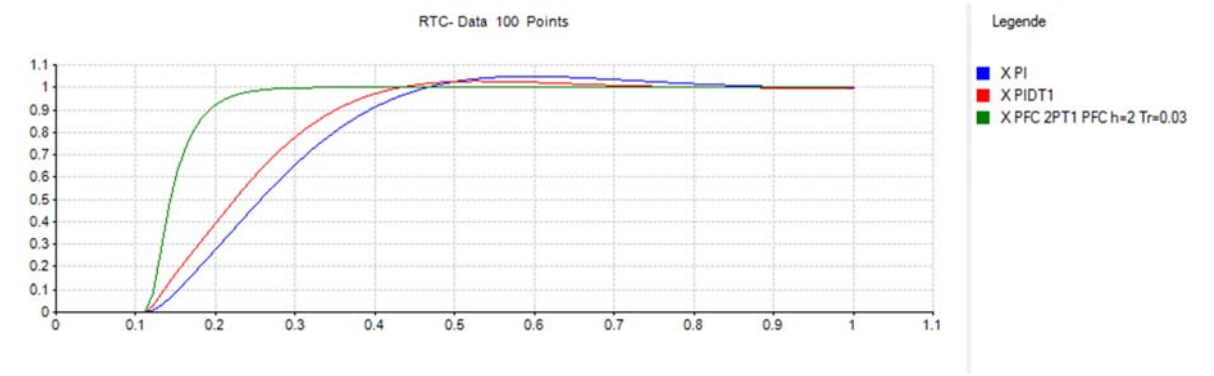
Controller design:



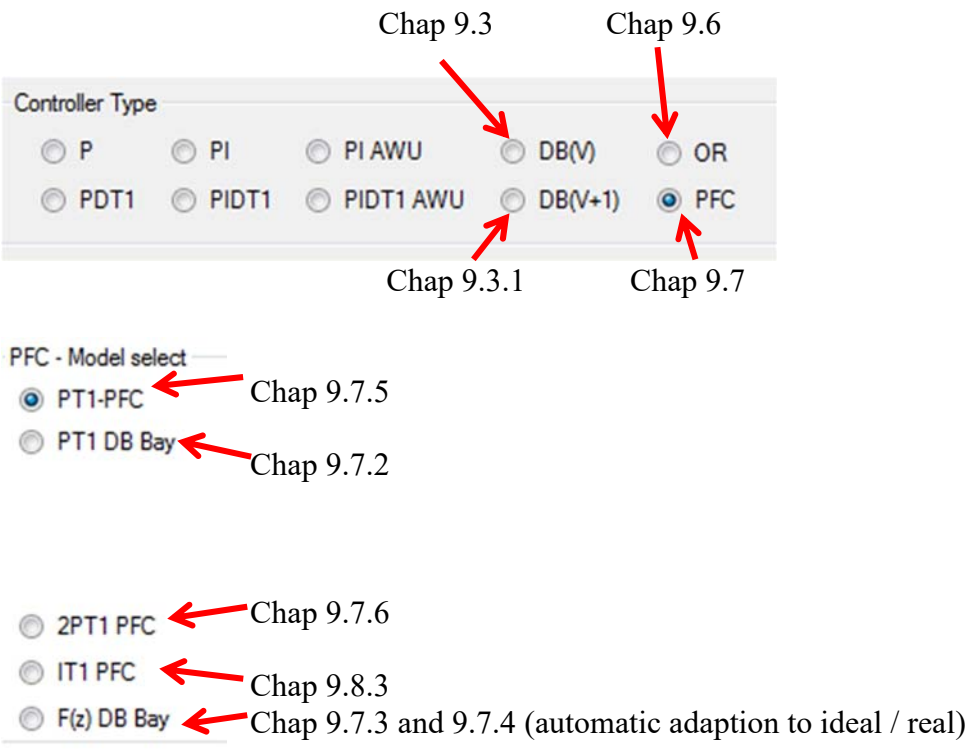
PFC- Parameters: (Since V1.7 d=10, internal automatic change for ideal /real algorithm)



Resulting PI / PIDT1 and PFC- reference step responses:

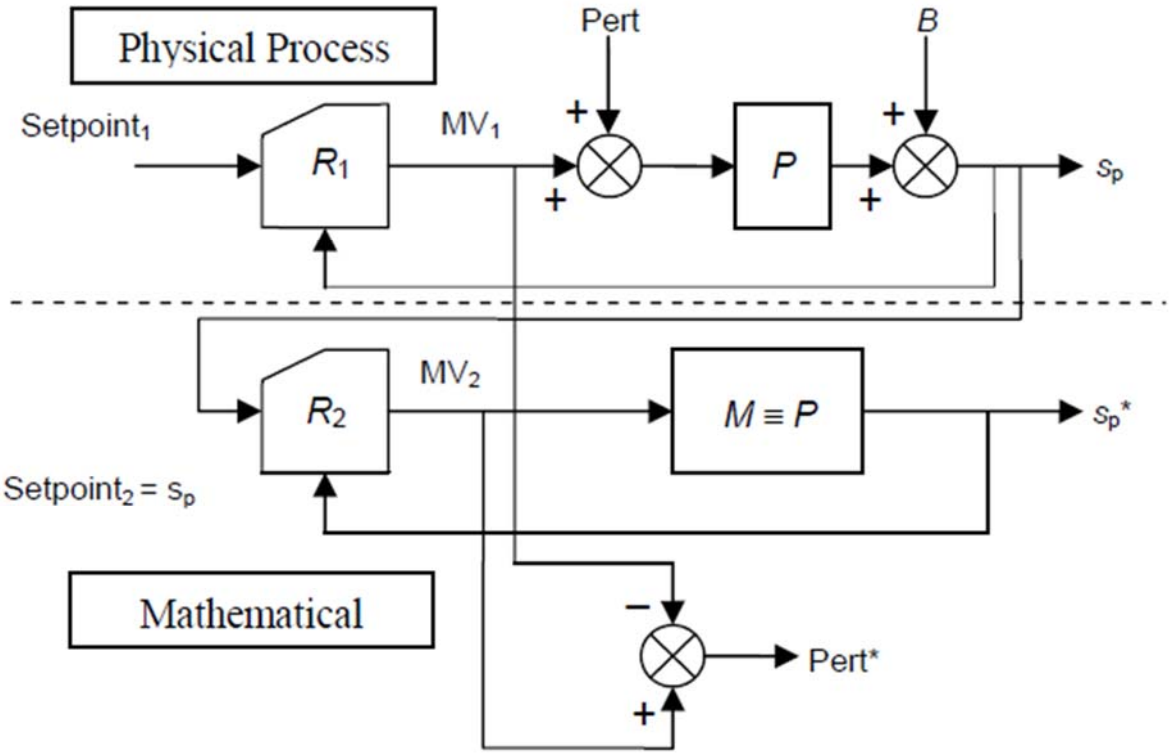


Help in finding the correct controller with description in chapter:



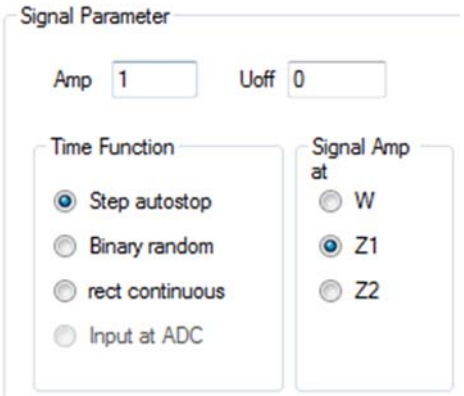
9.8.1 Measurement of disturbance with PFC

A very nice idea of PFC- control is the trick to measure an unknown internal disturbance value, for example to compensate it by disturbance feed forward. The picture is from [9.2]. Assume that B=0, Pert is the disturbance (comes from perturbation).

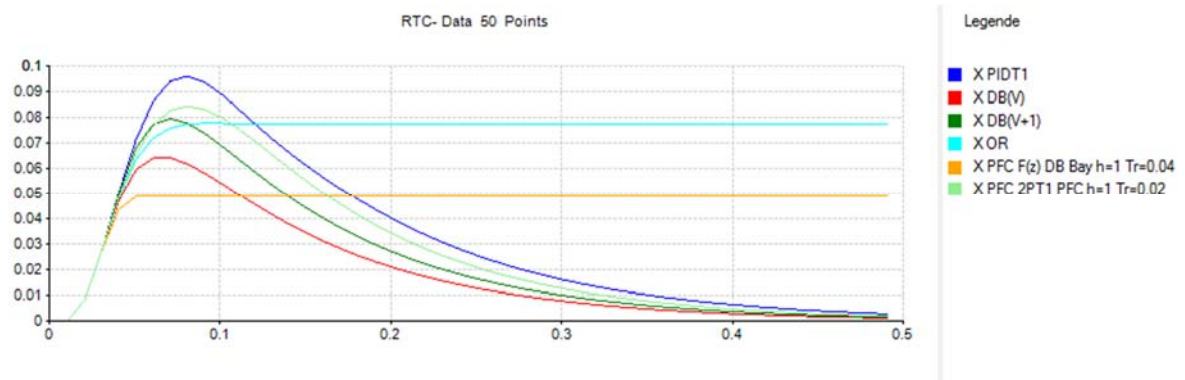


9.8.2 Disturbance behaviour of DBC and PFC

Last remark should be the simulation of a disturbance step in front of process of 1 V. See the reaction of the controllers in the 2 PT1 – process. Settings:



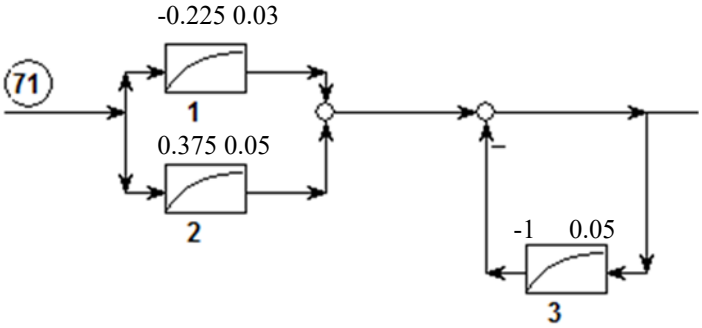
Results:



You can see, that the Orientation controller “OR” and “my” DB are not able to compensate this disturbance, but the other and especially the PFC can! See Chap. 9.13 for more Info.

9.8.3 Theory of PFC with IT1- process with any delay d*To

The principle of PFC needs PT1- elements a1 / b1 and integrators are not directly usable. There is a trick to replace Integrator with decomposition:



The above SBD replaces an IT1 with K=3 and T=0.03. The T_{dec} of Block 2 is a free choice, and the author of 9.2 recommends a value of T_{dec}=Tc/3, if Tc is the settling time of closed loop. All the next equations are coming from [9.2], there chap 4.6.

With the conversions:

$$K_{2m} = \frac{KT_{dec}^2}{T_{dec} - T} \text{ and } K_{1m} = KT_{dec} - K_{2m}.$$

Without presented calculation the new PFC- controller output is:

$$y(k) = A_0\left[w - CVpred\right] + A_1 x_{m1}(k) + A_2(x_{m2}(k) + x_{mr}(k) - CVpred)$$

With

$$CVpred = w + x_{m1}(k) + x_{m2}(k) + x_{mr}(k) - x_{m1}(k-d) - x_{m2}(k-d) - x_{mr}(k-d)$$

$x_{m1}(k) = -a_{1m} * x_{m1}(k-1) + K_{1m}*(1+a_{1m}) * y(k-1)$

$x_{m2}(k) = -a_{2m} * x_{m2}(k-1) + K_{2m}*(1+a_{2m}) * y(k-1)$

$x_{mr}(k) = -a_{2m} * x_{mr}(k-1) + (1+a_{2m}) * CVpred$

and

a_{1m}=- exp(-T_0/T), a_{2m}=- exp(-T_0/T_{dec}),

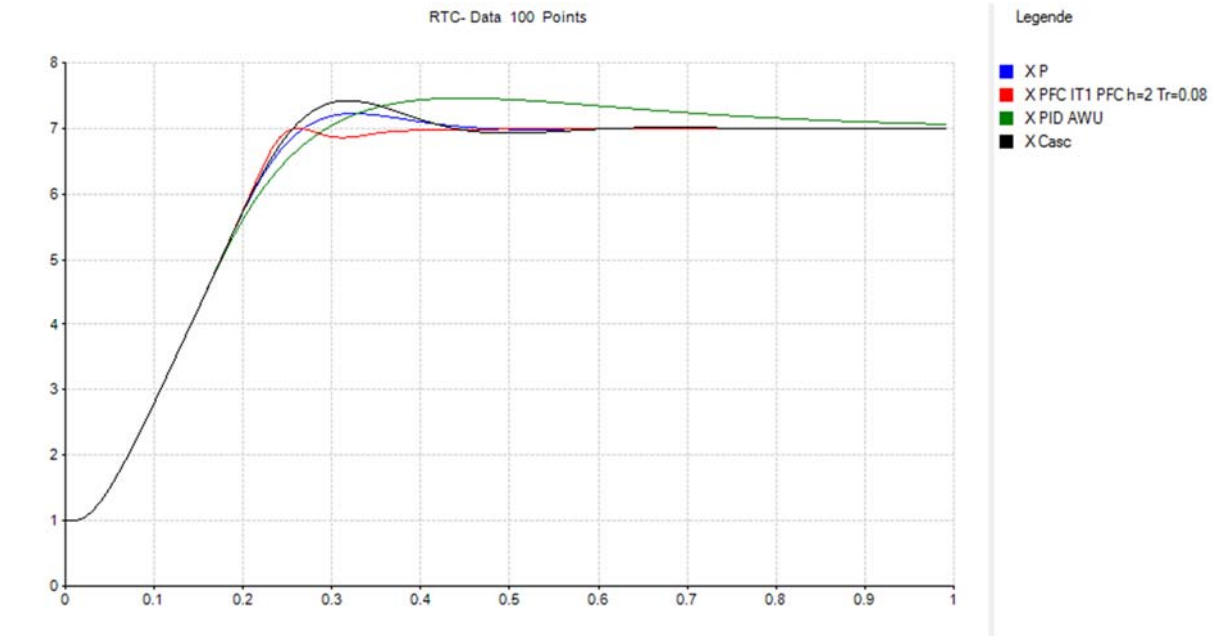
$$A_0 = \frac{(1-\lambda_v^h)}{K_{1m}(1-(-a_{1m})^h) + K_{2m}(1-(-a_{2m})^h)}$$

$$A_1 = \frac{(1-(-a_{1m})^h)}{K_{1m}(1-(-a_{1m})^h) + K_{2m}(1-(-a_{2m})^h)}$$

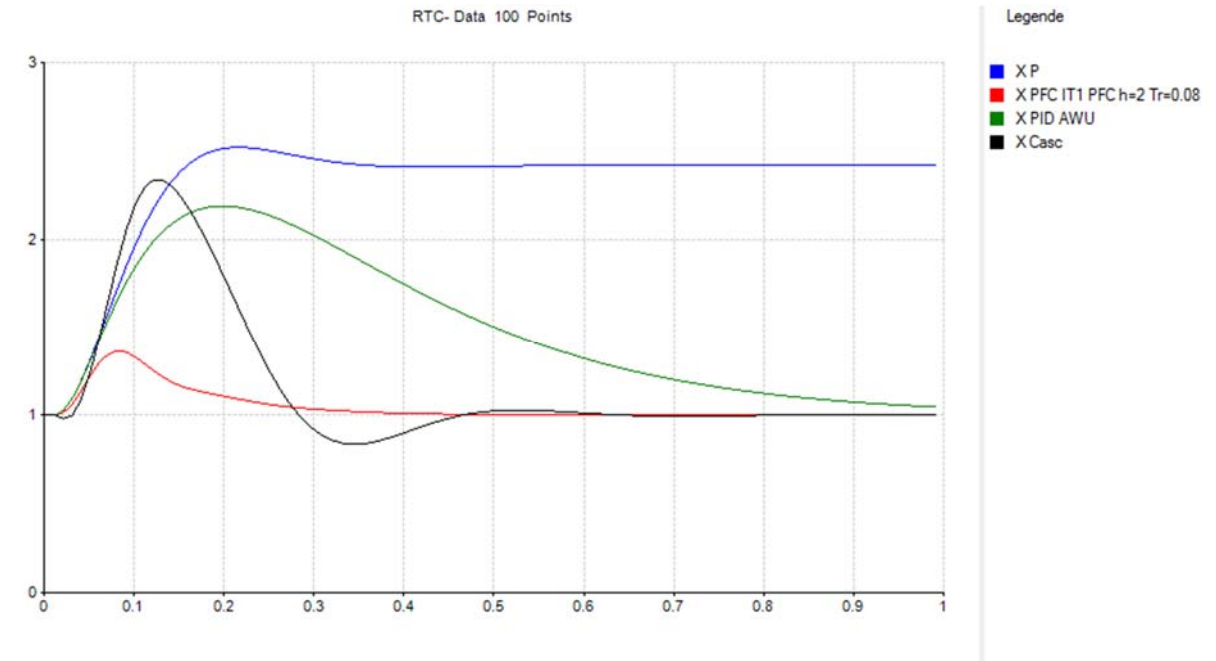
$$A_2 = \frac{(1-(-a_{2m})^h)}{K_{1m}(1-(-a_{1m})^h) + K_{2m}(1-(-a_{2m})^h)}$$

Software realized in Windfc# since 7.4.36

Results with simulated IT1 – block with K=3 and T=30 ms, To=10 ms, T_{dec}=0.08

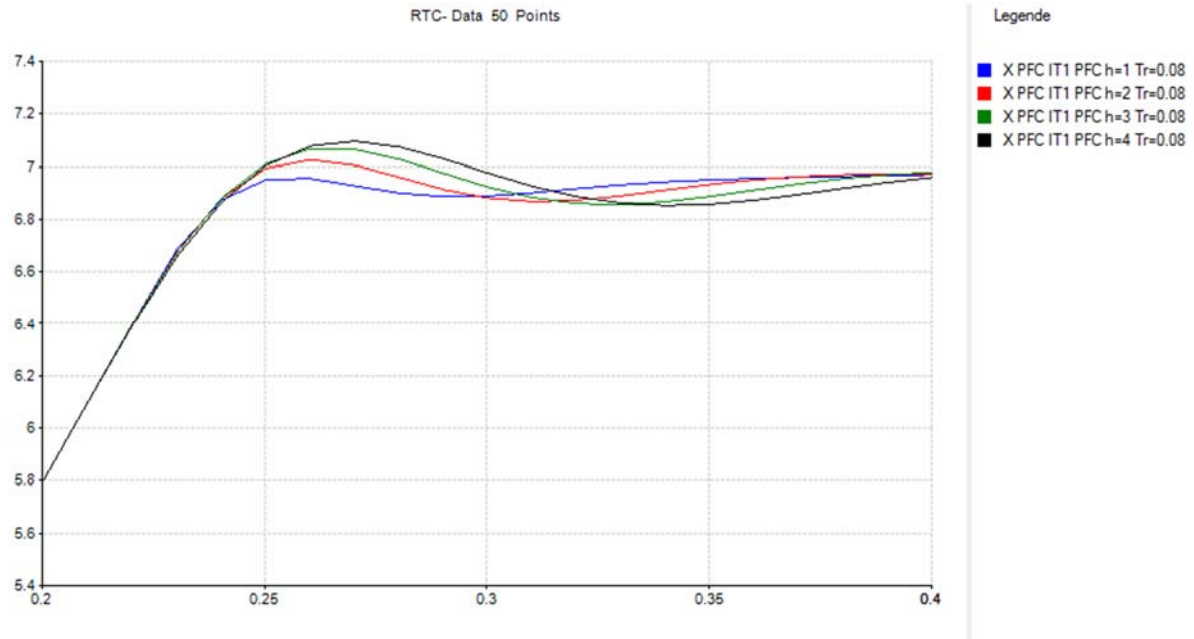


Disturbance behavior:

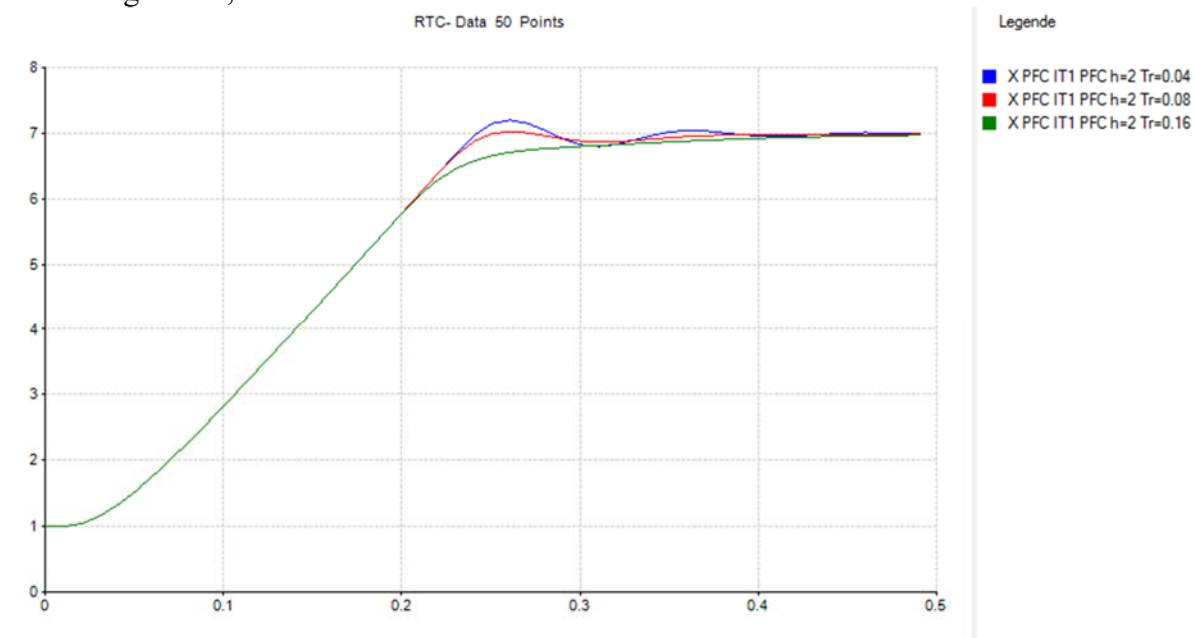


PFC is the best!

Parameter change, first h from 1 to 4

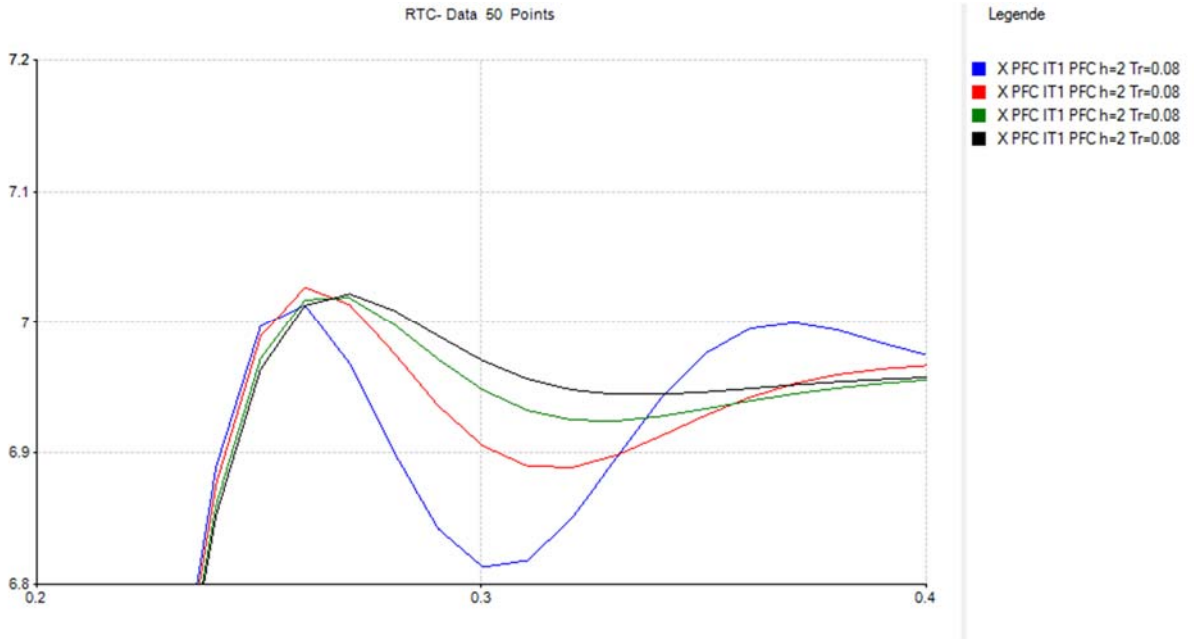


Then change of Tr,h=2:

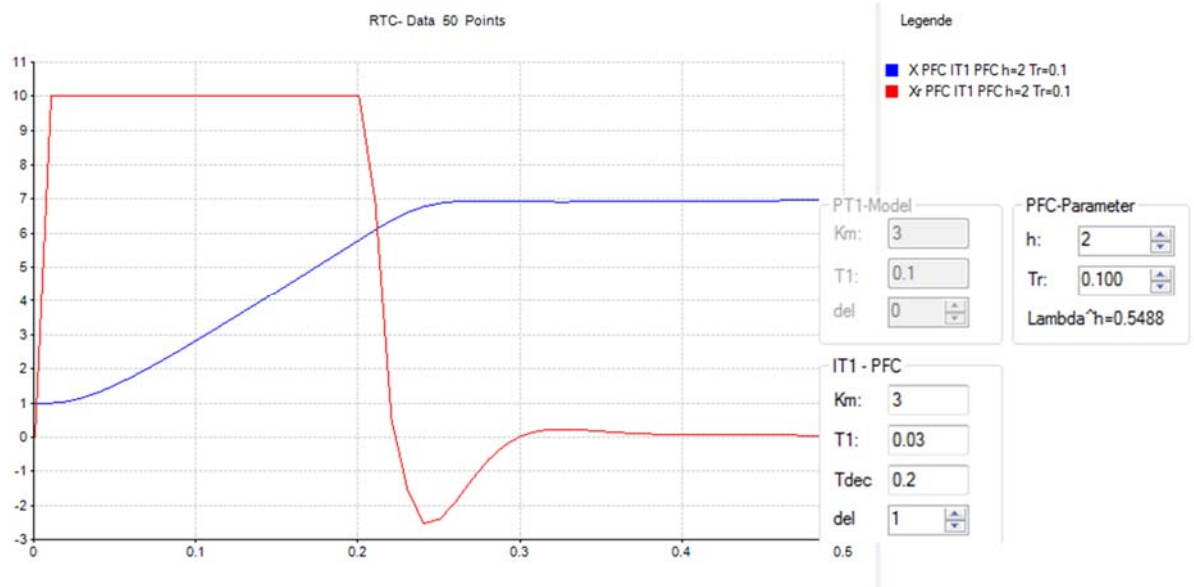


Change of the decomposition time T_{dec}:, h=2, Tr=0.08

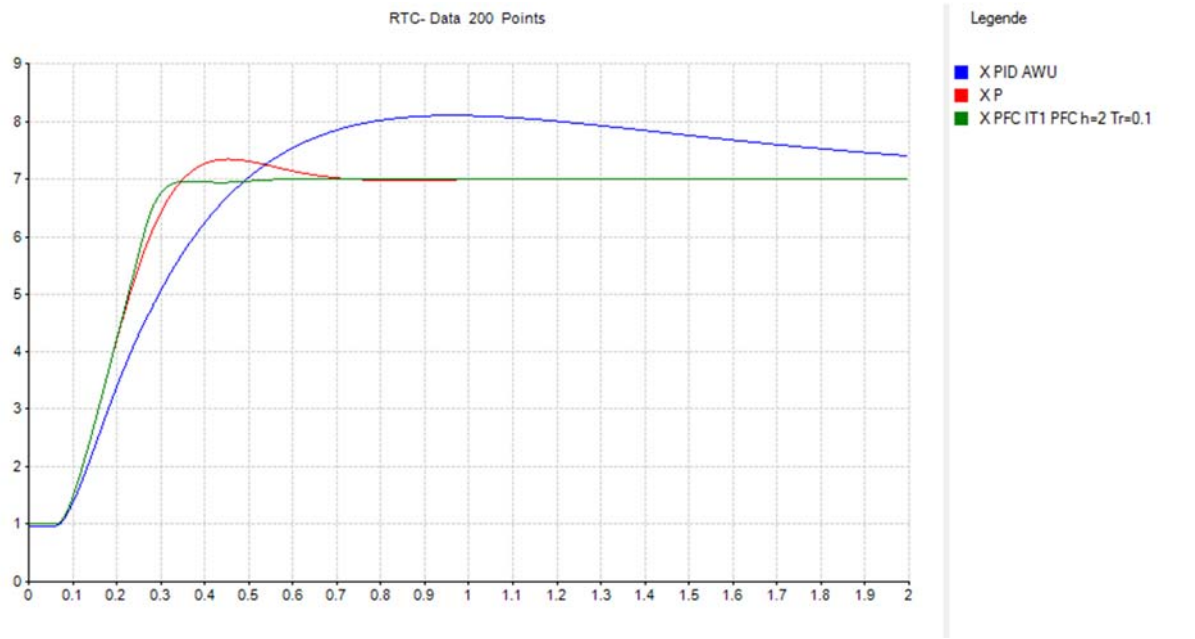
Blue: T_{dec}= 50 ms
Red: T_{dec}= 100 ms
Green: T_{dec}= 200 ms
Black: T_{dec}= 400 ms



Favorite:



Now with d=5 (additional delay of 50 ms), T_{dec}=0.1



9.9 Literature of Chapter 9

[9.1] *Orientierungsregler Ehrlich TU Chemnitz1.pdf*
[9.2] *Predictive Functional Control - Principles and Industrial Applications - Richalet, O'Donovan*.
[9.3] atp April 2011– Prof. Haber FH Köln, *„Predictive Functional Control: Algorithmus und Testbetrieb"*.

Prof. Dr. Bayerlein Donnerstag, 14. März 2019

9.10 Theory of DB –Bay- controller degree of 4

Now the solution for a process with a degree of 4 should be described. With same ideas as above we get following equations. A 4rd order process need at least four steps, so x(k+4) should reach the desired value w and all further x(k+x) too.

$$H_0F(z)=\frac{b_1z^{-1}+b_2z^{-2}+b_3z^{-3}+b_4z^{-4}}{1+a_1z^{-1}+a_2z^{-2}+a_3z^{-3}+a_4z^{-4}}=\frac{X(z)}{Y(z)}=\frac{output}{input}$$

$$Equ(1):\ \boxed{x_m(k+1)}=-a_1x(k)-\ldots-a_4x(k-3)+b_1\boxed{y(k)}+\ldots+b_4y(k-3)$$

$$Equ(2):\ \boxed{x_m(k+2)}=-a_1\boxed{x_m(k+1)}-\ldots-a_4x(k-2)+b_1\boxed{y(k+1)}\ldots+b_4y(k-2)$$

$$Equ(3):\ \boxed{x_m(k+3)}=-a_1\boxed{x_m(k+2)}-\ldots-a_4x(k-1)+b_1\boxed{y(k+2)}+\ldots+b_4y(k-1)$$

$$Equ(4):\ x_m(k+4)=-a_1\boxed{x_m(k+3)}-\ldots-a_4x_m(k)+b_1\boxed{y(k+3)}+\ldots+\boxed{b_4y(k)}=w$$

$$Equ(5):\ x_m(k+5)=-a_1w-a_2\boxed{x_m(k+3)}-\ldots-a_4\boxed{x_m(k+1)}+b_1y_{DC}+\boxed{b_2y(k+3)}+\ldots+b_4\boxed{y(k+1)}=w$$

$$Equ(6):\ x_m(k+6)=-a_1w-a_2w-a_3\boxed{x_m(k+3)}-a_4\boxed{x_m(k+2)}+b_1y_{DC}+b_2y_{DC}+b_3\boxed{y(k+3)}+b_4\boxed{y(k+2)}=w$$

$$Equ(7):\ x_m(k+7)=-a_1w-a_2w-a_3w-a_4\boxed{x_m(k+3)}+b_1y_{DC}+b_2y_{DC}+b_3y_{DC}+b_4\boxed{y(k+3)}=w$$

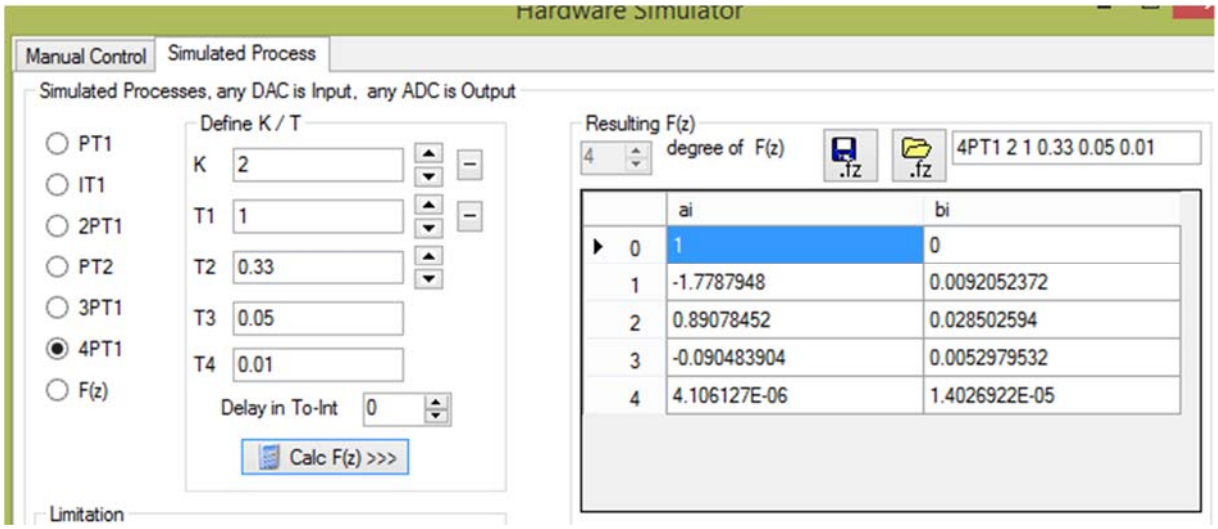
Starting with k+4 the y should be constant and is again the y(∞) = y_{DC}. All unknowns are indicated with red boxes. We have 7 unknowns with 7 equations, so solvable.
The solution is easy described in matrix- form. Sorted to unknowns left and knowns right:

$$\begin{pmatrix}-a_1x(k)-\ldots-a_4x(k-3)+b_2y(k-1)+\ldots+b_4y(k-3)\\-a_2x(k)-\ldots-a_4x(k-2)+b_3y(k-1)+b_4y(k-2)\\-a_3x(k)-a_4x(k-1)+b_4y(k-1)\\-w-a_4x(k)\\-w-a_1w+b_1y_{DC}\\-w-a_1w-a_2w+b_1y_{DC}+b_2y_{DC}\\-w-a_1w-a_2w-a_3w+b_1y_{DC}+b_2y_{DC}+b_3y_{DC}\end{pmatrix}=\begin{pmatrix}-b_1&0&0&0&1&0&0\\-b_2&-b_1&0&0&a_1&1&0\\-b_3&-b_2&-b_1&0&a_2&a_1&1\\-b_4&-b_3&-b_2&-b_1&a_3&a_2&a_1\\0&-b_4&-b_3&-b_2&a_4&a_3&a_2\\0&0&-b_4&-b_3&0&a_4&a_3\\0&0&0&-b_4&0&0&a_4\end{pmatrix}\begin{pmatrix}y(k)\\y(k+1)\\y(k+2)\\y(k+3)\\x_m(k+1)\\x_m(k+2)\\x_m(k+3)\end{pmatrix}$$

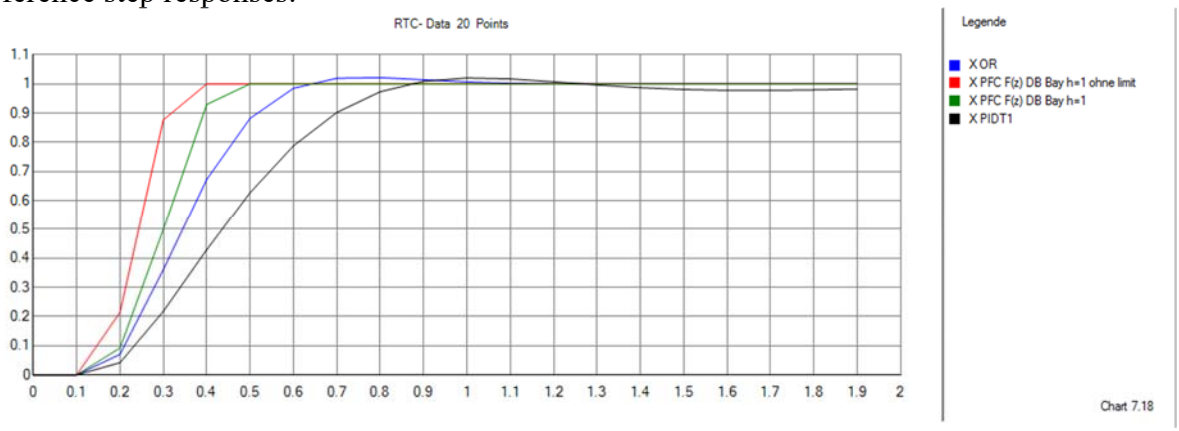
Z=A * Y
With the following solution:
$Y=A^{-1}*Z$
The controller output y(k) can then be calculated. A matrix inversion is necessary, but no problem, because A contains only a and b coefficients. So the inversion can be done one time before start of controlling. If online adaptive method is used, inversion is necessary in each To- step.
Delays can be added with same idea than before.
If process has a delay d with Tdel=d*To, again replace x(k) with xm (k) +x(k) – xm(k-d) as described before. Undelayed model output xm(k) can be calculated with

$$x_m(k)=-a_1x(k-1)-\ldots-a_4x(k-4)+b_1y(k-1)+\ldots+b_4y(k-4)$$

Results with 4- degree 4PT1- system K=2, T1=1, T2=0.33, T3=0.05 and T4=0.01, To=100ms

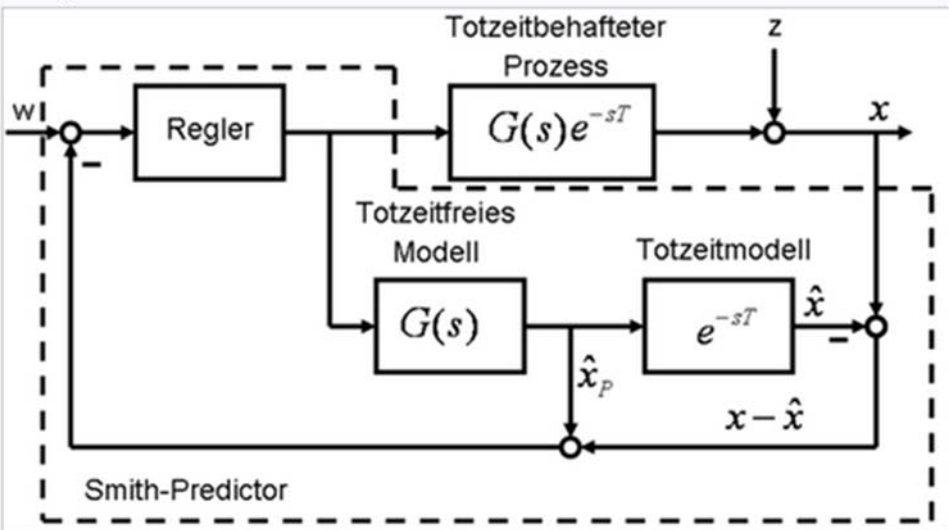


Reference step responses:



9.11 Smith predictor

Wikipedia:



In 9.7.2. is the replacement of x(k) with x(k) + x_m(k) –x_m(k-d) used. This is known as Smith-Predictor:

$$y(n) = \frac{w + a_1 x(n)}{b_1} \text{ without and } y(k) = \frac{w + a_1\left(x_m(k) + x(k) - x_m(k-d)\right)}{b_1} \text{ with delay d.}$$

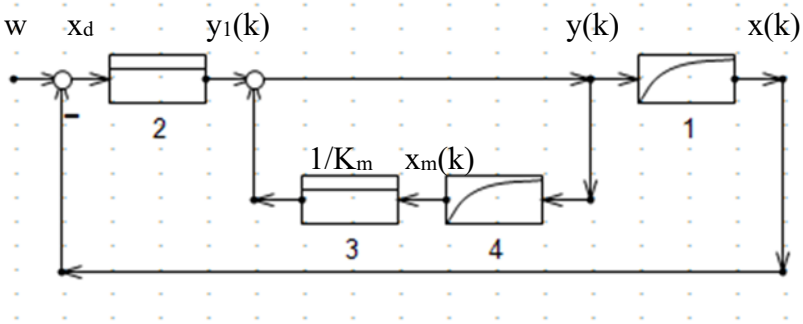
The signal block diagram of a Wikipedia- picture shows this as picture.

9.12 PFC behaves like a PI

The PFC equation from chapter 9.7.5 was this:

$$y(k) = A_0(w - x(k)) + A_1 x_m(k) \quad with \quad A_0 = \frac{(1-\lambda_v^h)}{K_m(1-(-a_1)^h)} \quad and \quad A_1 = \frac{1}{K_m}$$

We can draw this as signal block diagram. Block 1 is the real PT1 - process, block 2 is the gain A_0, block 4 the PT1 model and block 3 the gain A_1. The inputs of block 1 and 4 is the calculated PFC- controller output y(k).



Now model equation block 4:

$$H_0F(z) = \frac{x_m(z)}{y(z)} = \frac{b_1 z^{-1}}{1 + a_1 z^{-1}}$$ with $a_1 = -\exp(-T_o/T)$ and $b_1 = K_m(1+a_1)$.

Model equation:

$x_m(k) = b_1 * y(k-1) - a_1 * x_m(k-1)$;

Summing point: $y(k) = y_1(k) + 1/K_m * x_m(k)$

Summing point one step before: $y(k-1) = y_1(k-1) + 1/K_m * x_m(k-1)$

Solved to $x_m(k-1) = K_m * (y(k-1) - y_1(k-1))$

Now again summing point with these replacements:

$y(k) = y_1(k) + 1/K_m * (K_m(1+a_1) * y(k-1) - a_1 * K_m * (y(k-1) - y_1(k-1)))$

results in

$y(k) = y(k-1) + A_o * x_d(k) + A_o * a_1 * x_d(k-1)$.

Compare this with PI – equation $y(k) = y(k-1) + q_0 * xd(k) + q_1 * x_d(k-1)$ this is identical. With coefficient comparison we can extract the controller PI gain and time constant. With the PI coefficients $q_0 = K_c$ and $q_1 = K_c(-1 + T_o/T_N)$ we get

$K_c = A_o$ and $T_N/T_o = 1/(1-\exp(-T_o/T))$.

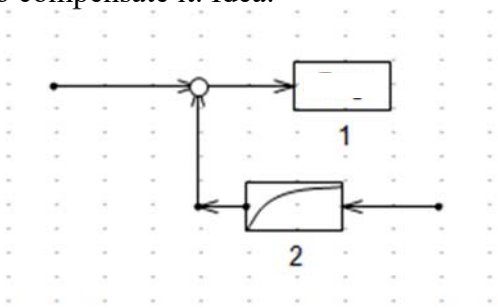
In a table I have calculated different T_N – values which depends on the process time constant:

T/T_o	T_N/T_o
1	1.58
2	2.54
5	5.52
10	10.5
20	20.5
50	50.5
100	100.5

This is very near to polecompenation. Advantage of PFC: ideal AWU- function.

9.13 Disturbance forward compensation in DB Bay case

Caused by bad disturbance behaviour of „my“ DB bay- control algorithm I tried to use the disturbance measurement method of 9.8.1 to measure the disturbance and then find a method to compensate it. Idea:

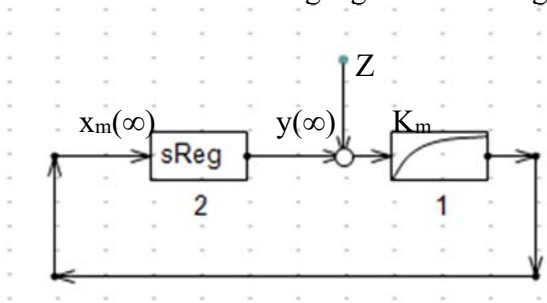


Block1 is the existing loop with standard controllers, block 2 is a disturbance filter with gain=-disturbance gain and a selectable time constant in steps of the sampling time To. The input of block 2 gets the measured disturbance. The disturbance gain has first to be calculated. Because “my “ algorithm has no F(z) function we have to find an other way to calculate a DC- gain: take the y(k)- calculation equation and set w=0: In a PT1- system this is:

$$y(n) = \frac{w + a_1 x(n)}{b_1}.$$

With w=0 we get for constant values:

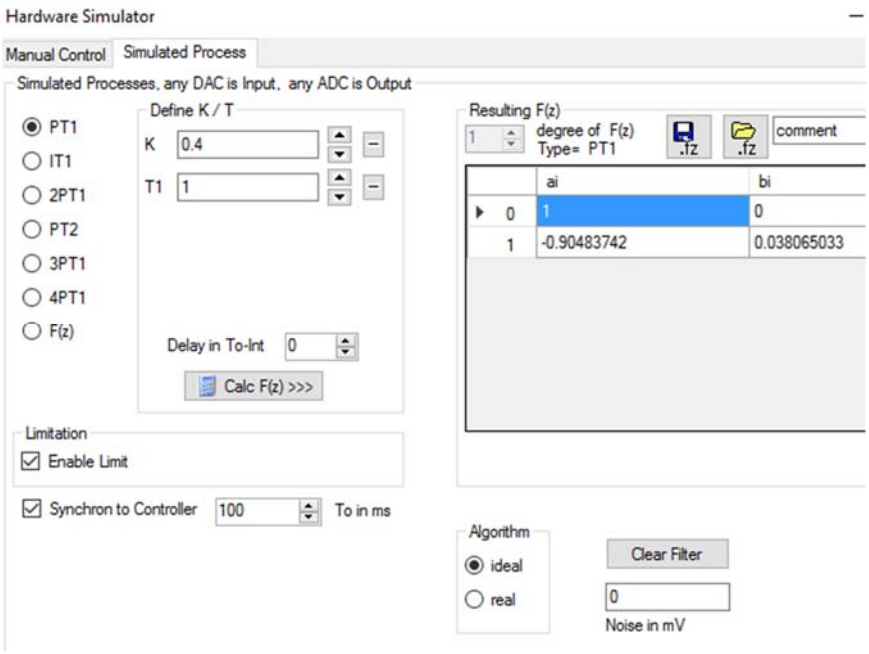
$y(\infty) = \frac{a_1 x(\infty)}{b_1}$ or the DC-gain a_1/b_1 of block 2. The disturbance gain Z/x_m now can be calculated with following signal block diagram:



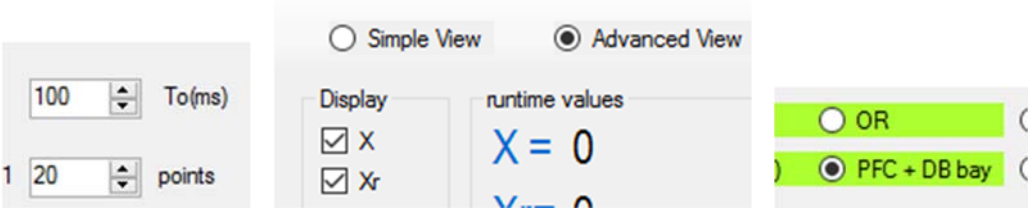
$$G_z = \frac{K}{1 - K * \frac{a_1}{b_1}} = \ldots = b_1$$

Example: with K_m=0.4, T_m=1 and To=0.1 we get a_1=-0.9048, b_1=0.03807 and disturbance Gain G_z = 0.03807.

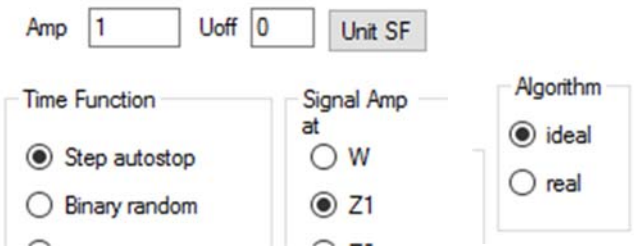
Screenshots of simulation WINDFC# version 7.6.51 or higher:



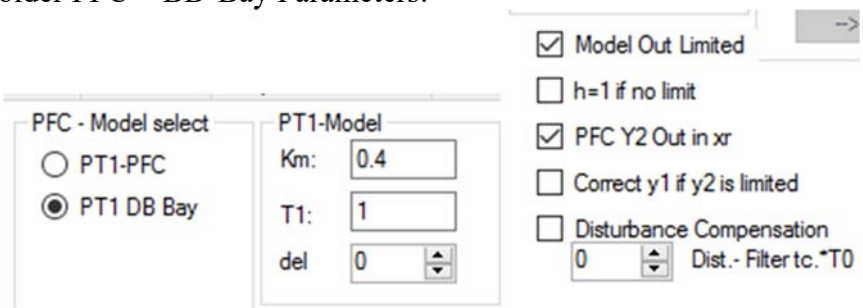
In Advanced Adaptive Control window following settings:



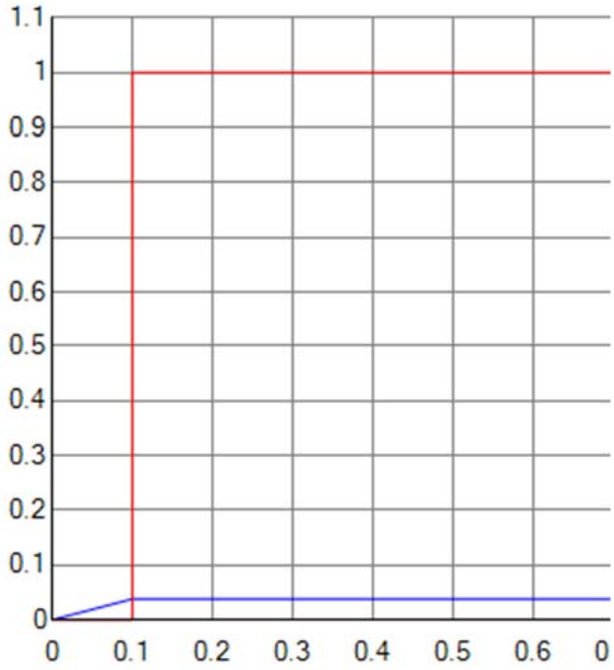
Folder Parameters:



Folder PFC + BD-Bay Parameters:

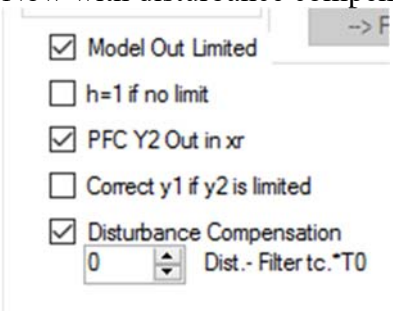


Thenn you start RUN and RSR-button and see this:

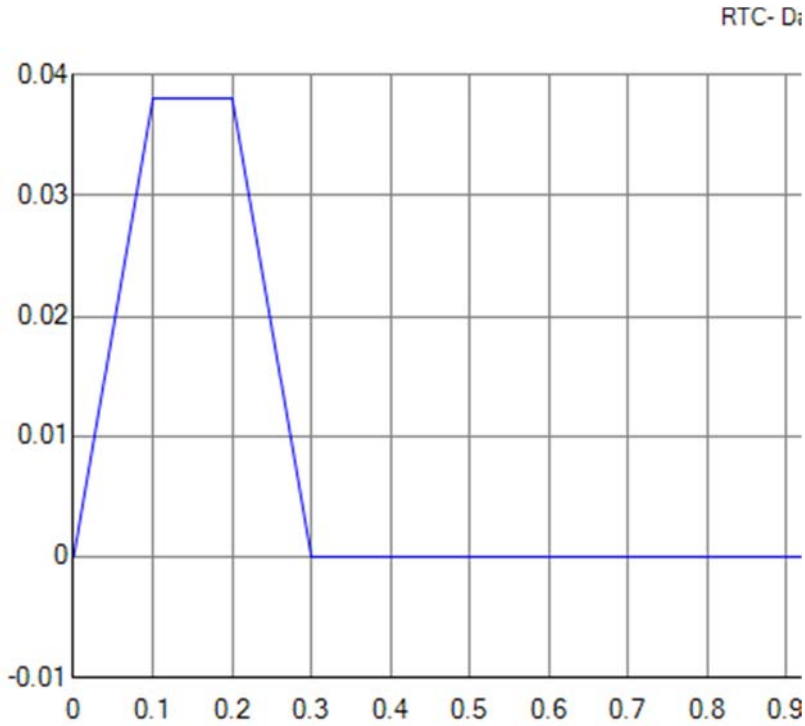


Blue curve is the process output with the final value in one step = 0.03807 the result of disturbance = 1 multiplied with disturbance gain G_z.
Red curve is the measured disturbance, of course with a delay of one step.

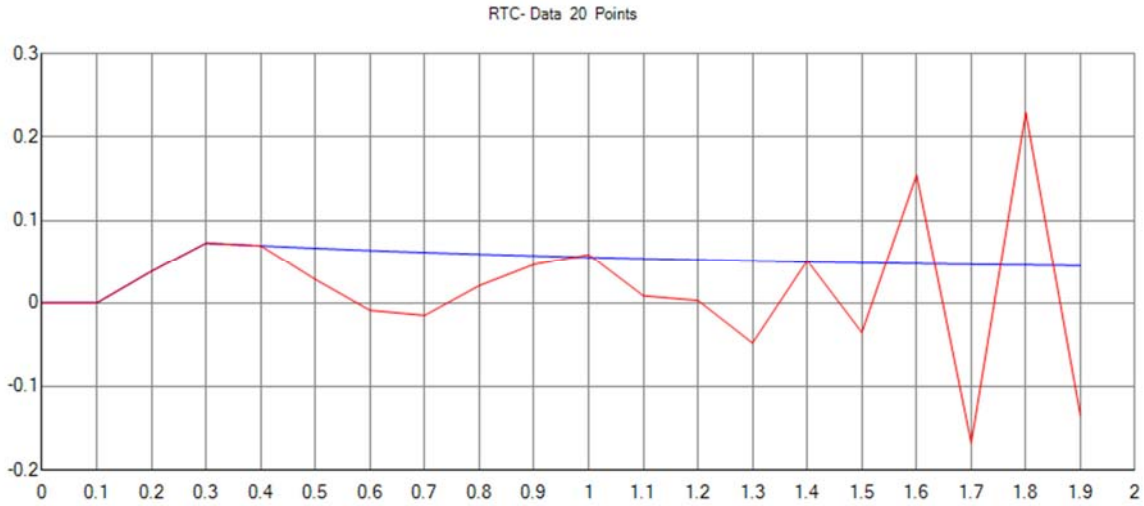
Now with disturbance compensation:



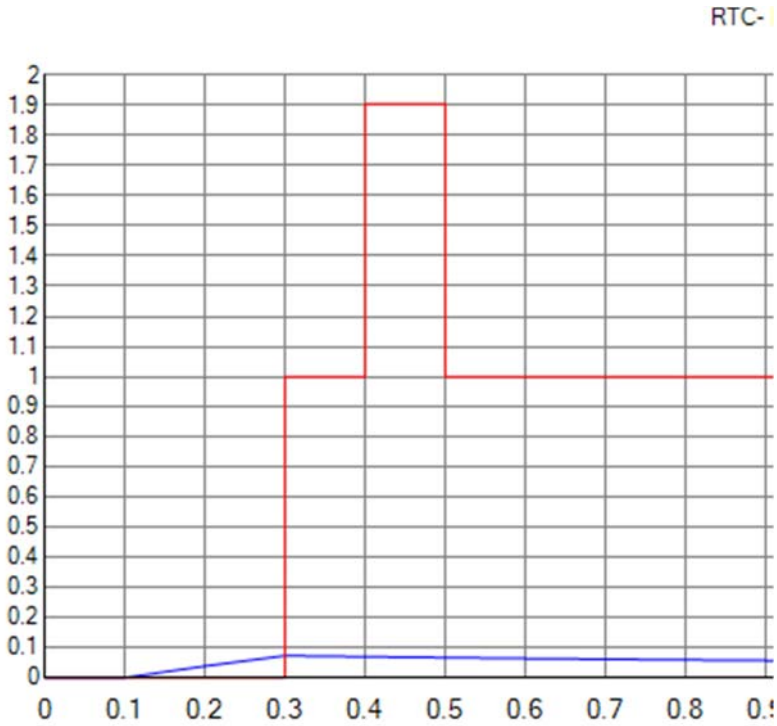
Only repetition of the blue curve:



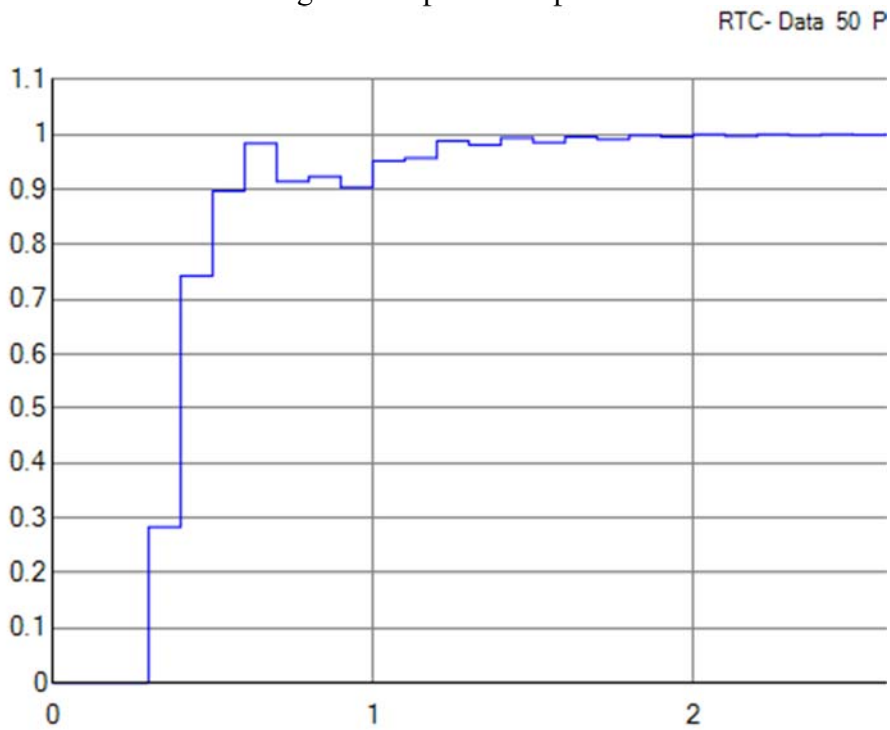
This is really impressive, but if there is a delay d in the process or we use real algorithm this doesn't work. See results only with change into real mode (both Hardware simulator and controller!): Blue curve without compensation, red with compensation, you see loop becomes unstable:



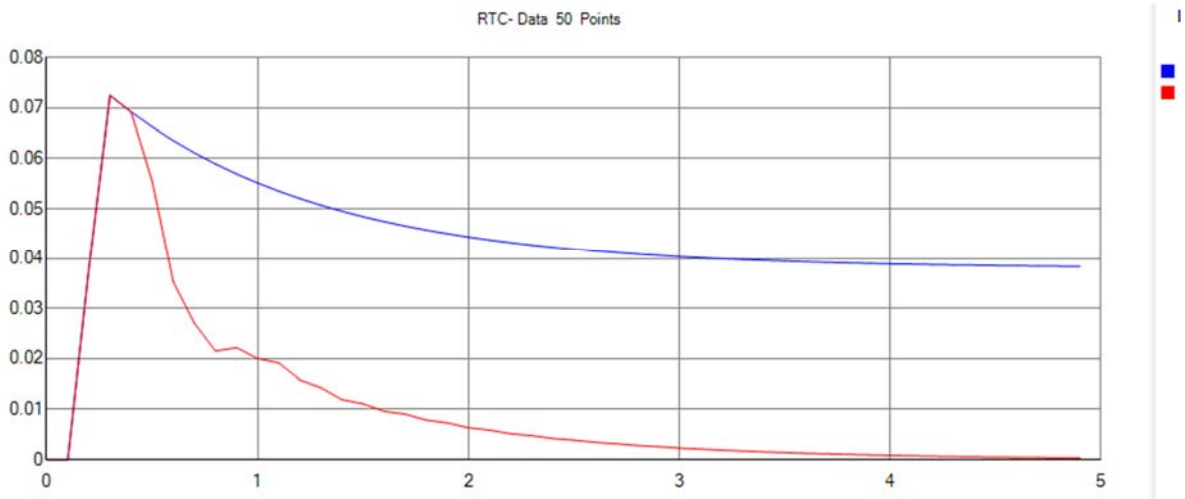
Now the measurement of disturbance fails: See calculated Z- values:



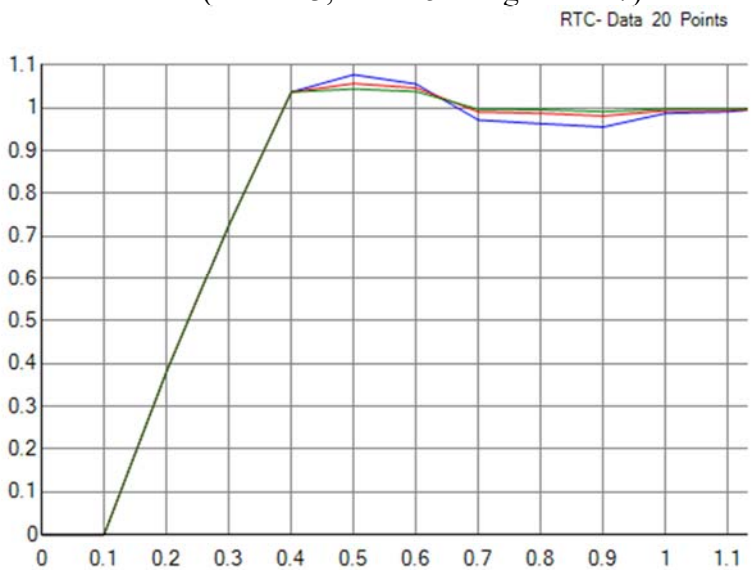
So after 5 steps z =1 is OK, but from 0.4 to 0.5 the signal is too large. So I try a PT1- Filter with n=3 (time constant 3*To=300 ms) to remove this "overshoot' to stabilize the loop, first the filtered disturbance during the compensation process:

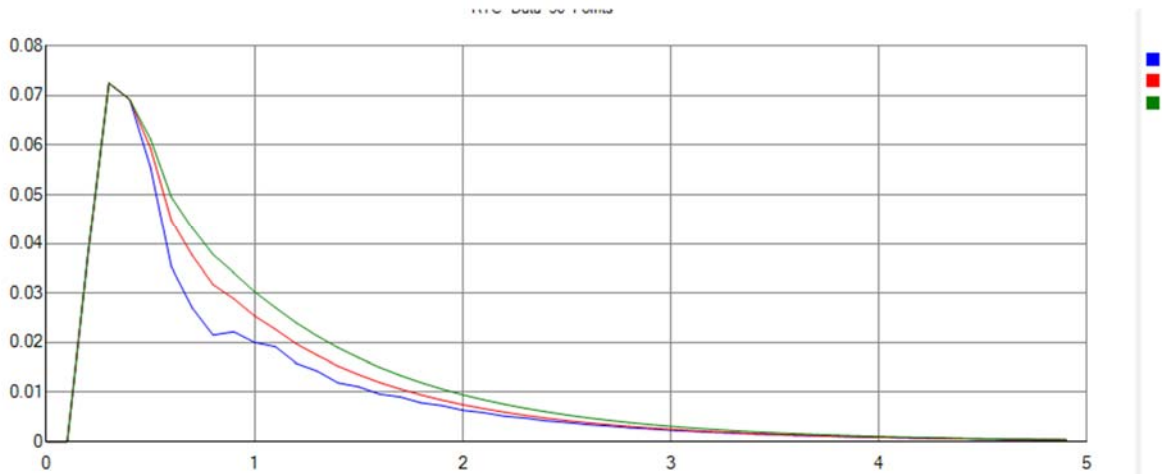


Then finally the disturbance step response, red previous original DB- Bay with control error, now blue: compensated to zero!!!:



But unfortunately the reference step response is also influenced and worse than previous: Without compensation ideally blue curve, with disturbance compensation in red:
So inspite of no disturbance because of the delay this method measures a fictive disturbance and system tries to compensate it. It is a pity. So increasing of filtered time constant value improves the RSR and makes disturbance behavior worse. See variation of n = 3,5, 7 in RSR case and DSR case (blue n=3, red n=5 and green n=7):





Same case now with second order system (2PT1 or IT1- process):
The calculation of the disturbance gain is much more difficult. The equations to calculate the y(k) were: (see 9.7.3)

$$Equ(7):\ z_1 = -a_1 x(k) - a_2 x(k-1) + b_2 y(k-1)$$

$$Equ(8):\ z_2 = -a_2 x(k)$$

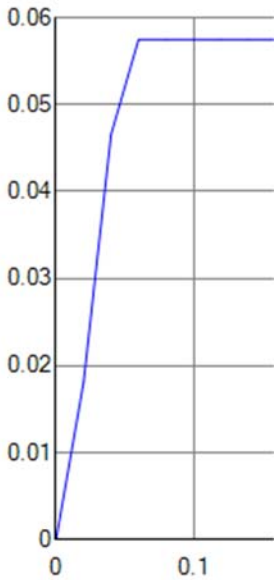
$$Equ(9):\ x_m(k+1) = \frac{b_1\left[b_2\left(w - z_2 + b_2 z_1/b_1\right) - b_1\left(w + a_1 w - b_1 y(DC)\right)\right]}{b_2^2 - a_1 b_1 b_2 + a_2 b_1^2}$$

$$Equ(10):\ y(k) = \left(x_m(k+1) - z_1\right)/b_1$$

The steady state condition in disturbances case is w=0, y(DC) =w/DCgain = 0 and old y values =y(k)= y(∞) and old x- values = x(k)=x(∞). I have used Equ(7) to (9) and solved these to

$$\frac{y(\infty)}{x(\infty)} = \frac{b_1 b_2 a_2 + (a_1 + a_2)(-a_1 b_1 b_2 + a_2 b_1^2)}{(b_1 + b_2)(b_2^2 - a_1 b_1 b_2 + a_2 b_1^2) - b_2^3} = A \text{ and } G_z = \frac{K}{1 - K * A}.$$

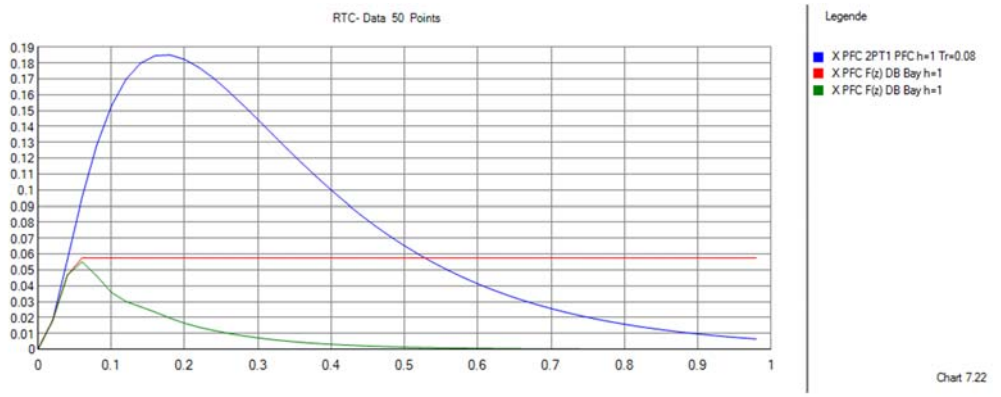
Example: in 2PT1 with K=2. T_1=0.1, T_2=0.2 and To=20ms I have got an A=-16.8856, this gives a disturbance gain of G_z=0.057519. In simulator this is proven, see screenshot of disturbance step response of this system with th DB- Bay version:



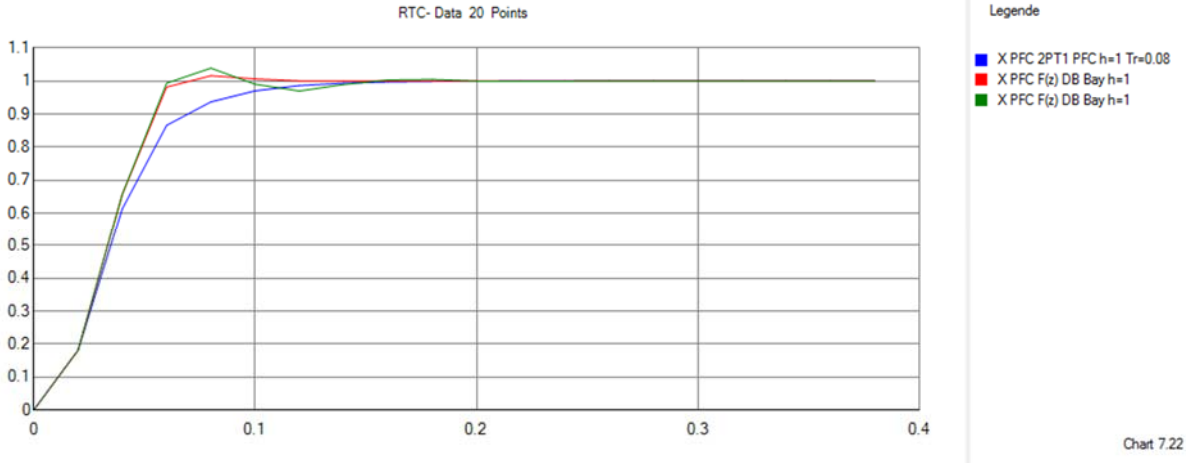
	0X PFC F(z) DB Bay h=1: X	0X PFC F(z) DB Bay h=1: Y
1	0	0
2	0.02	0.0181118
3	0.04	0.0465866
4	0.06	0.0575229
5	0.08	0.0575229
6	0.1	0.057523
7	0.12	0.057523
8	0.14	0.057523
9	0.16	0.057523
10	0.18	0.057523
11	0.2	0.057523
12	0.22	0.057523
13	0.24	0.057523

Four significant digits are equal!.

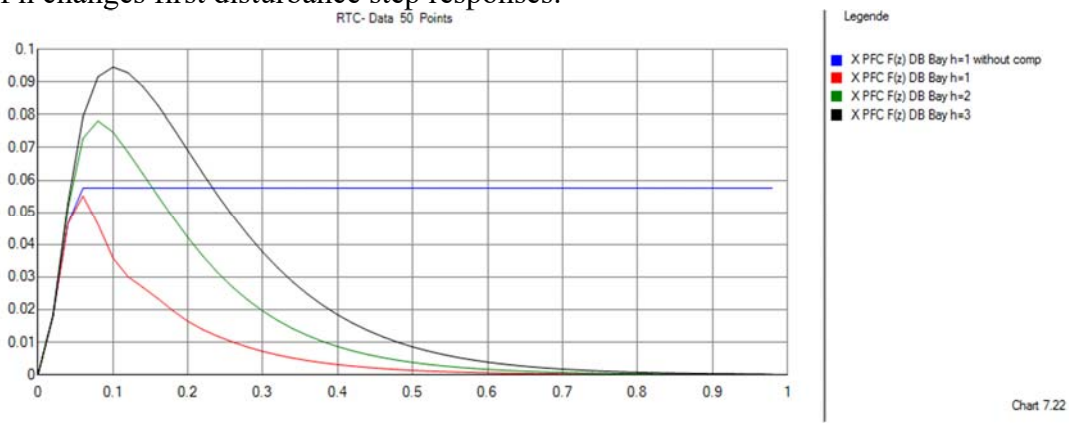
Now with compensation mechanism (green) and comparison with unchanged DB Bay (red) and PFC blue with the disturbance filter n=6:



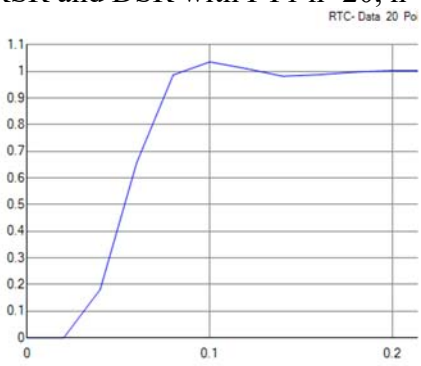
And finally the reference step responses

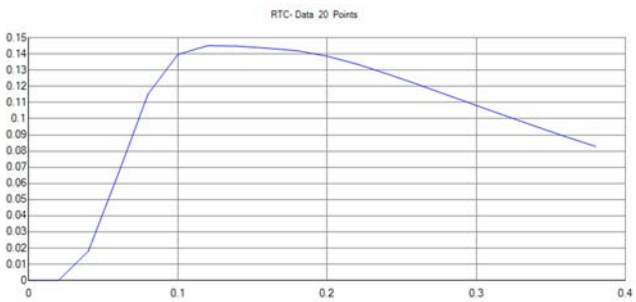


So with the PT1 for the disturbance and h find a good compromises:
If h changes first disturbance step responses:

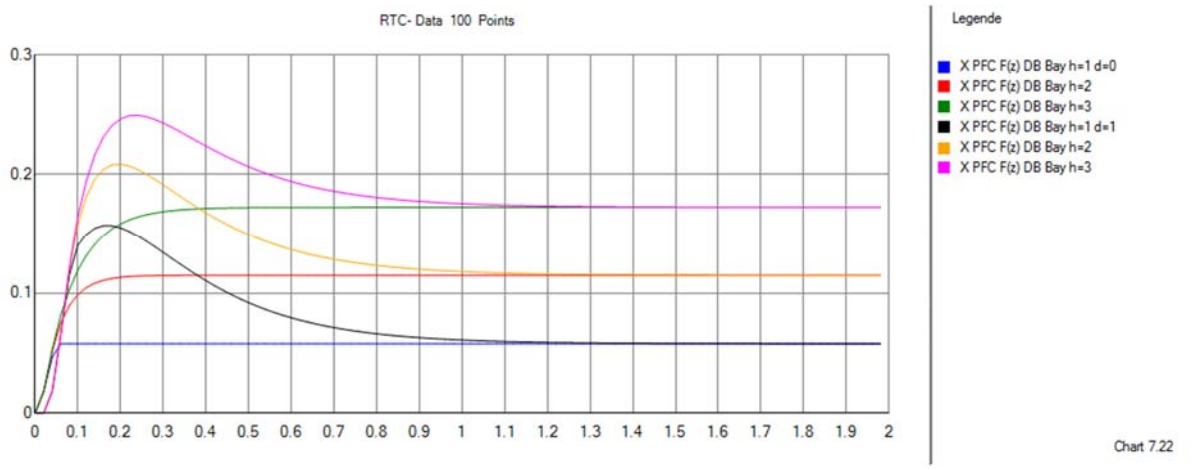


RSR and DSR with PT1 n=20, h=1:

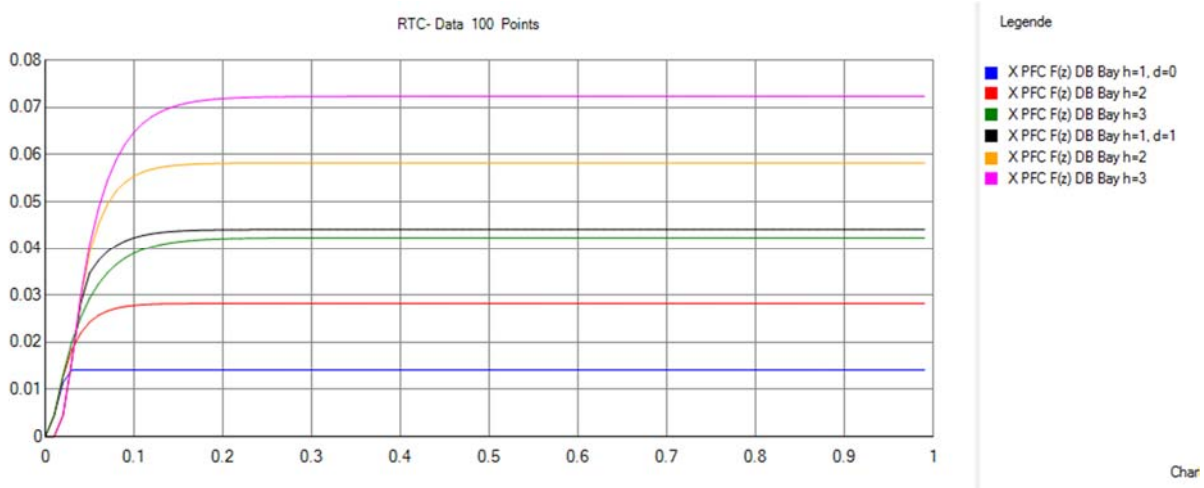




Now dependencies of final value in disturbance case from h and d, first 2PT1 process:



So final value in 2PT1 is independent of d (delay).
This is different in IT1 case:



If you compare the final values, it is evident that value is larger with parameter $+K_iTo$. To compensate this it is necessary to recalculate this value with given a_i and b_i. This works with the IT1- equations from chap. 9.2:

$$a_1 = -1 - \exp(-T_0/T), \quad a_2 = \exp(-T_0/T), \quad b_1 = KT(T_0/T - 1 + \exp(-T_0/T))$$

$$b_2 = -KT(T_0/T\exp(-T_0/T) - 1 + \exp(-T_0/T)).$$

Here $K=K_i$. if you add b_1 and b_2 you get:

$$b_1 + b_2 = KT(T_0/T - T_0/T\exp(-T_0/T));$$

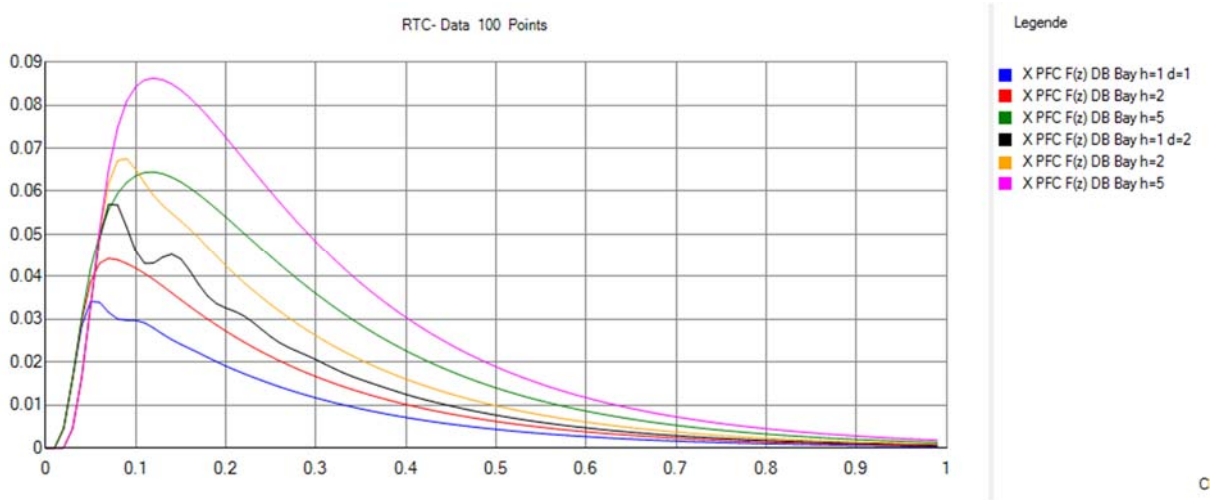
With a_2 we get

$$KT_0 = \frac{b_1 + b_2}{1 - a_2}$$ and can be used to correct this.

The correction expression if $K_m > 1e5$ (IT1)

$$G_z = \frac{K_m}{1 - K_m * A} - d/h * KT_0.$$ (d=delay and h is horizon)

Results with IT1 disturbance step responses with different d and h:



(Disturbance filter with n=20).