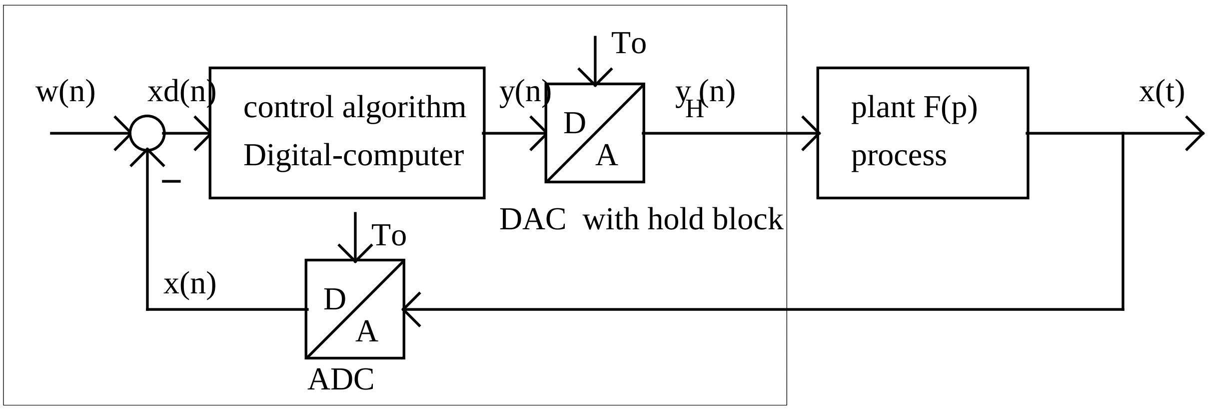
Content

6 Digital PID Controllers

6.1 Structure of System

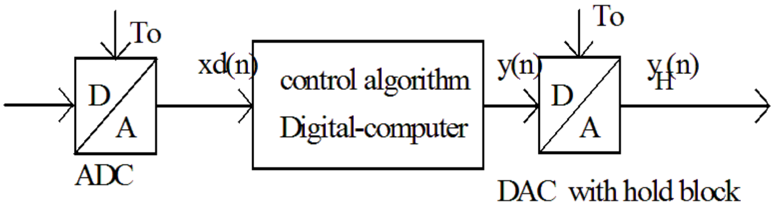
6.1.1 Overview

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

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

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).

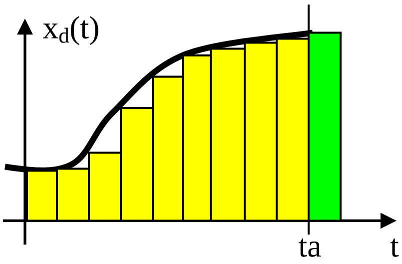
6.2.2 I- control algorithm

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 analog 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 can not 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.

$F(p) = \frac{K_c(1+pT_N)}{pT_N} = \frac{K_c}{pT_N} + K_c = \frac{Y(p)}{X_d(p)}$. 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

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

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

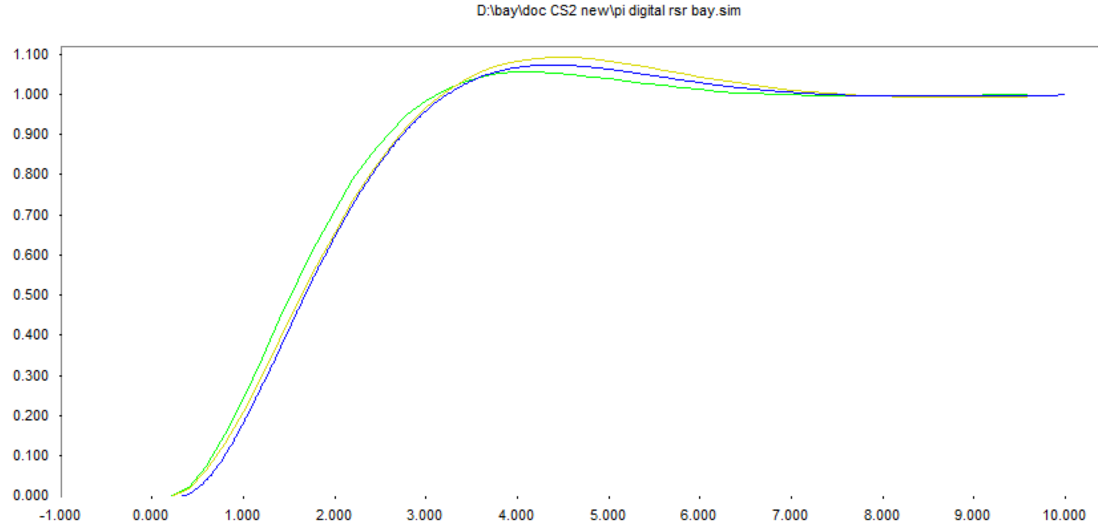
$y(n) = y(n-1) + q_0 x_d(n) + q_1 x_d(n-1)$ with

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

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.

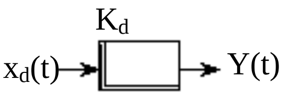
If you design a PI- controller with pole compensation and 60° phase margin, this results in exactly the previously used PI- controller with K_c=2 and T_N=1s. The following picture gives idea of the resulting unit reference step responses.



The blue curve is the analog 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

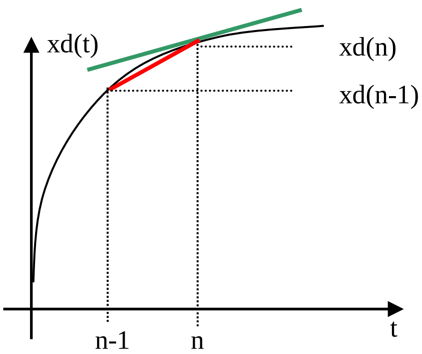
The next step is to include a differentiator. So, I will start by discussing a pure differentiator, after that the total PD, and finally the PID- algorithm.



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

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

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*

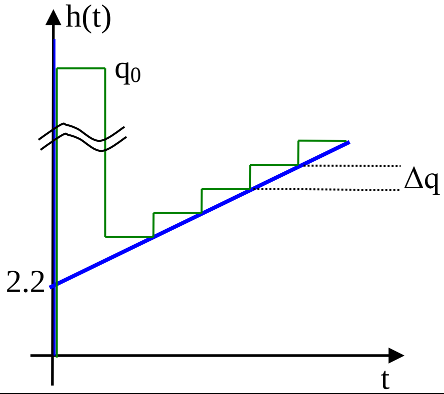
Task: Convert a PID with the parameters K_R=2, T_N=1, T_v=0.1 into a digital PID with sampling time T_0=0.01. Compare the unit step responses.
First convert given bode form parameters into summing form parameters with WB p 21:

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

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



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 $T_0 \le 0.1$ 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.

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

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

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.

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

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

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

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 $T_0 \ge 25.6$ 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

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

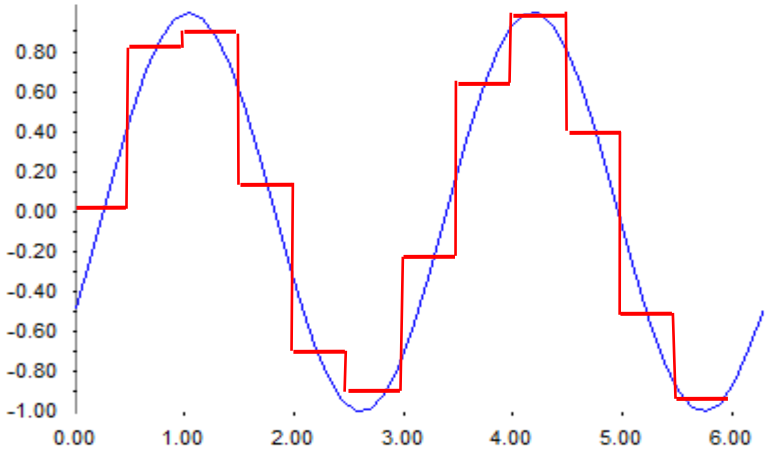
Next step is to develop a more realistic substitute of a digital PID for use in standard simulation Programs like Regdelph or for use in a standard simple FRA- design.
The substitution is simple and very possible and consists of analog blocks. I found the following analog replacement:

→ [Digital PID q_0, q_1, q_2,] → → [analog PIDT1, st] → [Sample & Hold] → [Calculation time] →

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 analog 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



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 $F_H(j\omega) = \dfrac{1-e^{-j\omega T_0}}{j\omega T_0}$

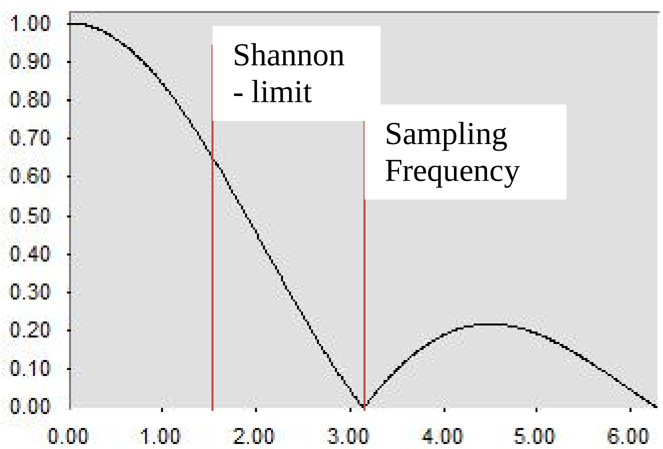
To separate the magnitude and the phase- function we use a trick. With Leonhard Euler the e-function e^{jx} can be replaced with cos x + j sin x.

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

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 zeros at $x_0=\pi$, 2* π, …
x in our Sample & Hold is

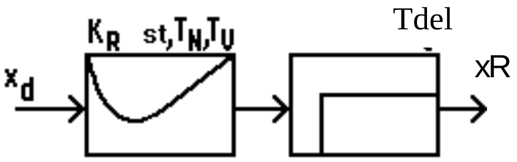
$x = \dfrac{\omega T_0}{2}$.

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

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;	**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 $\varphi_{del}= -\omega T_{del}$ *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 $\varphi_{PDT1}=\arctan(\omega T_v)-\arctan(\omega * T_v/st)$.
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 $q_0=K(1+\frac{T_0}{T_I}+\frac{T_D}{T_0}),\quad q_1=-K(1+2\frac{T_D}{T_0}),\quad q_2=K\frac{T_D}{T_0}$ and $K=K_R(T_N+T_V)/T_N,\ T_I=T_N+T_V$ and $T_D=T_N T_V/(T_N+T_V)$.

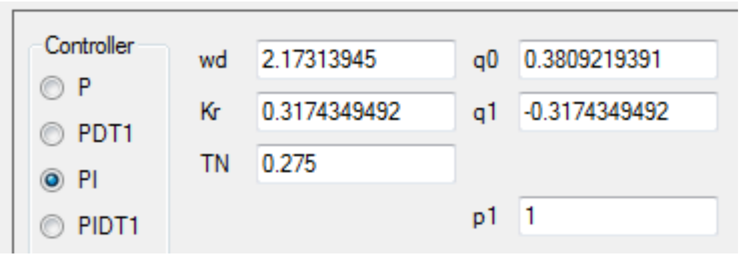
Now an exercise with a 2PT1- process:

6.7.5 Example

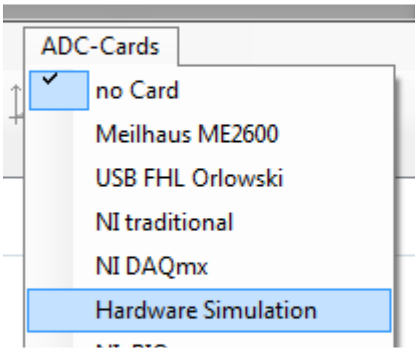
We will now design a digital controller with a sampling time of T_0=55 ms. The process transfer function reads
$F_S=K_S/(1+pT_1)(1+pT_2)$ with K_S=2, $T_1=5T_0$ =0.275 s and $T_2=3T_0$=0.165s.
It should be a real PIDT1 - controller, the phase margin being φ_R=60°.

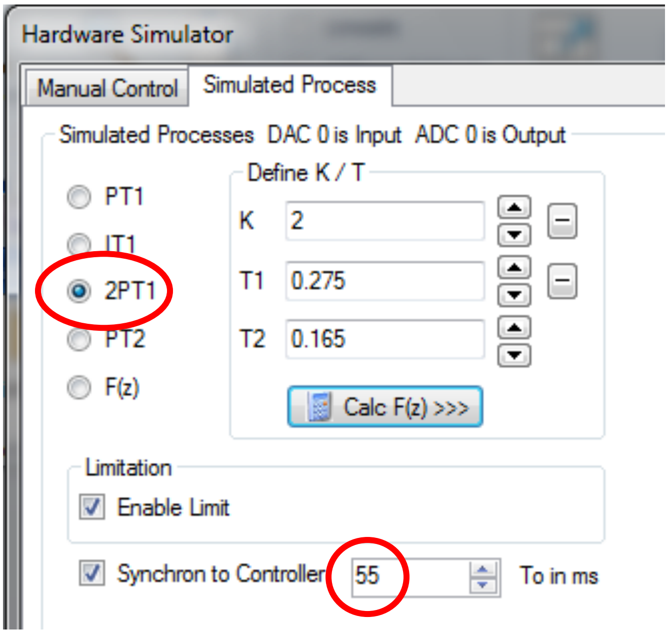
The method to convert a continuous PIDT1 into a discrete / digital in *Regdelph* works with the following steps:

PI design for this example:



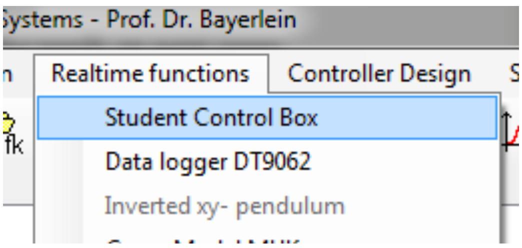
Now in WindfC# this PI and PIDT1 can be tested. The 2PT1- process can be simulated in the menu ADC-Cards :→ Hardware Simulation:

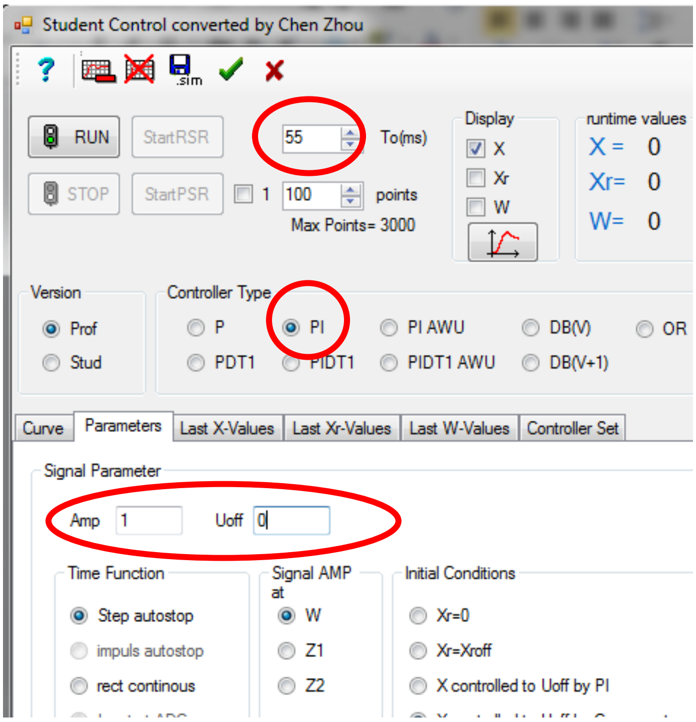




Select this process:

Now real time test in the menu Realtime functions → Student Control Box:





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^{-pTo}. 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

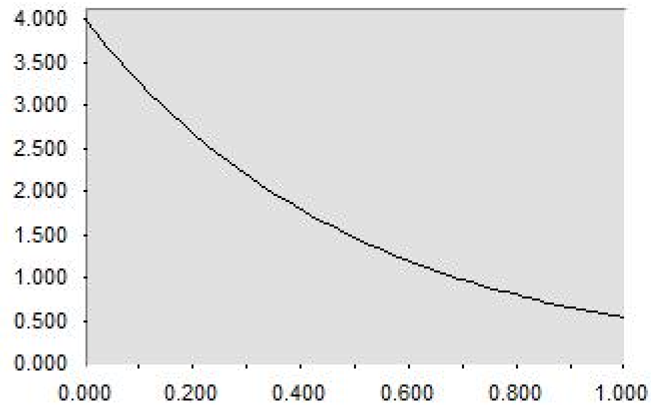
$$F(z) = \frac{2.8436 - 4.5021 z^{-1} + 1.7771 z^{-2}}{1 - z^{-1}}.$$

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

With the next four methods you can convert any continuous, analog 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:

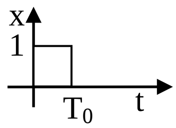


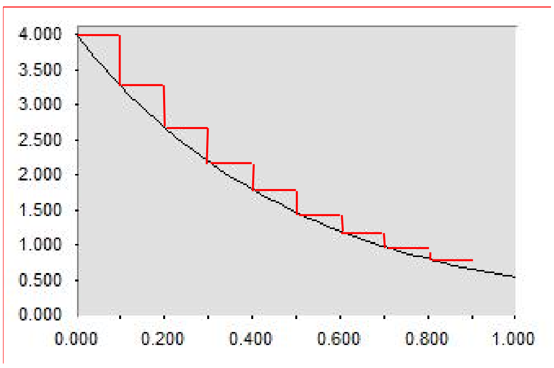
t	y(t)	q
0	4	q_0
0.1	3.27492301	q_1
0.2	2.68128018	q_2
0.3	2.19524654	q_3
0.4	1.79731586	q_4
0.5	1.47151776	q_5
0.6	1.20477685	q_6
…	…	…

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.

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

$F(z=1) = \frac{4}{1-0.8187307531} = 22.0666$, 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

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

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 can not 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 $h(t)=2(1-e^{-t/0.5})$:

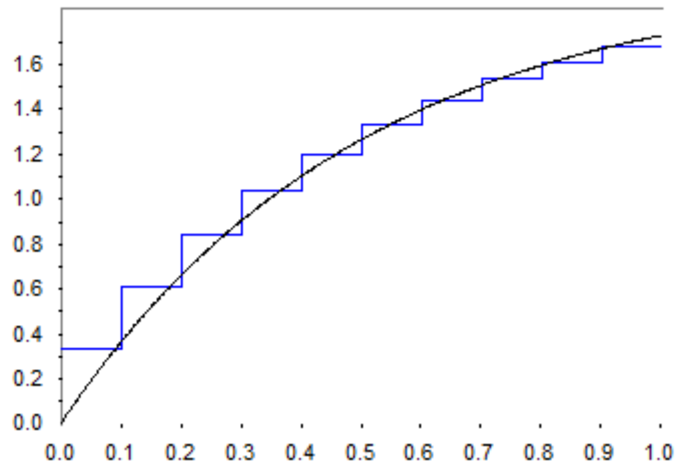
n	t	x(n)	y(n)	$h(t)=2(1-e^{-t/0.5})$
0	0	1	0	0
1	0.1	1	0.3625384938	0.3625384938
2	0.2	1	0.6593599078	0.6593599078
3	0.3	1	0.9023767277	0.9023767277
4	0.4	1	1.1013422072	1.1013422072
5	0.5	1	1.264241118	1.264241118
…	….	…	….	….
∞	∞	1	2	2

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

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

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”.

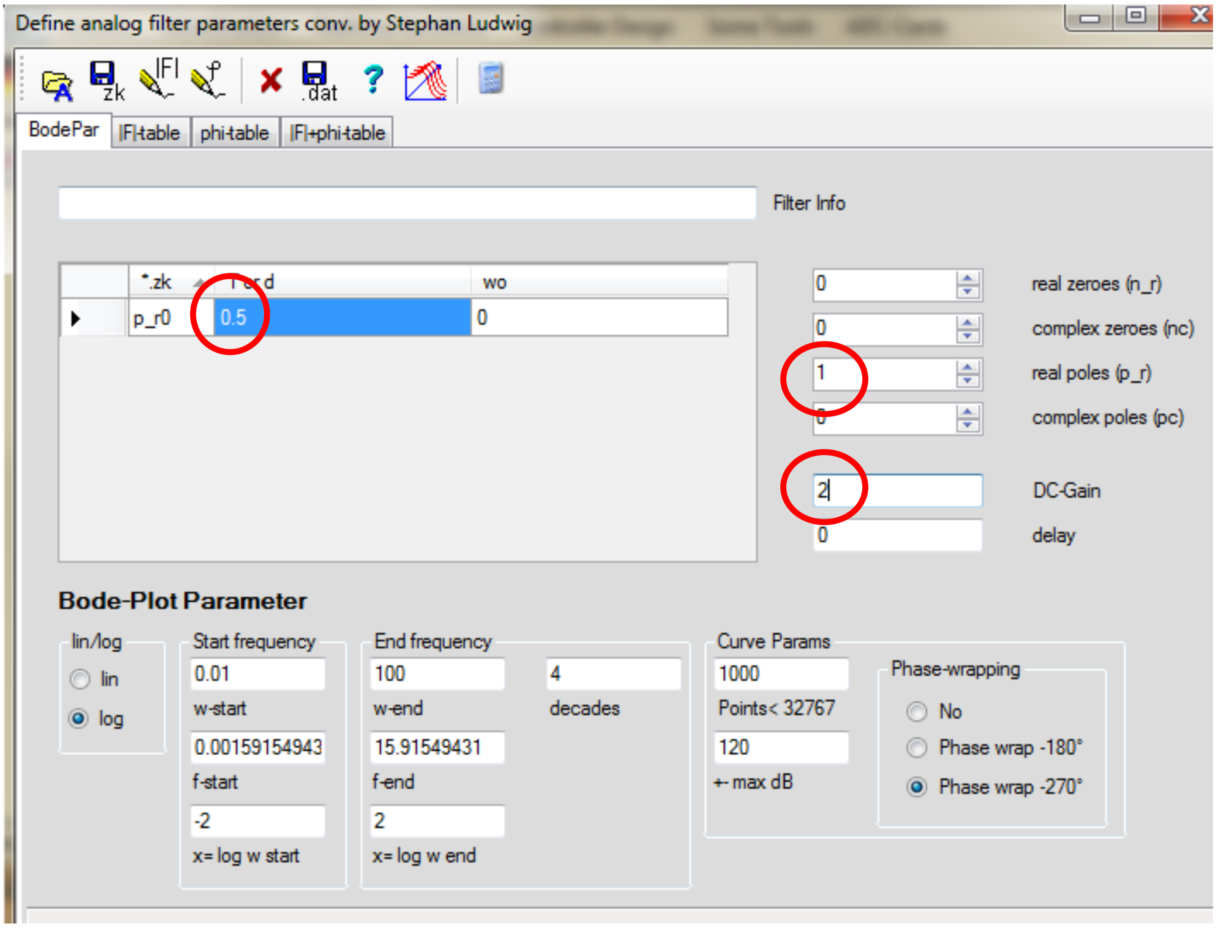
If we would do this replacement with the pure integrator, we get a result identical to a replacement of the area by a sum of trapezoids. This is the reason for the name. Now apply this to our PT1:

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

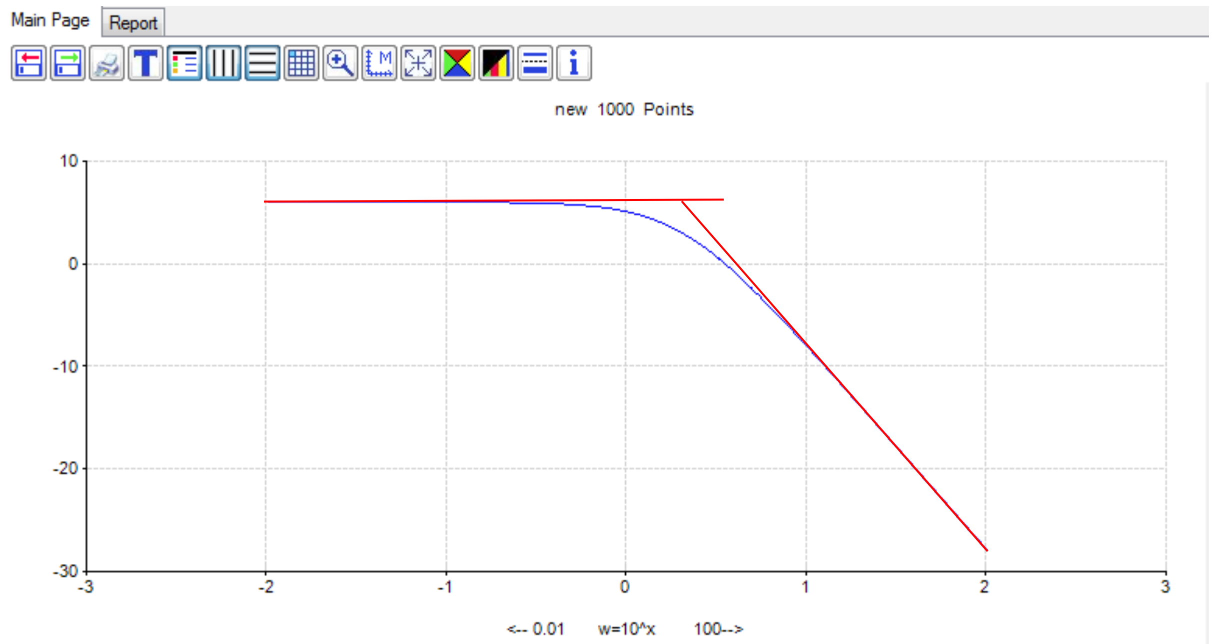
Corresponding algorithm:

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

n	x(n)	y(n)
0	1	0.181818181818182
1	1	0.512396694214876
2	1	0.782870022539444
3	1	1.004166382077730

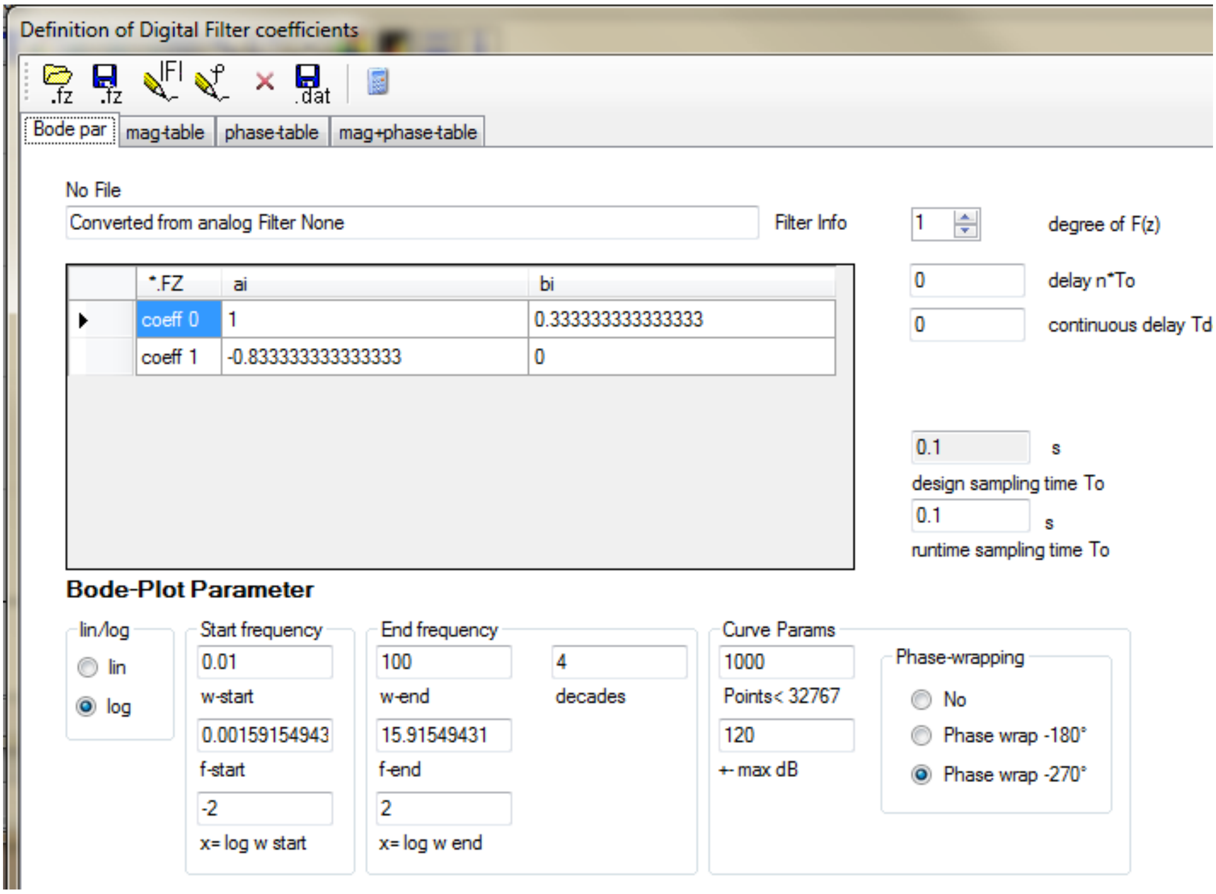


Now you can close this window with the **|F|** - button and the magnitude curve of the bode plot appears in the main window.



Note that horizontal axis carries the log value of ω. To get ω read the value x and calculate $\omega=10^x$. With MS-Word I have added the asymptotical red straight lines. The crossing defines the corner frequency at 0.3. $\omega_c = 10^{0.3} = 1.995 = 1 / T$. 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.



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

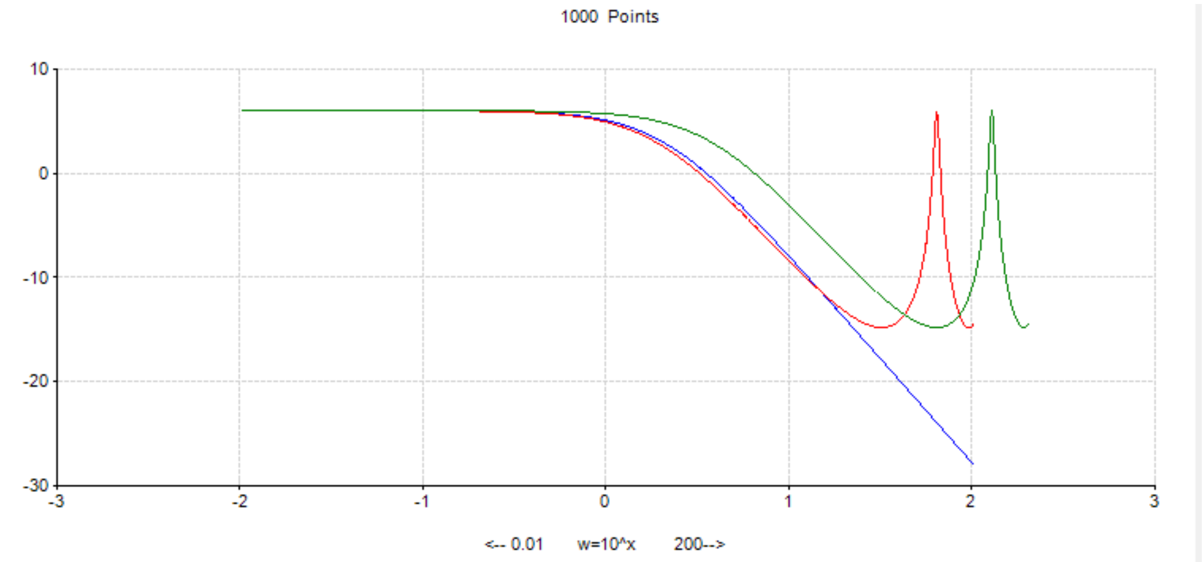
$$F(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_m z^{-m}}{1 + a_1 z^{-1} + \ldots + a_m z^{-m}} z^{-d}$$

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

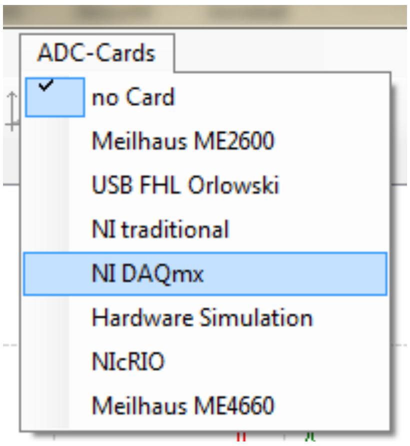
$$F(z) = \frac{q_0 + q_1 z^{-1} + \ldots + q_m z^{-m}}{1 - p_1 z^{-1} - \ldots - p_m z^{-m}} z^{-d}$$

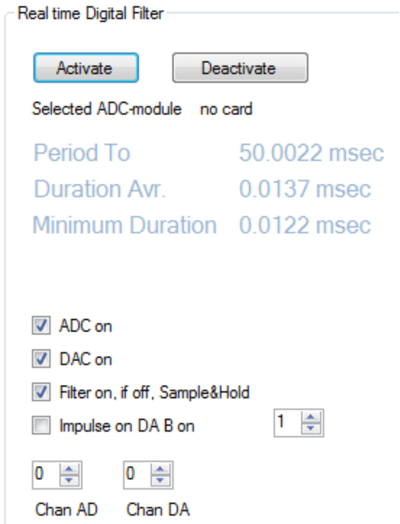
is more common for controllers. The difference is the sign of the coefficients in the denominator.

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



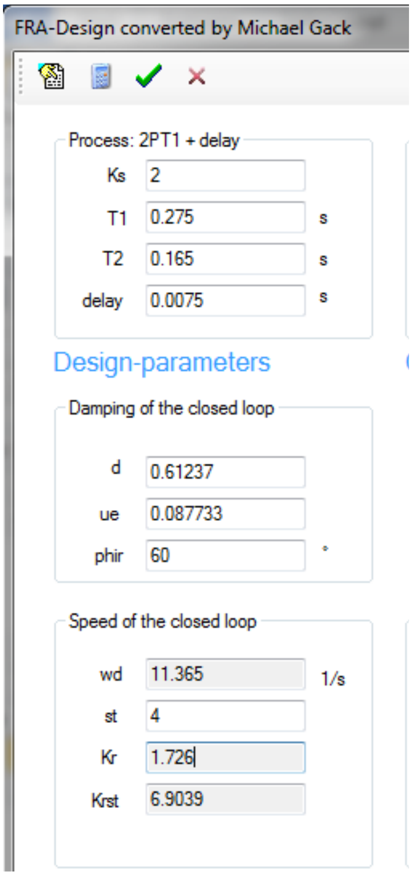
The green curve is the new magnitude curve. It is the red curve shifted one octave right.



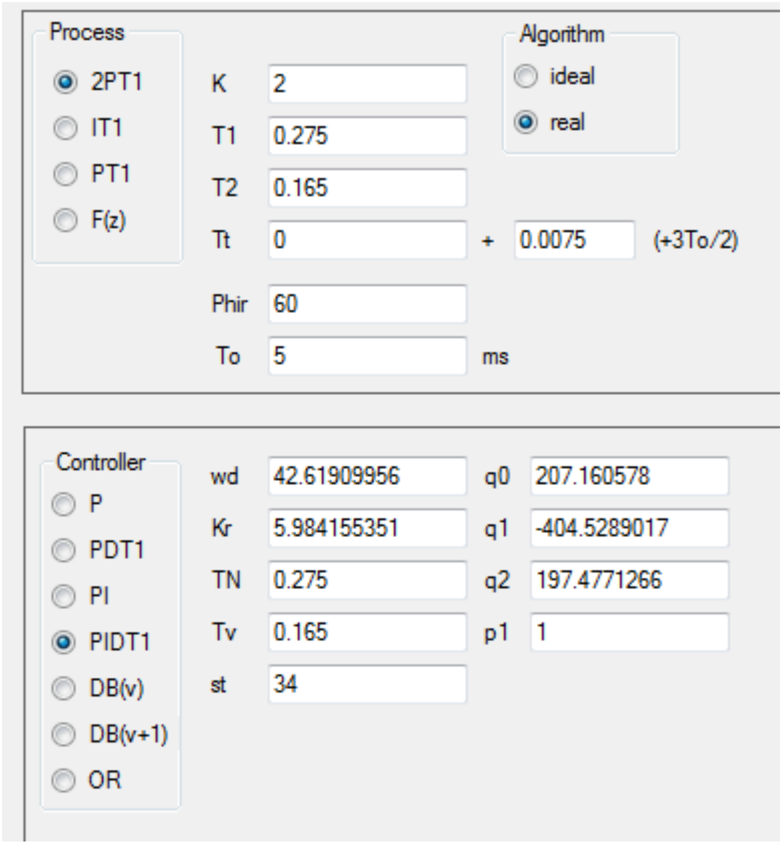


In the tool program WindfC# you can test this filter, if an ADC Card is available. First select your card in the main menu, then open the digital filter window again and start the running with click on the button “Activate”. Then connect signal generator with the selected AD input and measure the output 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.

Example: Design the analogous PIDT1 with K_c=1.726, T_N=0.275, T_V=0.165, st=4 and T_0=0.005s= 5 ms. These values are resulting from PIDT1- design with above 2PT1- process (K=2, T_1=0.275s, T_2=0.165s) of previous example calculated with simple FRA- design via *Regdelph*.(process + delay of 0.0075s =3/2T_0 , PID with polcomp. st=4, phase margin 60°). This gives K_c=1.726.



If you ignore the desired stepdepth 4 and use *design I* with 5 ms sampling time you get



An analog PIDT1 has a starting impulse at controller output after reference step of K_c*st = 6.9039.

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 6.904, but 207.16.

With the trapezoidal-*Design IV* 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 6.908, but very close to this value.

You now can choose the best fitting design for your application. Design IV with good K_cst – value should be the best choice.

All designs can be quickly computed with the program *WindfC#*.

Here are the screen shots:

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 $T_{del} = 3T_0/2$

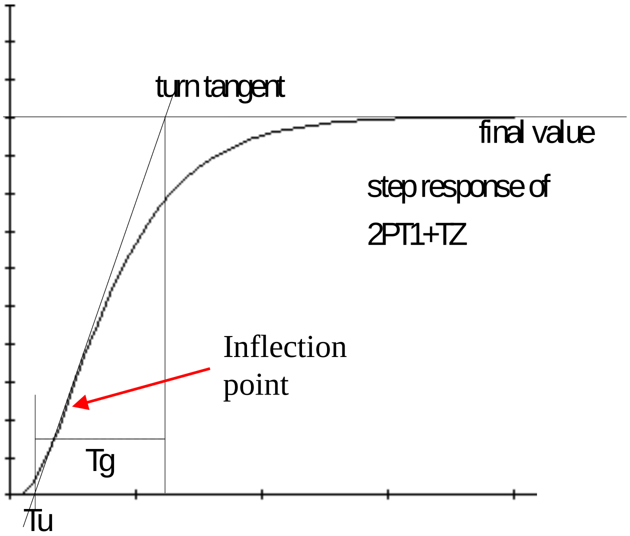
$$y(n) = p_1 y(n-1) + p_2 y(n-2) + q_0 x_d(n-1) + q_1 x_d(n-2) + q_2 x_d(n-3)$$

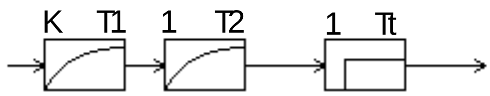
Type	F(p)	q_i, p_i, all missing q_i,p_i=0	remarks
P	K	$q_0=K$	non recurs.
P	K	$q_0=K$, $q_1=-K$, $p_1=1$	recursive
PI	$K_R(1+pT_N)/pT_N$	$q_0=K_R(1+T_0/T_N)$, $q_1=-K_R$, $p_1=1$	recursive
PD	$K(1+pT_V)$	$q_0=K(1+T_V/T_0)$, $q_1=-KT_V/T_0$, $st\approx 1+T_V/T_0$	non recursive
PD	$K(1+pT_V)$	$q_0=K(1+T_V/T_0)$, $q_1=-K(2T_V/T_0+1)$, $q_2=KT_V/T_0$, $p_1=1$, $st\approx 1+T_V/T_0$	recursive
PID	$K(1+pT_D+1/pT_i)=$ $\dfrac{K_R(1+pT_N)(1+pT_V)}{pT_N}$	$q_0=K(1+T_D/T_0+T_0/T_i)$, $q_1=-K(2T_D/T_0+1)$, $q_2=KT_D/T_0$, $p_1=1$, $st\approx 1+T_V/T_0$	recursive Design I
PIDT1	$K(1+pT_D+1/pT_i)/(1+pT_1)$, $\dfrac{K_R(1+pT_N)(1+pT_V)}{pT_N(1+pT_1)}$ $st_{max}=1+T_V/T_0$,	$q_0 = b(1+2\frac{T_N}{T_0})(1+2\frac{T_V}{T_0}),\ b=\frac{K_R T_0}{2T_N(1+2T_1/T_0)}$ $q_1 = b\left((1-2\frac{T_N}{T_0})(1+2\frac{T_V}{T_0})+(1+2\frac{T_N}{T_0})(1-2\frac{T_V}{T_0})\right)$ $q_2 = b(1-2\frac{T_N}{T_0})(1-2\frac{T_V}{T_0})$, $p_1 = \frac{4T_1/T_0}{(1+2T_1/T_0)}$ and $p_2 = 1-p_1$	recursive Design IV trapezoids,

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

Procedure:

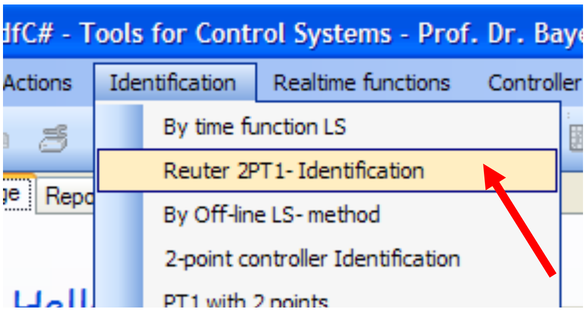
1. If the process can be approximated with 2PT1 + delay time block, the step response should look like this:



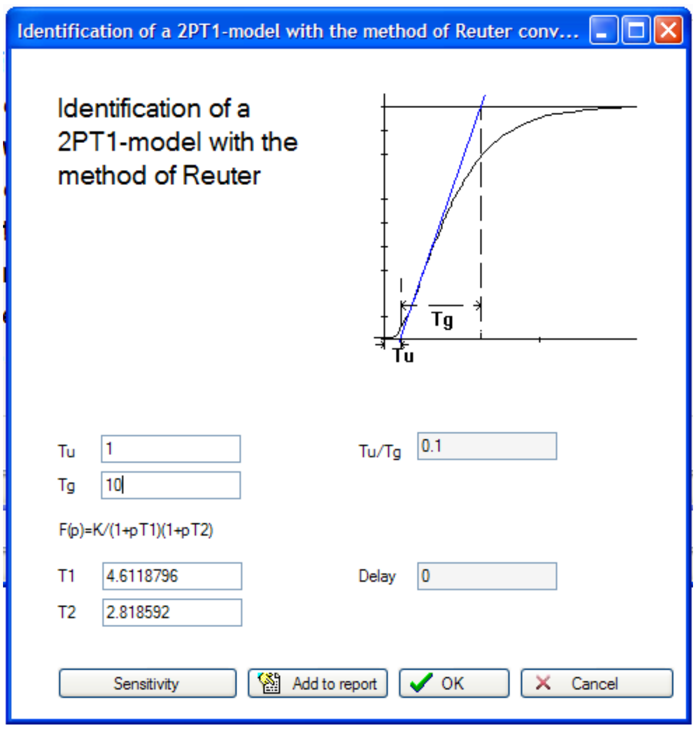


$K = fv / U_0$.

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.



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

Now derive s to the single parameters:

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

$$\vdots$$

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

With the abbreviations

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

and

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

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^1 are the starting values or old parameters. The above matric equation can be solved:

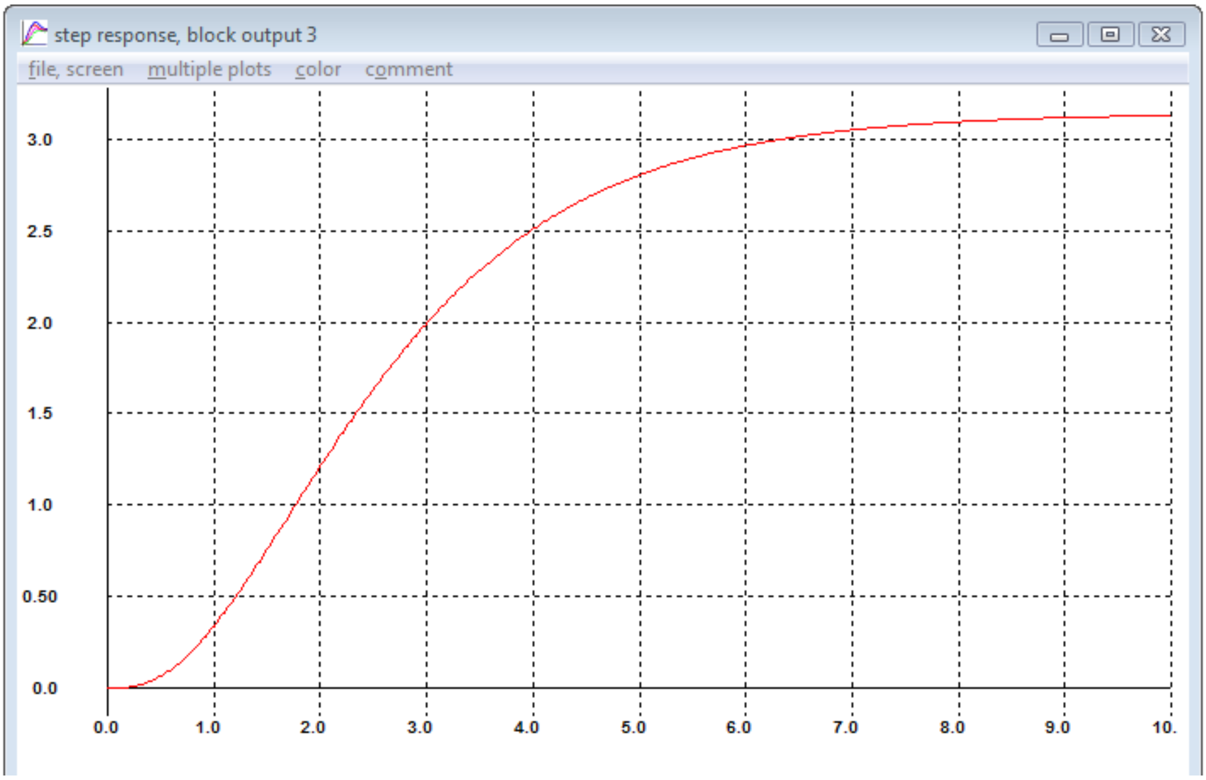
$$\begin{pmatrix} A_1 \\ A_2 \\ \\ A_M \end{pmatrix} = \begin{pmatrix} A_1^l \\ A_2^l \\ \\ A_M^l \end{pmatrix} + \begin{pmatrix} A_{11} & A_{12} & & A_{1M} \\ A_{21} & A_{22} & & A_{2M} \\ & & & \\ A_{M1} & & & A_{MM} \end{pmatrix}^{-1} \begin{pmatrix} q_1^l \\ q_2^l \\ \\ q_M^l \end{pmatrix}$$

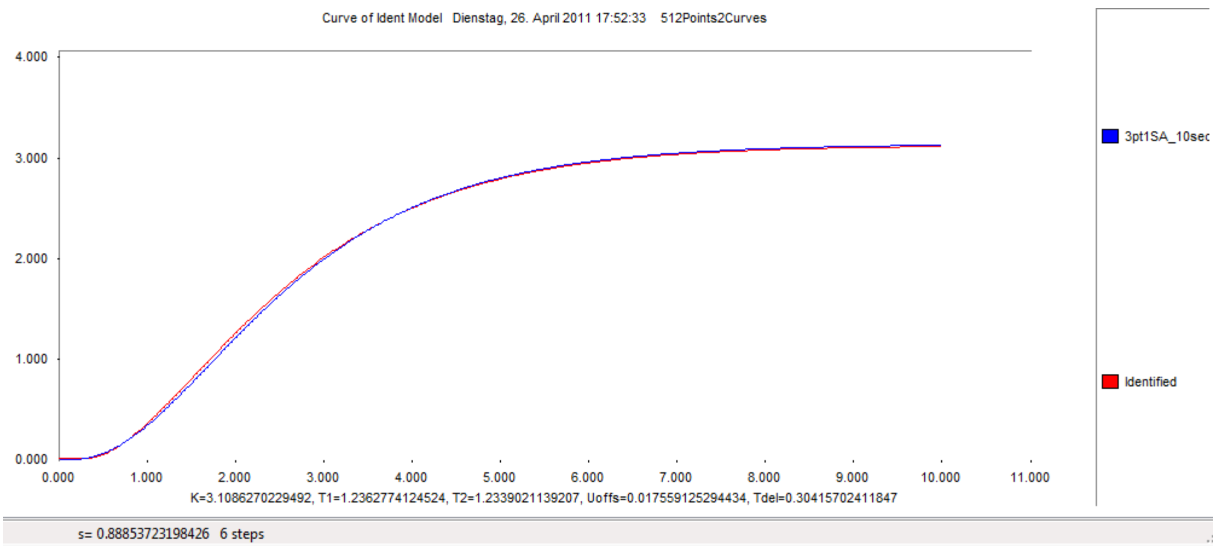
So a matric – Inversion is necessary, but not a problem, because several algorithms are available since years.
Convergence is possible (the new sum of squares with the new parameters is smaller than the previous one), if starting values are near to the valley (minimum).
If the function f(x) is a parabolic 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:

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	$F(p)=\frac{U_2}{U_1}=\frac{Kp}{1+\frac{2d}{\omega_0}p+\frac{1}{\omega_0^2}p^2}$	$h(t)=\frac{K\omega_0}{\sqrt{1-d^2}}e^{-d\omega_0t}\sin\omega_Et$, $\omega_E=\omega_0\sqrt{1-d^2}$

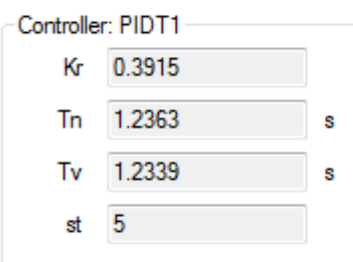
8.2.2 Example for Controller design purpose

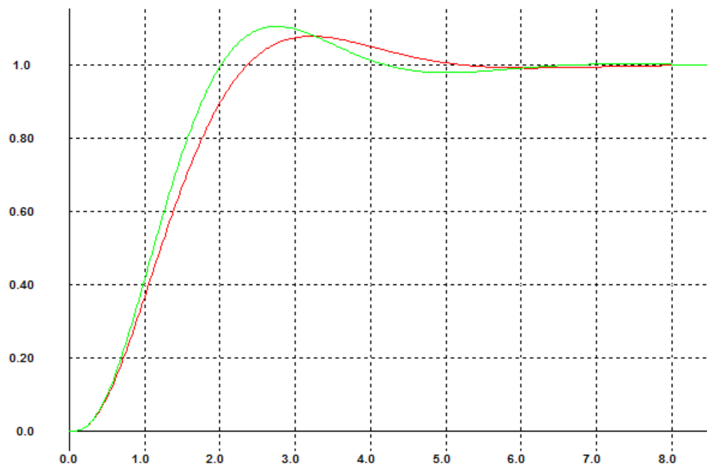
Now a 3PT1- process should be identified via step response and used to design a controller, which is tested at the original 3PT1- Process. The data of the 3 PT1 are K=3.1415, T1=1.5, T2=0.6 andT3=0.7. The step response is created with program Regdelph and is stored in the file *3pt1SA_10sec.sim.* This 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 Parmeters to 5 and then you should get the following result:





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





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

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

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

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

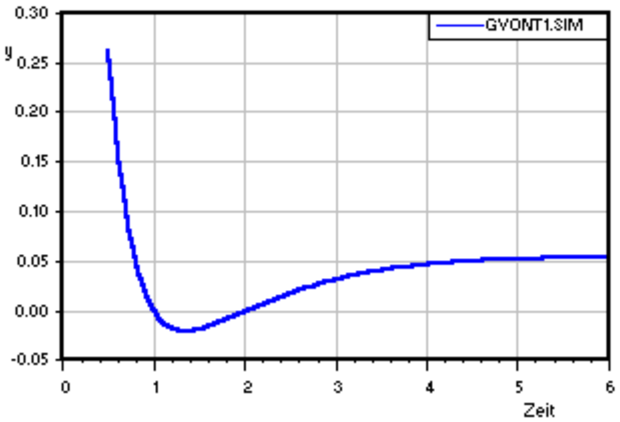
Herr you have two equations with the three unknowns BK, T_1 und T_z. Now divide x_{max} by x_{min} and BK can be reduced and you have one equation with two unknowns T_1 and T_z.

Start with T_z =0 (no delay) and look for a solution without delay (2PT1-process). Solve the

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

equation $g(T_1)$=0 e.g. with nested intervals. If there is no solution add a delay.

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 mor 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} und e=t_{max} und ea=exp(-a/T_1), eb=exp(-b/T_1) usw. und Λ=X_{max}/x_{min}

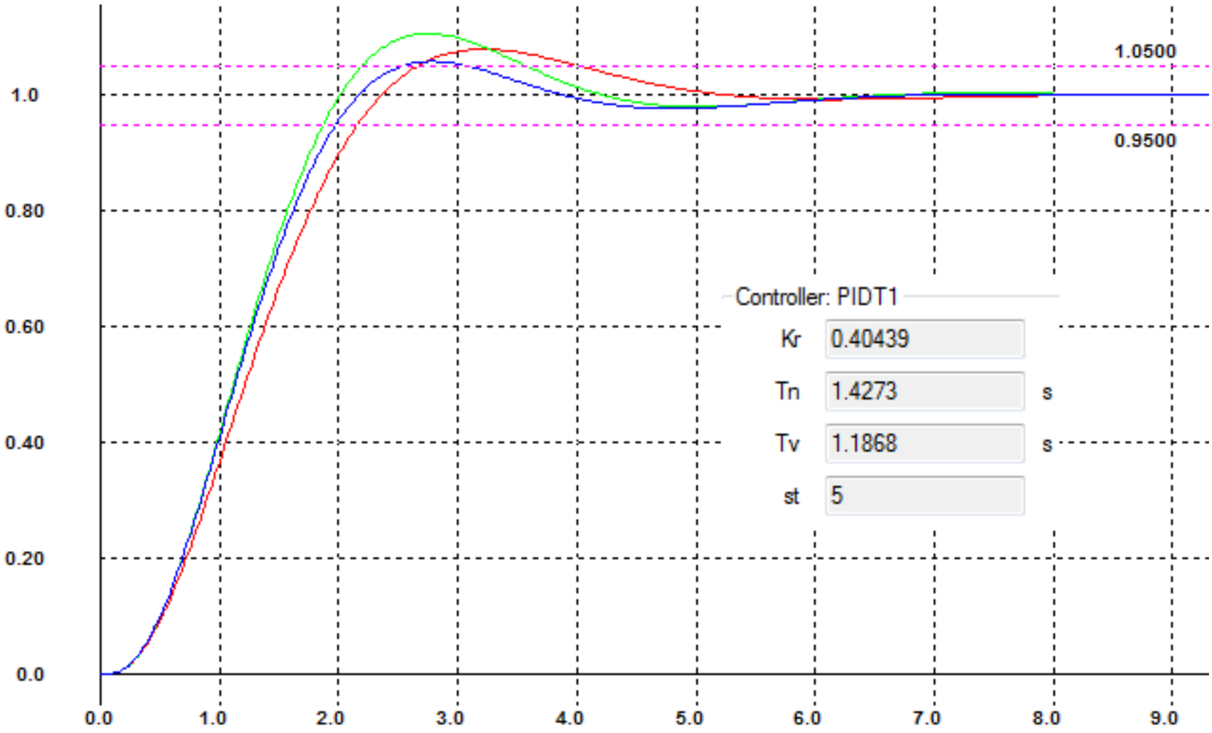
$g(T_1)$= Λ(ea-eb-ec+1)-ed+ee und

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

The factor $(-1/T_1^2)$ has no influence tot he 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

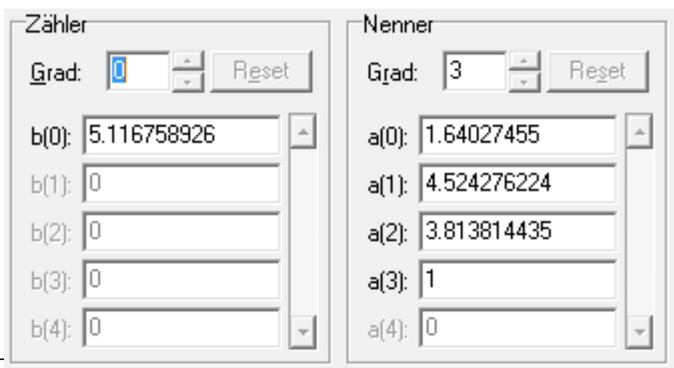
A controller design with these values gives the following numbers and RSR:



Red: Original controller
Green: Step response identification
Blue: Two-Point-Controller identification.

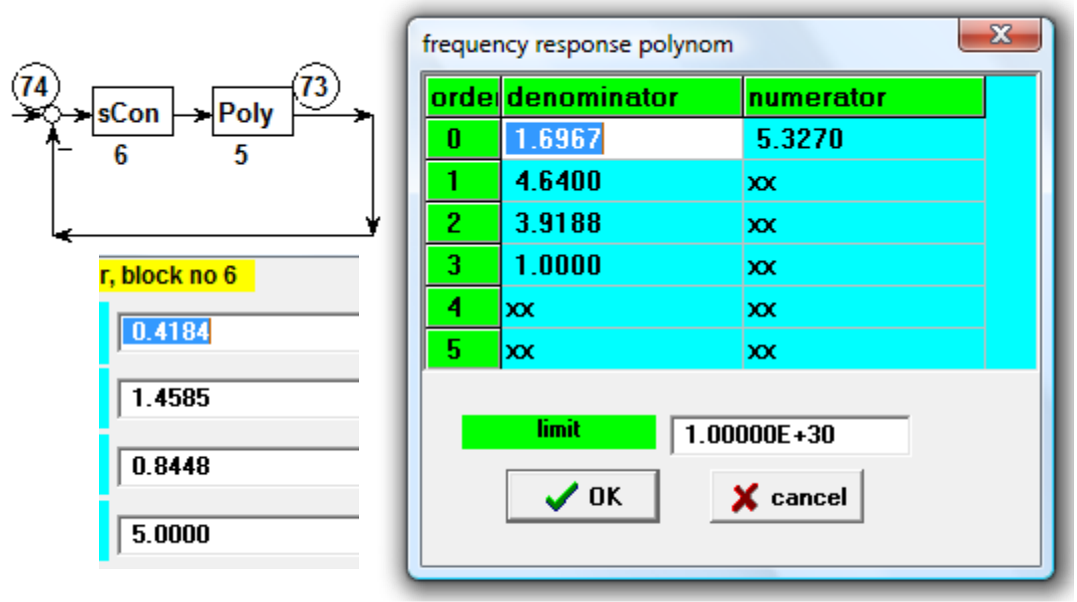
8.4 *Identification with program IDA.exe*

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:

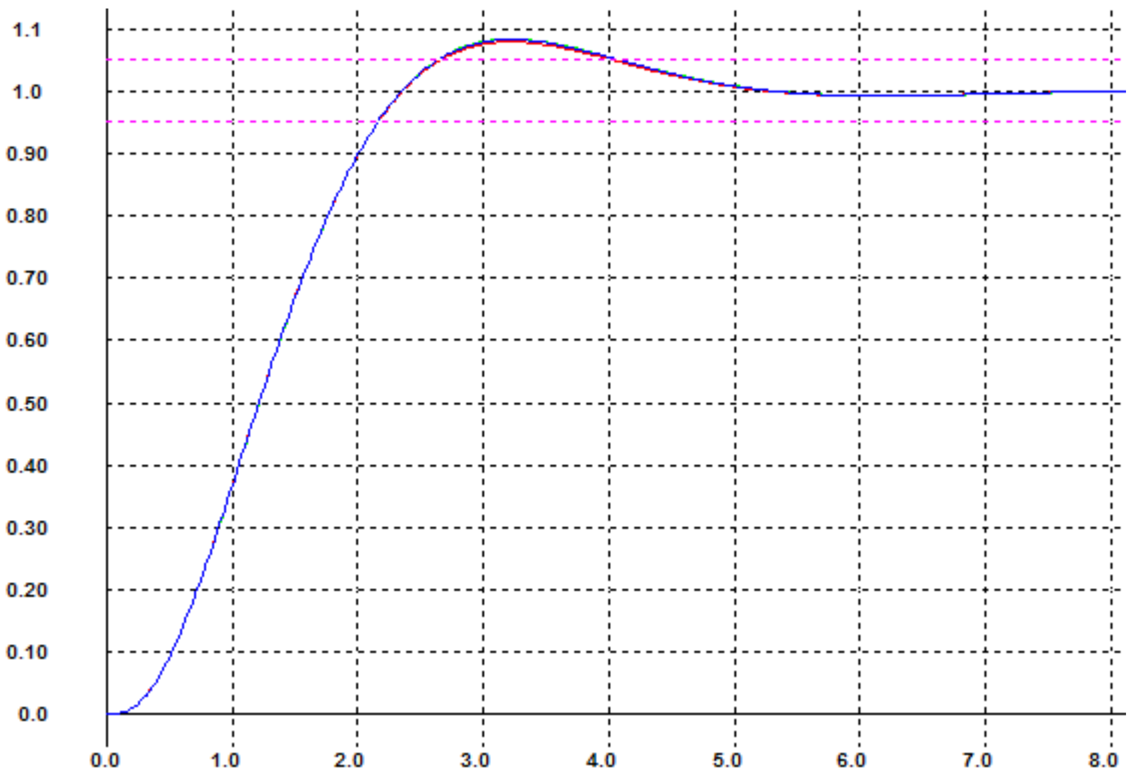


Now in menu “Datei → Steuerparameter” set the “Nennergrad “ = Denominator-degree” to 4 and checkBox “n anpassen”. Close this window and start “approximation” . Each click on “Weiter”increase

With the files *idaPIDprocess* the following data can be calculated:



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!).

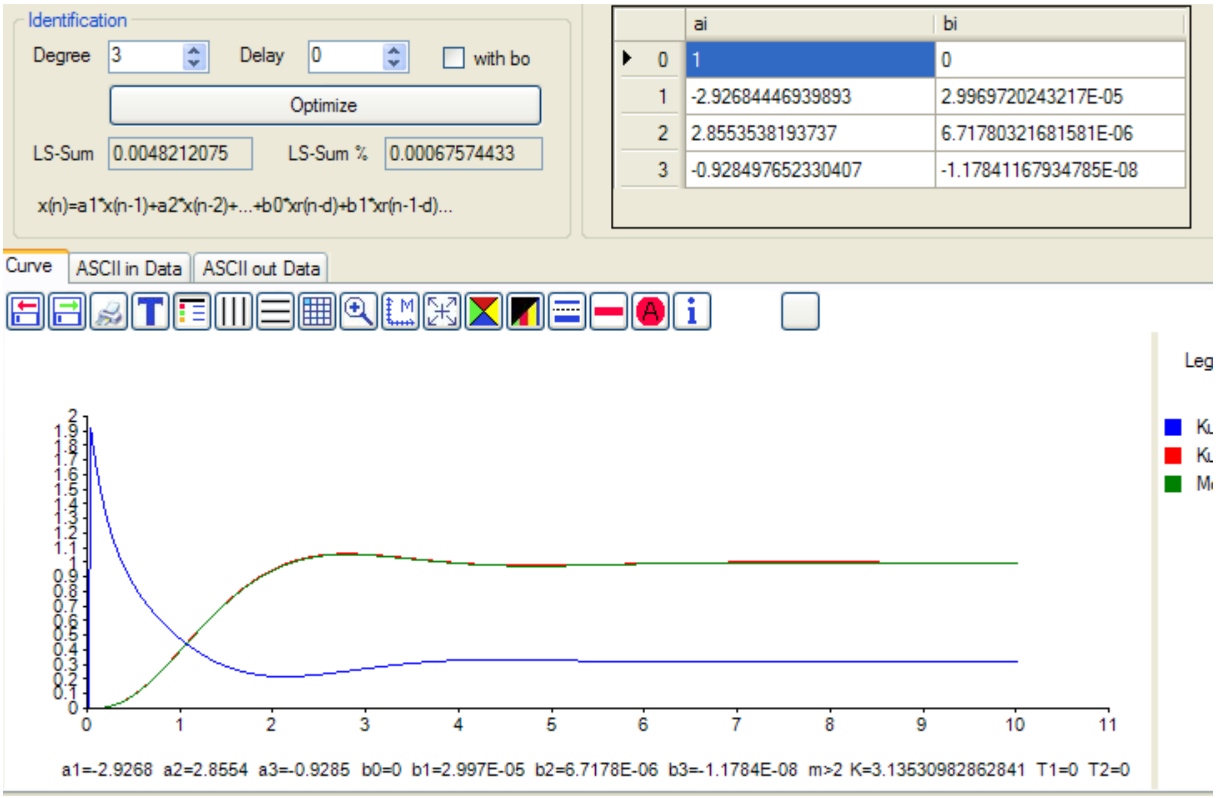


Last step is the comparison of the three time constants. For this purpose the transfer function F(p) has to be converted into a time-constant form usind tool Windfc#- You can load the ufk-files with the ufk- button, then click on button "Factorise" and you get the following results, compared with the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

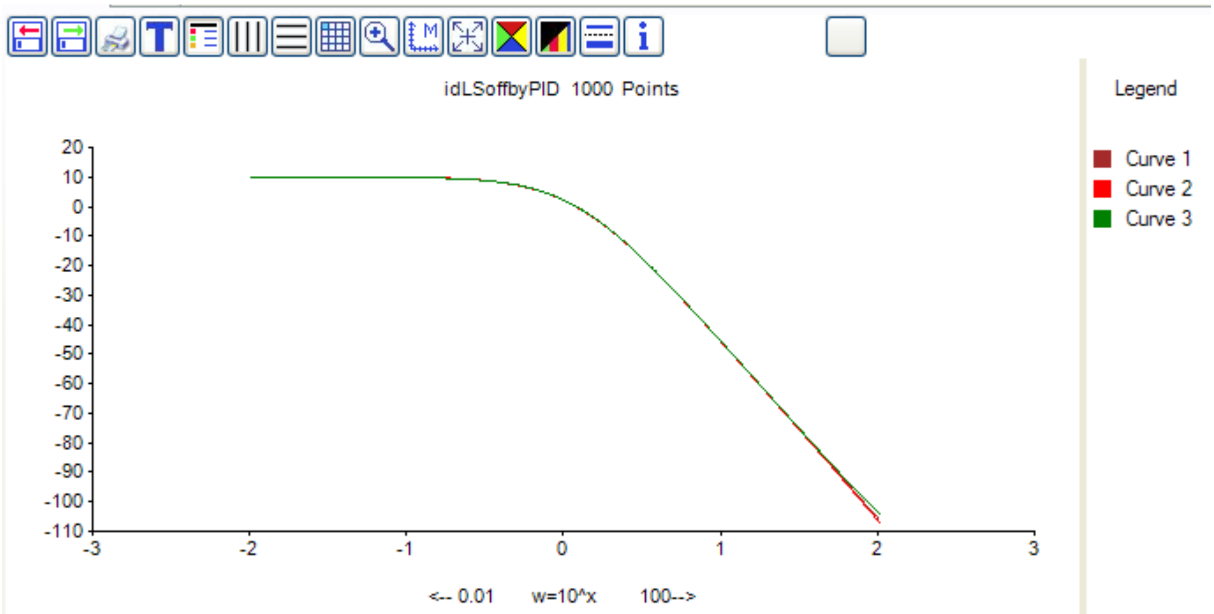
T or d
1.4315258
0.78238697
0.54433049

T or d
1.3370105
0.91714963
0.4806527

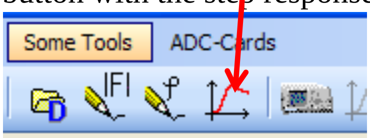
You can see, that difference in control system behavior is small in spite of the differences in time constants are big, more than 20%-



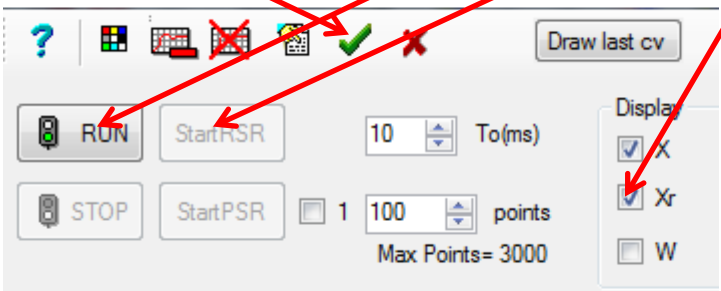
These both results can be compared as bode plot with the original 3PT1- bodeplot: This results in three nearby identic curves:



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.



This module is prepared to use this method in a realtime- measurement with the controller-module. Open either *"Real-Time- Controller→ Student control Box"* or *"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

$$F(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_m z^{-m}}{1 + a_1 z^{-1} + \ldots. + a_m z^{-2}} z^{-d}$$

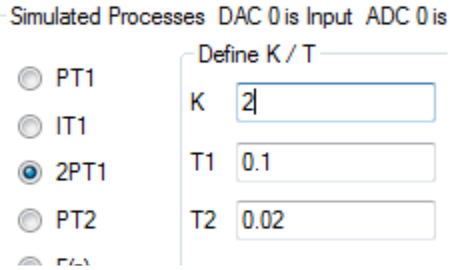
With m is the degree of the function and d is a delay. The unknowns are the a_i and b_i. A PTn-process has $b_0=0$. This gives the algorithm (output y and input u)

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

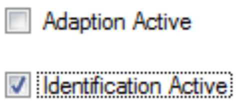
The same equation one step before:

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

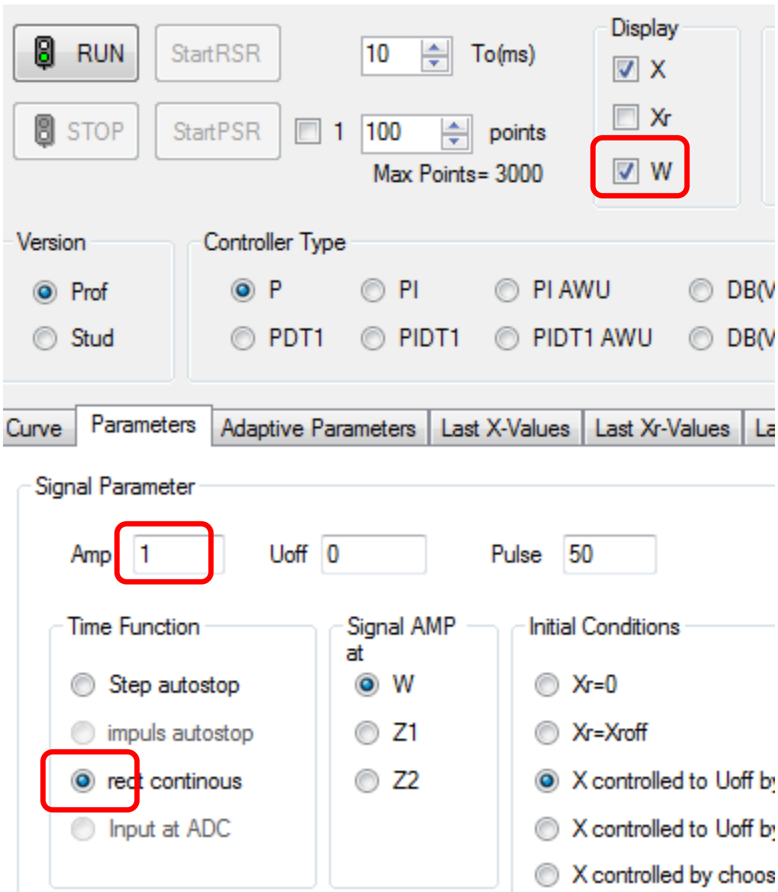
No do this as long there are as many equations as unknowns. This equation system looks like this:



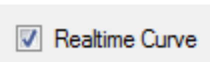
Then design controllers with menu "*Controller Design* → *Design of digital PIDT1+div*", select the same process with same parameters and leave with green arrow button.
Activate identification on "*Real-Time- Controller* → *Adaptive Advanced controller*" on the page "Adaptive parameters":

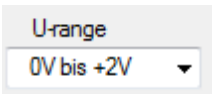


Follow the settings below on page Parameters:



Activate Realtime Curve with





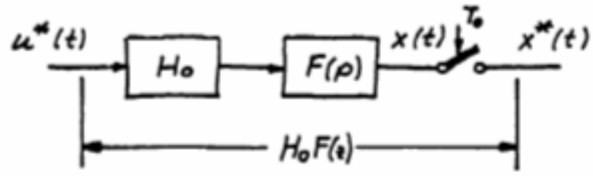
and set the amplitude scale U- range to 0 – 2V.

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.

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 following mechanism:



$$H_0F(z) = \mathcal{Z}\{H_0(p)\cdot F(p)\} = \mathcal{Z}\left\{\frac{(1-e^{-pT_0})}{p}\cdot F(p)\right\}$$

$$H_0F(z) = \mathcal{Z}\left\{\frac{F(p)}{p} - \frac{F(p)}{p}\cdot e^{-pT_0}\right\} = 1\cdot\mathcal{Z}\left\{\frac{F(p)}{p}\right\} - z^{-1}\cdot\mathcal{Z}\left\{\frac{F(p)}{p}\right\}$$

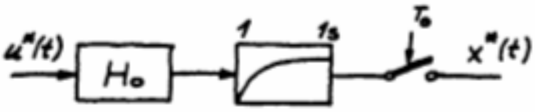
$$H_0F(z) = (1-z^{-1})\cdot\mathcal{Z}\left\{\frac{1}{p}\cdot F(p)\right\}$$

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

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.

Example:

Beispiel:



$$H_0F(z) = \frac{X(z)}{U(z)} = \frac{z-1}{z}\cdot\mathcal{Z}\left\{\frac{1}{p}\cdot\frac{1}{1+p}\right\}$$

$$= \frac{z-1}{z}\cdot\frac{(1-e^{-T_0})z}{(z-1)(z-e^{-T_0})}$$

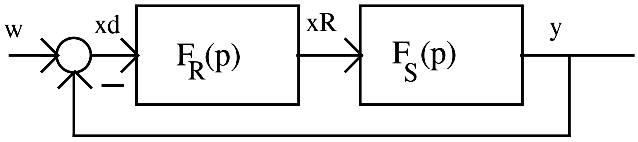
$$H_0F(z) = \frac{1-e^{-T_0}}{z-e^{-T_0}}$$

Für $T_0 = 0{,}5\,s$ gilt

$$H_0F(z) = \frac{1-0{,}607}{z-0{,}607} = \frac{0{,}393}{z-0{,}607}$$

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

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



Is the general $H_0F(z)$ written as

$$H_0F(z)=\frac{b_0+b_1z^{-1}+...+b_mz^{-m}}{1+a_1z^{-1}+a_2z^{-2}+...+a_mz^{-m}}z^{-d}=\frac{y}{x_R}$$

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

$$F_R(z)=\frac{q_0+q_1z^{-1}+...+q_mz^{-m}}{1-p_1z^{-1}-p_2z^{-2}-...-p_mz^{-m}}=\frac{x_R}{x_d}$$ with the algorithm

$$x_R(n)=p_1x_R(n-1)+p_2x_R(n-2)+...+q_0x_d(n)+q_1x_d(n-1)+q_2x_d(n-2)+...$$

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

$$F_R(z)=\frac{q_0z^{-1}+q_1z^{-2}+...+q_mz^{-m-1}}{1-p_1z^{-2}-p_2z^{-3}-...-p_mz^{-m-1}}$$ with the algorithm

$$x_R(n)=p_1x_R(n-2)+p_2x_R(n-3)+...+q_0x_d(n-1)+q_1x_d(n-2)+q_2x_d(n-3)+...$$

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

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

and the real:

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

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:

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:

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

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:

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

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

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

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

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

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

$$F_R(z) = \frac{Y(z)}{X_d(z)} = \frac{q_0 z^{-1} + q_1 z^{-2} + \ldots + q_m z^{-m-1}}{1 - p_1 z^{-2} - p_2 z^{-3} - \ldots - p_m z^{-m-1}} = \frac{q_0 z^{-1} + q_1 z^{-2}}{1 - p_1 z^{-2}}$$

The ps and qs :

$q_0 = 1/(b_0 + b_1 + \ldots + b_m) = 1/b_1 = 1.8388852$ and

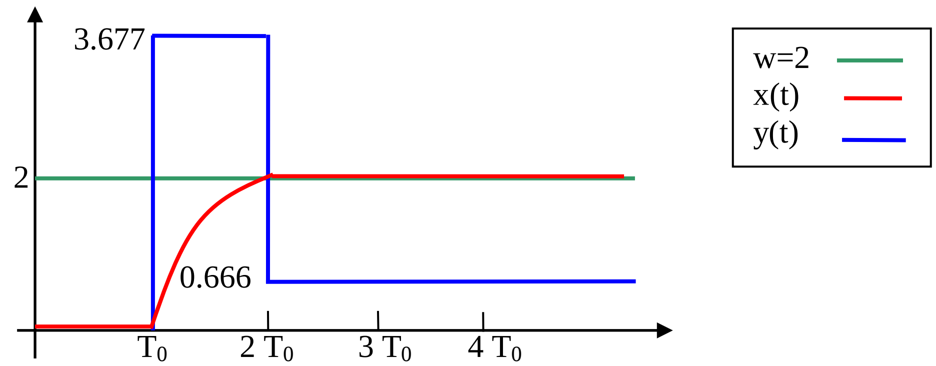
$q_1 = q_0 * a_1 =$- 1.50555185 and $p_1 = q_0 * b_1 = 1$. We get the transfer function of the DB-controller

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

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

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

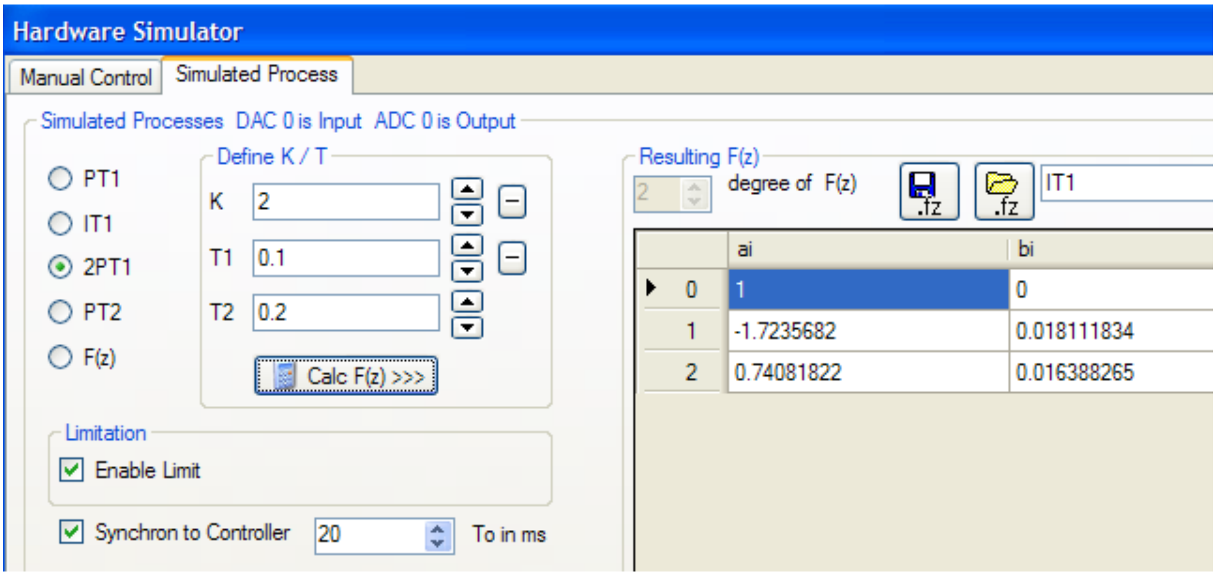


Description of the function in words: Because it is a real function the first cycle is empty, nothing happens. Then the dead beat controller throws out a first pulse with an amplitude of 3.677. This value is exactly the amout which is necessary to move the PT1- output in one T_0 to the desired value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



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 ps and qs :

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

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

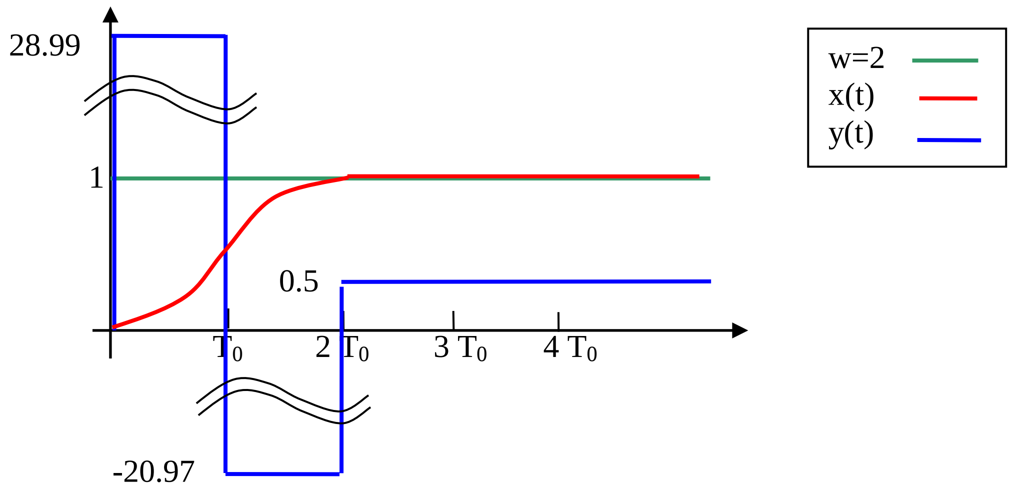
$q_2 = q_0 * a_2 = 21.4729$ and $p_2 = q_0 * b_2 = 0.475014$.

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

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

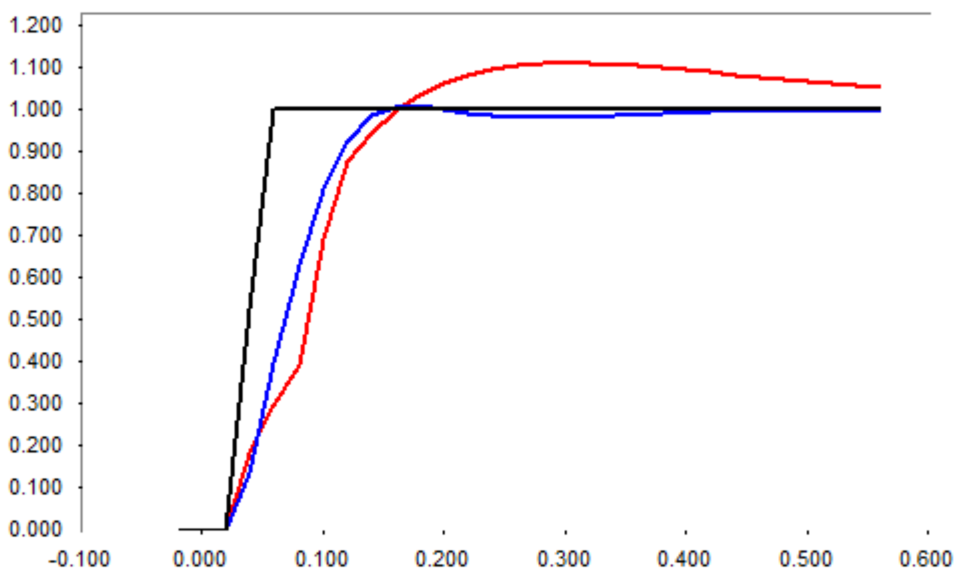
$y(n) = 0.5250 y(n-1) + 0.4750 y(n-2) + 28.99 x_d(n) - 49.95 x_d(n-1) + 21.47 x_d(n-2)$. With 4 significant digits. Compare the WindfC#- results, in the main menu “Controller-Design” the Item “Design of digital PI/PIDT1+div” has following results (part of the window):

	qi	pi
1	28.985424	1
2	-49.958354	0.52497919
	21.47293	0.47502081



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 : *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:

```
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];
}
```

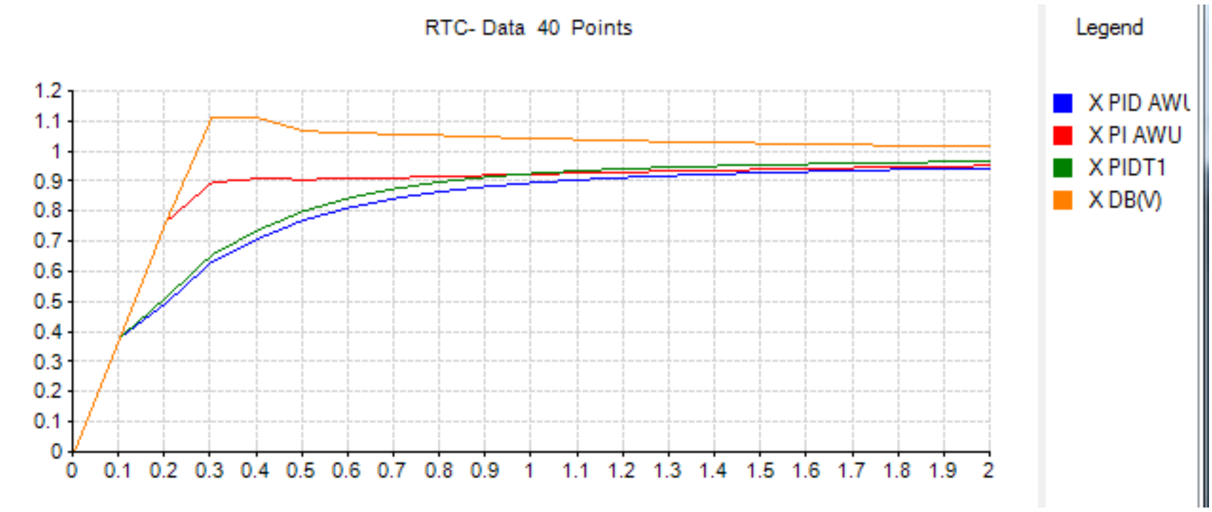
uk is output of new controller value
hr=pr and gr=qr are the controller coefficients
u[] are the old controller outputs
y[] are the old process outputs
m degree
teta the a_i and b_i in one vector
wk the actual desired value
f31 the f_3 value
yk the actual process output.

9.7 PFC- Predictive Functional Control

See Book: *Predictive Functional Control - Principles and Industrial Applications - Richalet, O'Donovan*.
See German diploma thesis Graeper: file: *DA Graeper PFC.pdf*

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.



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

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

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):

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

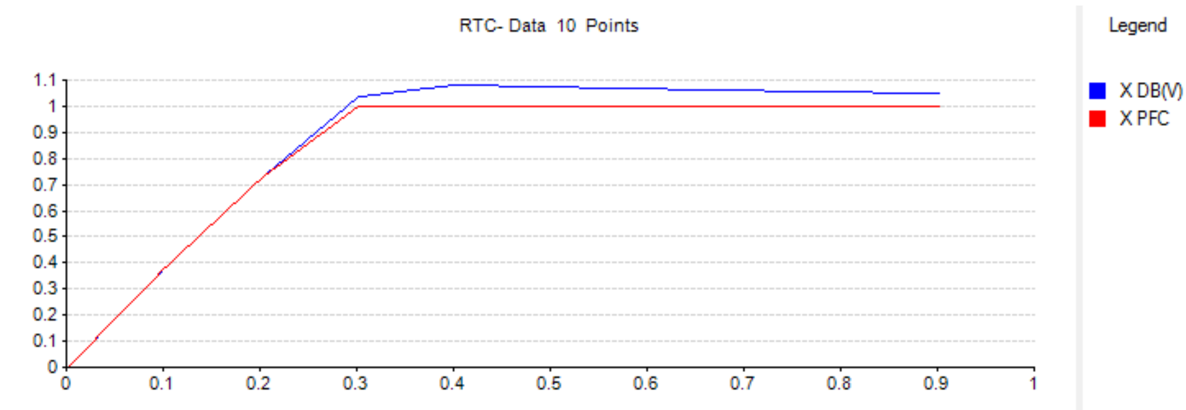
In simple PT1- case the solution is simple:

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

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:

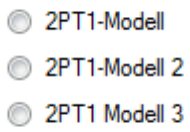


If process is a 2PT1, then the design has to be changed, because with one positive impulse the desired value cannot be reached, there must be a breaking step. So the time for the inflection point t_w is calculated of the step response of the model:

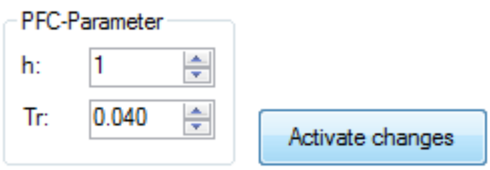
$$t_w = \frac{T_1 T_2}{T_1 - T_2} \ln(T_1 / T_2)$$

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)



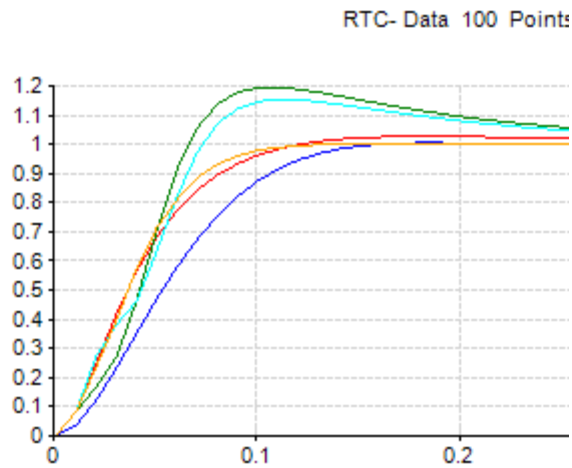
There are two versions of 2 PT1 programmed by Mr. Graeper and a third developed by myself. If a change is made in the two parameters h and desired settling time Tr, you must click on button "Activate changes".



Play with the different models. My third version automatically adapt to ideal / real version, the both two versions are only valid in ideal mode.
Example: 2PT1 model with K=0.4, T_1=0.1s T_2=0.02s and T_0=10ms. Ideal algorithms
First the conventional solutions:

	ai	bi
0	1	0
1	-1.5113681	0.008234357
2	0.54881164	0.0067430664

	qi	pi
1	66.767158	1
2	-100.90975	0.54978462
	36.642594	0.45021538



Pi is slow, PIDT1 and OR OK, but DB – controllers with high overshoot.
Now the PFC together with PIDT1:
1st version with h=4 and Tr=30 ms
2nd version with h=3 and Tr=10 ms
3rd version with h=2, Tr no influence

$$x_m(k) = -a_1 x(k-1) - a_2 x(k-2) + b_1 y'(k-1) + b_2 y'(k-2)$$
$$x_m(k) = -a_1 x(k-1) - a_2 x(k-2) + b_1 y(k-2) + b_2 y(k-3)$$
$$x_m(k+1) = -a_1 x(k) - a_2 x(k-1) + b_1 y(k-1) + b_2 y(k-32)$$
$$x_m(k+2) = -a_1 x_m(k+1) - a_2 x(k) + b_1 y(k) + b_2 y(k-1) = w_{new}$$

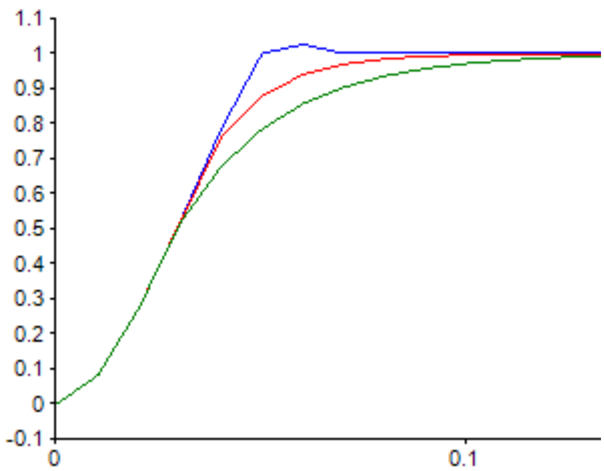
Last equation solved to y(k) is

$$y(k) = \left(w_{new} + a_1 x_m(k+1) + a_2 x(k) - b_2 y(k-1)\right) / b_1$$

In C – program:

```
xmodPred = -fa1 * fax - fa2 * valold + fb1 * fYrold + fb2 * fYrold2;
wnew = xmodPred + (w - xmodPred) / iH;
fYr = (wnew + fa1 * xmodPred + fa2 * xmodpredold - fb2 * fYr) / fb1;
```

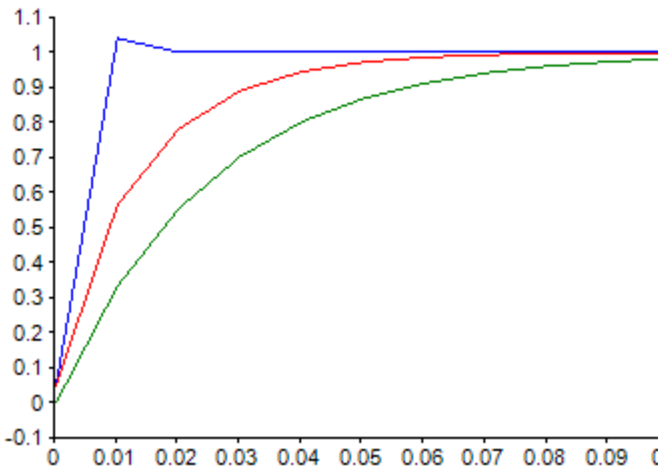
Results:
Again 2PT1 model with K=0.4, T_1=0.1s T_2=0.02s and T_0=10ms. With actuator limit +-10V.

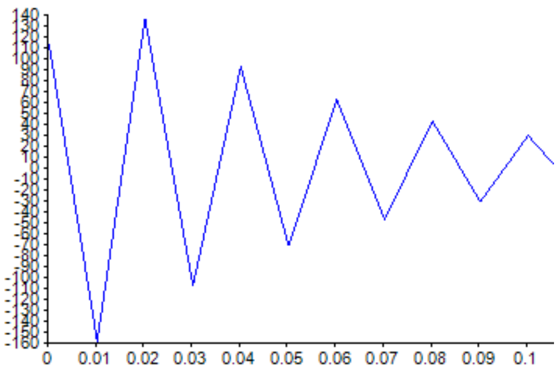


Blue: h=1
Red : h=2
Green: h=3

In real case I have got exactly the same curves one To shift right.

But if I deactivate the actuator limitations, following results happen:

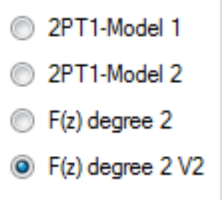




The process outputs seemed OK, but the controller outputs have strong oscillations (see blue right curve in the case of h=1).

9.7.2 Theory of 4th version 2nd order PFC

Now I want to introduce a fourth 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

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

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

I got the following solutions:

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

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

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

$$Equ(8d):\ \ z_2 = -a_2 x(k+1)$$

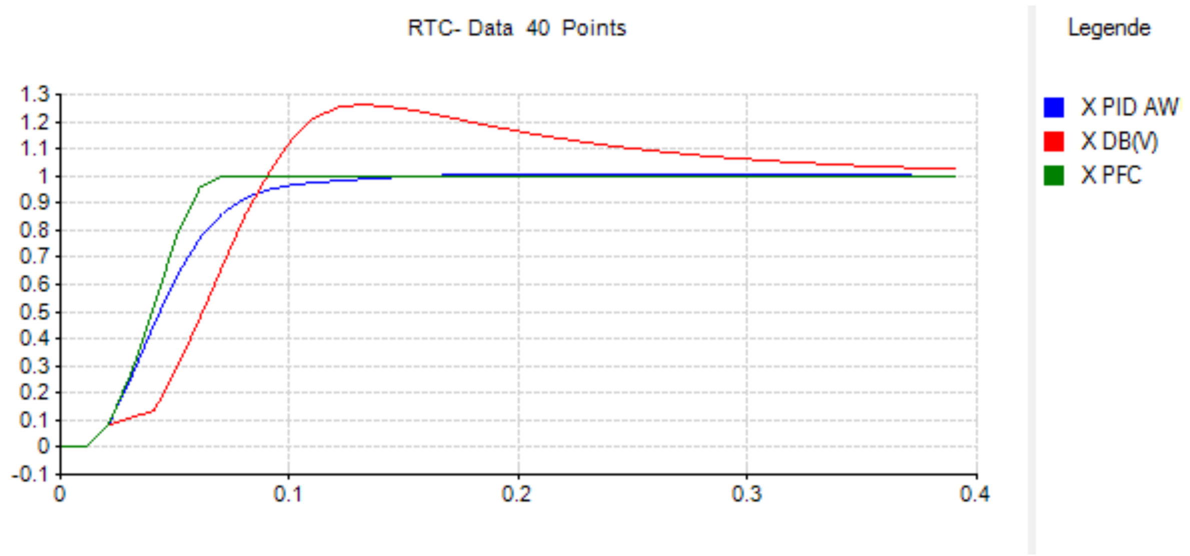
I got the following solutions:

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

$$Equ(10d):\ \ y(k) = \left(x_m(k+2) - z_1\right)/b_1$$

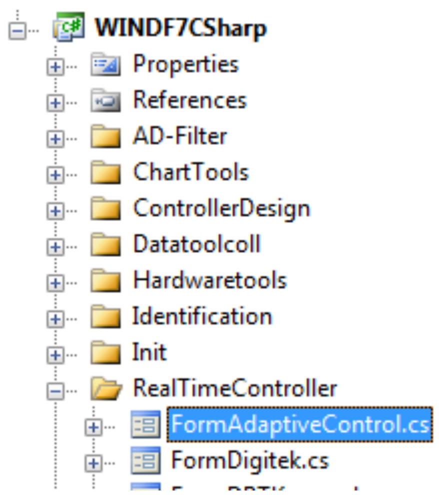
$$Equ(11d):\ \ y(k+1) = \left(w + a_1 x_m(k+2) + z_2 - b_2 y(k)\right)/b_1$$

Resulting reference step responses with one To – delay:

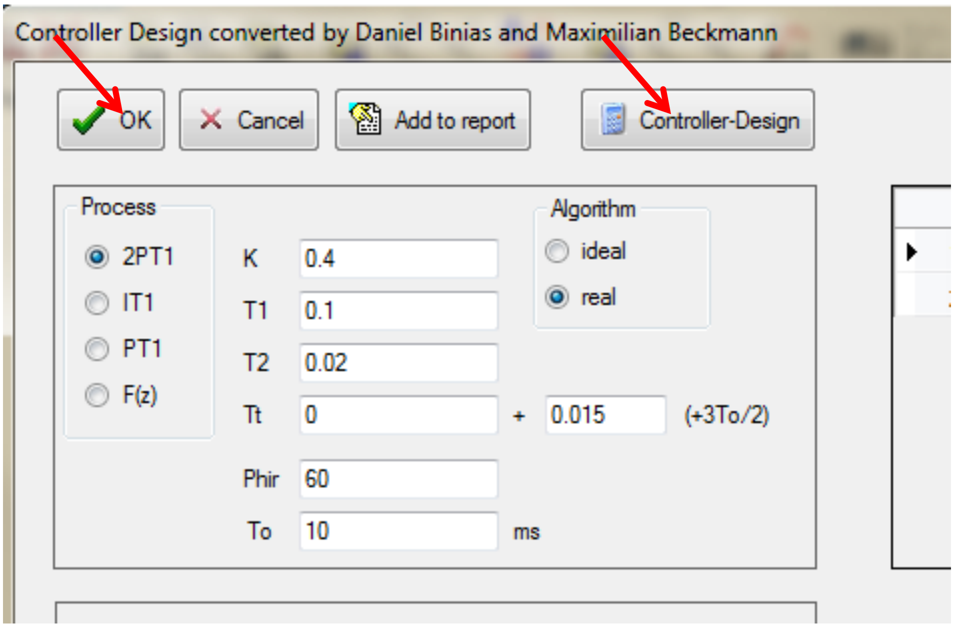


9.8 PFC – Controller in tool program Windfc#

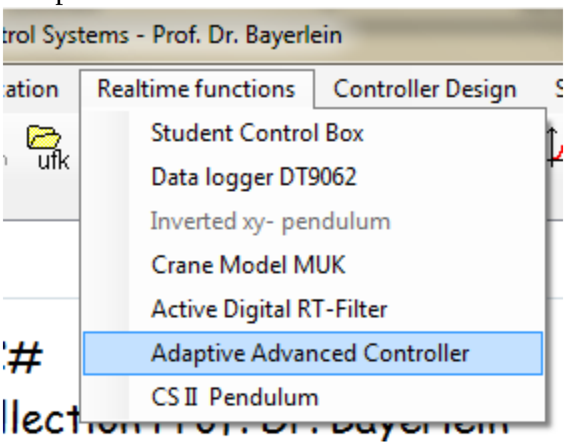
Here you can find some hints to use and test these PFC- controllers in my tool program Windfc# starting with version nb. 7.4.17. the source code is also published, so it should be easy to implement these controllers in other hardware combinations.



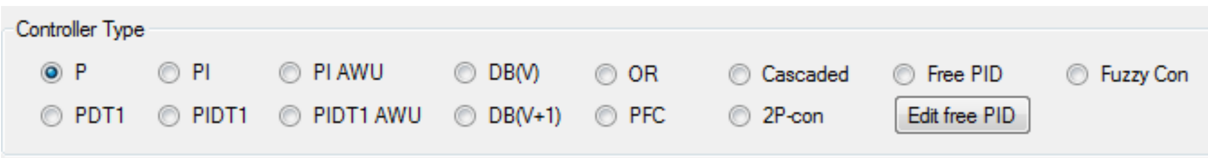
You can find the PFC- source in the file “FormAdaptiveControl.cs”. the sour e runs under MS Studio 2008 or MS Studio 2010.



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

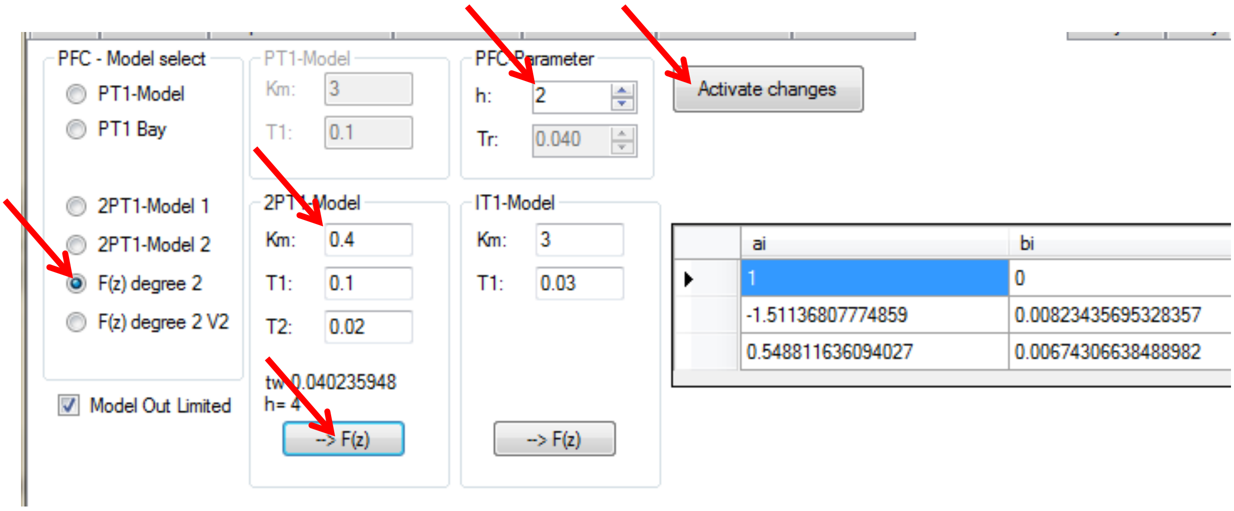


Here you can select different types of controllers.



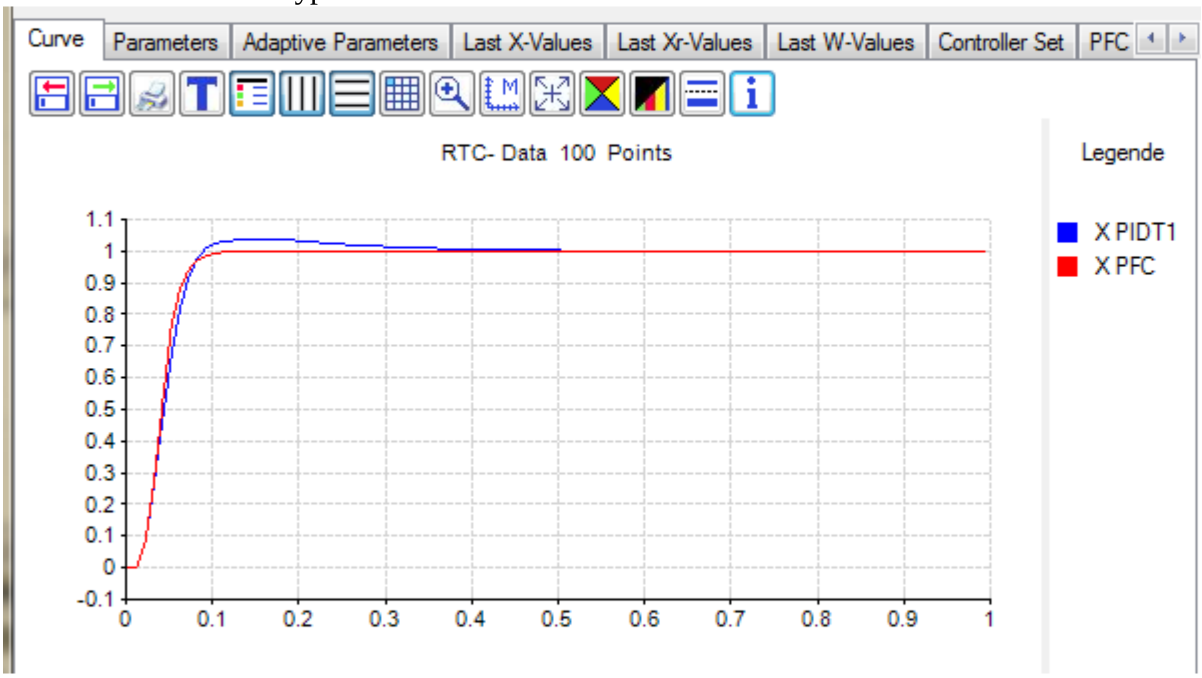
This module can now calculate a single reference step response (RSR) or a continuous response on a square reference signal jumping up and down. The single RSRs are displayed in the folder “Curve”, the continuous signal is displayed in real time in a new window.
Lets start with the single RSRs. In folder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

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:

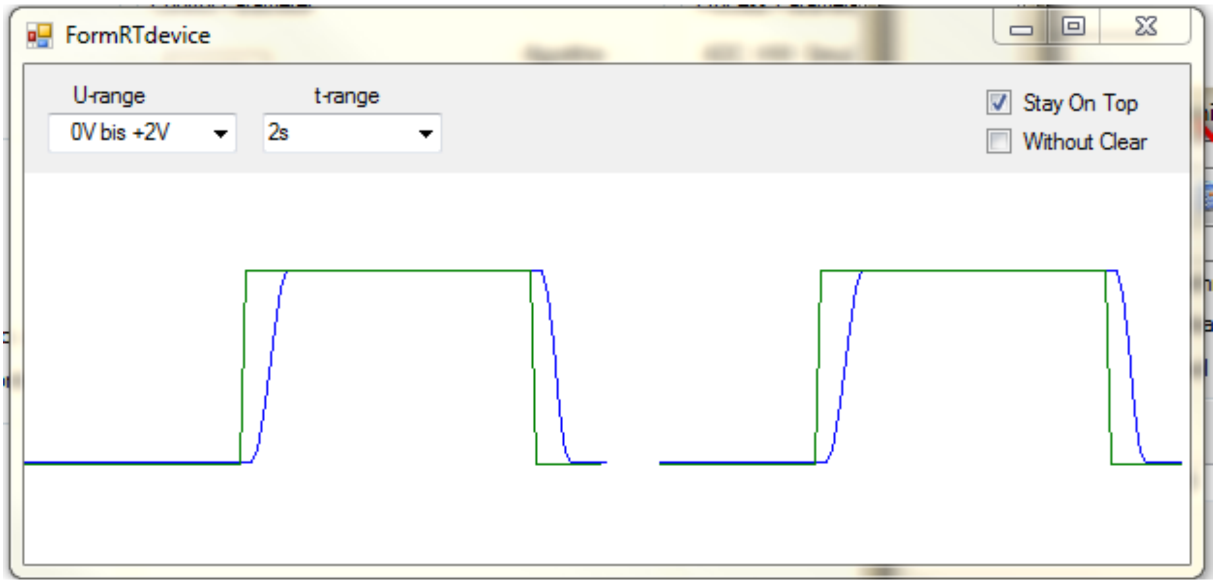


Start first with first second order PFC, change the 2PT1- parameters of the PFC to our process values, select h=2 (two step horizon), click on button “→F(z)” to get the F(z) of the model and click on “Activate changes” – button.

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



Red curve is this PFC. It takes 90ms to reach 2% final value. Time scale changed to 0.2 sec gives this picture:



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.

Prof. Dr. Bayerlein 1/17/2012 2:55:00 PM