Introduction to Industrial Communication Networks, Integration Systems and Control (PCWorx Express) - Apostilled material of the practical class - 003_A: PID Controllers

Technical Report · May 2018

CITATIONS
0

2 authors:

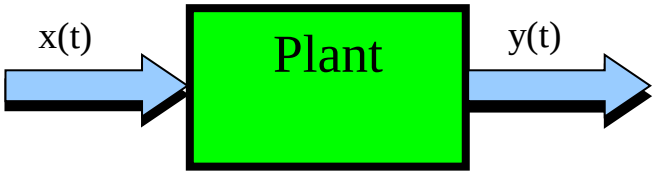
Hermes Jose Loschi
University of Campinas
53 PUBLICATIONS **37** CITATIONS
SEE PROFILE

Yuzo Iano
University of Campinas
122 PUBLICATIONS **204** CITATIONS
SEE PROFILE

PID controller in PC Worx

1. The simple mathematical model of a plant

If some plant has single input and single output (SISO) it might be represented as:



where:
x(t) – input of the system;
y(t) – response of the system.

A first order system has a differential equation – given below

$$T\frac{dy}{dt} + y(t) = k \cdot x(t) \quad (1)$$

with this differential equation, the terms are:
T – the system time constant;
k – the gain of the system.

Such model can be used for representing the simple heating or another object. If the differential equation is Laplace transformed we get:

$$T \cdot y(s) \cdot s + y(s) = k \cdot x(s) \quad (2)$$

The transfer function of the system:

$$W(s) = \frac{y(s)}{x(s)} = \frac{k}{T \cdot s + 1} \quad (3)$$

Some plants have a transport delay. The transfer function with time delay will be:

$$W(s) = \frac{k}{T \cdot s + 1} \cdot e^{-\tau \cdot s} \quad (4)$$

where τ is the time delay in seconds.

The second order transfer function in most cases is more accurate:

$$W(s) = \frac{k}{(T_1 \cdot s + 1)(T_2 \cdot s + 1)} \cdot e^{-\tau \cdot s} \quad (5)$$

and it has two system time constants: T1 and T2.

If we need to control some process parameter y(t) we can use the closed-loop system with PID controller:

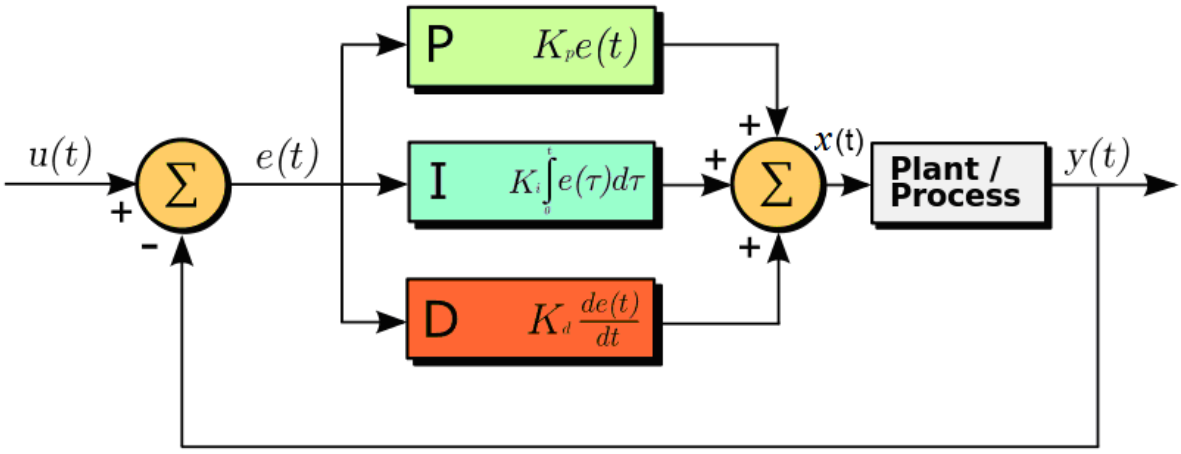
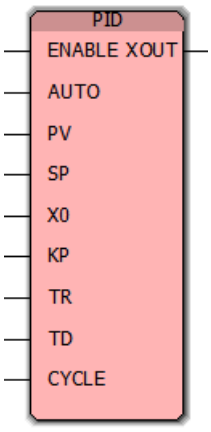


Fig. 1 – A block diagram of a PID controller in a feedback loop

PID controller calculates an "error" value *e(t)* as the difference between a measured value of a process and a desired setpoint *u(t)*. The controller attempts to minimize the error in outputs by adjusting the process control inputs [3].

2. Basic PID controller function block in PC Worx

Standart PC Worx library has the PID function block [1]:



XOUT signal is formed according to expression:

$$XOUT = KP \cdot \left(e(t) + \frac{1}{TR} \cdot \int e(t)dt + TD \cdot \frac{de(t)}{dt} \right)$$

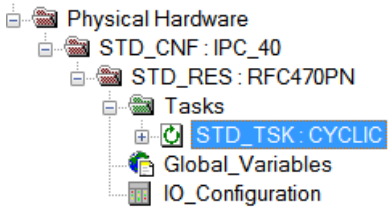
where KP, TR, TD are the adjustable parameters of PID controller.

Description of FB inputs and outputs [1]:

Parameter	Data type	Description
Input parameters		
ENABLE	BOOL	FALSE = Control disabled TRUE = Control enabled This input parameter may be used when a cyclic signal shall control the execution of the PID function block.
AUTO	BOOL	TRUE = Automatic mode FALSE = Manual mode If AUTO = FALSE, the output value XOUT is calculated as follows: XOUT = KP * X0
PV	REAL	Measured value of the process.
SP	REAL	Set point.
X0	REAL	Manual output adjustment (typical for transfer station).
KP	REAL	Proportional constant (amplification factor). This constant has to be negative.
TR	REAL	Reset time constant (time factor of integral term - time after that XOUT reaches the value of KP without consideration of the influence of the derivative term).
TD	REAL	Derivative time constant (time factor of derivative term - time before the derivative term leads to XOUT = KP in comparison to the proportional term without consideration of the influence of the integral term).
CYCLE	TIME	Sampling period, i.e., time interval after which the FB is calculated again.
Output parameters		
XOUT	REAL	Output value to the process.

IMPORTANT! The value of the operand connected to CYCLE has to correspond to the cycle time of the task in which the function block is used [1].

We will work with User-defined cyclic task, so, we can change the cycle time of the task by right click on **STD_TSK** :



Set the CYCLIC type of task (in Properties menu) and the interval of task (in Settings menu).

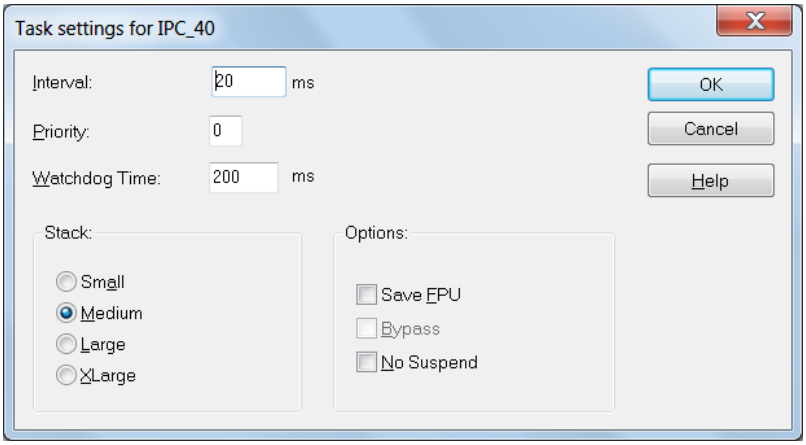


Fig. 2 – Task settings in PC Worx

In this course, we will work, always with TASK DEFAULT (See pag. 43, class 001).

3. PC Worx programming of the first order transfer function

If we use the Euler method for derivative approximation:

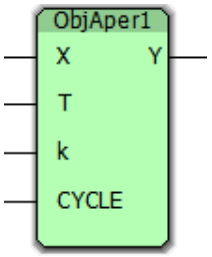
$$\frac{dy}{dt} = \frac{y_i - y_{i-1}}{dt}$$

we can rewrite the equation (1) :

$$y_i = y_{i-1} + \left(-\frac{1}{T} y_{i-1} + \frac{k}{T} x_i \right) \cdot dt \quad (6)$$

where *dt* is a time step. The expression (6) can be used for PLC programming because it has an iterative form.

To development the user function block, shown below:



Inputs: X – input signal (REAL); T – the system time constant (TIME); k – the gain of the system (REAL) ; CYCLE – time of the task (TIME).

Outputs: Y – response of the system (REAL).

Programming on ST:

```
1    fCYC:=DINT_TO_REAL(TIME_TO_DINT(CYCLE))/1000.0;
2    Y:=prY+(-(1.0/T)*prY + k/T*prX)*fCYC;
3    prX:=X;
4    prY:=Y;
```

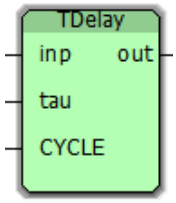
Variables table:

	Name	Type	Usage
	⊟ **Default**		
	X	REAL	VAR_INPUT
	T	REAL	VAR_INPUT
	k	REAL	VAR_INPUT
	CYCLE	TIME	VAR_INPUT
	Y	REAL	VAR_OUTPUT
	fCYC	REAL	VAR
	prX	REAL	VAR
	prY	REAL	VAR

The first string just converts TIME type to REAL type. The second string calculates the current iteration according to expression (5), first term. The third and fourth strings prepare the next iteration.

4. PC Worx programming of the time delay function

To development the user function block, shown below:



Inputs: inp – input signal (REAL); ***tau*** – time delay in seconds (TIME); CYCLE – time of the task (TIME).

Outputs: out – output signal, delayed relative to the input signal on the ***tau*** time (REAL).

ST programming:

```
1   IF CYCLE>=t#10ms and tau>=t#10ms THEN (*if tau or CYCLE is not too small*)
2
3    dtI:=DINT_TO_REAL(TIME_TO_DINT(CYCLE))/1000.0; (*converts time to int*)
4    tauI:=DINT_TO_REAL(TIME_TO_DINT(tau))/1000.0; (*converts time to int*)
5    n1:=REAL_TO_INT(tauI/dtI); (*n1 - length of memory we need*)
6
7    IF (n1<=49) THEN
8       (*shift the memory array to the right*)
9       FOR i1:=0 TO 48 DO
10          ind1:=50-i1;
11          ind2:=50-i1-1;
12          mas[ind1]:=mas[ind2];
13      END_FOR;
14      mas[1]:=inp; (*writes the new input value to first element*)
15      out:=mas[n1]; (*writes the n1 element to output*)
16   END_IF;
17
18  ELSE
19      out:=inp; (*if tau is too small*)
20  END_IF;
```

Variables table:

	Name	Type	Usage
	Default		
	inp	REAL	VAR_INPUT
	tau	TIME	VAR_INPUT
	CYCLE	TIME	VAR_INPUT
	out	REAL	VAR_OUTPUT
	n1	INT	VAR
	mas	memT	VAR
	dtl	REAL	VAR
	taul	REAL	VAR
	ind1	INT	VAR
	ind2	INT	VAR
	i1	INT	VAR

In this program we use the array type **memT** initialized in Data Types directory:

```
1    TYPE
2        memT : ARRAY[1..50] of REAL;
3    END_TYPE
```

This algorithm works like a FIFO memory (Fig.3). Every task cycle the array is shifted to the right. The number *n1* of array element to be connected to output pin is calculated with the input parameters of FB: CYCLE and *tau*.

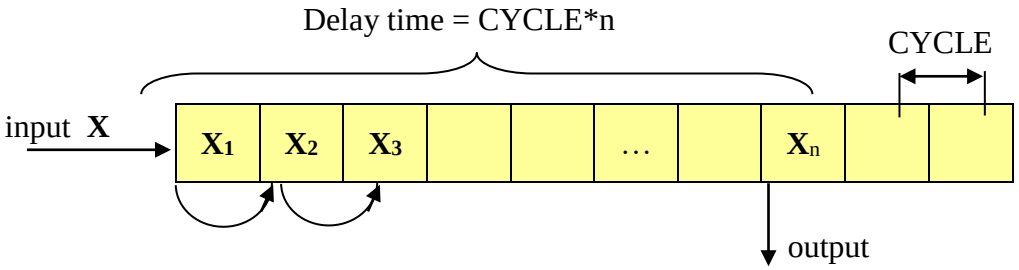


Fig. 3. The principle of Time Delay block working

This algorithm has disadvantage: the maximum possible delay time depends on the dimension of main array. For example, if it has 50 elements and the task interval is 50 ms the maximum possible time delay is 50*50 = 2500 ms.

Test of delay FB work:

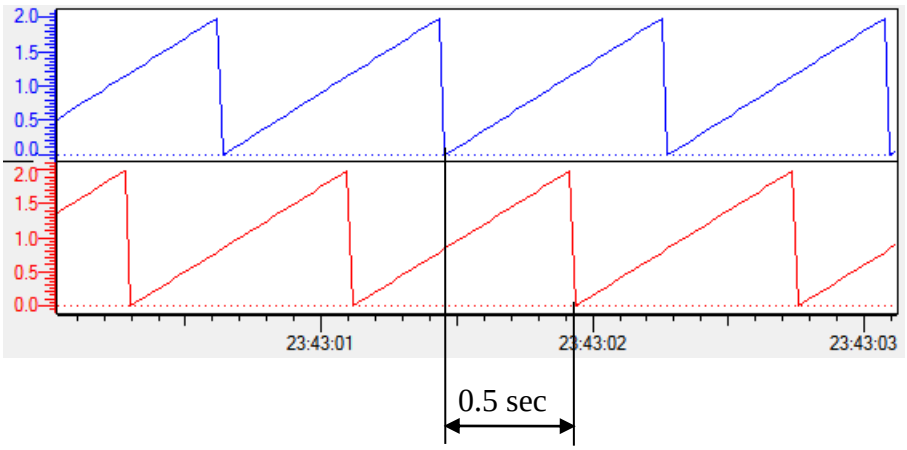


Fig. 4. Delay of the sawtooth signal

5. Closed-loop system in PC Worx

Now we have all function blocks to create the closed-loop system and to test the PID controller in action.

Let's take a plant with the following transfer function:

$$W(p) = \frac{2 \cdot e^{-0.5p}}{10 \cdot p + 1}$$

FBD programming of closed-loop system will be:

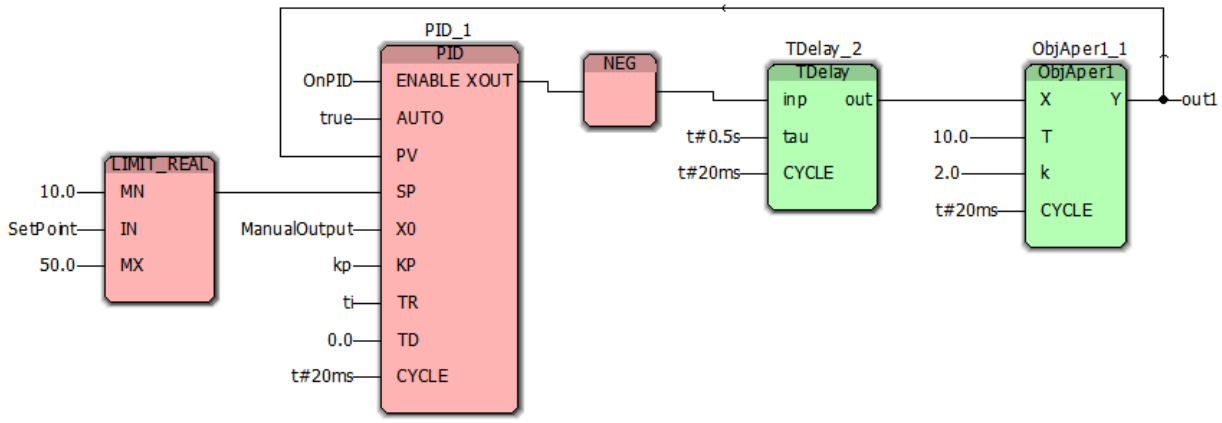


Fig. 5 – Closed-loop system with PID controller

We also need to calculate the parameters of PID controller. With the parameters KP=5, TI=10, TD=0.4 and setpoint=30 the closed-loop response is shown below:

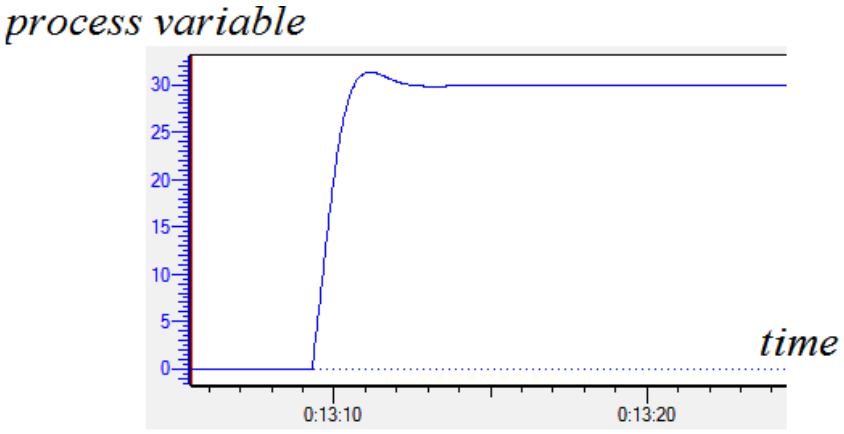


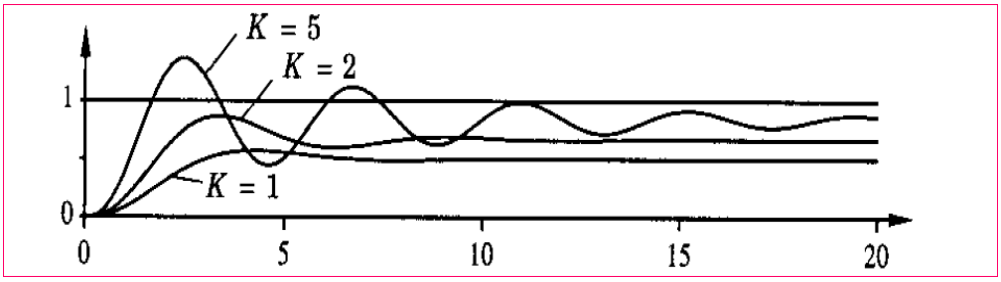
Fig. 6 – Closed-loop response

The task interval in this example is set to 20 ms. The CYCLE input of every block has the same value. For limitation of setpoint value the LIMIT_REAL function block is used.

Task

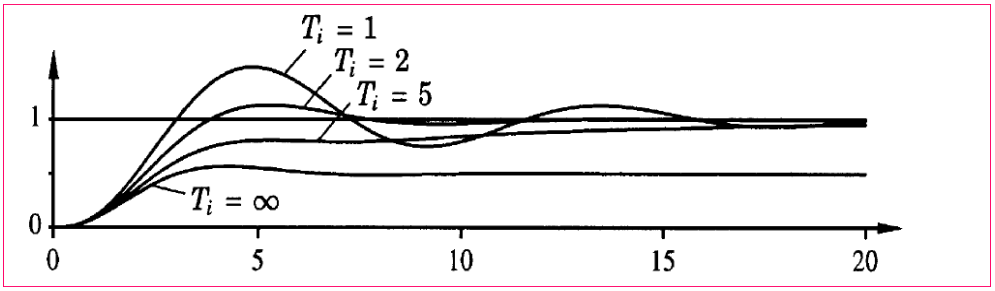
The behavior of PID controllers follows certain empirical rules:

By reducing the proportional band (equivalent to increasing the gain), the system responds more quickly, but has a larger envelope:



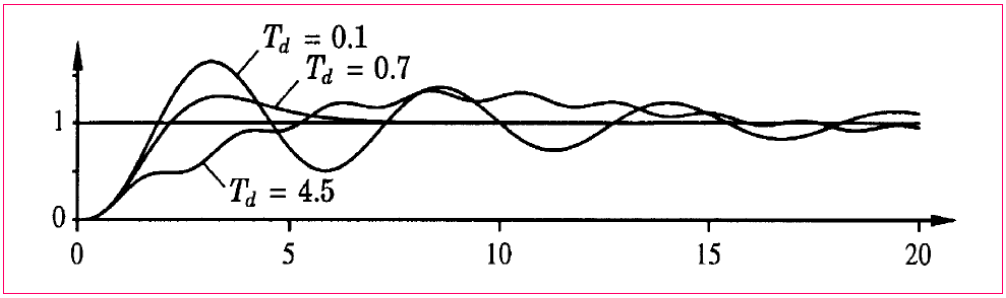
Simulated closed-loop system outputs with multiple proportional controller gain values (Kp).

By reducing the Reset time constant (TR), the system error rate drops more quickly, but the system becomes more unstable, with greater oscillations. if TR = ∞ corresponds to proportional control only.



Simulated closed-loop system outputs with PI controller and multiple full-time values. Proportional gain = 1.

By increasing the derivative time constant, the damping of the system increases to a certain point and then decreases again and the system becomes unstable.



Simulated closed loop system outputs with PID controller and several values of derivative time. Proportional gain = 3. Full time = 2.

Reproduce the graphics above in PC Worx Express. Remember to use Logic Analyzer.

6. References

1. PCWorx IEC61131-Programming, Phoenix Contact GmbH & Co, Blomberg, Germany.
2. Michel Van Dessel. Control of an industrial process using PID control blocks in automation controller. Lessius Mechelen Campus De Nayer, St.-Katelijne-Waver, Belgium.
3. http://en.wikipedia.org/wiki/PID_controller.