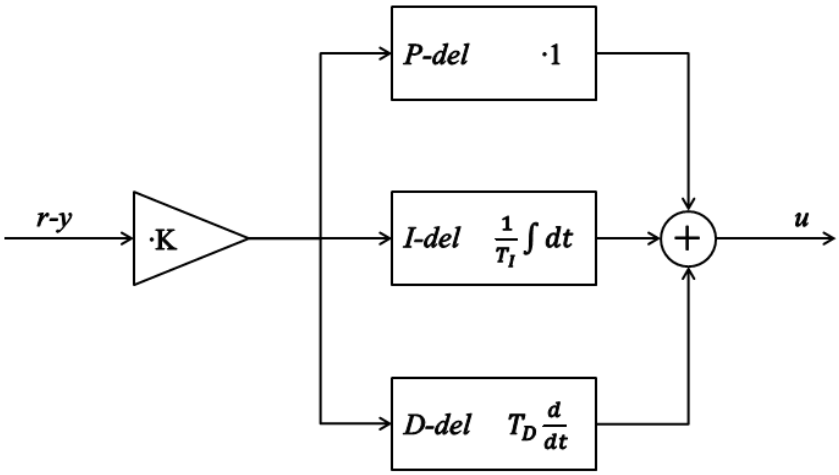
LTI PID Regulator

Frans Schreuder

April 16, 2020



Contents

Revision History

Revision	Date	Author(s)	Description
1.0	30-10-2014	F.P. Schreuder	created
1.1	31-10-2014	F.P. Schreuder	Added section about tuning

1 Introduction

To regulate the temperature with a microcontroller, the formula for a PID regulator must be converted to an LTI (Linear Time Invariant) system, because differentiating and integrating is difficult for a microcontroller. If the calculations are converted to the N domain via the Z domain with Tustin's formula, the PID code is a very short piece of code.

In this LTI system, the three constants K_p, K_i and K_d are defined and an extra constant T_s is introduced. T_s is the sample-time and the interval in which the function PID() is executed

2 Equations

The formula for a PID regulator is given below see [**?**]

$$H(t) = K_p \cdot X(t) + K_i \cdot \int X(t)dt + K_d \cdot \frac{dX(t)}{dt} \tag{1}$$

In the s domain, this function is given as

$$H(s) = K_p + K_i \cdot \frac{1}{s} + K_d \cdot s \tag{2}$$

Tustins formula let's us replace S with the following Z replacement

$$s = \frac{2}{T} \cdot \frac{1 - z^{-1}}{1 + z^{-1}} \tag{3}$$

The PID regulator in the Z domain is given as:

$$H(z) = 1 + \frac{1}{2 \cdot (1 - \frac{1}{z})} \cdot (1 + \frac{1}{z}) + 2 \cdot \frac{1 - \frac{1}{z}}{1 + \frac{1}{z}} \tag{4}$$

The Z function can be transferred into a discrete N domain function

$$Y_n = Y_{n-1} + K_1 \cdot K_n + K_2 \cdot X_{n-1} + K_3 \cdot X_{n-2} \tag{5}$$

The 3 constants K1, K2 and K3 can be calculated from Kp, Ki and Kd:

$$K_1 = K_p + \frac{T_s \cdot K_i}{2} + \frac{K_d}{T_s} \tag{6}$$

$$K_2 = -K_p - \frac{2 \cdot K_d}{T} + \frac{T \cdot K_i}{2} \tag{7}$$

$$K_3 = \frac{K_d}{T} \tag{8}$$

3 The PID controller in C code

In C-Code this PID regulator can be implemented as follows:

```
#include "pid.h"

//These 3 constants are calculated from Kp, Ki and Kd, defined in pid.h
//(See "PID_Regulator.pdf")
#define K1 ((Kp)+((Ts*Ki)/2.0)+(Kd/Ts))
#define K2 ((-1*Kp)-((2.0*Kd)/Ts)+((Ts*Ki)/2.0))
#define K3 (Kd/Ts)

//These constants represent Y[n], Y[n-1] etc.
//where:
//Xn[0] = X[n]
//Xn[1] = X[n-1]
//Xn[2] = X[n-2]
//Yn[0] = Y[n]
//Yn[1] = Y[n-1]
static float Xn[3]={0,0,0};
static float Yn[2]={0,0};

float PID (float setpoint,float actual)
{
  float pid_temp;
  //Shift the X[n] and Y[n] registers one position in the n domain
  Yn[1]=Yn[0];
  Xn[2]=Xn[1];
  Xn[1]=Xn[0];
  //X[n] is the actual error
  Xn[0]=setpoint-actual;
  //Calculate new PID, Y[n] is the current output
  Yn[0]=Yn[1]+(K1*Xn[0])+(K2*Xn[1])+(K3*Xn[2]);
  //limit the output value to 1023 because this is the maximum of
  //the PWM generator (10 bit)
  pid_temp=Yn[0];
  if(pid_temp>MAX_PWM)pid_temp=MAX_PWM;
  if(pid_temp<0)pid_temp=0;
  return pid_temp;
}

/*
 * This function must be called before starting the regulation,
 * if old values remain in the Xn and Yn arrays, unexpected things
 * could happen when starting regulating...
 **/

void reset_PID(void)
{
 Xn[0]=0;
 Xn[1]=0;
 Xn[2]=0;
 Yn[0]=0;
 Yn[1]=0;
}
```

Listing 1: PID Regulator source

```
/**********************************************************************
 * HOW TO TUNE THE PID REGULATOR                                      *
 *                                                                    *
 * -First set Ki and Kd to 0 and choose a (random) value for Kp       *
 * -Tune Kp the way that the regulator just regulates below           *
 *  the setpoint                                                      *
 * -Choose Ki so that there won't be too much overshoot               *
 * -Choose Kd so that the regulation will stay stable.                *
 **********************************************************************/

//proportional constant 180
#define Kp 180
//integral constant     4
#define Ki 4
//differential constant 0.02
#define Kd 0.02
//0.02
//sample time (seconds)
#define Ts 1.0

//This value is the limit for the return value of PID, set it to the
//maximum value of your PWM module, DAC or other actuator.
#define MAX_PWM 1023


/**********************************************************************
 * Call this function every Ts (eg using a timer) with the setpoint  *
 * and measured (actual) value. The function returns a value from     *
 * 0..MAX_PWM which can directly go into a PWM module or actuator     *
 **********************************************************************/

float PID (float setpoint,float actual);


/**********************************************************************
 * Resets the PID regulator by setting all internal shift registers   *
 * to 0                                                               *
 **********************************************************************/

void reset_PID(void);
```

Listing 2: PID Regulator header

4 Tuning the PID regulator

The 3 constants that the PID regualtor is depending on K_p, K_i and K_d are strongly dependent on the properties of the system you are regulating. A system where temperature is regulated on a massive object has a time constant of several minutes, while a voltage regulator can have a time constant of microseconds.

Before the three constants can be determined, T_s must be chosen. It must be much smaller than the time constant of the system, but big enough so that your processor is able to handle the timer accurately.

The section on tuning is obtained from an external website by Emile van de Logt. He has done a good job on explaining the different parameters of a PID controller. Additionally, you will learn the basics of home-brewing. [**?**] The contents of van de Logt's page are kept here for future reference.

Tuning the PID controller means finding values for Kc, Ti and Td, such that the PID controller responds quickly to changes in setpoint value and/or temperature, while minimising any overshoot in temperature. Tuning a PID controller is always necessary, even if you bought a ready-to-use PID controller. The advantage of such a ready-to-use PID controller is that they contain an auto-setup function, which finds the optimum values automatically for you (well, most of the time). But even such a ready-to-use PID controller uses the same algorithms as the ones I describe here. So this information might also be useful to you if you did not create your own PID controller, but bought one.

In order to understand the specific measurements which we are about to do, some terms and definitions have to be explained:

- **Open loop response**
  the open loop response means that the PID controller output is set to a fixed value (e.g. 20%) and the response of the system (in our case the HLT temperature) is measured. The term open loop comes from the fact that the PID controller is not in control, but set to a fixed value, the loop is not closed.

- **Closed loop response**
  the closed loop response means that the PID controller is enabled and that we apply a step to the setpoint value (the HLT temperature reference). Then we measure how the PID controller is doing by measuring the response of the system (the HLT temperature). For example: we could set the HLT reference temperature from 20 °C up to 50 °C, thereby creating a step of 30 °C.

- **Dead-Time (TD)**
  the dead-time of the system is the time delay between the initial response and a 10% increase in the process variable (which is the HLT temperature in our case). Unity is given in seconds.

- **Normalized slope (a)**
  the normalized slope of the step response is the increase in temperature per second per percentage of the PID controller output. Stated in a formula: a = change in temperature divided by time-interval divided by PID controller output.

- **Gain (K)**
  the gain of the HLT system. We can model our HLT system as having one input

(the PID controller output in %) and one output (the HLT temperature). The Gain of this HLT system is then defined as output divided by input and has a unity of °C / %.

- **Time-constant (tau)**
  the time-constant of the system is another important parameter of the system. It describes how quick the temperature will increase, given a particular output of the PID controller. Unity is given in seconds.

4.1 Determine Dead-Time and normalized slope of the HLT system

Manually set the PID-controller output to a certain value (e.g. 20 %). In case of a heavy load, 100 % is recommended (more accurate), but if you do not know the performance of the system, a lower value to start with is better. The temperature starts to increase and follows the following curve:

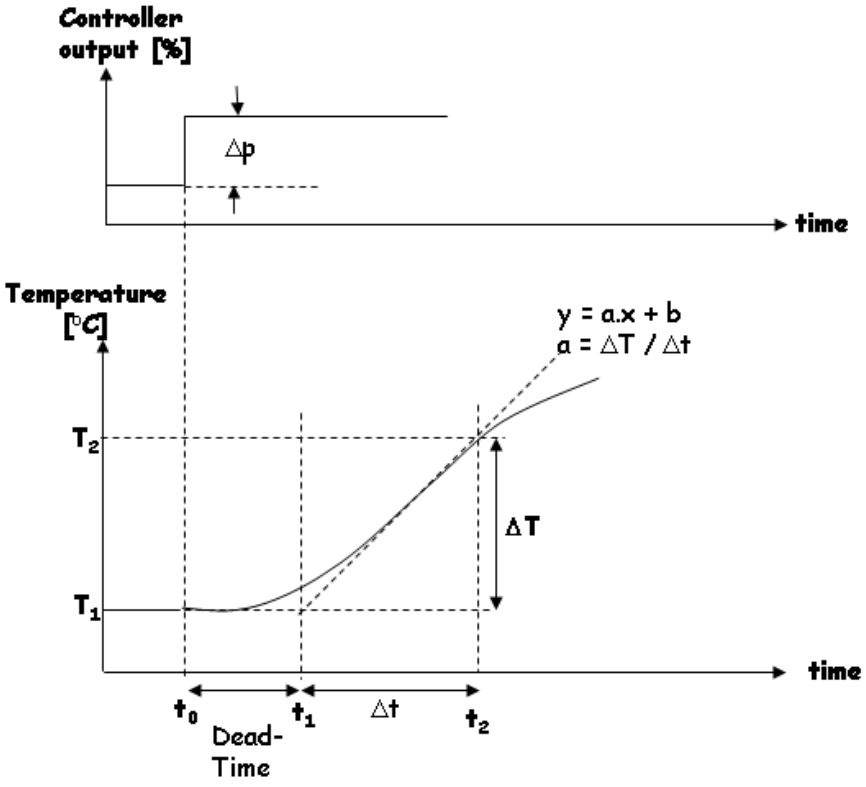


Figure 1: Determining the dead time of the system

After doing this experiment with my HLT system (90 L of water, no lid on the pan, controller output set to 100 %, heating element only), combined with some regression analysis (for more accuracy), I found out that the dead-time for my HLT system is equal to 115 seconds.

During the experiment, the temperature change was 0.4 °C per minute when the PID controller output was set to 100 %. The normalized slope a is therefore equal to 0.004 °C

/ (%.minute) or 6.68E-5 °C/(%.s). BUT... since I switched over to gas-burners, and more recently, to a new HLT kettle, I may have to do this experiment again!.

4.2 Determine Time-Constant of the HLT system

Because of the large time-constant presumably present in the HLT system, the previous experiment is not very accurate in determining the dead-time as well. This is the mean reason to conduct two experiments (theoretically, the step-response would give you all the required information to determine the three parameters).

Manually set the PID-controller output to a certain value (e.g. 20 %). The temperature starts to increase and follows the following curve:

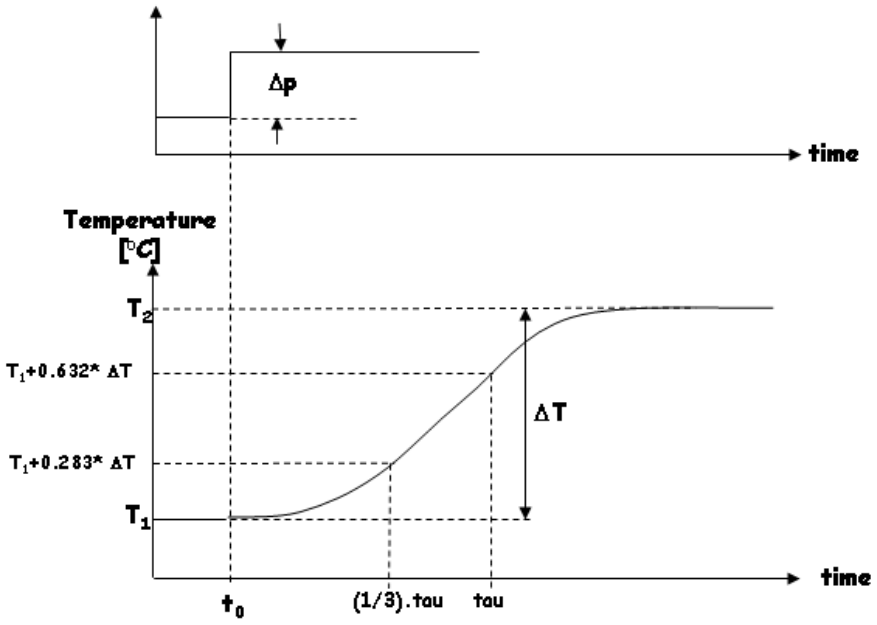


Figure 2: Determining the time constant of the system

The difference with the previous experiment is that we let the PID controller settle, meaning that the HLT temperature will converge to a new value (in my experiment, this took a complete day, because I wanted it as reliable as possible).

After doing this experiment with my HLT system (90 L of water, no lid on the pan, controller output set to 20 %, heating element only), I collected the following data:

- $T_0 = 09:49:04, T_1 = 19.20°C$
- $T_2 = 52.99°C, T_2 - T_1 = 33.79°C$
- $T_1 + 0.283 \cdot (T_2 - T_1) = 28.76°C$. Check the HLT temperature graph when this temperature was reached. In the experiment, this temperature was reached 6599 seconds after the start (T_0). Therefore: $\frac{\tau}{3} = 6599 seconds$.
- $T_1 + 0.632 * (T2 - T1) = 40.56°C$. Check the HLT temperature graph when this temperature was reached. In the experiment, this temperature was reached 16573 seconds after the start (t0). Therefore: $\tau = 16573 seconds$.

- Solving tau from the previous two results: $\tau - \frac{\tau}{3} = 16573 - 6599 \iff \frac{2}{3} \cdot \tau = 9974 \iff \tau = 14961 seconds$.
- The Gain is calculated directly from this data as well: $Gain = \frac{T_2 - T_1}{20\%} = \frac{33.79°C}{20\%} = 1.69°C/\%$.

4.3 Calculate the Optimum values for the PID controller (K_c, T_i and T_d)

A summary of the results from the two experiments for my particular HLT system:

- Dead-Time $T_D = 115 seconds$
- Normalized slope a $= 6.68E - 5°C/(\%.s)$.
- Gain $(K) = 1.69°C/\%$.
- Time-constant $\tau = 14961 seconds$.

There are a few well-known algorithms that calculate the optimum settings for a PI and a PID controller. These algorithms calculate a value for K_c, T_i and T_d based upon the values found for T_D, a, K and τ. The algorithms are:

- Ziegler-Nichols Open Loop
- Ziegler-Nichols Closed Loop
- Cohen-Coon
- Integral of Time weighted Absolute Error (ITAE-Load): this algorithm normally gives the best results. The error signal is minimized over time.

The detailed formulas for every algorithm are as follows:

$K_c[\%/(°C)]$	$Ti[sec.]$	$Td[sec.]$
$\frac{1.2}{TD*a}$	$2.0 \cdot TD$	$0.5 \cdot TD$

Table 1: Ziegler-Nichols Open Loop, PID

$K_c[\%/(°C)]$	$Ti[sec.]$
$\frac{0.9}{TD*a}$	$3.33 \cdot TD$

Table 2: Ziegler-Nichols Open Loop, PI, no D term

$K_c[\%/(°C)]$	$Ti[sec.]$	$Td[sec.]$
$\frac{1.2}{TD*K}$	$2.0 \cdot TD$	$0.5 \cdot TD$

Table 3: Ziegler-Nichols Closed Loop, PID

$K_c[\%/(^\circ C)]$	$Ti[sec.]$
$\frac{0.9}{TD*K}$	$3.33 \cdot TD$

Table 4: Ziegler-Nichols Closed Loop, PI, no D term

$K_c[\%/(^\circ C)]$	$Ti[sec.]$	$Td[sec.]$
$(\frac{\tau}{K \cdot T_D}) \cdot (\frac{T_D}{4 \cdot \tau} + \frac{4}{3})$	$T_D \cdot \frac{32 \cdot \tau + 6 \cdot T_D}{13 \cdot \tau + 8 \cdot T_D}$	$\frac{4 \cdot T_D \cdot \tau}{2 \cdot T_D + 11 \cdot \tau}$

Table 5: Cohen-Coon, PID

$K_c[\%/(^\circ C)]$	$Ti[sec.]$
$\frac{\tau}{K \cdot T_D} \cdot (\frac{T_D}{12 \cdot \tau} + \frac{9}{10})$	$T_D \cdot \frac{30 \cdot \tau + 3 \cdot T_D}{9 \cdot \tau + 20 \cdot T_D}$

Table 6: Cohen-Coon, PI, no D term

$K_c[\%/(^\circ C)]$	$Ti[sec.]$	$Td[sec.]$
$\frac{1.357}{K} \cdot \frac{T_D}{\tau}^{-0.947}$	$\frac{\tau}{0.842} \cdot \frac{T_D}{\tau}^{0.738}$	$0.381 \cdot \tau \cdot \frac{T_D}{\tau}^{0.995}$

Table 7: Integral of Time weighted Absolute Error (ITAE-Load): this algorithm normally gives the best results. The error signal is minimized over time

$K_c[\%/(^\circ C)]$	$Ti[sec.]$
$\frac{0.859}{K} \cdot \frac{T_D}{\tau}^{-0.977}$	$\frac{\tau}{0.674} \cdot \frac{T_D}{\tau}^{0.680}$

Table 8: Integral of Time weighted Absolute Error (ITAE-Load), PI, no D term

When these formulas are applied to the values found for my HLT system, the following results were obtained:

Algorithm	$Kc[\%/(^\circ C)]$	$Ti[sec.]$	$Td[sec.]$
Ziegler-Nichols Open Loop	156.2 117.2	230.0 383.0	57.5
Ziegler-Nichols Closed Loop	92.4 69.3	230.0 383.0	57.5
Cohen-Coon	102.8 69.4	282.3 377.2	41.8
ITAE-Load	80.8 59.2	489.0 810.2	44.9

Table 9: Numeric values for different algorithms

For further results, tests and more information see [**?**]

References

[1] Good reference on PID regulators and Tuning
Emile's Home-Brewing Site: http://www.vandelogt.nl/htm/regelen_pid_uk.htm

[2] Mathworks explanation of LTI systems and Tustins approximation
http://www.mathworks.nl/help/control/ug/continuous-discrete-conversion-methods.html?nocookie=true