AVR 8-bit Microcontrollers

AVR221: Discrete PID Controller on tinyAVR and megaAVR devices

APPLICATION NOTE

Introduction

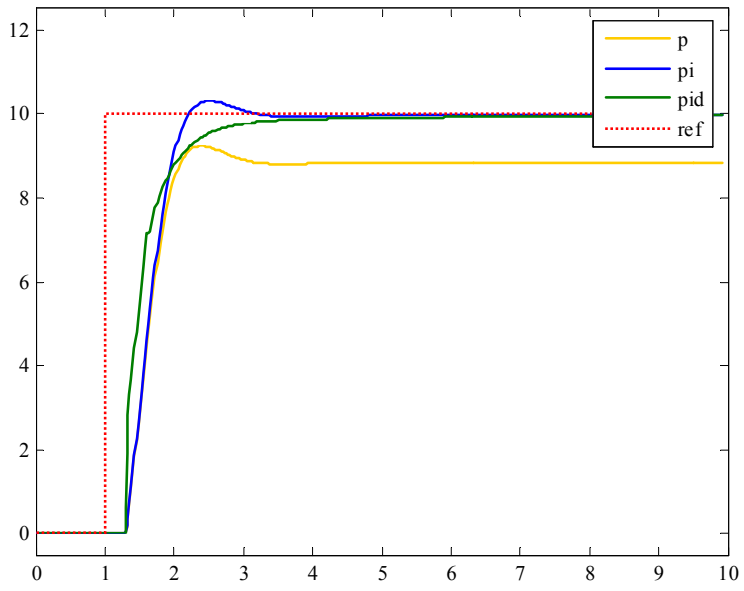
This application note describes a simple implementation of a discrete Proportional- Integral-Derivative (PID) controller.

When working with applications where control of the system output due to changes in the reference value or state is needed, implementation of a control algorithm may be necessary. Examples of such applications are motor control, control of temperature, pressure, flow rate, speed, force, or other variables. The PID controller can be used to control any measurable variable, as long as this variable can be affected by manipulating some other process variables.

Many control solutions have been used over the time, but the PID controller has become the ‘industry standard’ due to its simplicity and good performance.

For further information about the PID controller and its implications, the reader should consult other sources, e.g. PID Controllers by K. J. Astrom & T. Hagglund (1995).

Figure -1. Typical PID Regulator Response to Step Change in Reference Input



Features

- Simple discrete PID controller algorithm
- Supported by all Atmel® AVR® devices

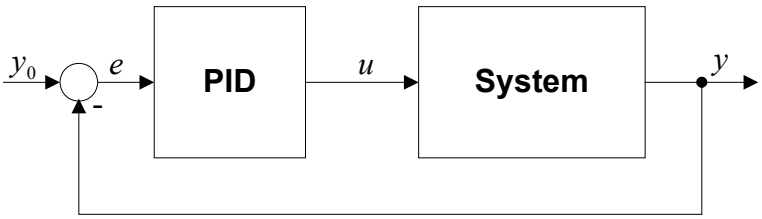
Table of Contents

1. PID Controller

In the figure below a schematic of a system with a *PID* controller is shown. The *PID* controller compares the measured process value *y* with a reference setpoint value, y_0. The difference or error, *e*, is then processed to calculate a new process input, *u*. This input will try to adjust the measured process value back to the desired setpoint.

The alternative to a closed loop control scheme such as the PID controller is an open loop controller. Open loop control (no feedback) is in many cases not satisfactory, and is often impossible due to the system properties. By adding feedback from the system output, performance can be improved.

Figure 1-1. Closed Loop System with *PID* Controller

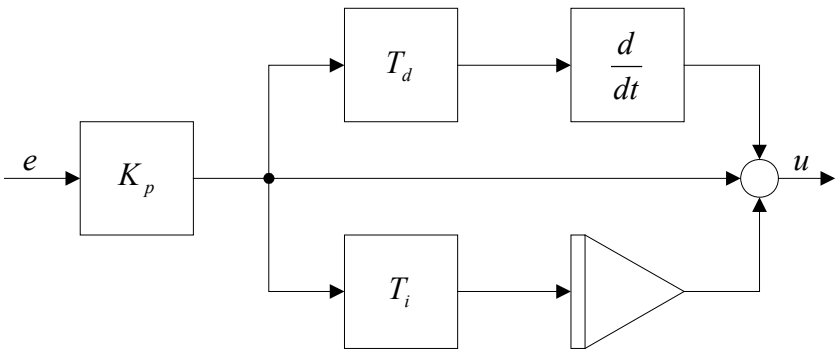


Unlike simple control algorithms, the *PID* controller is capable of manipulating the process inputs based on the history and rate of change of the signal. This gives a more accurate and stable control method.

The basic idea is that the controller reads the system state by a sensor. Then it subtracts the measurement from a desired reference to generate the error value. The error will be managed in three ways; to handle the present through the proportional term, recover from the past using the integral term, and to anticipate the future through the derivate term.

The figure below shows the *PID* controller schematics, where T_p, T_i, and T_d denote the time constants of the proportional, integral, and derivative terms respectively.

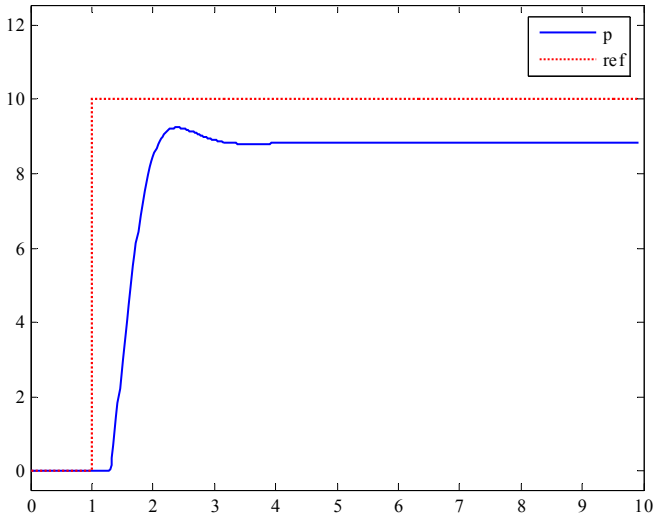
Figure 1-2. PID Controller Schematic



1.1. Proportional Term

The proportional term (*P*) gives a system control input proportional with the error. Using only *P* control gives a stationary error in all cases except when the system control input is zero and the system process value equals the desired value. In the figure below the stationary error in the system process value appears after a change in the desired value (*ref*). Using a too large P term gives an unstable system.

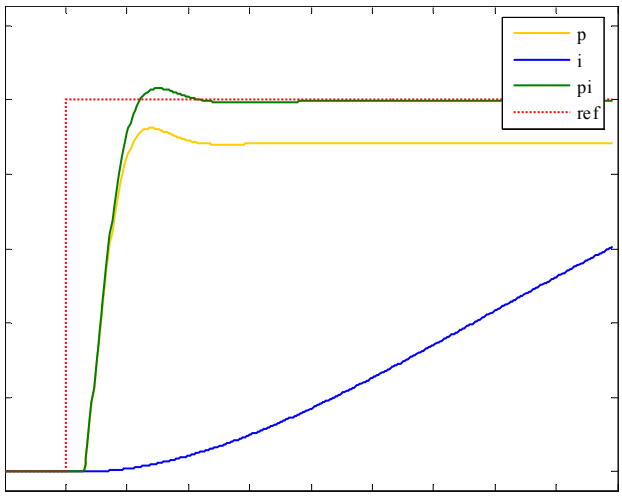
Figure 1-3. Step Response P Controller



1.2. Integral Term

The integral term (*I*) gives an addition from the sum of the previous errors to the system control input. The summing of the error will continue until the system process value equals the desired value, and this results in no stationary error when the reference is stable. The most common use of the *I* term is normally together with the *P* term, called a *PI* controller. Using only the *I* term gives slow response and often an oscillating system. The figure below shows the step responses to a *I* and *PI* controller. As seen the *PI* controller response have no stationary error and the *I* controller response is very slow.

Figure 1-4. Step Response *I* and *PI* Controller

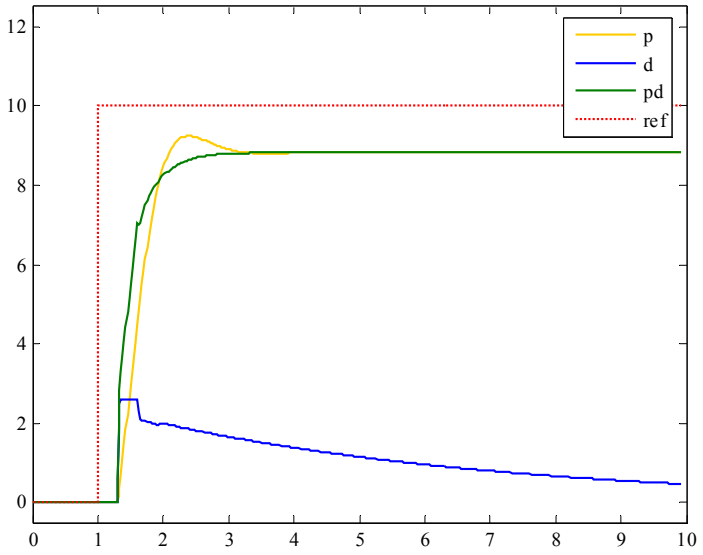


1.3. Derivative Term

The derivative term (*D*) gives an addition from the rate of change in the error to the system control input. A rapid change in the error will give an addition to the system control input. This improves the response to a sudden change in the system state or reference value. The *D* term is typically used with the *P* or *PI* as a *PD* or *PID* controller. A too large *D* term usually gives an unstable system. The figure below shows *D* and *PD* controller responses. The response of the *PD* controller gives a faster rising system process value

than the *P* controller. Note that the *D* term essentially behaves as a high-pass filter on the error signal and thus easily introduces instability in a system and make it more sensitive to noise.

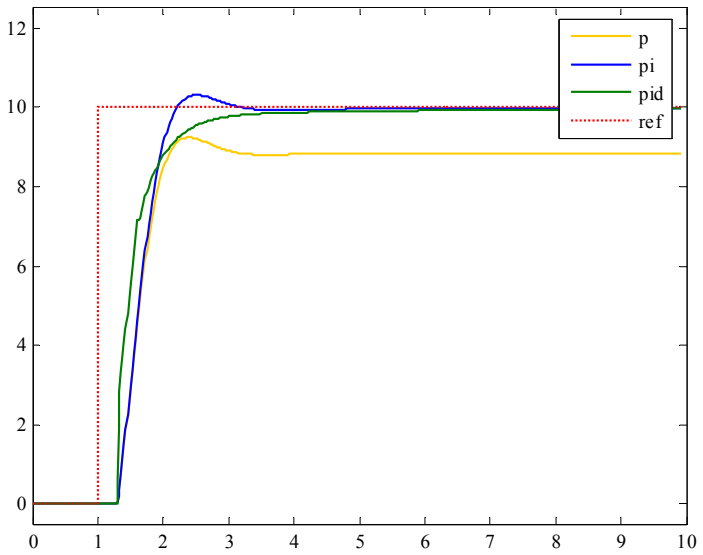
Figure 1-5. Step Response *D* and *PD* Controller



1.4. Proportional, Integral, and Derivative Terms Together

Using all the terms together, as a *PID* controller usually gives the best performance. The figure below compares the *P*, *PI*, and *PID* controllers. *PI* improves the *P* by removing the stationary error, and the *PID* improves the *PI* by faster response and no overshoot.

Figure 1-6. Step Response *P*, *PI*, and *PID* Controller



1.5. Tuning the Parameters

The best way to find the needed *PID* parameters is from a mathematical model of the system, parameters can then be calculated to get the desired response. Often a detailed mathematical description of the system is unavailable, experimental tuning of the *PID* parameters has to be performed. Finding the terms for the *PID* controller can be a challenging task. Good knowledge about the systems properties and the

way the different terms work is essential. The optimum behavior on a process change or setpoint change depends on the application at hand. Some processes must not allow overshoot of the process variable from the setpoint. Other processes must minimize the energy consumption in reaching the setpoint. Generally, stability is the strongest requirement. The process must not oscillate for any combinations or setpoints. Furthermore, the stabilizing effect must appear within certain time limits.

Several methods for tuning the *PID* loop exist. The choice of method will depend largely on whether the process can be taken off-line for tuning or not. Ziegler-Nichols method is a well-known online tuning strategy. The first step in this method is setting the *I* and *D* gains to zero, increasing the *P* gain until a sustained and stable oscillation (as close as possible) is obtained on the output. Then the critical gain K_c and the oscillation period P_c is recorded and the *P*, *I*, and *D* values adjusted accordingly using the table below.

Table 1-1. Ziegler-Nichols Parameters

Controller	Kp	Ti	Td
P	0.5 * Kc		
PD	0.65 * Kc		0.12 * Pc
PI	0.45 * Kc	0.85 * Pc	
PID	0.65 * Kc	0.5 * Pc	0.12 * Pc

Further tuning of the parameters is often necessary to optimize the performance of the *PID* controller.

The reader should note that there are systems where the *PID* controller will not work very well, or will only work on a small area around a given system state. Non-linear systems can be such, but generally problems often arise with *PID* control when systems are unstable and the effect of the input depends on the system state.

1.6. Discrete PID Controller

A discrete *PID* controller will read the error, calculate and output the control input at a given time interval, at the sample period *T*. The sample time should be less than the shortest time constant in the system.

1.6.1. Algorithm Background

Unlike simple control algorithms, the *PID* controller is capable of manipulating the process inputs based on the history and rate of change of the signal. This gives a more accurate and stable control method.

Figure 1-2 shows the *PID* controller schematics, where *Tp, Ti*, and *Td* denotes the time constants of the proportional, integral, and derivative terms respectively. The transfer function of this system is:

$$\frac{u}{e}\left(s\right) = H\left(s\right) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right)$$

This gives *u* with respect to *e* in the time domain:

$$u\left(t\right) = K_p\left(e\left(t\right) + \frac{1}{T_i}\int_0^t e(\sigma)\,d\sigma + T_d\frac{de(t)}{dt}\right)$$

Approximating the integral and the derivative terms to get the discrete form, using:

$$\int_0^t e(\sigma)\,d\sigma \approx T\sum_{k=0}^{n} e(k) \qquad \frac{de(t)}{dt} \approx \frac{e(n) - e(n-1)}{T} \qquad t = nT$$

Where *n* is the discrete step at time *t*.

This gives the controller:

$$u\left(n\right) = K_p e\left(n\right) + K_i \sum_{k=0}^{n} e(k) + K_d\left(e\left(n\right) - e\left(n-1\right)\right)$$

Where:

$$K_i = \frac{K_p T}{T_i} \qquad K_d = \frac{K_p T_d}{T}$$

To avoid that changes in the desired process value makes any unwanted rapid changes in the control input, the controller is improved by basing the derivative term on the process value only:

$$u\left(n\right) = K_p e\left(n\right) + K_i \sum_{k=0}^{n} e(k) + K_d\left(y\left(n\right) - y\left(n-1\right)\right)$$

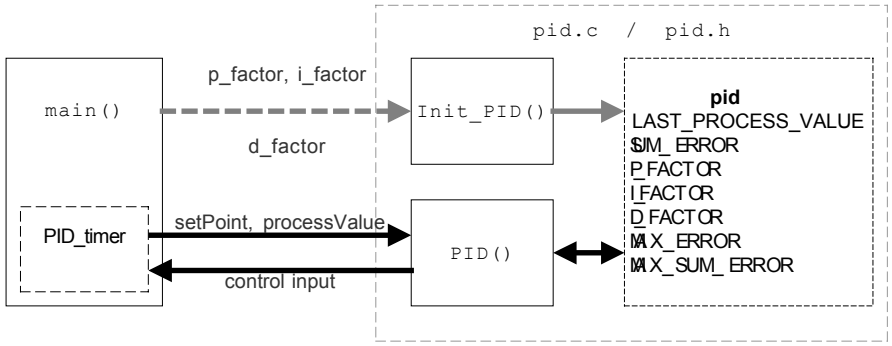
2. Implementation

The example code is written for "Atmel START". It can be downloaded from "BROWSE EXAMPLES" entry of Atmel START for both Atmel Studio 7 and IAR™ IDE.

Double click the downloaded .atzip file and the project will be imported to Atmel Studio 7.

To import the project in IAR, refer to "Atmel START in IAR" , select 'Atmel Start Output in External Tools → IAR'.

Figure 2-1. Block Diagram of Demo Application



In the figure above a simplified block diagram of the demo application is shown.

The *PID* controller uses a struct to store its status and parameters. This struct is initialized in main, and only a pointer to it is passed to the Init_PID() and PID() functions.

The PID() function must be called for each time interval *T*. This is done by a timer, which sets the PID_timer flag when the time interval has passed. When the PID_timer flag is set the main routine reads the desired process value (setPoint) and system process value, calls PID(), and outputs the result to the control input.

To increase the accuracy the p_factor, i_factor, and d_factor are scaled with a factor 1:128. The result of the *PID* algorithm is later scaled back by dividing by 128. The value 128 is used to allow for optimizing in the compiler.

$$PFactor = 128K_p$$

Furthermore the effect of the IFactor and DFactor will depend on the sample time *T*.

$$IFactor = 128K_p\frac{T}{T_i}$$

$$DFactor = 128K_p\frac{T_d}{T}$$

2.1. Integral Windup

When the process input, *u*, reaches a high enough value, it is limited in some way. Either by the numeric range internally in the PID controller, the output range of the controller, or constraints in amplifiers or the process itself. This will happen if there is a large enough difference in the measured process value and the reference set point value, typically because the process has a larger disturbance/load than the system is capable of handling.

If the controller uses an integral term, this situation can be a problematic. The integral term will sum up as long as the situation last, and when the larger disturbance/load disappear, the PID controller will overcompensate the process input until the integral sum is back to normal.

This problem can be avoided in several ways. In this implementation the maximum integral sum is limited by not allowing it to become larger than MAX_I_TERM. The correct size of the MAX_I_TERM will depend on the system and sample time used.

3. Further Development

The *PID* controller presented here is a simplified example. The controller should work fine, but it might be necessary to make the controller even more robust (limit runaway/overflow) in certain applications. Adding saturation correction on the integral term, basing the proportional term on only the system process value can be necessary.

In the calculating of IFactor and DFactor the sample time *T* is a part of the equation. If the sample time *T* used is much smaller or larger than 1 second, the accuracy for either IFactor or DFactor will be poor. Consider rewriting the *PID* algorithm and scaling so accuracy for the integral and derivative terms are kept.

4. Literature References

K. J. Astrom & T. Hagglund, 1995: *PID Controllers: Theory, Design, and Tuning*. International Society for Measurement and Con.

5. Revision History

Doc Rev.	Date	Comments
2558C	09/2016	Chapter "Features" has been updated. Chapter "Implementation" has been updated by adding the relation to Atmel START.
2558B	08/2016	New template and some minor changes
2558A	05/2006	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**