David Lee

Spring 2017

PID Control.

Introduction:

PID (proportional. Integral, derivative) controllers are amongst the most common types of process controllers in industrial applications. It is possible to use an Arduino as a PID controller.

Background:

PID controllers, as their name implies, consist of three parts: proportional, integral, and derivative. We'll start with the proportional portion.

A proportional controller varies its output based on the error from the set point.

$$P_{output} \propto Error$$

This means that a larger error will elicit a larger response from the controller. Next, the integral control portion is proportional to the sum of the error over time.

$$I_{output} \propto \int Error\ dt$$

This means that the response of the controller increases over time for persistent errors. Integral controllers can susceptible to what is known as "integral windup" which occurs when a system accrues a large error that takes a large amount of time to work its way out in the controller. This can occur when large set point changes are passed to a controller. There are several ways to deal with this but the simplest is to simply clear the integral periodically. Finally, the derivative controller is proportional to the rate of change of the error.

$$D_{output} = \propto \frac{d}{dt} Error$$

Tuning of the parameters for the proportional, integral, and derivative components is often considered something of an artform. Wikipedia provides a good table with some rules of thumb for adjusting each constant.

Effects of *increasing* a parameter independently[22] [23]

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect in theory	Improve if K_d small

Source: https://en.wikipedia.org/wiki/PID_controller

Library:

This library makes implementing PID control in Arduino relatively simple.

https://playground.arduino.cc/Code/PIDLibrary

To create a new PID controller, the procedure is as follows:

$$PID\ myPID(\&Input, \&Output, \&Setpoint, Kp, Ki, Kd, direction);$$

Where &Input, &Output, &Setpoint are previously defined variables for the input, output, and set point.

Kp, Ki, and Kd are the proportional, integral and derivative control constants respectively. Direction determines how the output variable changes. 'DIRECT' output makes the output variable increase with increasing error. This is the most commonly used mode.

The input variable can be the reading from a sensor (ie fan speed or temperature.)

The output variable is a double that ranges from 0 to 100. How this is interpreted is up to the user. For analog write, one can convert to a range of 0 to 255 as follows:

$$round(255/output);$$

To update the output variable, use the compute function.

$$myPID.Compute();$$

Other notes:

- A special case for PID controllers that may be useful is to nest several controllers. Have the output of one controller be the input of another. This is useful when there are multiple process variables with very different time constants. An example of this would be controlling temperature with a fan. The fan has a very fast response time while the temperature is much slower. Thus we would have one controller monitor the temperature and change the set point of the second controller. This controller would control the fan. In many circumstances, this provides for vastly improved system stability.