PID analysis

Author: Yu Weifu

Date: 14/09/2017

Last Modified: 03/10/2017

Discretization methods

- Euler's forward differentiation method,

In time domain, $\dot{x}_k \approx \frac{x_{k+1} - x_k}{T_s}$

In frequency domain, replace with, $s = \frac{x_{k+1} - x_k}{T_s}$

- Euler's backward differentiation method,

In time domain, $\dot{x}_k \approx \frac{x_k - x_{k-1}}{T_s}$

- Zero Order Hold (ZOH) method
- Tustin's method (bilinear transform method)
- Tustin's method with frequency prewarping, or bilinear transformation

PID controller in time domain

The PID equation in time domain is,

$$u(t) = K_p \left[e(t) + \frac{1}{T_i} \int_0^t e(t)dt + T_d \dot{e}(t) \right]$$

Where $e(t) = r(t) - y(t)$, is the error signal.

And $K_i = \frac{K_p}{T_i}$, $K_d = K_p T_d$.

Differentiating above equation gives,

$$\dot{u}(t) = K_p \left[\dot{e}(t) + \frac{1}{T_i} e(t)dt + T_d \ddot{e}(t) \right]$$

Discretization of PI controller with Backward differentiation method

Treat PI controller as a whole, and applying the Backward differentiation method to the PI controller,

$$\frac{u_k - u_{k-1}}{T_s} = K_p \left[\frac{e_k - e_{k-1}}{T_s} + \frac{e_k}{T_i} \right]$$

solving u_k,

$$u_k = u_{k-1} + K_p\left(e_k - e_{k-1}\right) + \frac{K_p T_s}{T_i} e_k = K_p e_k + u_{k-1} - K_p e_{k-1} + \frac{K_p T_s}{T_i} e_k$$

Code implementation:

Define integral item as,

$$u_{ik} = u_{k-1} - K_p e_{k-1} + \frac{K_p T_s}{T_i} e_k$$

In pseudo-code, the non-incremental form is,

```
func_pi() {
ui += Ki * Ts * err;
u = Kp * err + ui;
ui = ui - Kp * err;
}
```

The step change is,

$$\Delta u_k = u_k - u_{k-1} = K_p e_k - K_p e_{k-1} + \frac{K_p T_s}{T_i} e_k$$

Define the incremental `integral` components:

$$\Delta u_{ik} = -K_p e_{k-1} + \frac{K_p T_s}{T_i} e_k$$

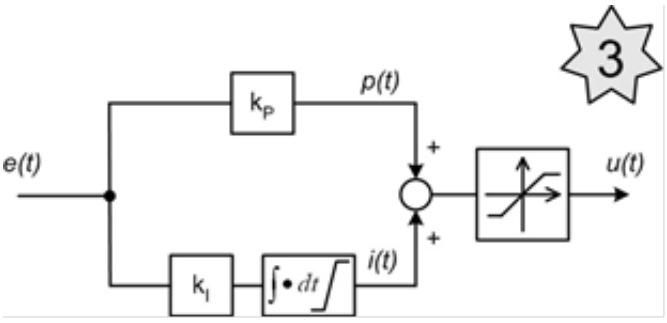
The incremental form is,

$$\Delta u_{ik} = -K_p e_{k-1} + \frac{K_p T_s}{T_i} e_k$$

$$\Delta u_k = K_p e_k + \Delta u_{ik}$$

$$u_k = u_{k-1} + \Delta u_k$$

It's easy to impment the PI controller anti-windup with this incremental form.



The above form of anti-windup is implemented,

1. Rate limitation applied to Δu_{ik}, which limits the integration speed.
2. Magnitude limitation applied to u_k, which limits the output.

The incremental form that suitable for the anti-windup limitation is,

```
func_pi() {
delta_ui += Ki * Ts * err;
<rate limit here>;
u += Kp * err + delta_ui;
<magnitude limit here>;
delta_ui = -Kp * err;
}
```

This incremental form implementation is proposed.

Seperately differentiation of proportional item and integral item

If the PID equation is broke into proportional and integral component,

$$u_p(t) = K_p e(t)$$

$$u_i(t) = \frac{K_p}{T_i}\int_0^t e(t)dt$$

$$u(t) = u_p(t) + u_i(t)$$

Seperately discretize the proportional and integral component, we have

$$u_{pk} = K_p e_k$$

$$\frac{u_{ik} - u_{i(k-1)}}{T_s} = \frac{K_p}{T_i} e_k$$

solving u_{ik},

$$u_{ik} = u_{i(k-1)} + \frac{K_p T_s}{T_i} e_k$$

solving u_k,

$$u_k = u_{pk} + u_{ik} = K_p e_k + u_{i(k-1)} + \frac{K_p T_s}{T_i} e_k$$

In pseudo-code,

```
func_pi() {
delta_ui = Ki * Ts * err;
<rate limit here>;
ui += delta_ui;
u = Kp * err + ui;
<magnitude limit here>;
}
```