Function Approximation and Adaptive PID Control using Neural Networks

PID

The strategy of *PID* control has been one of sophisticated and most frequently used methods in the industry. By taking the time-derivative and discretizing the resulting equation and of the both sides of the continuous-time *PID* equation, a typical discrete-time *PID* controller can be expressed as:

$$u(k) = u(k-1) + K_p\ (\mathrm{e(k)\text{-}e(k\text{-}1)}) + \frac{K_I T}{2}\ (\mathrm{e(k)+e(k\text{-}1)})\ + \frac{K_D}{2}(\mathrm{e(k)\text{-}2e(k\text{-}1)+e(k\text{-}2)}) \qquad (1)$$

where *u(k)* is the control effort at time *k*, K_P, K_I and K_D are proportional, integral and derivative gains, respectively. *T* is the sampling period. *e(k)* is the tracking error defined as $e(k)=y_d(k)-y(k)$; $y_d(k)$ is the desired plant output, *y(k)* is the actual plant output.

$$\Delta u(k) = u(k) - u(k-1) = K_p\ e_p(k) + K_I\ e_I(k) + K_D\ e_D(k) \qquad (2)$$

Where

$$\begin{aligned} e_p(k) &= e(k) - e(k-1) \\ e_I(k) &= \frac{T}{2}\big(e(k) + e(k-1)\big) \\ e_D(k) &= \frac{1}{T}\big(e(k) - 2\,e(k-1) + e(k-2)\big) \end{aligned} \qquad (3)$$

Control system architecture

The structure of the newly proposed control algorithm of nonlinear *PID*-based neural networks is shown in Fig. 1. The *PID* is placed in cascade with the plant neural network model. The error signal obtained as the difference between the plant model and desired plant output is used to adjust only the *PID* parameters. In the proposed adaptive control scheme, the model plant weights are adjusted of-line then the fixed weights will be used only to adjust on-line the gains of the *PID* controller. The presented control algorithm has simple structure and provides a gain in computation time.

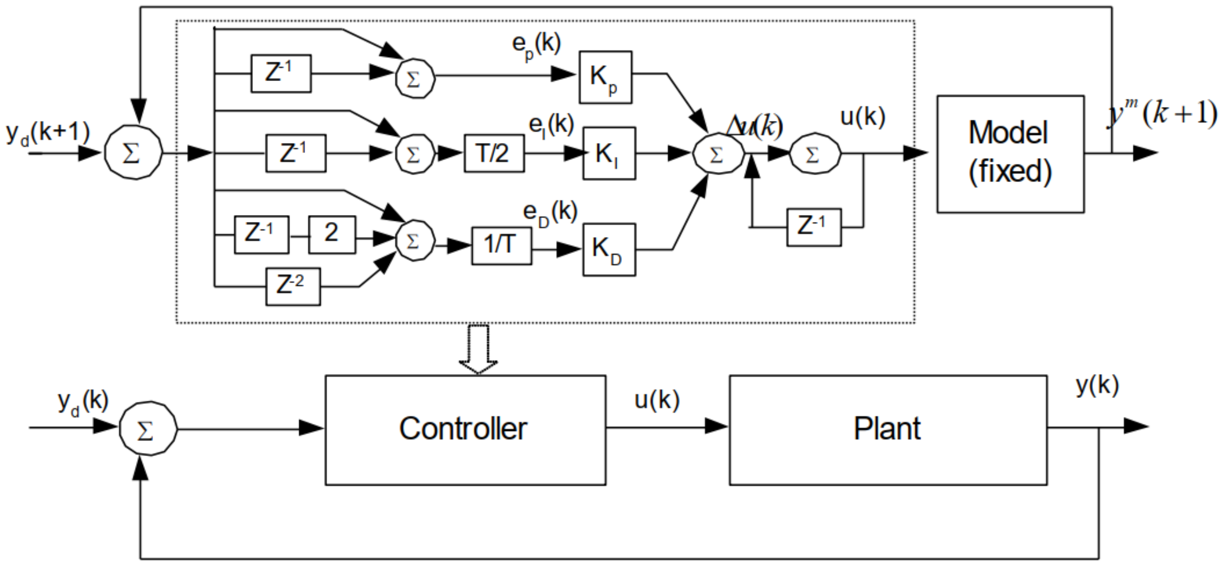

Fig. 1. Proposed *PID* control system architecture.

To get the neural network model plant, a feedforward neural network is used to learn the system and back-propagation algorithm is employed to train the weights. The block diagram of identification system is shown in Fig. 2 [16].

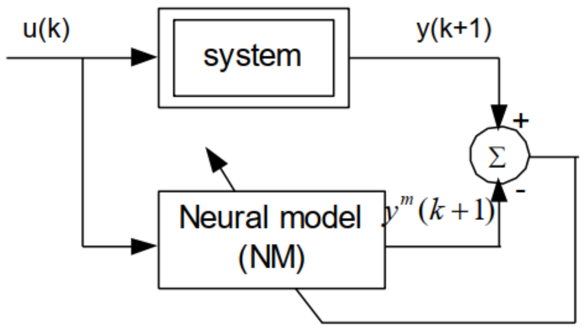

Fig. 2. The on-line Neural model training.

Assume that the unknown nonlinear system to be considered is expressed by:

$$y(k+1) = f\left[y(k), y(k-1), \ldots, y(k-n+1), u(k), u(k-1), \ldots, u(k-m+1)\right] \quad (7)$$

where *y(k)* is the scalar output of the system, *u(k)* is the scalar input to the system, *n* and *m* are the output and the input orders respectively, f[. . .] is the unknown nonlinear function to be estimated by a Neural Network.

The neural model for the unknown system can be expressed as:

$$y^m(k+1) = \hat{f}\left[y(k), y(k-1), \ldots, y(k-n+1), u(k), u(k-1), \ldots, u(k-m+1)\right] \quad (8)$$

Where y^m is the output of the neural model and $\hat{f}$ is the estimate of *f*.

The weights of the neural model are adjusted to minimize the cost function given by:

$$E = \frac{1}{2}\left(y(k+1) - y^m(k+1)\right)^2 \quad (9)$$

Adaptation of the PID controller gains

At each control cycle, the controller gains are adjusted. The cost function and the NM whose weights are now fixed are used to adjust the controller gains. In order to make the learning algorithm converges much faster while guaranteeing stability of the closed-loop system, the stability properties has been investigated in weight update. The idea consists in incorporating in the present weight update some influence of past iterations by using a momentum. The influence of the momentum depends on the stability properties of the closed-loop system. Thus, the influence can became large in cases where a desired stability performances is not well attained (small margin stability).
The weight tuning update for the gains using descent method is given by:

$$\Delta k_{p,I,D}(t) = -\alpha \frac{\partial E(t)}{\partial k_{p,I,D}} + \beta(t)\Delta k_{p,I,D}(t-1) \quad (10)$$

α is the learning rate and β is the momentum rate tuned according the flowing equation:

$$\beta(t) = \beta_0 \exp(-b_{P,C}) \quad (11)$$

Where β_0 is its initial value.

The presented adaptive momentum tuning method is developed to accelerate the adaptation of the controller parameters when the controller has poor robustness performance and to slow down the adaptation when the controller has good robustness performance.

The derivates in equation (10) is computed as:

$$\begin{aligned}
\frac{\partial E}{\partial k_p} &= \frac{\partial E(t)}{\partial y^m}\frac{\partial y^m}{\partial u(t)}\frac{\partial u(t)}{\partial K_p} = -(y-y^m)\frac{\partial y^m}{\partial u(t)}e_p(t) \\
\frac{\partial E}{\partial k_I} &= \frac{\partial E}{\partial y^m}\frac{\partial y^m}{\partial u(t)}\frac{\partial u(t)}{\partial K_I} = -(y-y^m)\frac{\partial y^m}{\partial u(t)}e_I(t) \\
\frac{\partial E}{\partial k_D} &= \frac{\partial E(t)}{\partial y^m}\frac{\partial y^m}{\partial u(t)}\frac{\partial u(t)}{\partial K_D} = -(y-y^m)\frac{\partial y^m}{\partial u(t)}e_D(t)
\end{aligned} \tag{12}$$

As the hidden and output neuron functions were defined by the logistic sigmoid function $f(x) = 1/(1+\exp(-x))$ then $\frac{\partial y^m}{\partial u(t)}$ is expressed as:

$$\frac{\partial y^m}{\partial u} = y^m(1-y^m)\sum_j W_j^m O_j^m (1-O_j^m) W_{1j}^m \tag{13}$$

O_j^m is the output of *j*th neuron in the hidden layer of the *NM*, W_{1j}^m and W_j^m are the weights of the *NM* from the input neurons to intermediate layer, and from the intermediate layer to the output.

Simulation Equations

$$\begin{cases} \dot{x}_1 = -x_1 + D_a(1-x_1)\exp\left(\dfrac{x_2}{1+x_2/\gamma}\right) \\ \dot{x}_2 = -x_2 + BD_a(1-x_1)\exp\left(\dfrac{x_2}{1+x_2/\gamma}\right) + \beta(u-x_2) \\ y = x_2 \end{cases}$$

where x_1, x_2, and u are the dimensionless reagent conversion, the temperature (output), and the coolant temperature (input), respectively and the nominal values for the constants in are $D_a=0.72$, $B=8$, $\gamma=20$ and $\beta=0.3$.