Design and Convergence of Iterative Learning Control Based on Neural Networks

Krzysztof Patan, Maciej Patan

Institute of Control and Computation Engineering
University of Zielona Góra, Poland

Introduction

- Iterative Learning Control – modern control strategy
- Neural networks – useful when dealing with nonlinear problems
- Purpose of the paper – to adopt artificial neural networks to develop iterative learning control
    1. to build an accurate neural model of the nonlinear plant based on the measurement from the previous trials
    2. to train the neural controller based on data provided by the model assuring convergence and stability,

Motivations

- Industrial robotics
  (S. Arimoto (1984), R.W. Longman (1994), M. Norrlow (2002))
- Numerical control machine tools (CNC)
  (D.-I. Kim, S.Kim (1996))
- Semiconductor lithography steppers
  (D. de Roover, O.H. Bosgra (2000), B.G. Dijkstra (2003))
- Chemical batch reactors
  (M. Mezghani, G.Roux (2002))
- Signal processing
  (H. Elci, R.W. Longman, M.Q. Phan (2002))
- Robots in rehabilitation and health care
  (Z. Cai, D. Tong, E. Rogers (2010))
- and many more . . .

General idea of neural network based ILC

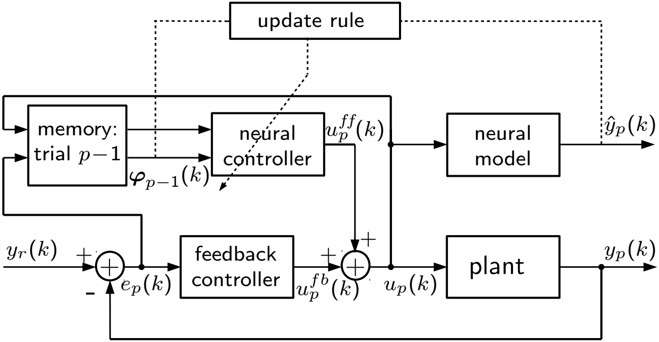
- Adaptation of so-called first-order ILC scheme (*current iteration ILC*)
    - use of existing feedback controller for stabilization,
    - adding supporting feedforward neural controler for tracking improvement,

$$u_p(k) = u_p^{fb}(k) + u_p^{ff}(k)$$

where p – trial number
$u_p^{fb}(k)$ – feedback control
$u_p^{ff}(k)$ – ILC update

- Model of the process required for synthesis of the ILC controller – reasonable application of data-driven neural modelling

Structure of the neural based ILC scheme



- Objective of neural controller – significant improvement of tracking for reference trajectory $y_r(k)$
- Efficient scheme in case of control of nonlinear processes supporting existing feedback controller

System description

Consider a class of dicrete-time nonlinear systems

$$\begin{aligned} \boldsymbol{x}_p(k+1) &= g(\boldsymbol{x}_p(k), u_p(k)), \ \ k = 0, \ldots, N-1, \\ y_p(k) &= \boldsymbol{C}\boldsymbol{x}_p(k) \end{aligned}$$

where $p \geq 0$ – a trial number

N – a trial length

$\boldsymbol{x}_p(k)$, $u_p(k)$, $y_p(k)$ – system state, input and response

g – some nonlinear function

Assumptions

Assumption A1. Let $y_r(k)$ be a reference trajectory defined over a discrete time k, which is assumed to be realizable, that is there exists a unique $u_r(k)$ and an initial state $\boldsymbol{x}_d(0)$, i.e.

$$\begin{aligned} \boldsymbol{x}_r(k+1) &= g(\boldsymbol{x}_r(k), u_r(k)) \\ y_r(k) &= \boldsymbol{C}\boldsymbol{x}_r(k) \end{aligned}.$$

Assumption A2. The identical initial condition holds for all trials, i.e.

$$\forall p \;\; \boldsymbol{x}_p(0) = \boldsymbol{x}_r(0).$$

Assumption A3. The nonlinear function g satisfies the global Lipschitz condition

$$\|g(\boldsymbol{x}_1, u_1) - g(\boldsymbol{x}_2, u_2)\| \leq L\left(\|\boldsymbol{x_1} - \boldsymbol{x}_2\| + |u_1 - u_2|\right),$$

where $L > 0$ stands for the Lipschitz constant.

Neural model

- System modelling – state space neural network model

$$\hat{\boldsymbol{x}}_p(k+1) = \hat{g}(\hat{\boldsymbol{x}}_p(k), u_p(k))$$
$$\hat{y}_p(k) = \boldsymbol{C}\hat{\boldsymbol{x}}_p(k)$$

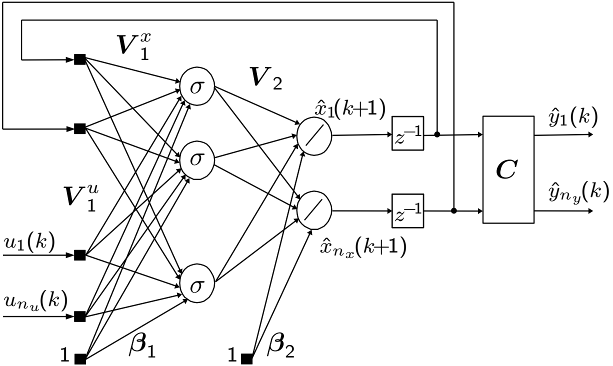
where $\hat{\boldsymbol{x}}_{\boldsymbol{p}} \in \mathbb{R}^{n_x}$ – model state, $u_p \in \mathbb{R}^1$, $\hat{y}_p \in \mathbb{R}^1$ – model input and output

- Implementation of nonlinear function $\hat{g}(\cdot, \cdot)$:

$$\hat{g}(\hat{\boldsymbol{x}}_p(k), u_p(k)) = \boldsymbol{V}_2\sigma(\boldsymbol{V}_1^x\hat{\boldsymbol{x}}_p(k) + \boldsymbol{V}_1^u u_p(k) + \boldsymbol{\beta}_1) + \boldsymbol{\beta}_2),$$

gdzie $\boldsymbol{V}_1^u \in \mathbb{R}^{v_m \times 1}$, $\boldsymbol{V}_1^x \in \mathbb{R}^{v_m \times n_x}$ and $\boldsymbol{V}_2 \in \mathbb{R}^{n_x \times v_m}$ – layers weight matrices
$\boldsymbol{\beta}_1 \in \mathbb{R}^{v_m}$, $\boldsymbol{\beta}_2 \in \mathbb{R}^{n_x}$ – bias vectors
$\sigma : \mathbb{R}^{v_m} \to \mathbb{R}^{v_m}$ – hidden neurons activation functions
v_m – number of hidden neurons
n_x – model order

Structure of the neural network model



- Training in batch mode (off-line) based on previous measurement data
- Training algorithms
  - dynamic backward propagation
  - dynamic Newton methods
  - modified Levenberg-Marquardt

Neural controller

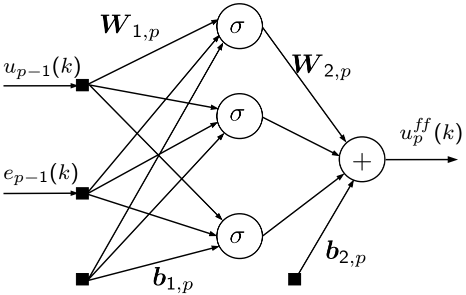
- The key idea – use the neural network to provide the realization of the function $u_p^{ff}(k)$ (being implicitly an inverted model of the plant)
- Let consider the controller in the form:

$$u_p^{ff}(k) = f(\boldsymbol{\varphi}_{p-1}(k)),$$

  where f is a nonlinear function,

- $\boldsymbol{\varphi}_{p-1}(k)$ – regression vector, e.g.
  - $\boldsymbol{\varphi}_{p-1}(k) = [u_{p-1}(k), e_{p-1}(k)]^\mathsf{T}$ – P-type controller
  - $\boldsymbol{\varphi}_{p-1}(k) = [u_{p-1}(k), e_{p-1}(k+1)]^\mathsf{T}$ – D-type controller
  - $\boldsymbol{\varphi}_{p-1}(k) = [u_{p-1}(k), e_{p-1}(k), e_{p-1}(k+1)]^\mathsf{T}$ – PD-type controller

Structure of the neural network controller



- Neural network with one hidden layer

$$u_p^{ff}(k) = f(\boldsymbol{\varphi}_{p-1}(k)) = \boldsymbol{W}_{2,p}\sigma(\boldsymbol{W}_{1,p}\boldsymbol{\varphi}_{p-1}(k) + \boldsymbol{b}_{1,p}) + \boldsymbol{b}_{2,p},$$

where $\boldsymbol{W}_{1,p}$, $\boldsymbol{W}_{2,p}$ – weight matrices
$\boldsymbol{b}_{1,p}$, $\boldsymbol{b}_{2,p}$ – bias vectors
σ – hidden neurons activation function

- Network parameters are updated after each process trial

Update rule

- After each trail the controller parameters are updated according to:

$$\boldsymbol{\theta}_p = \boldsymbol{\theta}_{p-1} + \Delta\boldsymbol{\theta}_p$$

where $\boldsymbol{\theta}_p$ – the generalized network parameter
$\Delta\boldsymbol{\theta}_p$ – a correction term

- Learning objective – at each trial p minimize the criterion

$$J_p = \frac{1}{2}\sum_{k=1}^{N}(y_r(k) - y_p(k))^2 + \frac{1}{2}\mu\sum_{i=1}^{M}\theta_{p,i}^2$$

where M – the number of the controller parameters
μ – a parameter governing how strongly large weights are penalized

- Using the gradient descent

$$\Delta\boldsymbol{\theta}_p = -\eta\frac{\partial J_p}{\partial\boldsymbol{\theta}_p}$$

where η – the learning rate

- The gradient of the cost function J with respect to the parameter θ

$$\frac{\partial J_p}{\partial \boldsymbol{\theta}_p} = \sum_{k=1}^{N} \left(e_p(k) \frac{\partial y_p(k)}{\partial u_p(k-1)} \frac{\partial u_p(k-1)}{\partial \boldsymbol{\theta}_p} \right) + \lambda \boldsymbol{\theta}_p \qquad (*)$$

- The first partial derivative in $(*)$, due to the equivalence rule, can be calculated using the neural model of the system:

$$\frac{\partial y_p(k)}{\partial u_p(k-1)} \approx \frac{\partial \hat{y}_p(k)}{\partial u_p(k-1)} = \boldsymbol{C}\boldsymbol{V}_2 \left(\sigma' \circ \boldsymbol{V}_1^u\right)$$

where $\boldsymbol{V}_1^{u\mathsf{T}}$ is the weight vector associated with the input $u(k-1)$
$\circ$ – the Hadamard product (element-wise)

- The second derivative can be calculated assuming that

$$u_p(k) = u_p^{ff}(k)$$

- for weights of the first layer $\boldsymbol{W}_1$:

$$\frac{\partial u_p(k-1)}{\partial \boldsymbol{W}_{1,p}} = \left(\boldsymbol{W}_{2,p} \circ \sigma^{'}\right) \varphi_{p-1}^{\mathsf{T}}(k-1)$$

- for biases of the first layer $\boldsymbol{b}_1$:

$$\frac{\partial u_p(k-1)}{\partial \boldsymbol{b}_{1,p}} = \boldsymbol{W}_{2,p} \circ \sigma^{'}$$

- for weights of the second layer $\boldsymbol{W}_2$:

$$\frac{\partial u_p(k-1)}{\partial \boldsymbol{W}_{2,p}} = \sigma(\boldsymbol{W}_{1,p}\boldsymbol{\varphi}_{p-1}(k-1) + \boldsymbol{b}_{1,p})$$

- for biases of the second layer $\boldsymbol{b}_2$:

$$\frac{\partial u_p(k-1)}{\partial \boldsymbol{b}_{2,p}} = \mathbf{1}$$

Convergence analysis

- Controlled system

$$
\begin{aligned}
\boldsymbol{x}_p(k+1) &= g(\boldsymbol{x}_p(k), u_p(k)), \;\; k = 0, \dots, N-1, \\
y_p(k) &= \boldsymbol{C}\boldsymbol{x}_p(k)
\end{aligned}
\tag{1}
$$

- Neural controller

$$
u_p^{ff}(k) = f(\boldsymbol{\varphi}_{p-1}(k)) = \boldsymbol{W}_{2,p}\sigma(\boldsymbol{W}_{1,p}\boldsymbol{\varphi}_{p-1}(k) + \boldsymbol{b}_{1,p}) + \boldsymbol{b}_{2,p}, \tag{2}
$$

  with P-type regression vector $\boldsymbol{\varphi}_{p-1}(k) = [u_{p-1}(k), e_{p-1}(k)]^\mathsf{T}$

- Define controller sensitivities (with respect to input and error, respectively)

$$
f_u(k) = \frac{\partial f(u_p(k), e_p(k))}{\partial u_p(k)}, \quad f_e(k) = \frac{\partial f(u_p(k), e_p(k))}{\partial e_p(k)}
$$

Main result

Theorem

For nonlinear system (1)*, under the assumptions* **A1–A3** *hold, convergence of the control law* (2) *with the P-type regressor is guaranteed if*

$$\gamma_1 + \gamma_2 \cdot \frac{1 - \alpha^{-(\lambda - 1)N}}{1 - \alpha^{-(\lambda - 1)}} < 1 \tag{3}$$

where $\gamma_1 = \sup_k \|f_u(k)\|, \quad \gamma_2 = \sup_k \|f_e(k)\boldsymbol{C}\|$

α – *Lipschitz constant of controlled process,*

$\lambda > 0$,

Sketch of proof

- the proof can be obtained as an extension of the approach presented in: D. Shen, W. Zhang, J. Xu: Iterative learning control for discrete nonlinear systems with randomly iteration varying lengths, Systems & Control Letters, vol. 96, pp. 81-87, 2016.
- proof is based on deriving uniform convergence property

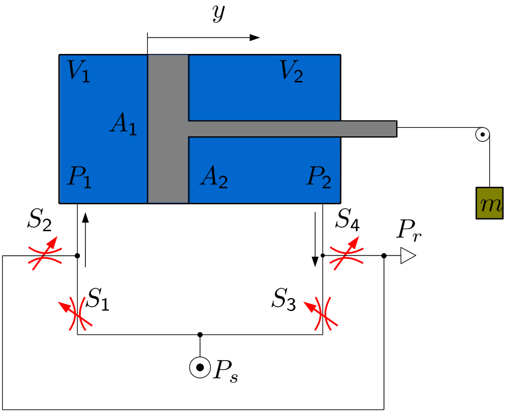
$$\lim_{p\to\infty} u_p(k) = u^*(k),$$

through analysis of the induced norms imposed on the control law

$$\|z(k)\|_\lambda = \sup_{k\in[0,N-1]} \alpha^{-\lambda k}\|z(k)\|$$

- to deal with a nonlinear representation, the learning controller is expanded into Taylor series
- possible generalizations toward D-type and PD-type update rules

Illustrative example – pneumatic servomechanism



V_1, V_2 – cylinder volumes
A_1, A_2 – chamber areas
P_1, P_2 – chamber pressures
P_s – supplied pressure
P_r – exhaust pressure
m – load mass
y – piston position
$S_1, \ldots, S_4$ – operating valves
u – control signal

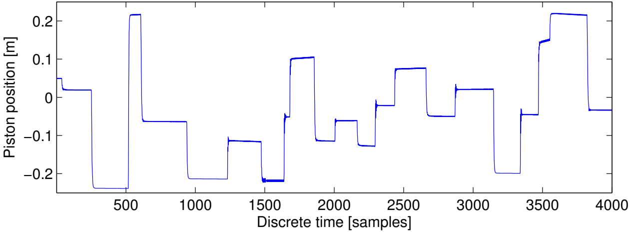
S_1 and S_4 are open for $u \geqslant 0$
S_2 and S_3 are open for $u < 0$

Modelling

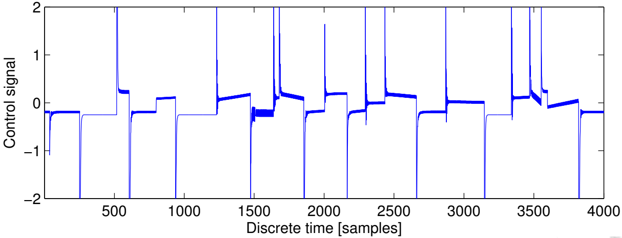
- investigated nonlinear system is poorly damped and includes integration action
- data recorded in the closed-loop control with the P controller
- reference: random steps trigerred randomly with levels covering possible piston positions from the interval $(-0.245, 0.245)$
- neural model setting: number of delayed outputs and inputs $n_x = 3$, number of hidden neurons $v_m = 5$, activation function of hidden neurons $\sigma_h \equiv \tanh$,
- training process carried out for 100 epochs
- modelling quality – Sum of Squared Errors is SSE=0.0438

Exemplary interval of training data: the output (a), the control (b)

a)



b)



Synthesis of ILC neural controller

- random initial neural controller parameters
- training dataset: $\{e(k), u(k)\}_{k=1}^{N}$ – recorded during the evaluation of the closed-loop control with the feedback controller
- structure of the neural controller: $v_c = 3$, $\sigma_h \equiv \tanh$
- training carried out after each trial
  - learning rate: $\eta = 0.05$
  - penalty factor: $\mu = 0.0001$
  - iteration number: 100
- performance index: norm of tracking error (for P-type controller: $\|e(k)\| = 0.4843$)

Convergence condition

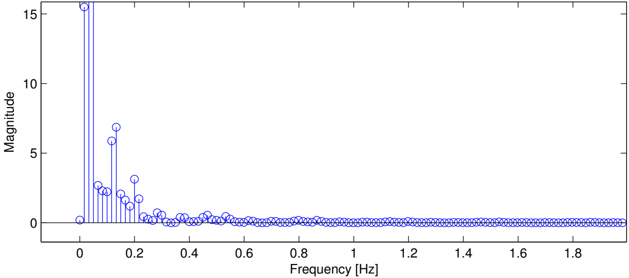
- Convergence condition can be rewritten in the form

$$\|\boldsymbol{W}_{2,p}\boldsymbol{W}^{u}_{1,p}\| + \|\boldsymbol{W}_{2,p}\boldsymbol{W}^{e}_{1,p}\| < 1$$

- easy to check during retraining of neural controller
- in the case of infeasibility parameter update is not executed at given trial
- safety margin can also be introduced (some value < 1); if the convergence condition if violated controller weights will bring back to the best ones stored during learning

Q-Filter

- spectrum of reference signal



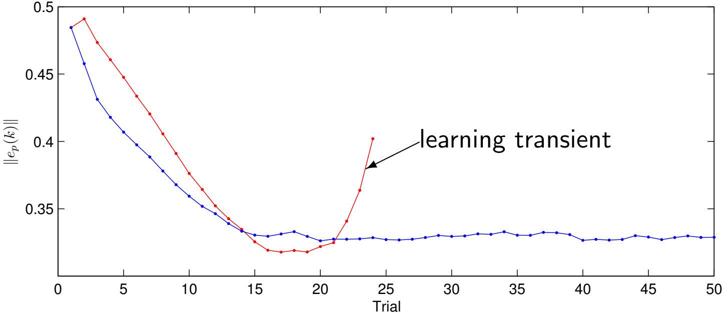
- cutoff frequency $f_c = 1,75Hz$
- transfer function

$$Q(z) = \frac{0.63}{z - 0.37}.$$

Experiment 1

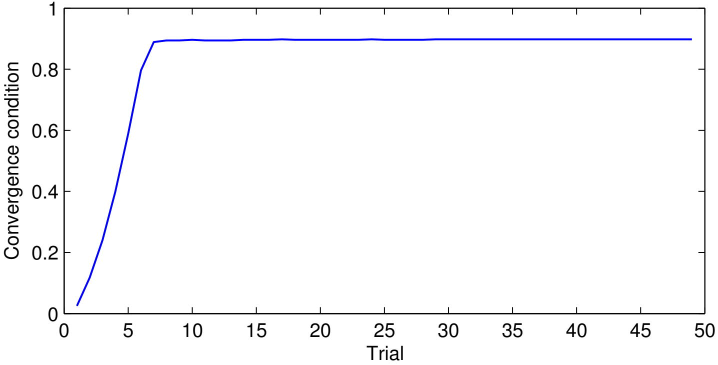
Control with convergence verification

Error norm convergence



a) without Q-filter – red,
b) with Q-filter – blue

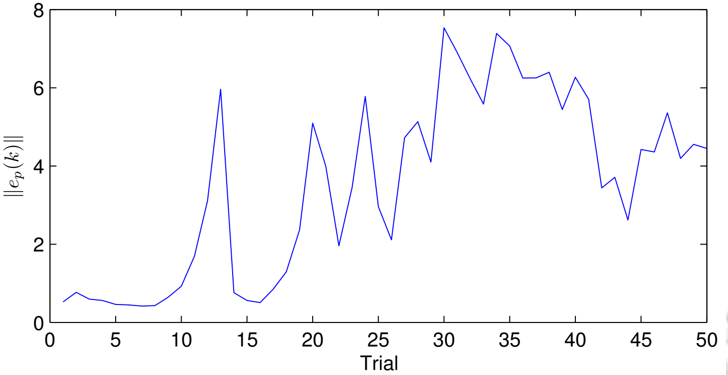
Evolution of the convergence condition - Exp. 1



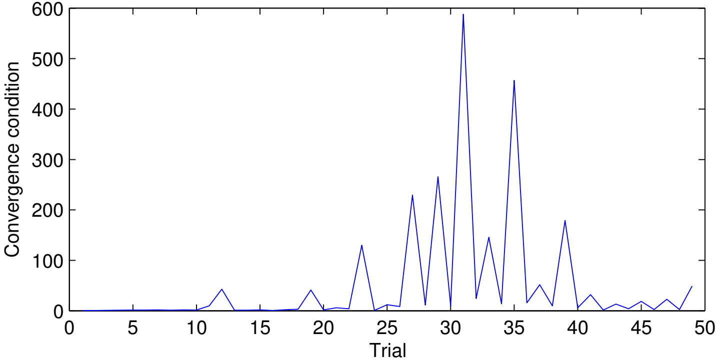
Experiment 2

Control without convergence verification

Error norm convergence



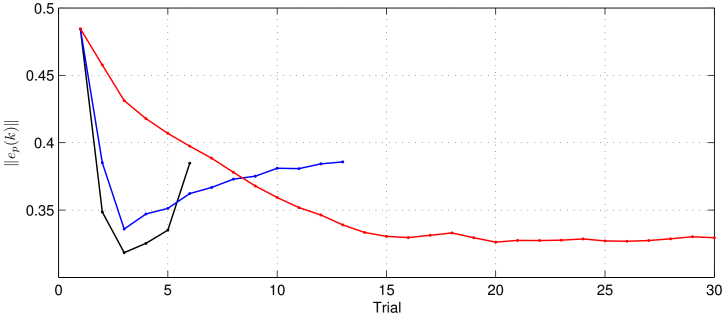
Evolution of the convergence condition - Exp. 2



Comparison to a conventional technique

Linear update rule: $u_{p+1}(k) = u_p(k) + 0.5e_p(k)$

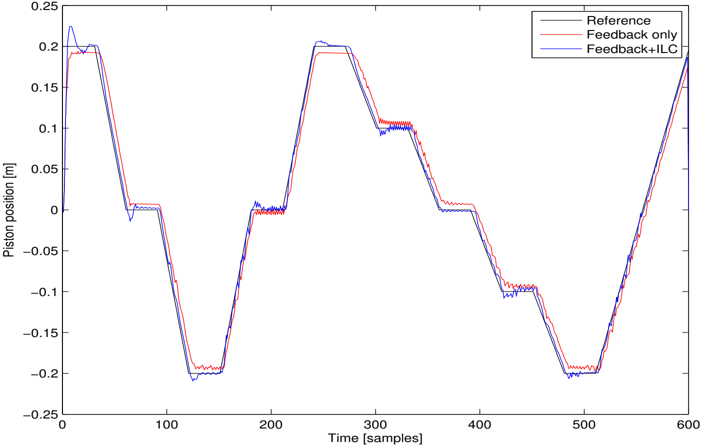
Error norm convergence



neural controller – red,
linear rule without Q-filter – black,
linear rule with Q-filter – blue

Control quality

Reference trajectory tracking



Concluding remarks

- A novel approach for ILC synthesis based on neural networks was proposed
- The proposed control scheme may lead to significant improvement of control system performance.
- Advantages of the proposed approach are the great flexibility of neural controller in adaptation to plant nonlinearities and simplicity of the ultimate training algorithm
- The solution was tested on the pneumatic servomechanism using different working conditions of the plant with promising results
- There is still a room for refinements:
  - improving the performance of neural controller
  - neural Q-filter implementation
  - adaptation of high-order ILC schemes
  - developing robust neural network based ILC