Sensors and Materials, Vol. 29, No. 6 (2017) 699–711
MYU Tokyo

S & M 1361

Design of Quaternion-Neural-Network-Based Self-Tuning Control Systems

Kazuhiko Takahashi,* Yusuke Hasegawa, and Masafumi Hashimoto[1]

Information Systems Design, Doshisha University,
1-3 Miyakodani, Tatara, Kyotanabe, Kyoto 610-0321, Japan
[1]Intelligent Information Engineering and Science, Doshisha University,
1-3 Miyakodani, Tatara, Kyotanabe, Kyoto 610-0321, Japan

(Received September 12, 2016; accepted January 5, 2017)

Keywords: quaternion neural network, self-tuning controller, PID controller, nonlinear plant, reference model

In this study, we investigate the control performance of an adaptive controller using a multilayer quaternion neural network. The control system is a self-tuning controller, the control parameters of which are tuned online by the quaternion neural network to track plant output to follow the desired output generated by a reference model. A proportional–integral–derivative (PID) controller is used as a conventional controller, the parameters of which are tuned by the quaternion neural network. Computational experiments to control a single-input single-output (SISO) discrete-time nonlinear plant are conducted to evaluate the capability and characteristics of the quaternion-neural-network-based self-tuning PID controller. Experimental results show the feasibility and effectiveness of the proposed controller.

1. Introduction

The use of hypercomplex-valued neural networks, such as complex neural networks and quaternion neural networks, to overcome classically hard-to-treat intractable problems has been investigated.[1,2] Quaternion neural networks have been demonstrated to perform better than real-number neural networks, because the former can cope with multidimensional issues more efficiently by employing quaternions directly. Many studies have successfully used quaternion neural networks in applications requiring multidimensional signal processing, for example, colour image processing,[3,4] signal processing,[5,6] filtering,[7] inverse problems,[8] and classification problems.[9] In previous studies, we presented robot control applications of quaternion neural networks to solve forward and inverse kinematics of a robot manipulator.[10,11]

In this study, we investigate the characteristics of an adaptive controller based on a quaternion neural network. The controller is a self-tuning feedback controller, comprising a reference model to generate the desired output and a feedback controller, the parameters of which are tuned online by the quaternion neural network. Moreover, we use a proportional–integral–derivative (PID) controller as the feedback controller. We conducted computational experiments for controlling a single-input single-output (SISO) discrete-time nonlinear plant to evaluate the feasibility of the proposed quaternion-neural-network-based self-tuning PID controller.

*Corresponding author: e-mail: katakaha@mail.doshisha.ac.jp
http://dx.doi.org/10.18494/SAM.2017.1468

ISSN 0914-4935

2. Self-tuning PID Controller Using Quaternion Neural Network

2.1 Quaternion neural network

A quaternion forms a class of hypercomplex number that consists of a real number and three imaginary numbers. A quaternion $\boldsymbol{q}$ is defined as

$$\boldsymbol{q} = q_0 + q_1\mathtt{i} + q_2\mathtt{j} + q_3\mathtt{k}, \tag{1}$$

where q_i (i = 0, 1, 2, 3) is the real number parameter. The real number unit is 1 and the three imaginary units are $\mathtt{i}$, $\mathtt{j}$, and $\mathtt{k}$. They are orthogonal spatial vectors. Quaternion algebra is not commutative and satisfies the following Hamilton rules:

$$\mathtt{i}^2 = \mathtt{j}^2 = \mathtt{k}^2 = -1, \mathtt{ij} = -\mathtt{ji} = \mathtt{k}, \mathtt{jk} = -\mathtt{kj} = \mathtt{i}, \mathtt{ki} = -\mathtt{ik} = \mathtt{j}.$$

The conjugate of a quaternion $\boldsymbol{q}^*$ is defined as

$$\boldsymbol{q}^* = q_0 - q_1\mathtt{i} - q_2\mathtt{j} - q_3\mathtt{k}, \tag{2}$$

and the multiplication between one quaternion and its conjugate is defined as

$$\boldsymbol{q} \otimes \boldsymbol{q}^* = q_0^2 + q_1^2 + q_2^2 + q_3^2. \tag{3}$$

The addition and subtraction of two quaternions, $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$, are defined as

$$\boldsymbol{q}_1 \pm \boldsymbol{q}_2 = (q_{0_1} \pm q_{0_2}) + (q_{1_1} \pm q_{1_2})\mathtt{i} + (q_{2_1} \pm q_{2_2})\mathtt{j} + (q_{3_1} \pm q_{3_2})\mathtt{k}. \tag{4}$$

The multiplication between a real number a and a quaternion $\boldsymbol{q}$ is given as

$$a\boldsymbol{q} = aq_0 + aq_1\mathtt{i} + aq_2\mathtt{j} + aq_3\mathtt{k}, \tag{5}$$

while the multiplication between two quaternions $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ is given as

$$\boldsymbol{q}_1 \otimes \boldsymbol{q}_2 = q_{0_1}q_{0_2} - \overrightarrow{q_1} \cdot \overrightarrow{q_2} + q_{0_2}\overrightarrow{q_1} + q_{0_1}\overrightarrow{q_2} + \overrightarrow{q_1} \times \overrightarrow{q_2}, \tag{6}$$

where

$$\overrightarrow{q_i} = \begin{bmatrix} q_{1_i} & q_{2_i} & q_{3_i} \end{bmatrix}^{\mathrm{T}}$$

(i = 1, 2); $\cdot$ and $\times$ represent scalar and vector products, respectively. The norm of the quaternion is defined as

$$|\boldsymbol{q}| = \sqrt{\boldsymbol{q} \otimes \boldsymbol{q}^*}, \tag{7}$$

and the inverse of the quaternion is given as

$$\boldsymbol{q}^{-1} = \frac{\boldsymbol{q}^*}{\boldsymbol{q} \otimes \boldsymbol{q}^*}. \tag{8}$$

To describe the training algorithm of the multilayer quaternion neural network, a multilayer quaternion neural network was considered. In the input layer of the quaternion neural network, the l-th neuron input $\boldsymbol{x}_l$ is a quaternion:

$$\boldsymbol{x}_l = x_{0_l} + x_{1_l}\mathbf{i} + x_{2_l}\mathbf{j} + x_{3_l}\mathbf{k}. \tag{9}$$

In the hidden layer connected with the input layer, the output from the m-th neuron unit $z_m^{(\mathrm{h}_1)}$ is defined as

$$z_m^{(\mathrm{h}_1)} = f\Big(\sum_l \boldsymbol{w}_{ml}^{(\mathrm{h}_{10})} \otimes \boldsymbol{x}_l + \boldsymbol{\phi}_m^{(\mathrm{h}_1)}\Big), \tag{10}$$

where $\boldsymbol{w}_{ml}^{(\mathrm{h}_{10})}$ is the weight between the l-th neuron of the input layer and the m-th neuron of the hidden layer, $\boldsymbol{\phi}_m^{(\mathrm{h}_1)}$ is the threshold of the m-th neuron in the hidden layer, and $f(\cdot)$ is an activation function of the neuron. In the p-th hidden layer, the output from the n-th neuron unit $z_n^{(\mathrm{h}_p)}$ is defined as

$$z_n^{(\mathrm{h}_p)} = f\Big(\sum_m \boldsymbol{w}_{nm}^{(\mathrm{h}_p\ p-1)} \otimes z_m^{(\mathrm{h}_{p-1})} + \boldsymbol{\phi}_n^{(\mathrm{h}_p)}\Big), \tag{11}$$

where $\boldsymbol{w}_{nm}^{(\mathrm{h}_p\ p-1)}$ is the weight between the m-th neuron of the $(p-1)$-th hidden layer and the n-th neuron of the p-th hidden layer, and $\boldsymbol{\phi}_n^{(\mathrm{h}_p)}$ is the threshold of the n-th neuron in the p-th hidden layer. In the output layer, the output from the s-th neuron unit $z_s^{(\mathrm{o})}$ is defined as

$$z_s^{(\mathrm{o})} = f\Big(\sum_n \boldsymbol{w}_{sn}^{(\mathrm{o}_{0p})} \otimes z_n^{(\mathrm{h}_p)} + \boldsymbol{\phi}_s^{(\mathrm{o})}\Big), \tag{12}$$

where $\boldsymbol{w}_{sn}^{(\mathrm{o}_{0p})}$ is the weight between the n-th neuron of the p-th hidden layer and the s-th neuron of the output layer, and $\boldsymbol{\phi}_s^{(\mathrm{o})}$ is the threshold of the s-th neuron in the output layer. Here, the weights and thresholds are quaternions and the activation function is a quaternion function.

The training of the quaternion neural network was conducted to minimise the cost function J:

$$J = \frac{1}{2}\sum_P \sum_s \boldsymbol{\epsilon}_s \otimes \boldsymbol{\epsilon}_s^*, \tag{13}$$

where $\boldsymbol{\epsilon}_s$ is the output error defined by $\boldsymbol{\epsilon}_s = \boldsymbol{d}_s - z_s^{(\mathrm{o})}$, $\boldsymbol{d}_s$ is the desired output of the s-th neuron in the output layer, and P indicates the index of training pattern. According to the steepest descent method applied to quaternion function,[12,13] the parameters of the multilayer quaternion neural network are given as

$$\boldsymbol{\omega}(k+1) = \boldsymbol{\omega}(k) - \eta\frac{\partial J}{\partial \boldsymbol{\omega}^*(k)}, \tag{14}$$

where k is the iteration number, η is the learning factor, and $\boldsymbol{\omega}(k)$ is composed of network parameters such as weights and thresholds.

In this study, the activation function was split[14] as

$$f(\boldsymbol{q}) = f_0(q_0) + f_1(q_1)\mathtt{i} + f_2(q_2)\mathtt{j} + f_3(q_3)\mathtt{k}, \tag{15}$$

where

$$f_i(x) = \frac{1}{1 + e^{-x}},$$

(i = 0, 1, 2, 3) is a real-valued function. This activation function is not analytic in the quaternion number domain; however, we use it for computational convenience to derive the training algorithm of the quaternion neural network. A back-propagation algorithm of the quaternion neural network can be obtained by calculating the gradient of the cost function with respect to the quaternion neural network's parameters as

$$\boldsymbol{w}_{sn}^{(\mathtt{o}_{0p})}(k+1) = \boldsymbol{w}_{sn}^{(\mathtt{o}_{0p})}(k) + \eta \sum_{P} \boldsymbol{z}_n^{(\mathtt{h}_p)} \otimes \boldsymbol{\delta}_s^{(\mathtt{o}_{0p})*}, \tag{16}$$

$$\boldsymbol{\phi}_s^{(\mathtt{o})}(k+1) = \boldsymbol{\phi}_s^{(\mathtt{o})}(k) + \eta \sum_{P} \boldsymbol{\delta}_s^{(\mathtt{o}_{0p})*}, \tag{17}$$

where

$$\boldsymbol{\delta}_s^{(\mathtt{o}_{0p})} = \boldsymbol{\epsilon}_s \odot f'\Big(\sum_n \boldsymbol{w}_{sn}^{(\mathtt{o}_{0p})} \otimes \boldsymbol{z}_n^{(\mathtt{h}_p)} + \boldsymbol{\phi}_s^{(\mathtt{o})}\Big).$$

$\odot$ denotes the component-by-component product and

$$\boldsymbol{w}_{nm}^{(\mathtt{h}_{p\ p-1})}(k+1) = \boldsymbol{w}_{nm}^{(\mathtt{h}_{p\ p-1})}(k) + \eta \sum_{P} \boldsymbol{z}_m^{(\mathtt{h}_{p-1})} \otimes \boldsymbol{\delta}_n^{(\mathtt{h}_{p\ p-1})*}, \tag{18}$$

$$\boldsymbol{\phi}_n^{(\mathtt{h}_p)}(k+1) = \boldsymbol{\phi}_n^{(\mathtt{h}_p)}(k) + \eta \sum_{P} \boldsymbol{\delta}_n^{(\mathtt{h}_{p\ p-1})*}, \tag{19}$$

where

$$\boldsymbol{\delta}_n^{(\mathtt{h}_{p\ p-1})} = \left\{\sum_t \boldsymbol{\delta}_t^{(\mathtt{h}_{p+1\ p})} \otimes \boldsymbol{w}_{tn}^{(\mathtt{h}_{p\ p-1})}\right\} \odot f'\Big(\sum_m \boldsymbol{w}_{nm}^{(\mathtt{h}_{p\ p-1})} \otimes \boldsymbol{z}_m^{(\mathtt{h}_{p-1})} + \boldsymbol{\phi}_n^{(\mathtt{h}_p)}\Big).$$

2.2 Self-tuning controller

Figure 1 shows a schematic of a self-tuning feedback controller, where u is the control input synthesized by a conventional controller, the parameters of which are tuned online by a quaternion neural network, y is the plant output, y_d is the desired plant output generated by a reference model, and r is the reference input. To simplify the controller design, we assume a linear SISO discrete-time plant as

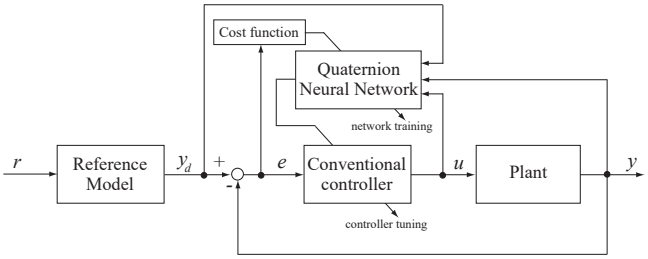

Fig. 1. Schematic of quaternion-neural-network-based self-tuning feedback controller.

$$y(k+d) = b_0 u(k) + \sum_{i=1}^{m+d-1} \alpha_{u_i} u(k-i) + \sum_{i=1}^{n} \alpha_{y_i} y(k-i+1), \tag{20}$$

where k is sampling number, $y(k)$ is plant output, $u(k)$ is plant input, n and m are plant orders, d is plant dead time, and α_{u_i}, α_{y_i}, and b_0 are plant parameters. Using a digital PID controller, the following control input $u(k)$ is provided

$$\begin{aligned} u(k) &= u(k-1) + g_1(k)\Delta e(k) + g_2(k)e(k) + g_3(k)\left(\Delta e(k) - \Delta e(k-1)\right) \\ &= u(k-1) + \boldsymbol{\xi}^{\mathrm{T}}(k)\boldsymbol{g}(k), \end{aligned} \tag{21}$$

where

$$\boldsymbol{\xi}(k) = \begin{bmatrix} \Delta e(k) & e(k) & \Delta e(k) - \Delta e(k-1) \end{bmatrix}^{\mathrm{T}},$$

$e(k)$ is the output error defined as the difference between the desired and actual plant outputs, $\Delta e(k)$ is the time difference of the output error, and

$$\boldsymbol{g}(k) = \begin{bmatrix} g_1(k) & g_2(k) & g_3(k) \end{bmatrix}^{\mathrm{T}},$$

is the gain parameter vector.

Consider the condition $\lim_{k\to\infty} e(k+d) = 0$ using an auxiliary parameter,[15] expressed as

$$\boldsymbol{g}(k) = \frac{1}{b_0}\left\{\boldsymbol{\xi}(k)\boldsymbol{\xi}^{\mathrm{T}}(k) + e_0\boldsymbol{E}\right\}^{-1}\boldsymbol{\xi}(k)\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{I}(k), \tag{22}$$

where e_0 is a constant that ensures the regularity of the matrix, $\boldsymbol{E}$ is an identity matrix, and

$$\boldsymbol{\alpha} = \begin{bmatrix} 1 & -\alpha_{y_1} & \cdots & -\alpha_{y_n} & -b_0 - \alpha_{u_1} & -\alpha_{u_2} & \cdots & -\alpha_{u_{m+d-1}} \end{bmatrix}^{\mathrm{T}},$$

and

$$\boldsymbol{I}(k) = \begin{bmatrix} y_d(k+d) & y(k) & \cdots & y(k-n+1) & u(k-1) & u(k-2) & \cdots & u(k-m-d+1) \end{bmatrix}^{\mathrm{T}}.$$

In this design, the output of the quaternion neural network is used as the gain parameter. By representing the mapping function of the quaternion neural network as $\boldsymbol{F}_{qnn}(\cdot)$, we describe the input-output relationship of the quaternion neural network as

$$\boldsymbol{z}^{(\mathrm{o})}(k) = \boldsymbol{F}_{qnn}(\boldsymbol{\omega}(k), \boldsymbol{x}(k)), \tag{23}$$

By comparing Eqs. (22) and (23), an input to the quaternion neural network $\boldsymbol{x}(k)$ should be composed of the elements of the vectors $\boldsymbol{I}(k)$ and $\boldsymbol{\xi}(k)$. The output of the quaternion neural network $\boldsymbol{z}^{(\mathrm{o})}(k) = z_0^{(\mathrm{o})}(k) + z_1^{(\mathrm{o})}(k)\mathtt{i} + z_2^{(\mathrm{o})}(k)\mathtt{j} + z_3^{(\mathrm{o})}(k)\mathtt{k}$ is set as the gain parameter: $g_1(k) = z_1^{(\mathrm{o})}(k)$, $g_2(k) = z_2^{(\mathrm{o})}(k)$, and $g_3(k) = z_3^{(\mathrm{o})}(k)$.

Considering the dead time of the plant, the multilayer quaternion neural network of the self-tuning PID controller was trained to minimize the cost function $J(k)$ online as

$$\boldsymbol{\omega}(k+1) = \boldsymbol{\omega}(k-d) - \eta \frac{\partial J(k)}{\partial \boldsymbol{\omega}^*(k-d)}, \tag{24}$$

where η is the learning factor and the cost function is defined by the output error:

$$J(k) = \frac{1}{2}e^2(k). \tag{25}$$

The gradient of the cost function with respect to the quaternion neural network's parameters can be calculated using the chain rule as

$$\frac{\partial J(k)}{\partial \boldsymbol{\omega}^*(k-d)} = \frac{\partial J(k)}{\partial e(k)} \frac{\partial e(k)}{\partial y(k)} \frac{\partial y(k)}{\partial u(k-d)} \frac{\partial u(k-d)}{\partial \boldsymbol{g}(k-d)} \frac{\partial \boldsymbol{z}^{(\mathrm{o})}(k-d)}{\partial \boldsymbol{\omega}^*(k-d)},$$

where $\dfrac{\partial y(k)}{\partial u(k-d)}$ is the Jacobian of the plant, $\dfrac{\partial u(k)}{\partial \boldsymbol{g}(k)}$ is the Jacobian of the controller, and the gradient $\dfrac{\partial \boldsymbol{z}^{(\mathrm{o})}(k)}{\partial \boldsymbol{\omega}^*(k)}$ can be calculated using the back-propagation algorithm extended to the quaternion described in Sect. 2.1.

Convergence analysis of the quaternion neural network is important and several studies have been conducted.[7,13,16] In this study, the difference in the cost function $\Delta J(k)$ is derived as

$$\begin{aligned}
\Delta J(k) &= J(k+1) - J(k-d) \\
&= \{\boldsymbol{\omega}(k+1) - \boldsymbol{\omega}(k-d)\} \otimes \frac{J(k+1) - J(k-d)}{\boldsymbol{\omega}(k+1) - \boldsymbol{\omega}(k-d)} \\
&\approx -\eta \frac{\partial J(k)}{\partial \boldsymbol{\omega}^*(k-d)} \otimes \frac{\partial J(k)}{\partial \boldsymbol{\omega}(k-d)} \\
&= -\eta \left| \frac{\partial J(k)}{\partial \boldsymbol{\omega}(k-d)} \right|^2 \leq 0,
\end{aligned}$$

where the learning factor η is nonnegative and the gradient of the cost function with respect to the quaternion neural network's parameter is approximated by the finite difference of the cost

function and the parameters. This condition indicates the stability of the quaternion neural network's training; however, it is difficult to guarantee the stability of the quaternion neural network throughout the training process, because the finite-difference approximation of the gradient is not satisfied when the quaternion neural network's parameters significantly change in the initial training stage or when they hardly change in the final training stage. Furthermore, the learning factor η has an upper limit because the finite-difference approximation of the gradient is not satisfied for a relatively large learning factor. It is difficult to analyze the upper limit of the learning factor due to the close relationship between the gradient and the plant dynamics; accordingly, we use numerical simulations to determine the learning factor that guarantees the stability of the quaternion neural network.

3. Computational Experiments of Adaptive Controller

To investigate the feasibility of the quaternion-neural-network-based self-tuning PID controller, we performed computational experiments using the following SISO discrete-time nonlinear plant with a dominant second-order system:[17]

$$y(k+1) = \sum_{i=1}^{2} \alpha_{y_i} y(k-i+1) + b_0 u(k) + \sum_{i=1}^{1} \alpha_{u_i} u(k-i) + \alpha_{add} y(k-2) + c_{non} y^2(k), \qquad (26)$$

where α_{add} is the coefficient of the parasitic term and c_{non} is the coefficient of the nonlinear term. Because the plant was assumed to be a linear, second-order plant ($n = 2$, $m = 1$, and $d = 1$) for designing the quaternion neural network, the input vector of the quaternion neural network was defined as

$$\begin{cases} \boldsymbol{x}_1(k) = y_d(k+1) + y(k)\mathtt{i} + y(k-1)\mathtt{j} + u(k-1)\mathtt{k} \\ \boldsymbol{x}_2(k) = \xi_1(k)\mathtt{i} + \xi_2(k)\mathtt{j} + \xi_3(k)\mathtt{k} \end{cases}.$$

In the computational experiments, the network topology was $2 - M_1 - M_2 - 1$, where M_i ($i = 1$, 2) denotes the number of quaternion neurons in the i-th hidden layer. To calculate the gradient of the cost function, the Jacobian of the plant was assumed to be 1 and its magnitude and sign were adjusted using the learning factor η. The Jacobian of the controller was derived from Eq. (21):

$$\frac{\partial u(k)}{\partial \boldsymbol{g}(k)} = [\ \Delta e(k) \quad e(k) \quad \Delta e(k) - \Delta e(k-1)\ ].$$

The initial values of the network parameters were selected randomly from the interval [−0.1, 0.1]. The learning factor was $\eta = 0.1$. The reference model was a linear, first-order system:

$$y_d(k+1) = 0.7 y_d(k) + 0.3 r(k), \qquad (27)$$

and the reference input $r(k)$ was a rectangular wave, where the number of samples within one wave period was 100 and the amplitude of the wave was ±0.5.

Figure 2 shows an example of plant response when controlling the plant using the developed quaternion-neural-network-based self-tuning PID controller. Here, the number of quaternion neurons in the hidden layers is $M_1 = M_2 = 4$, and the plant is linear; the plant parameters are $\alpha_{y1} = 1.3$, $\alpha_{y2} = 0.3$, $\alpha_{add} = 0$, $b_0 = 1$, $\alpha_{u1} = 0.7$, and $c_{non} = 0$. Figure 3 shows an example of the plant response for a nonlinear plant ($\alpha_{add} = 0.03$ and $c_{non} = 0.2$). As shown in Figs. 2 and 3, the controller can achieve the control task of ensuring that the plant output follows the desired plant output by tuning the gain parameters using the output from the quaternion neural network, as its training progresses with the sampling number. Figure 4 shows the relationship between the number of quaternion neurons in the hidden layers and the normalized cost function averaged within one period of the desired plant output. Here, the plant is nonlinear and the normalized cost function is calculated with respect to the 10th period of the desired plant output and is averaged using 100 results for each number of quaternion neurons in the hidden layers. As shown in Fig. 4, the normalized cost function decreases as the number of quaternion neurons in the hidden layers

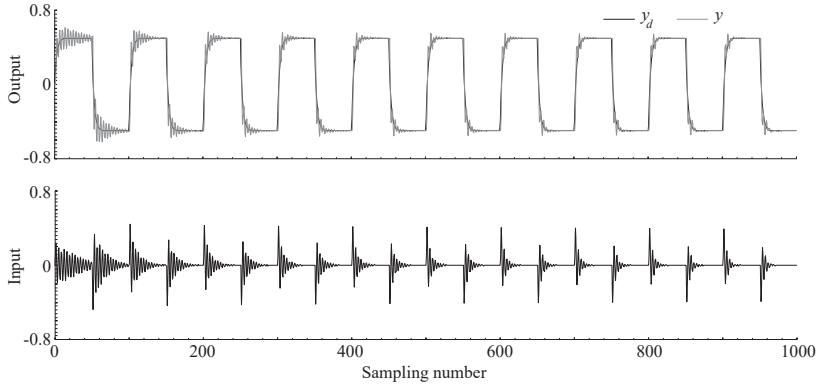

Fig. 2. Experimental result of controlling linear plant using the developed quaternion-neural-network-based self-tuning PID controller (top: plant output, where the black line indicates the desired output $y_d(k)$ and the grey line indicates the plant output $y(k)$; bottom: control input from PID controller).

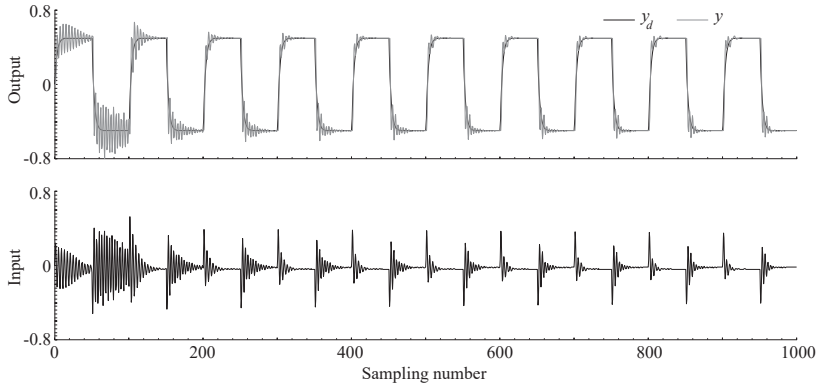

Fig. 3. Experimental result of controlling nonlinear plant using the developed quaternion-neural-network-based self-tuning PID controller (top: plant output, where the black line indicates the desired output $y_d(k)$ and the grey line indicates the plant output $y(k)$; bottom: control input from PID controller).

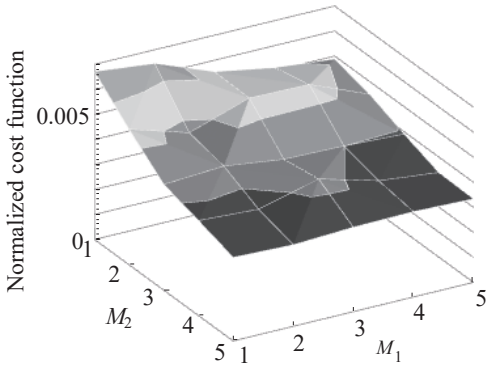

Fig. 4. Relationship between the normalized cost function and the number of quaternion neurons in hidden layers when controlling the nonlinear plant using the quaternion-neural-network-based self-tuning PID controller.

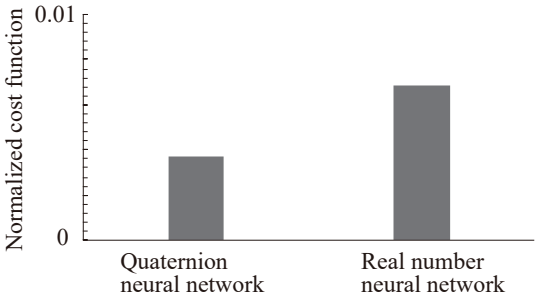

Fig. 5. Comparison of the normalized cost function obtained by the self-tuning PID controllers.

increases. To accurately control the plant output using the quaternion-neural-network-based self-tuning PID controller, a sufficient number of quaternion neurons are required in the hidden layers of the quaternion neural network. As a reference to compare the result of the quaternion neural network, we designed a self-tuning PID controller with a conventional real number neural network of 7-8-8-3 network topology. The topology of the real number neural network was defined so that the number of parameters, such as weights and thresholds, corresponded to that in the quantum neural network. The experimental conditions were the same as those shown in Fig. 3. Figure 5 shows the normalized cost functions of the self-tuning PID controllers. As shown in Fig. 5, the normalized cost function of the quaternion neural network is smaller than that of the real number neural network. This result indicates the effectiveness of the proposed quaternion-neural-network-based self-tuning PID controller.

Figure 6 shows an example of the plant response wherein the plant is nonlinear and random noise generated in the interval [−0.01, 0.01] is added to the control input and the plant output. Figures 7 and 8 show examples of plant response for the variation of the desired plant output wherein the reference input $r(k)$ changes from the rectangular wave to the sine wave in Fig. 7 and the amplitude of the reference input $r(k)$ changes randomly in Fig. 8. Here, the plant is nonlinear and affected by random noise. Figure 9 shows an example of plant response when a disturbance is added to the plant output in the interval [350, 650]. Here, the amplitude of the disturbance is 0.5 and the plant is affected by random noise. As shown in Figs. 6–9, the plant output follows the desired plant output using the learning capability of the quaternion neural network. These results indicate the robustness of the quaternion-neural-network-based self-tuning PID controller.

Figure 10 shows an example of the plant response wherein the plant parameter α_{yi} is time variant as follows:

$$\begin{cases} \alpha_{y_1} = 1.3\,(1 + 0.3\sin(0.01\pi k)) \\ \alpha_{y_2} = -0.3\,(1 + 0.3\sin(0.005\pi k + 0.25\pi)) \end{cases}.$$

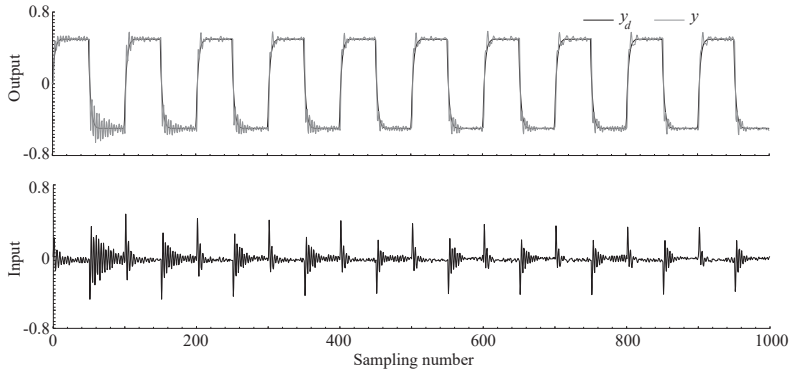

Fig. 6. Experimental result of controlling nonlinear plant using the developed quaternion-neural-network-based self-tuning PID controller, in which random noise is added to control input and plant output (top: plant output, where the black line indicates the desired output $y_d(k)$ and the grey line indicates the plant output $y(k)$; bottom: control input from PID controller).

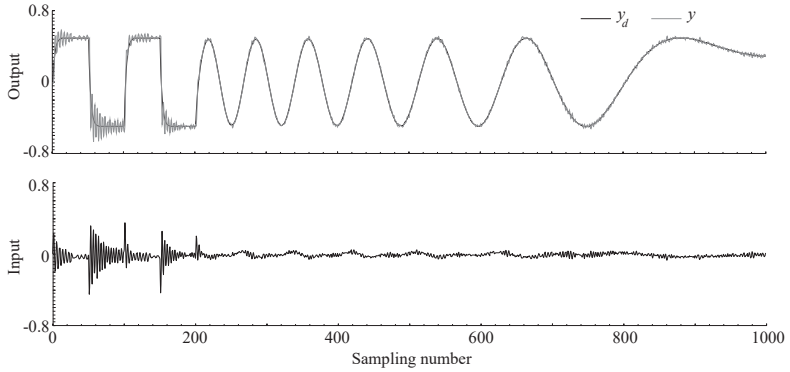

Fig. 7. Experimental result of controlling nonlinear plant (in which random noise is added to control input and plant output) using the developed quaternion-neural-network-based self-tuning PID controller, wherein the reference input changes from the rectangular wave to the sine wave (top: plant output, where the black line indicates the desired output $y_d(k)$ and the grey line indicates the plant output $y(k)$; bottom: control input from PID controller).

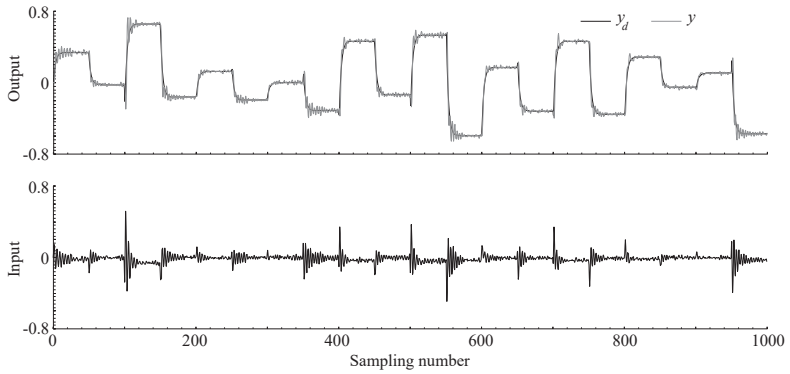

Fig. 8. Experimental result of controlling nonlinear plant (in which random noise is added to control input and plant output) using the developed quaternion-neural-network-based self-tuning PID controller, wherein the reference input amplitude changes randomly (top: plant output, where the black line indicates the desired output $y_d(k)$ and the grey line indicates the plant output $y(k)$; bottom: control input from PID controller).

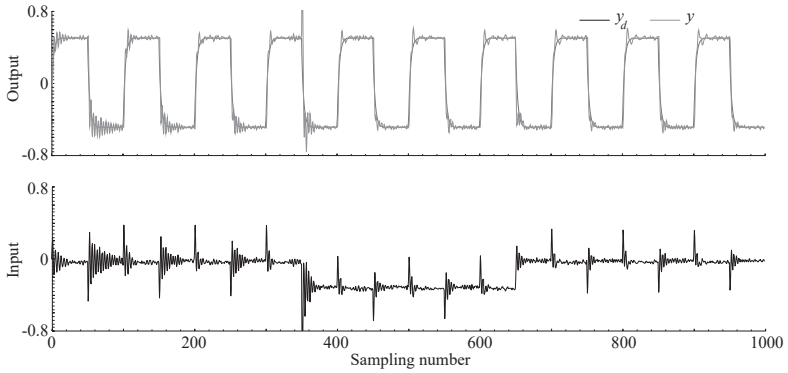

Fig. 9. Experimental result of controlling nonlinear plant (in which random noise is added to control input and plant output) using the developed quaternion-neural-network-based self-tuning PID controller, wherein the constant disturbance is added to the plant in the interval [350, 650] (top: plant output, where the black line indicates the desired output $y_d(k)$ and the grey line indicates the plant output $y(k)$; bottom: control input from PID controller).

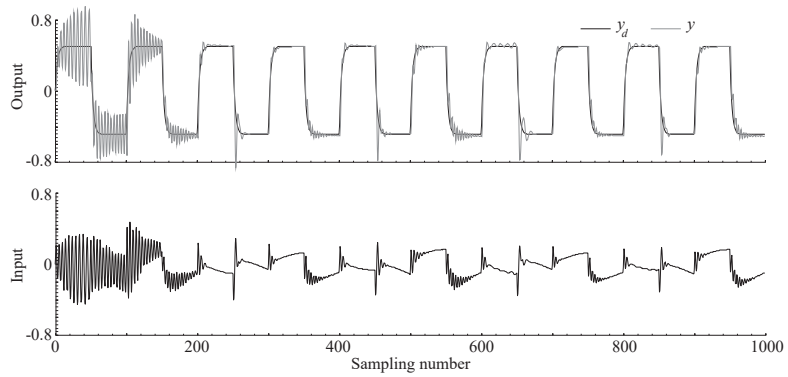

Fig. 10. Experimental result of controlling nonlinear plant using the developed quaternion-neural-network-based self-tuning PID controller, wherein the plant parameter α_{y_i} is time variant (top: plant output, where the black line indicates the desired output $y_d(k)$ and the grey line indicates the plant output $y(k)$; bottom: control input from PID controller).

Using the learning capability of the quaternion neural network, the controller can achieve the control task of ensuring that the plant output follows the desired plant output even if the plant has time variant parameters.

Figure 11 shows an example of the plant response to controlling the other nonlinear plant[18] that is based on the plant used by Narendra and Parthictsarathy[19] and we add a parasitic term as follows:

$$y(k+1) = \frac{2.5y(k)y(k-1)}{1+y^2(k)+y^2(k-1)+y^2(k-2)} + u(k) + 0.8u(k-1). \tag{28}$$

In this experiment, the output from the quaternion neural network was converted from [0, 1] to [0, 0.5]. Using the learning capability of the quaternion neural network, the plant output follows the desired plant output even though little residual vibration was observed in the plant output. These results indicated the effectiveness of the quaternion-neural-network-based self-tuning PID controller.

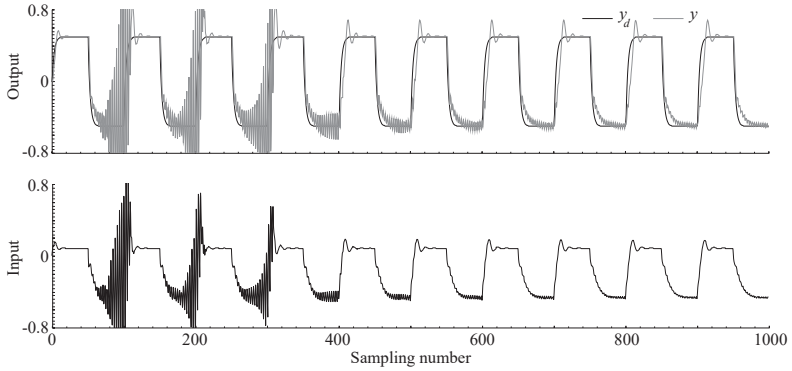

Fig. 11. Experimental result of controlling nonlinear plant [Eq. (28)] using the developed quaternion-neural-network-based self-tuning PID controller (top: plant output, where the black line indicates the desired output $y_d(k)$ and the grey line indicates the plant output $y(k)$; bottom: control input from PID controller).

4. Conclusions

In this study, we investigated the capability of a quaternion neural network and explored its application to control systems. A self-tuning PID controller, the control parameters of which were tuned online by the multilayer quaternion neural network, was designed and its characteristics were evaluated. Computational experiments were conducted to control a discrete-time nonlinear plant, and the simulation results confirmed the feasibility and effectiveness of the proposed quaternion-neural-network-based self-tuning PID controller.

All experiments conducted in this work are computational simulations. Therefore, the application of the quaternion-neural-network-based self-tuning PID controller into hardware should be a future work. One of the applicable control applications is gas/air flow control in gas-sensing systems.[20] The gas-sensing system, namely, electronic nose, which can detect and discriminate aroma/odor, is expected in various fields, such as food and beverage evaluation, environmental monitoring, and chemical quality control. In the gas-sensing system, gas sensors with broad and partially overlapping selectivity are used to detect a wide range of gases, and the sensor responses to various gases under modulating gas flow conditions such as flow profile, velocity, temperature, and humidity, are utilized to discriminate gas samples.[21,22] A servo valve is usually used to modulate the gas flow; however, the static and dynamic characteristics of the valve should be compensated by a feedback controller to achieve the gas flow modulation precisely. As the valve has several nonlinear characteristics caused by mechanical and electromagnetic effects: the self-tuning PID controller presented in this paper can be applied to the control problem in the gas-sensing system.

References

1 T. Nitta (ed.): Complex-Valued Neural Networks—Utilizing High-Dimensional Parameters— (Information Science Reference, Hershey PA, 2009).
2 A. Hirose (ed.): Complex-Valued Neural Networks—Advances and Applications— (IEEE Press, Piscataway NJ, 2013).
3 H. Kusamichi, T. Isokawa, N. Matsui, Y. Ogawa, and K. Maeda: Proc. 2nd ICARA (2004) pp. 101–106.

4 N. Matsui, T. Isokawa, H. Kusamichi, F. Peper, and H. Nishimura: J. Intell. Fuzzy Syst. **15** (2004) 149.
5 P. Arena, R. Caponetto, L. Fortuna, G. Muscato, and M. G. Xibilia: IEICE Trans. Fundamentals E79-A (1996) 1682.
6 S. Buchholz and N. Le Bihan: Proc. 14th EUSIPCO (2006) pp. 1–5.
7 B. C. Ujang, C. C. Took, and D. P. Mandic: IEEE Trans. Neural Netw. **22** (2011) 1193.
8 T. Iura and T. Ogawa: Proc. SICE Annual Conf. **2012** (2012) pp. 1802–1805.
9 F. Shang and A. Hirose: IEEE Trans. Geosci. Remote Sens. **52** (2014) 5693.
10 Y. Cui, K. Takahashi, and M. Hashimoto: Proc. SICE Annual Conf. **2013** (2013) pp. 1381–1386.
11 Y. Cui, K. Takahashi, and M. Hashimoto: Proc. IEEE/SICE SII **2013** (2013) pp. 527–532.
12 D. P. Mandic, C. Jahanchahi, and C. C. Took: IEEE Signal Process. Lett. **18** (2011) 47.
13 T. Isokawa, H. Nishimura, and N. Matsui: Information **3** (2012) 756.
14 P. Arena, L, Fortuna, G. Muscato, and M. G. Xibilia: Neural Netw. **10** (1997) 335.
15 T. Yamada, T. Yabuta, and K. Takahashi: Proc. IECON '91 (1991) pp. 1389–1394.
16 E. Hitzer: Math. Methods Appl. Sci. **36** (2013) 1042.
17 T. Yamada: Artif. Life Robot. **15** (2010) 413.
18 K. Takahashi, H. Kizaki, and M. Hashimoto: Proc. IFAC MIM '2013 (2013) pp. 1003–1008.
19 K. S. Narendra and K. Parthictsarathy: IEEE Trans. Neural Netw. **1** (1990) 4.
20 T. Nakamoto: Sens. Mater. **17** (2005) 365.
21 K. Arshak, E. Moore, G. M. Lyons, F. Harris, and S. Clifford: Sens. Rev. **24** (2004) 181.
22 S. Lakkis, R. Younes, Y. Alayli, and M. Sawan: Sens. Rev. **34** (2014) 24.