Remarks on Control of Robot Manipulator Using Quaternion Neural Network

Kazuhiko Takahashi
Doshisha University, Kyoto, Japan
E-mail: katakaha@mail.doshisha.ac.jp Tel/Fax: +81 774656434

***Abstract*—High–dimensional neural networks, in which all the network parameters, states, signals and activation functions are expressed using hypercomplex numbers, have received increasing attention as solutions to real–world problems in many fields of science and engineering. Quaternion numbers constitute a class of the hypercomplex number system, and several successful applications based on quaternion neural networks have been demonstrated. In this study, the application of a quaternion neural network to control systems is investigated. An adaptive–type servo controller, in which a quaternion neural network output is used as the control input of a plant to ensure the plant output matches the desired output, is presented. The quaternion neural network has a multi–layer feedforward network topology with a split–type quaternion function as the activation function of neurons, and a tapped–delay–line method is used to compose the network input. To train the network parameters by using the gradient error minimisation, a feedback error learning scheme is introduced into the control system. Computational experiments for controlling a two–link robot manipulator by using the proposed quaternion neural network–based controller are conducted, and the simulation results obtained demonstrate the feasibility of using the proposed controller in practical control applications.**

I. Introduction

Over the past decades, numerous studies have been conducted to use the capabilities of neural networks, such as the flexibility, non–linear mapping and learning ability, for solving real–world problems in many fields of science and engineering, and many successful applications have been demonstrated. Meanwhile, high–dimensional neural networks, in which all the network parameters, states, signals and activation functions are expressed using hypercomplex numbers, such as complex numbers and quaternion numbers, have received increasing attention as solutions to such problems [1], [2].

Quaternion numbers, introduced by W. R. Hamilton in 1843, constitute a four–dimensional hypercomplex number system. A quaternion number is defined as a generalised complex numbers: $\boldsymbol{q} = q_0 + q_1\mathtt{i} + q_2\mathtt{j} + q_3\mathtt{k}$, where q_0, q_1, q_2, $q_3 \in \mathbb{R}$ and $\mathtt{i}$, $\mathtt{j}$, and $\mathtt{k}$ are imaginary units that satisfy the following Hamilton rule: $\mathtt{i}^2 = \mathtt{j}^2 = \mathtt{k}^2 = \mathtt{ijk} = -1$, $\mathtt{ij} = -\mathtt{ji} = \mathtt{k}$, $\mathtt{jk} = -\mathtt{kj} = \mathtt{i}$, and $\mathtt{ki} = -\mathtt{ik} = \mathtt{j}$. Neural networks based on quaternion algebras have been proposed, and several engineering applications based on quaternion neural networks have been successfully demonstrated because such networks provide a method for handling the relationship between high–dimensional inputs and outputs naturally, for example, time–series signal processing [3], rigid body control [4], image processing [5], filtering [6], pattern classification [7], and inverse problems [8]. Moreover, the authors have also proposed a control systems application of quaternion neural networks and investigated its effectiveness in servo–level controllers by using mathematical representations of a non–linear plant as the target systems [9], [10].

In this study, an adaptive–type servo controller based on a quaternion neural network, in which the output of the quaternion neural network output is directly used as a control input of a plant to make the plant output match the desired output, is investigated and applied to trajectory control of a robot manipulator. The quaternion neural network has the multi–layer feedforward network topology, and a tapped–delay–line method is used in the input of the quaternion neural network to manage the information that conveys the system dynamics. A split–type quaternion function in which a real–valued function is applied independently to each component of the quaternion number is used as the activation function of the neuron, and the training rule of the network parameters is derived using gradient error minimisation. Computational experiments for controlling the robot manipulator – in this case, a two–link robot manipulator – by using the quaternion neural network–based controller are conducted to evaluate the feasibility of using this network in practical control applications.

II. Dynamics Model of Robot Manipulator

Figure 1 shows a two–link robot manipulator where θ_1 and θ_2 are the joint angles and τ_1 and τ_2 are the control torques. Here, acceleration due to gravity is downward along Y–axis. The dynamics of the two–link robot manipulator can

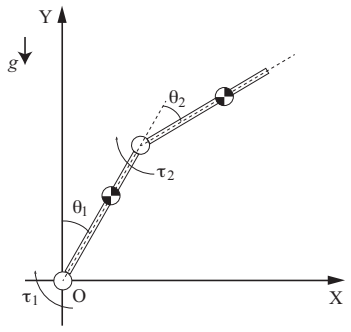


Fig. 1. Two–link robot manipulator.

be represented using the following equation of motion:

$$\boldsymbol{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + \boldsymbol{K}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\boldsymbol{\theta} + \boldsymbol{D}\dot{\boldsymbol{\theta}} + \boldsymbol{\xi}(\dot{\boldsymbol{\theta}}) = \boldsymbol{\tau} \tag{1}$$

where $\boldsymbol{\theta} \in \mathbb{R}^2$ is the vector of the joint angle; $\boldsymbol{M}(\boldsymbol{\theta}) \in \mathbb{R}^{2\times 2}$ is the inertia matrix; $\boldsymbol{K}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) \in \mathbb{R}^{2\times 2}$ represents the Coriolis force, centrifugal force, and gravity force where a sinc function is used in the components of the matrix to extract the variable $\boldsymbol{\theta}$; $\boldsymbol{D} \in \mathbb{R}^{2\times 2}$ is the viscous matrix; $\boldsymbol{\xi}(\dot{\boldsymbol{\theta}}) \in \mathbb{R}^2$ is the vector of the solid friction force; and $\boldsymbol{\tau} \in \mathbb{R}^2$ is the vector of the control torque.

III. Quaternion Neural Network–based Controller

To design the servo controller, the following discrete–time plant is considered as the target system:

$$\begin{aligned}\boldsymbol{y}(t+d) = \boldsymbol{F}\,(&\boldsymbol{y}(t), \boldsymbol{y}(t-1), \cdots, \boldsymbol{y}(t-\nu),\\ &\boldsymbol{u}(t), \boldsymbol{u}(t-1), \cdots, \boldsymbol{u}(t-\mu))\,,\end{aligned} \tag{2}$$

where $\boldsymbol{y} \in \mathbb{R}^q$ is the plant output, $\boldsymbol{u} \in \mathbb{R}^r$ is the control input, t is the sampling number, $\boldsymbol{F}(\cdot)$ is a function representing the plant characteristics, $\nu = n-1$, $\mu = m+d-1$, n and m are the plant orders, and $d \geq 1$ is the plant dead time. Assuming the output error $\boldsymbol{e}(t)$ defined by the difference between the desired output $\boldsymbol{y}_d$ and the plant output will converge to $\boldsymbol{0}$, the control input can be expressed using an arbitrary function $\tilde{\boldsymbol{F}}(\cdot)$ as follows:

$$\begin{aligned}\boldsymbol{u}(t) = \tilde{\boldsymbol{F}}\,(&\boldsymbol{y}_d(t+d), \boldsymbol{y}(t), \boldsymbol{y}(t-1), \cdots, \boldsymbol{y}(t-\nu),\\ &\boldsymbol{u}(t-1), \cdots, \boldsymbol{u}(t-\mu))\,.\end{aligned} \tag{3}$$

By representing the mapping function of the quaternion neural network as $\boldsymbol{F}_{qnn}(\cdot)$, the input–output relationship of the quaternion neural network can be represented as follows:

$$\boldsymbol{o}(t) = \boldsymbol{F}_{qnn}\,(\boldsymbol{s}(t), \boldsymbol{\omega}(t))\,, \tag{4}$$

where $\boldsymbol{s}(t)$ is the input vector to the quaternion neural network, $\boldsymbol{o}(t)$ is the output vector from the quaternion neural network, and $\boldsymbol{\omega}(t)$ is composed of network parameters such as weights and thresholds. Comparing Eqs. (3) and (4), the components of the input vector should consist of $\{\boldsymbol{y}_d(t+d), \boldsymbol{y}(t), \boldsymbol{y}(t-1), \cdots, \boldsymbol{y}(t-\nu), \boldsymbol{u}(t-1), \cdots, \boldsymbol{u}(t-\mu)\}$.

The quaternion neural network uses a multi–layer feedforward network topology. The output of the j-th neuron unit in the h-th layer can be defined as follows:

$$\begin{cases} \boldsymbol{o}_j^{(h)}(t) = \boldsymbol{f}(\boldsymbol{z}_j^{(h)}(t)) \\ \boldsymbol{z}_j^{(h)}(t) = \displaystyle\sum_i \boldsymbol{w}_{ji}^{(h)}(t) \otimes \boldsymbol{o}_i^{(h-1)}(t) + \boldsymbol{\phi}_j^{(h)}(t) \end{cases} \tag{5}$$

where $\boldsymbol{w}_{ji}^{(h)}$ is the weight between the i-th neuron of the $(h-1)$-th layer and the j-th neuron of the h-th layer, $\boldsymbol{\phi}_j^{(h)}$ is the threshold of the j-th neuron of the h-th layer, $\otimes$ denotes the product of quaternion numbers, and $\boldsymbol{f}(\cdot)$ is an activation function of the neuron unit. A split–type quaternion function with respect to a quaternion number $\boldsymbol{q} = q_0 + q_1\mathtt{i} + q_2\mathtt{j} + q_3\mathtt{k}$ is used as the activation function [11]:

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

where and the function $f_l(\cdot)$ is a real–valued non–linear function ($l = 0, 1, 2, 3$).

In training the quaternion neural network, the back–propagation algorithm extended to quaternions is used to conduct the gradient error minimisation with respect to the network parameters. A feedback error learning scheme is introduced to avoid the Jacobian problem in the control system, and thus, the network parameters are updated to minimise the cost function $J(t)$ by considering the dead time of the plant as follows:

$$\boldsymbol{\omega}(t+1) = (1-\sigma)\boldsymbol{\omega}(t-d) + \Delta\boldsymbol{\omega}(t), \tag{6}$$

where t is the iteration number; $\eta, \sigma \in \mathbb{R}^+$ are the training factor and the weight decay factor, respectively; and $\Delta\boldsymbol{\omega}(t)$ is the increment of the network parameters defined as follows:

$$\Delta\boldsymbol{\omega}(t) = -\eta\nabla_{\boldsymbol{\omega}(t-d)}J(t), \tag{7}$$

where

$$J(t) = \frac{1}{2}\boldsymbol{u}_c(t) \otimes \boldsymbol{u}_c^*(t), \tag{8}$$

where $\boldsymbol{u}_c(t)$ is the feedback controller output, and $\boldsymbol{u}_c^*(t)$ is its conjugate.

Owing to use of the feedback error learning scheme, the control system uses the feedback controller in parallel with the quaternion neural network. Therefore, the control input is synthesised by the sum of the quaternion neural network output and the feedback controller output. Considering Eqs. (3) and (4), the input vector of the quaternion neural network $\boldsymbol{s}(t)$ should contain the feedback controller output $\boldsymbol{u}_c(t)$ as one of its components [10].

IV. Controller Design for Robot Manipulator

To define the input to the quaternion neural network, the transfer function of the robot manipulator is considered. To simplify, the control toque $\boldsymbol{\tau}$ is assumed to be proportional to the control input of the robot manipulator $\boldsymbol{u}$: $\boldsymbol{\tau} = \boldsymbol{T}_r\boldsymbol{u}$ where $\boldsymbol{T}_r \in \mathbb{R}^{2\times 2}$ is the coefficient matrix. Representing Eq. (1) by a state space form with the state vector $\boldsymbol{x}(t) = \begin{bmatrix} \boldsymbol{\theta} & \dot{\boldsymbol{\theta}} \end{bmatrix}^{\mathsf{T}}$ and applying zero–order hold of the sampling interval T to the state space form under linear approximations yields

$$\begin{cases} \boldsymbol{x}(t+1) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \\ \boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) \end{cases}, \tag{9}$$

where $\boldsymbol{A} = e^{\boldsymbol{A}_c T}$, $\boldsymbol{A}_c = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{E} \\ -\boldsymbol{M}(\boldsymbol{\theta})^{-1}\boldsymbol{K}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) & -\boldsymbol{M}(\boldsymbol{\theta})^{-1}\boldsymbol{D} \end{bmatrix}$, $\boldsymbol{E}$ is an identity matrix, $\boldsymbol{B} = \int_0^T e^{\boldsymbol{A}_c t}dt\boldsymbol{B}_c$, $\boldsymbol{B}_c = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{M}(\boldsymbol{\theta})^{-1}\boldsymbol{T}_r \end{bmatrix}$, and $\boldsymbol{C} = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \end{bmatrix}$. The transfer function from the control input to the joint angle can be expressed by

$$\boldsymbol{y}(t) = \frac{1}{D(\mathtt{z})}\begin{bmatrix} N_{11}(\mathtt{z}) & N_{12}(\mathtt{z}) \\ N_{21}(\mathtt{z}) & N_{22}(\mathtt{z}) \end{bmatrix}\boldsymbol{u}(t), \tag{10}$$

where $D(\mathtt{z}) = \mathtt{z}^4 + a_3\mathtt{z}^3 + a_2\mathtt{z}^2 + a_1\mathtt{z} + a_0$, $N_{ij}(\mathtt{z}) = b_{3_{ij}}\mathtt{z}^3 + b_{2_{ij}}\mathtt{z}^2 + b_{1_{ij}}\mathtt{z} + b_{0_{ij}}$ $(i, j = 1, 2)$, and $\mathtt{z}$ is a shift operator.

Applying the form of the transfer function ($d = 1$, $n = 4$, $m = 3$) to design the quaternion neural network–based controller described in Section III, the input vector of the quaternion neural network is defined as follows:

$$
\begin{aligned}
\boldsymbol{s}_1(t) &= y_{d_1}(t+1) + y_{d_2}(t+1)\mathtt{i},\\
\boldsymbol{s}_2(t) &= y_1(t) + y_2(t)\mathtt{i} + u_{c_1}(t)\mathtt{j} + u_{c_2}(t)\mathtt{k},\\
\boldsymbol{s}_3(t) &= y_1(t-1) + y_2(t-1)\mathtt{i} + u_1(t-1)\mathtt{j} + u_2(t-1)\mathtt{k},\\
\boldsymbol{s}_4(t) &= y_1(t-2) + y_2(t-2)\mathtt{i} + u_1(t-2)\mathtt{j} + u_2(t-2)\mathtt{k},\\
\boldsymbol{s}_5(t) &= y_1(t-3) + y_2(t-3)\mathtt{i} + u_1(t-3)\mathtt{j} + u_2(t-3)\mathtt{k}.
\end{aligned}
$$

Using the imaginary part of the quaternion neural network output $\boldsymbol{o}_1(t)$, the control input is synthesised as follows:

$$
\begin{aligned}
u_1(t) &= o_{1_2}(t) + u_{c_1}(t),\\
u_2(t) &= o_{1_3}(t) + u_{c_2}(t),
\end{aligned}
$$

where the feedback controller uses the P–control law and its output can be given by

$$
u_{c_i}(t) = G_{p_i}e_i(t) + G_{d_i}T^{-1}(e_i(t) - e_i(t-1)),
$$

where G_{p_i} and G_{d_i} $(i = 1, 2)$ are the gain parameters. In the cost function of Eq. (8), the output of the feedback controller is defined as follows:

$$
\boldsymbol{u}_c(t) = u_{c_1}(t)\mathtt{j} + u_{c_2}(t)\mathtt{k}.
$$

V. Numerical Simulations

In computational experiments, trajectory control of the two–link robot manipulator was conducted using the proposed quaternion neural network–based controller.

The parameters of the robot manipulator are shown in Table I. In the experiments, the force due to solid friction at the joint was given by $S_i\mathtt{sgn}(\dot{\theta}_i)$, where S_i $(i = 1, 2)$ is the amplitude of the solid friction force. The desired joint angles, $y_{d_1}(t)$ and $y_{d_2}(t)$, were calculated in advance by solving the inverse kinematics of the robot manipulator to ensure that the end effector of the robot manipulator tracked the circular trajectory of the radius R. Here, $R = 0.1$, and the trajectory consisted of 1000 data points per round. Hereinafter, one round of the trajectory is called one cycle.

TABLE I
Parameters of robot manipulator

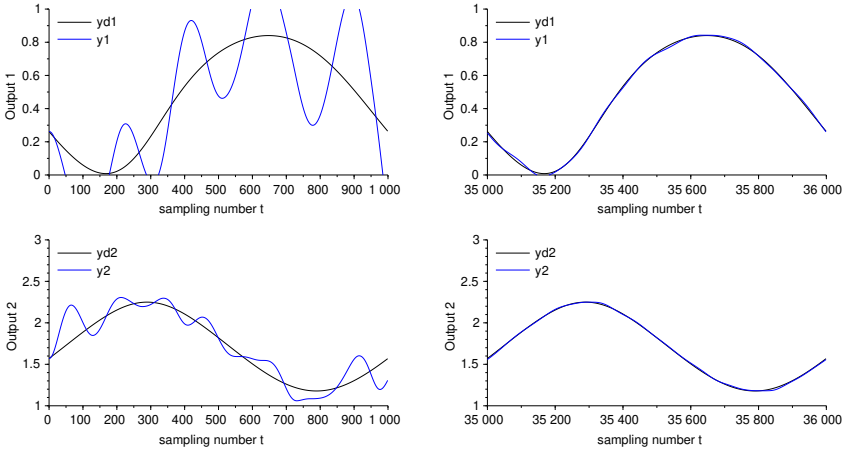
parameter	value
Length of link 1	0.25
Length of link 2	0.25
Distance of centre of mass from joint 1	0.125
Distance of centre of mass from joint 2	0.125
Mass of link 1	0.2
Mass of link 2	0.15
Damping factor 1	0.005
Damping factor 2	0.005
Amplitude of sold friction force 1	0.2
Amplitude of sold friction force 2	0.1
Torque coefficient 1	5.0
Torque coefficient 2	5.0

In the controller, the quaternion neural network contained one hidden layer, and the number of neurons in the hidden layer was five. The real–valued function of the activation function in the neuron was a sigmoid function: $f_l(x) = 1/(1+e^{-x})$ where $l = 0, 1, 2, 3$. To use the output from the quaternion neural network $\boldsymbol{o}_1(t)$ as a part of the control input, it was converted to the range [-1, 1] with gain and shift factors because the output of the sigmoid function was in the range [0, 1]. The initial values of the network parameters were selected randomly from the interval $[-1, 1]$, and the training factors were $\eta = 0.01$ and $\sigma = 10^{-5}$. The gain parameters of the feedback controller were $G_{p_1} = 2.0$, $G_{p_2} = 1.0$, $G_{d_1} = 0.02$, and $G_{d_2} = 0.01$.

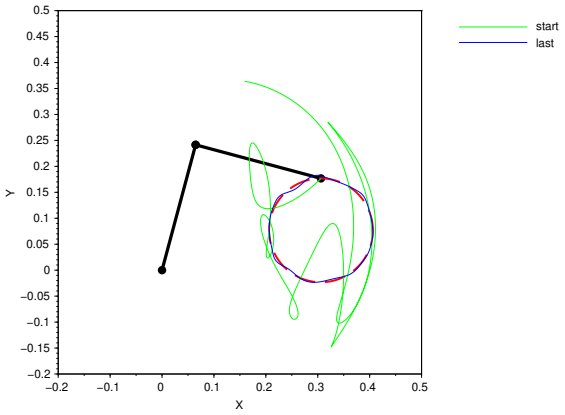
In the simulation experiments, training of the quaternion neural network was terminated when the normalised cost function was smaller than 3×10^{-4} for five continuous cycles, while the total number of cycles in the simulation was 100. Here, the normalised cost function is defined as the mean of the cost function J within one cycle of the desired plant output. Furthermore, the training of the quaternion neural network was restarted with new initial values of the network parameters if the normalised cost function was larger than 10 in the training process.

Figure 2 shows an example of system response where the simulation is terminated at the 36th cycle and the root mean square of the output error is 0.0068. The controller could achieve the control task of ensuring that the joint angles followed the desired output by using the output of the quaternion neural network as its training progressed. A real–valued neural network was used as a controller that can provide the data with which to compare the results of the quaternion neural network. Figure 3 shows an example of the system response obtained using the real–valued neural network. Here, the upper limit of the normalised cost function in the stop criteria was the same as that of the quaternion neural network. To satisfy the stop criteria, an 18–24–2 network topology was required. The simulation was terminated at the 65th cycle and the root mean square of the output error was 0.0106. The output error using the quaternion neural network is less than that when using the real–valued neural network and the normalised cost function of the quaternion neural network decreases faster than that of the real–valued neural network. Using 40 results, the cycle that terminated the training of the quaternion neural network was with a mean of 35.5 (standard deviation of 18.7), while that of the real-valued neural network was with a mean of 63.6 (standard deviation of 4.6). These results indicate that the learning performance of the quaternion neural network is better than that of the real–valued neural network. By contrast, the performance obtained by a complex–valued neural network of 9–7–1 network topology was similar to that of the quaternion neural network.

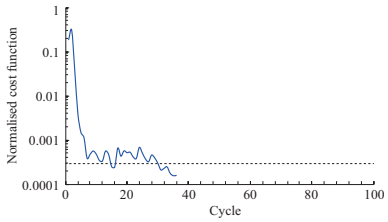
In training of the quaternion neural network, quick propagation (Quickprop) [12] and resilient back propagation (Rprop) [13] were compared with standard back propagation. To sim-



(a) Time responses. In each figure, the top panels show response of joint angle 1, while the bottom panels show response of joint angle 2. Left: system response within the first cycle; right: system response within the last cycle.



(b) Trajectory in $O-XY$ plane.



(c) Training process.

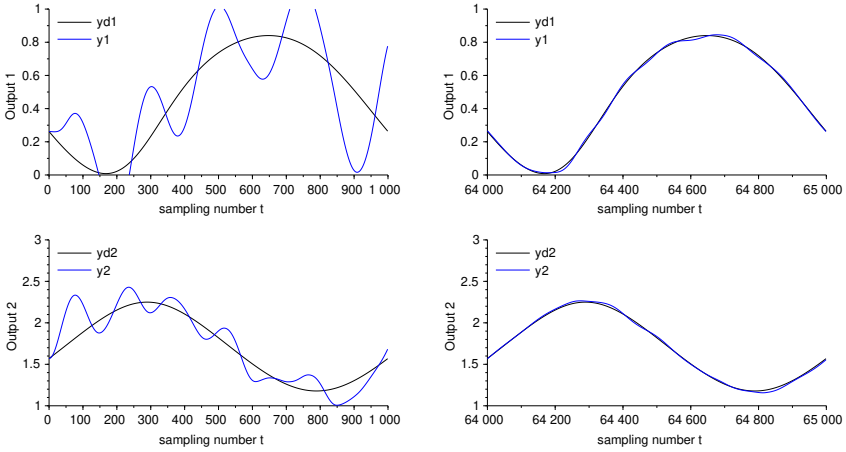
Fig. 2. Example of simulation result where the robot manipulator is controlled using proposed quaternion neural network–based controller.

plify the application of these methods for calculating network parameter increments, each component of the quaternion number representing the network parameter increments and the gradients was processed independently.

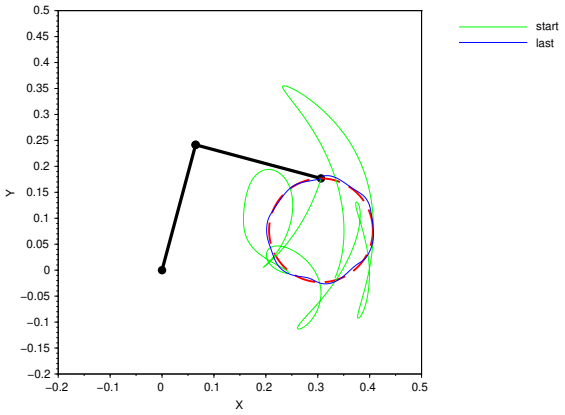
Quickprop is an iterative second–order optimisation algorithm in which the cost function is approximated using a quadratic function for each network parameter independently from the others. Defining $\Omega(t) = \nabla_{\omega_{l_{ji}}} J(t)/(\nabla_{\omega_{l_{ji}}} J(t-1) - \nabla_{\omega_{l_{ji}}} J(t))$, the process of calculating the increment of the network parameter is as follows:

1) If the absolute value of the current gradient is smaller than the previous one but the signs of the gradients are the same, then

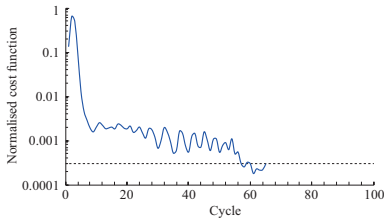
$$\Delta\omega_{l_{ji}}(t) = -\eta\nabla_{\omega_{l_{ji}}} J(t) + \Omega(t)\Delta\omega_{l_{ji}}(t-1),$$



(a) Time responses. In each figure, the top panels show response of joint angle 1, while the bottom panels show response of joint angle 2. Left: system response within the first cycle; right: system response within the last cycle.



(b) Trajectory in $O - XY$ plane.



(c) Training process.

Fig. 3. Example of simulation result where the robot manipulator is controlled using real–valued neural network–based controller.

where η is the training factor.
2) If the signs of the gradients are different, then

$$\Delta\omega_{l_{ji}}(t) = \Omega(t)\Delta\omega_{l_{ji}}(t-1).$$

3) If the absolute value of the current gradient is larger or equal to the previous one, then

$$\Delta\omega_{l_{ji}}(t) = -\eta\nabla_{\omega_{l_{ji}}}J(t) + \alpha\Delta\omega_{l_{ji}}(t-1),$$

where α is the momentum factor.
In the implementation, the maximum growth factor ϕ is used to limit the growth of a step size.

Rprop is a local adaptive learning scheme in which the network parameter increments are calculated by using only the sign of the gradient without using its value. The magnitude of

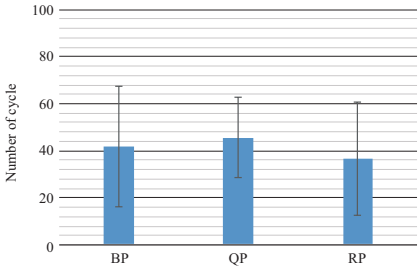


Fig. 4. Comparison of training performance (BP: back propagation, QP: QuickProp, RP: Rprop).

the increment is determined as follows:

$$\Delta\omega_{l_{ji}}(t) = -\eta_{l_{ji}}(t)\mathtt{sgn}\left(\nabla_{\omega_{l_{ji}}}J(t)\right),$$

where the training factor $\eta_{l_{ji}}(t)$ is defined individually for each parameter as follows:

$$\eta_{l_{ji}}(t) = \begin{cases} \min(\eta^+ \times \eta_{l_{ji}}(t-1), \eta_{max}) \\ \qquad (\nabla_{\omega_{l_{ji}}}J(t) \times \nabla_{\omega_{l_{ji}}}J(t-1) > 0) \\ \max(\eta^- \times \eta_{l_{ji}}(t-1), \eta_{min}) \\ \qquad (\nabla_{\omega_{l_{ji}}}J(t) \times \nabla_{\omega_{l_{ji}}}J(t-1) < 0) \\ \eta_{l_{ji}}(t-1) \qquad \text{(otherwise)} \end{cases},$$

where η^+ and η^- are the increase and decrease factors, respectively, that satisfy $0 < \eta^- < 1 < \eta^+$, and η_{max} and η_{min} are the limits of the maximum and minimum step size, respectively. In the implementation, a window of N_w samples, which is continuously renewed, is used to apply Rprop in on–line training.

Figure 4 shows a comparison of the training performance of the algorithms in terms of the number of cycles required to satisfy the stop criterion. Here, the number of cycles was averaged using 50 results, and the error bar depicts the standard deviation of the mean. In the experiments, a momentum term $\beta\Delta\boldsymbol{\omega}(t-1)$ was introduced in the standard back propagation and Rprop. The factors in the standard back propagation were $\eta = 0.01$ and $\beta = 0.1$. The factors of Quickprop were $\eta = 0.01$, $\alpha = 0.5$, and $\phi = 1.75$, while the factors of Rprop were $\eta^+ = 1.2$, $\eta^- = 0.5$, $\eta_{max} = 1$, $\eta_{min} = 10^{-6}$, $\beta = 0.1$, and $N_w = 5$. When using either Quickprop or Rprop, the number of cycle was almost the same as that when using the standard back propagation. This shows that the simplest approach that handles each component of the quaternion number independently in these methods seems to be ineffective to improve the training speed of the quaternion neural network.

VI. Conclusions

In this study, we investigated an adaptive–type servo controller based on a quaternion neural network, in which the network output was directly used as the control input of a plant to ensure that the plant output follows the desired output, and applied the proposed controller to trajectory control of a robot manipulator. The controller was composed of a multi–layer feedforward quaternion neural network, in which a tapped–delay–line method was used as its input, the activation function of the neuron was a split–type quaternion function, and the training of the network parameters was based on the gradient error minimisation. A feedback error learning scheme was introduced in the control system to conduct the training of the quaternion neural network. Computational experiments for controlling a two–link robot manipulator by using the quaternion neural network–based controller were conducted to evaluate the feasibility of using this network in practical control applications. The simulation results demonstrated the effectiveness and characteristics of the quaternion neural network–based controller for trajectory control of a two–link robot manipulator.

Acknowledgement

This research was partially funded by MEXT–Supported Program for the Strategic Research Foundation at Private Universities, 2014–2018.

References

[1] T. Nitta (ed.), Complex–Valued Neural Networks - Utilizing High–Dimensional Parameters -, Information Science Reference, 2009.
[2] A. Hirose (ed.), Complex–Valued Neural Networks - Advances and Applications -, IEEE Press, WILEY, 2013.
[3] P. Arena, R. Caponetto, L. Fortuna, G. Muscato and M. G. Xibilia, "Quaternionic Multilayer Perceptrons for Chaotic Time Series Prediction", *IEICE Transactions Fundamentals*, Vol. E79–A, No. 10, pp. 1682–1688, 1996.
[4] L. Fortuna, G. Muscato and M. G. Xibilia, "A Comparison Between HMLP and HRBF for Attitude Control", *IEEE Transactions on Neural Networks*, Vol. 12, No. 2, pp. 318–328, 2001.
[5] H. Kusamichi, T. Isokawa, N. Matsui, Y. Ogawa and K. Maeda, "A New Scheme Color Night Vision by Quaternion Neural Network", in *Proceedings of the 2nd International Conference on Autonomous Robots and Agents*, pp. 101–106, 2004.
[6] B. C. Ujang, C. C. Took and D. P. Mandic, "Quaternion–Valued Nonlinear Adaptive Filtering", *IEEE Transactions on Neural Networks*, Vol. 22, No. 8, pp. 1193–1206, 2011.
[7] F. Shang and A. Hirose, "Quaternion Neural–Network–Based PolSAR Land Classification in Poincare–Sphere–Space", *IEEE Transactions on Geoscience and Remote Sensing*, Vol. 52, No. 9, pp. 5693–5703, 2014.
[8] T. Iura and T. Ogawa, "Quaternion Neural Network Inversion for Solving Inverse Problems", in *Proceedings of SICE Annual Conference 2012*, pp. 1802–1805, 2012.
[9] K. Takahashi, Y. Hasegawa and M. Hashimoto, "Remarks on Quaternion Neural Network-based Self–tuning PID Controller", in *Proceedings of the 3rd International Symposium on Computer, Consumer and Control*, pp.231–234, 2016.
[10] K. Takahashi, "Remarks on Adaptive–type Hypercomplex–valued Neural Network–based Feedforward Feedback Controller", in *Proceedings of 17th IEEE International Conference on Computer and Information Technology*, pp. 151-156, 2017.
[11] E. Hitzer, "Algebraic Foundations of Split Hypercomplex Nonlinear Adaptive Filtering", *Mathematical Methods in the Applied Sciences*, Vol. 36, No. 9, pp. 1042–1055, 2013.
[12] S. E. Fahlman, "Fast Learning Variations on Back–propagation: An empirical study", in *Proceedings of the 1988 Connectionist Models Summer School*, pp. 38–51, 1989.
[13] M. Riedmiller and H. Braun, "A Direct Adaptive Method for Faster Back–propagation Learning: The RPROP Algorithm", in *Proceedings of International Conference on Neural Networks*, Vol. 1, pp. 586–591, 1993.