Article

Feedforward–Feedback Controller Based on a Trained Quaternion Neural Network Using a Generalised $\mathbb{HR}$ Calculus with Application to Trajectory Control of a Three-Link Robot Manipulator [†]

Kazuhiko Takahashi [1,*], Eri Tano [1] and Masafumi Hashimoto [2]

[1] Department of Information Systems Design, Faculty of Science and Engineering, Doshisha University, Kyoto 610-0394, Japan; cgud1073@mail4.doshisha.ac.jp
[2] Department of Intelligent Information Engineering and Sciences, Faculty of Science and Engineering, Doshisha University, Kyoto 610-0394, Japan; mhashimo@mail.doshisha.ac.jp
* Correspondence: katakaha@mail.doshisha.ac.jp

† This paper is an extended version of our paper published in The 11th International Conference on Advanced Mechatronic Systems (ICAMechS 2021).

Abstract: This study derives a learning algorithm for a quaternion neural network using the steepest descent method extended to quaternion numbers. This applies the generalised Hamiltonian–Real calculus to obtain derivatives of a real–valued cost function concerning quaternion variables and designs a feedback–feedforward controller as a control system application using such a network. The quaternion neural network is trained in real-time by introducing a feedback error learning framework to the controller. Thus, the quaternion neural network-based controller functions as an adaptive-type controller. The designed controller is applied to the control problem of a three-link robot manipulator, with the control task of making the robot manipulator's end effector follow a desired trajectory in the Cartesian space. Computational experiments are conducted to investigate the learning capability and the characteristics of the quaternion neural network used in the controller. The experimental results confirm the feasibility of using the derived learning algorithm based on the generalised Hamiltonian–Real calculus to train the quaternion neural network and the availability of such a network for a control systems application.

Keywords: hypercomplex numbers; quaternion neural network; generalised Hamiltonian–Real calculus; feedforward–feedback controller; robot manipulator

Citation: Takahashi, K.; Tano, E.; Hashimoto, M. Feedforward–Feedback Controller Based on a Trained Quaternion Neural Network Using a Generalised $\mathbb{HR}$ Calculus with Application to Trajectory Control of a Three-Link Robot Manipulator. *Machines* **2022**, *10*, 333. https://doi.org/10.3390/machines10050333

Academic Editors: Mingcong Deng, Takao Sato, Changan Jiang and Lin Meng

Received: 30 March 2022
Accepted: 29 April 2022
Published: 2 May 2022

1. Introduction

Recently, artificial intelligence, including machine learning, has been frequently used to find solutions for intractable problems in science and engineering and other fields, such as social science, economics and finance. Their ability to learn by example makes them capable of applicable solutions in real-world problems [1]. For instance, artificial intelligence could offer approaches for resolving robotics issues in control engineering, especially for autonomous robotics [2]. Neural networks (NNs) have been applied to solve numerous engineering problems over the last half-century due to their flexibility, learning capability and highly complex mappings [3]. Additionally, NNs incorporating deep-learning techniques and reinforcement learning integrating deep NNs are currently being used to handle challenging problems in the real world. They have demonstrated successful applications in many engineering fields [4,5]. Although deep learning is commonly utilised to deal with such problems, high-dimensional NNs, where all the network parameters and signals are based on complex and hypercomplex numbers including quaternion and octonion, have become an attractive alternative for handling such difficulties more efficiently and easily. Since the high-dimensional representation using such numbers can easily address multi-dimensional correlated data as a single entity and has many advantages, including parameter and computational efficiency [6], high-dimensional NNs have been explored to solve multi-dimensional and complicated problems [7–12].

A servo-level controller using a high-dimensional NN based on quaternion numbers was proposed in our previous work. Its characteristics for controlling nonlinear dynamical systems were investigated as a control system application of quaternion NNs (QNNs) [13–15]. Moreover, we found that the QNN outperformed the other networks in the learning and control of a three-link robot manipulator [16], such as real- and hypercomplex-valued NNs. It applies the steepest descent against the real-valued cost function defined by the error between the desired and system output to train the QNN. Since the cost function's derivative to the quaternion variables representing the network parameters was required to derive the QNN's learning algorithm, the derivative was calculated using the pseudo-derivative that calculates component-wise real derivatives concerning the real variables of the quaternion number. The pseudo-derivative is often used to derive the learning algorithm in many QNN applications, e.g., time-series signal processing, image processing, speech processing and pattern classification [10,17]. However, the formal derivatives of a real-valued function for the quaternion variables, including their involution, have been proposed using Hamiltonian–Real ($\mathbb{HR}$) calculus, which is similar to the Wirtinger calculus relating complex numbers to quaternion numbers, to develop learning algorithms for signal processing using quaternion numbers [18]. Furthermore, a generalised form of the $\mathbb{HR}$ calculus called generalised $\mathbb{HR}$ (G$\mathbb{HR}$) calculus has been proposed to develop a more suitable learning algorithm [19]. Despite an increasing interest in learning algorithms based on the G$\mathbb{HR}$ calculus [20–23], its applicability on QNNs has not been adequately investigated. Moreover, practical, real-world applications using QNNs trained by such algorithms have not been fully explored, although such a QNN has been applied to control of a robot manipulator in a previous study [24].

Considering the aforementioned research gap, the main aim of this study is to determine how the specific properties of the learning algorithm based on the G$\mathbb{HR}$ calculus affect the QNN in control systems. In this study, the learning algorithm of a multi-layer QNN based on the G$\mathbb{HR}$ calculus is derived, and the possibility of using such a QNN in control system applications is explored. A servo controller based on the QNN is presented, where the input to the QNN consists of several input–output sets of the plant to handle plant dynamics and the output from the QNN is used to synthesise the control input to the plant. A feedback error learning (FEL) [25] framework extended to the quaternion numbers is employed to train the QNN-based controller. We used the QNN-based controller to perform computational experiments on the trajectory-tracking control of a three-link robot manipulator. We evaluated the feasibility and characteristics of the QNN trained by the learning algorithm based on the G$\mathbb{HR}$ calculus.

2. Feedforward–Feedback Controller Based on a Quaternion Neural Network

2.1. Generalised $\mathbb{HR}$ Calculus

Suppose the algebra of the quaternion numbers is $\mathbb{H} = \{q \mid q = q_0 + q_1\imath + q_2\jmath + q_3\kappa, q_0, q_1, q_2, q_3 \in \mathbb{R}\}$ using four real numbers and the basis $\{1, \imath, \jmath, \kappa\}$, where the units $\imath$, $\jmath$ and κ satisfy Hamilton's rules: $\imath^2 = \jmath^2 = \kappa^2 = \imath\jmath\kappa = -1, \imath\jmath = -\jmath\imath = \kappa, \jmath\kappa = -\kappa\jmath = \imath$ and $\kappa\imath = -\imath\kappa = \jmath$. It forms a four-dimensional noncommutative, associative normed division algebra over real numbers.

Some basics of the $\mathbb{HR}$ calculus [18] are summarised first. Let the transformation for any quaternion number q be denoted as $q^\mu = \mu q \mu^{-1}$ where μ is any nonzero quaternion number. The real variables of the quaternion number q_i $(i = 0, 1, 2, 3)$ can be expressed using the quaternion involutions: $q_0 = \dfrac{1}{4}(q + q^\imath + q^\jmath + q^\kappa)$, $q_1 = \dfrac{1}{4\imath}(q + q^\imath - q^\jmath - q^\kappa)$, $q_2 = \dfrac{1}{4\jmath}(q - q^\imath + q^\jmath - q^\kappa)$ and $q_3 = \dfrac{1}{4\kappa}(q - q^\imath - q^\jmath + q^\kappa)$. Let $f : \mathbb{H} \to \mathbb{H}$ be the quaternion-valued function $f(q) = f_0(q_0, q_1, q_2, q_3) + f_1(q_0, q_1, q_2, q_3)\imath + f_1(q_0, q_1, q_2, q_3)\jmath + f_3(q_0, q_1, q_2, q_3)\kappa$, the function f is real differentiable when the components f_i $(i = 0, 1, 2, 3)$

are differentiable as functions of real variables q_i ($i = 0, 1, 2, 3$). If the function f is real differentiable, the $\mathbb{HR}$ derivatives of the function f to the involution are defined as follows:

$$\begin{cases} \dfrac{\partial f}{\partial q} = \dfrac{1}{4}\left(\dfrac{\partial f}{\partial q_0} - \dfrac{\partial f}{\partial q_1}\imath - \dfrac{\partial f}{\partial q_2}\jmath - \dfrac{\partial f}{\partial q_3}\kappa\right) \\ \dfrac{\partial f}{\partial q^\imath} = \dfrac{1}{4}\left(\dfrac{\partial f}{\partial q_0} - \dfrac{\partial f}{\partial q_1}\imath + \dfrac{\partial f}{\partial q_2}\jmath + \dfrac{\partial f}{\partial q_3}\kappa\right) \\ \dfrac{\partial f}{\partial q^\jmath} = \dfrac{1}{4}\left(\dfrac{\partial f}{\partial q_0} + \dfrac{\partial f}{\partial q_1}\imath - \dfrac{\partial f}{\partial q_2}\jmath + \dfrac{\partial f}{\partial q_3}\kappa\right) \\ \dfrac{\partial f}{\partial q^\kappa} = \dfrac{1}{4}\left(\dfrac{\partial f}{\partial q_0} + \dfrac{\partial f}{\partial q_1}\imath + \dfrac{\partial f}{\partial q_2}\jmath - \dfrac{\partial f}{\partial q_3}\kappa\right) \end{cases}, \tag{1}$$

and its conjugates are as follows:

$$\begin{cases} \dfrac{\partial f}{\partial q^*} = \dfrac{1}{4}\left(\dfrac{\partial f}{\partial q_0} + \dfrac{\partial f}{\partial q_1}\imath + \dfrac{\partial f}{\partial q_2}\jmath + \dfrac{\partial f}{\partial q_3}\kappa\right) \\ \dfrac{\partial f}{\partial q^{\imath *}} = \dfrac{1}{4}\left(\dfrac{\partial f}{\partial q_0} + \dfrac{\partial f}{\partial q_1}\imath - \dfrac{\partial f}{\partial q_2}\jmath - \dfrac{\partial f}{\partial q_3}\kappa\right) \\ \dfrac{\partial f}{\partial q^{\jmath *}} = \dfrac{1}{4}\left(\dfrac{\partial f}{\partial q_0} - \dfrac{\partial f}{\partial q_1}\imath + \dfrac{\partial f}{\partial q_2}\jmath - \dfrac{\partial f}{\partial q_3}\kappa\right) \\ \dfrac{\partial f}{\partial q^{\kappa *}} = \dfrac{1}{4}\left(\dfrac{\partial f}{\partial q_0} - \dfrac{\partial f}{\partial q_1}\imath - \dfrac{\partial f}{\partial q_2}\jmath + \dfrac{\partial f}{\partial q_3}\kappa\right) \end{cases}. \tag{2}$$

Consequently, the $\mathbb{GHR}$ derivatives of the function f with respect to q^μ and $q^{\mu *}$ ($\mu \neq 0$, $\mu \in \mathbb{H}$) are defined as follows [19]:

$$\begin{cases} \dfrac{\partial f}{\partial q^\mu} = \dfrac{1}{4}\left(\dfrac{\partial f}{\partial q_0} - \dfrac{\partial f}{\partial q_1}\imath^\mu - \dfrac{\partial f}{\partial q_2}\jmath^\mu - \dfrac{\partial f}{\partial q_3}\kappa^\mu\right) \\ \dfrac{\partial f}{\partial q^{\mu *}} = \dfrac{1}{4}\left(\dfrac{\partial f}{\partial q_0} + \dfrac{\partial f}{\partial q_1}\imath^\mu + \dfrac{\partial f}{\partial q_2}\jmath^\mu + \dfrac{\partial f}{\partial q_3}\kappa^\mu\right) \end{cases}, \tag{3}$$

where $\{1, \imath^\mu, \jmath^\mu, \kappa^\mu\}$ is the the generalised basis. Some properties of the $\mathbb{GHR}$ derivatives are given as follows:

$$\text{product rule:} \quad \frac{\partial (fg)}{\partial q} = \frac{\partial f}{\partial q^g} g + f\frac{\partial g}{\partial q}, \frac{\partial (fg)}{\partial q^*} = \frac{\partial f}{\partial q^{g*}} g + f\frac{\partial g}{\partial q^*}, \tag{4}$$

$$\text{rotation rule:} \quad \left(\frac{\partial f}{\partial q}\right)^\mu = \frac{\partial f^\mu}{\partial q^\mu}, \left(\frac{\partial f}{\partial q^*}\right)^\mu = \frac{\partial f^\mu}{\partial q^{\mu *}}, \tag{5}$$

$$\text{chain rule:} \quad \frac{\partial f(g(q))}{\partial q} = \sum_{\mu \in \{1,\imath,\jmath,\kappa\}} \frac{\partial f}{\partial g^\mu}\frac{\partial g^\mu}{\partial q}, \frac{\partial f(g(q))}{\partial q^*} = \sum_{\mu \in \{1,\imath,\jmath,\kappa\}} \frac{\partial f}{\partial g^\mu}\frac{\partial g^\mu}{\partial q^*}. \tag{6}$$

where the functions $f : \mathbb{H} \to \mathbb{H}$ and $g : \mathbb{H} \to \mathbb{H}$ are real differentiable.

Using the $\mathbb{GHR}$ derivatives, Taylor's theorem for quaternion-valued functions of quaternion variables can be formulated as a useful tool in calculus. The first-order Taylor series expansion of the function $f(q) \in \mathbb{H}$ is derived as follows:

$$f(q + dq) = f(q) + \sum_{\mu \in \{1,\imath,\jmath,\kappa\}} \frac{\partial f(q)}{\partial q^\mu} dq^\mu + O(dq^2). \tag{7}$$

If and only if the direction of the steepest descent dq is set to $dq = -\eta\left(\frac{\partial f}{\partial q}\right)^* = -\eta\frac{\partial f}{\partial q^*}$ with the learning factor $\eta \in \mathbb{R}^+$, the following relationship can be obtained:

$$\begin{aligned} f(q+dq) - f(q) &\approx \sum_{\mu\in\{1,\imath,\jmath,\kappa\}} \tfrac{\partial f(q)}{\partial q^\mu} dq^\mu \\ &= \mathtt{Re}\left(4\tfrac{\partial f}{\partial q}\right)dq \le -4\eta\left\|\tfrac{\partial f}{\partial q}\right\|^2 < 0. \end{aligned} \tag{8}$$

This yields the steepest descent method for quaternion numbers, which could be used as a learning algorithm for the QNN. Since the learning factor η defines a step size in the steepest descent, it functions to adjust the learning speed of the QNN.

2.2. *Quaternion Neural Network*

Considering a three-layer QNN with an input layer, a hidden layer and an output layer, the network's input–output relationship can be given as:

$$\sigma_l = \sum_j \omega_{lj}\psi(\sum_i w_{ji}s_i), \tag{9}$$

where $s_i \in \mathbb{H}$ is an external input to the input layer's i-th unit, $w_{ji} \in \mathbb{H}$ is the connection weight between the input layer's i-th unit and the hidden layer's j-th unit, $\omega_{lj} \in \mathbb{H}$ is the connection weight between the hidden layer's j-th unit and the output layer's l-th unit, $\sigma_l \in \mathbb{H}$ is the output layer's l-th output and $\psi : \mathbb{H} \to \mathbb{H}$ is the unit's activation function in the hidden layer. The unit's threshold in the hidden and output layers can be treated as a connection weight against a constant input.

Assuming that $\epsilon_l \in \mathbb{H}$ is the output error defined as $\epsilon_l = t_l - \sigma_l$, where $t_l \in \mathbb{H}$ is the teaching signal of the output layer's l-th unit, the QNN's connection weights are trained using the steepest descent method that minimises the following cost function defined using the norm of the output error:

$$E = \frac{1}{2}\sum_l \|\epsilon_l\|^2. \tag{10}$$

Although the cost function is a real-valued function since the norm of the quaternion number is a real number, it is also a function of the quaternion number and its conjugate due to the definition of the norm: $\|q\| = \sqrt{qq^*}$ where $q \in \mathbb{H}$. Calculating the cost function gradient to the connection weights using the G$\mathbb{HR}$ calculus updates the connection weights according to the steepest descent method extended to quaternion numbers derived as follows:

$$\begin{aligned} \omega_{lj} &= \omega_{lj} - \eta\tfrac{\partial E}{\partial \omega_{lj}^*} \\ &= \omega_{lj} - \eta\left[-\tfrac{1}{2}\left\{\frac{\partial \sigma_l}{\partial \omega_{lj}^{\epsilon_l^* *}}\epsilon_l^* + \epsilon_l\frac{\partial \sigma_l^*}{\partial \omega_{lj}^*}\right\}\right] \\ &= \omega_{lj} + \tfrac{1}{4}\eta\epsilon_l\nu_j^*, \end{aligned} \tag{11}$$

$$\begin{aligned} w_{ji} &= w_{ji} - \eta\tfrac{\partial E}{\partial w_{ji}^*} \\ &= w_{ji} - \eta\left[-\tfrac{1}{2}\sum_l\left\{\frac{\partial \sigma_l}{\partial w_{ji}^{\epsilon_l^* *}}\epsilon_l^* + \epsilon_l\frac{\partial \sigma_l^*}{\partial w_{ji}^*}\right\}\right] \\ &= w_{ji} + \tfrac{1}{4}\eta\sum_l\left\{\mathtt{Re}\left[\epsilon_l\left(\tfrac{\partial \nu_j}{\partial x_{j_0}}\right)^*\omega_{lj}^*\right] + \mathtt{Re}\left[\epsilon_l\left(\tfrac{\partial \nu_j}{\partial x_{j_1}}\right)^*\omega_{lj}^*\right]\imath\right. \\ &\qquad \left. + \mathtt{Re}\left[\epsilon_l\left(\tfrac{\partial \nu_j}{\partial x_{j_2}}\right)^*\omega_{lj}^*\right]\jmath + \mathtt{Re}\left[\epsilon_l\left(\tfrac{\partial \nu_j}{\partial x_{j_3}}\right)^*\omega_{lj}^*\right]\kappa\right\}s_i^*. \end{aligned} \tag{12}$$

where $v_j = \psi(x_j)$ is the hidden layer's j-th output, and $x_j = \sum_i w_{ji} s_i$ is an internal state of the hidden layer's j-th unit. These update rules are the back–propagation algorithm extended to quaternion numbers.

2.3. Feedforward–Feedback Controller

In this section, the QNN is used to design a controller which makes a plant output follow a reference signal. The control input to the plant is composed of both the QNN output and a feedback controller output in this controller because a FEL framework is introduced to train the controller's QNN. To simplify designing the controller, the following discrete-time linear plant is considered:

$$y(k+d) = \sum_{i=0}^{n-1} \alpha_i y(k-i) + \sum_{i=0}^{m+d-1} \beta_i u(k-i), \tag{13}$$

where $y \in \mathbb{H}$ is the plant output, $u \in \mathbb{H}$ is the control input, $\alpha_i \in \mathbb{H}$ and $\beta_i \in \mathbb{H}$ are the plant's coefficients, n and m are the plant orders, d is the dead time and k is the sampling number. The control input consists of the QNN output σ and the feedback controller output v, such as $u(k) = \sigma(k) + v(k)$. In defining the control error between the reference signal r and the plant output y: $e(k) = r(k) - y(k)$, it can be rewritten using Equation (6) as:

$$\begin{aligned} e(k+d) &= r(k+d) - y(k+d) \\ &= \beta_0\{\boldsymbol{\xi}^{\mathrm{T}} \boldsymbol{s}(k) - \sigma(k)\}, \end{aligned} \tag{14}$$

where $(\cdot)^{\mathrm{T}}$ is the transpose operator,

$$\boldsymbol{\xi} = \beta_0^{-1} \begin{bmatrix} 1 & -\alpha_0 & \cdots & -\alpha_{n-1} & -\beta_1 & \cdots & \beta_{p-1} & -\beta_0 \end{bmatrix}^{\mathrm{T}}, \tag{15}$$

$$\boldsymbol{s}(k) = \begin{bmatrix} r(k+d) & y(t) & \cdots & y(k-n+1) & u(k-1) & \cdots & u(k-p+1) & v(k) \end{bmatrix}^{\mathrm{T}}, \tag{16}$$

and $p = m + d$. Assuming the control error becomes zero, the three-layer QNN is designed using the vector $\boldsymbol{s}(k)$ as the network's input:

$$\sigma(k) = \boldsymbol{\omega}^{\mathrm{T}}(k) \boldsymbol{\psi}(\boldsymbol{W}(k)\boldsymbol{s}(k)), \tag{17}$$

where the component of the matrix $\boldsymbol{W} \in \mathbb{H}^{M \times L}$ comprises the connection weights between the input and hidden layers, and the component of the vector $\boldsymbol{\omega} \in \mathbb{H}^M$ consists of the connection weights between the hidden and output layers. Here L is the number of units in the input layer defined as $L = n + p + 1$, and M is the number of units in the hidden layer.

The connection weights are updated with the learning algorithm using Equations (11) and (12) at every sampling number, trained in real-time. Thus, the QNN acts as an adaptive controller. When the generalised δ-rule is applied to the QNN training by considering the plant's dead time, the connection weight training in the controller is performed as follows:

$$\boldsymbol{\omega}(k+1) = \boldsymbol{\omega}(k-d) - \eta \frac{\partial E(k)}{\partial \boldsymbol{\omega}^*(k)}, \tag{18}$$

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k-d) - \eta \frac{\partial E(k)}{\partial \boldsymbol{W}^*(k)}. \tag{19}$$

where $(\cdot)^*$ is the quaternion conjugate operator. The cost function, Equation (10), is defined using the feedback controller output instead of the output error; namely, the feedback controller output is minimised during the QNN training process according to the FEL. While the feedback controller is expected to keep the control system stable at the beginning of the QNN training process, the QNN comes to be a feedforward controller when the QNN training is ideally completed. Thus, such a controller is called a feedforward–feedback controller [26]. Figure 1 shows the resulting control system schematic.

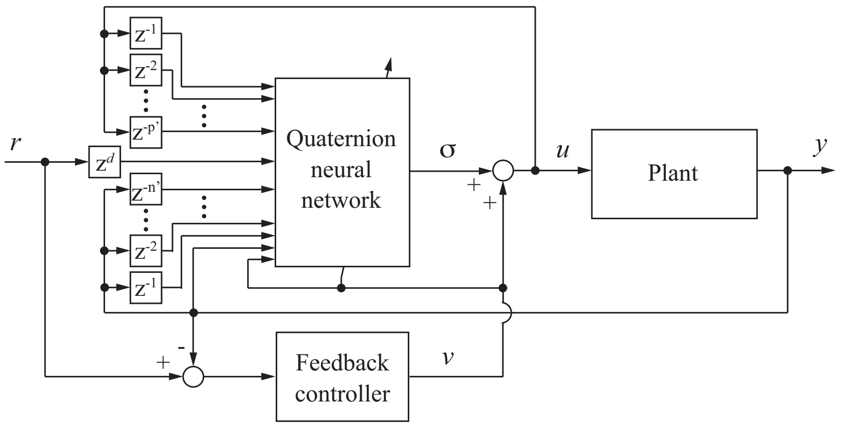

Figure 1. Schematic of a QNN-based feedforward–feedback controller, where **z** is a shift operator, $n' = n - 1$ and $p' = p - 1$.

2.4. Remarks on the Stability Condition of the Controller

The stability analysis of the QNN-based feedforward–feedback controller is important. However, it is generally difficult, owing to the nonlinearity of the controller and the plant, the noncommutative multiplication of quaternion numbers and the analyticity of the quaternion function used in the activation function. Therefore, the local stability condition of the controller is discussed by introducing the following assumptions:

(i) The plant is represented by Equation (13), where the orders of the plant, the dead time and the sign of the high–frequency gain are known.
(ii) The QNN's activation function is a split-type function using a component-wise linear function even though it is not analytic in the field of quaternion number.
(iii) The feedback controller is a P–controller, and the QNN's connection weights allow the feedback controller output to be sufficiently small.

Using assumption (ii), the QNN, Equation (17) and the connection weight's learning algorithm, Equations (18) and (19) can be rewritten as follows:

$$\sigma(k) = \boldsymbol{\omega}^{\mathrm{T}}(k)\boldsymbol{W}(k)\boldsymbol{s}(k), \tag{20}$$

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k-d) + \frac{1}{2}\eta\boldsymbol{\omega}^{*}(k)v(k)\boldsymbol{s}^{\mathrm{H}}(k), \tag{21}$$

$$\boldsymbol{\omega}(k+1) = \boldsymbol{\omega}(k-d) + \frac{1}{4}v(k)\left\{\boldsymbol{s}^{\mathrm{H}}(k)\boldsymbol{W}^{\mathrm{H}}(k)\right\}^{\mathrm{T}}, \tag{22}$$

where $(\cdot)^{\mathrm{H}}$ is the Hermitian operator. Using assumption (i), the control error is represented by Equation (14). Substituting Equation (20) into Equation (14) yields:

$$\begin{aligned} e(k+d) &= \beta_0\{\boldsymbol{\xi}^{\mathrm{T}} - \boldsymbol{\omega}^{\mathrm{T}}(k)\boldsymbol{W}(k)\}\boldsymbol{s}(k) \\ &= \beta_0\boldsymbol{\zeta}^{\mathrm{T}}(k)\boldsymbol{s}(k), \end{aligned} \tag{23}$$

where $\boldsymbol{\zeta}^{\mathrm{T}}(k) = \boldsymbol{\xi}^{\mathrm{T}} - \boldsymbol{\omega}^{\mathrm{T}}(k)\boldsymbol{W}(k)$ denotes the parameter error vector. Substituting Equations (21) and (22) into the parameter error vector and using the condition $\|v(k)\| \ll 1$ obtained by assumption (iii) yields:

$$\begin{aligned} \boldsymbol{\zeta}^{\mathrm{T}}(k+1) &= \boldsymbol{\xi}^{\mathrm{T}} - \boldsymbol{\omega}^{\mathrm{T}}(k+1)\boldsymbol{W}(k+1) \\ &\approx \boldsymbol{\zeta}^{\mathrm{T}}(k-d)\Big\{\boldsymbol{E} - \tfrac{1}{4}h\eta\boldsymbol{\Omega}(k-d)\Big\}, \end{aligned} \tag{24}$$

where

$$\boldsymbol{\Omega}(k-d) = \boldsymbol{s}(k-d)\Big\{\boldsymbol{s}^{\mathrm{H}}(k)\boldsymbol{W}^{\mathrm{H}}(k)\boldsymbol{W}(k-d) + 2v^{-1}(k)\boldsymbol{\omega}^{\mathrm{T}}(k-d)\boldsymbol{\omega}^{*}(k)v(k)\boldsymbol{s}^{\mathrm{H}}(k)\Big\}, \quad (25)$$

$\boldsymbol{E}$ is the identity matrix, and the feedback controller is assumed $v(k) = h\beta_0^{-1}e(k)$ where $h \in \mathbb{R}^{+}$ is the feedback gain. Assuming the function $V(k) = \|\boldsymbol{\zeta}(k)\|^2 \geq 0$ as the Lyapunov function candidate yields:

$$\begin{aligned} \Delta V(k) &= V(k+1) - V(k-d) \\ &= -\tfrac{1}{4}h\eta\boldsymbol{\zeta}^{\mathrm{T}}(k-d)\boldsymbol{\Phi}(k-d)\boldsymbol{\zeta}^{*}(k-d), \end{aligned} \quad (26)$$

where

$$\boldsymbol{\Phi}(k-d) = \boldsymbol{\Omega}(k-d) + \boldsymbol{\Omega}^{\mathrm{H}}(k-d) - \frac{1}{4}h\eta\boldsymbol{\Omega}(k-d)\boldsymbol{\Omega}^{\mathrm{H}}(k-d). \quad (27)$$

Consequently, the stability of the parameter error vector $\boldsymbol{\zeta}(k)$ is guaranteed when $\Delta V(k) \leq 0$ in Equation (26), namely, the matrix $\boldsymbol{\Phi}(k-d)$ is positive semi-definite. The matrix $\boldsymbol{\Phi}(k-d)$ is Hermitian because of its definition, so its eigenvalues are real numbers. This result clarifies the stability characteristics qualitatively, i.e., the stability depends on the learning factor, the feedback gain and the connection weight, which is strongly related to both the initial value and the convergence trajectory of the connection weight. Therefore, the brute force approach which changes the learning factor, the feedback gain and the connection weight's initial values is necessary to find the stability condition quantitatively. Additionally, the convergence of the parameter error vector $\boldsymbol{\zeta}(k)$ would not always satisfy the relationship $\boldsymbol{\xi}^{\mathrm{T}} = \boldsymbol{\omega}^{\mathrm{T}}(k)\boldsymbol{W}(k)$ even though the feedback controller output becomes sufficiently small; namely, the control error approaches zero. The connection weight's learning algorithm for the QNN using the steepest descent method primarily minimises the feedback controller output by matching the response between the plant output and the reference signal but does not estimate the true values of the plant parameter as in the adaptive control's parameter identification. This shows that the QNN has a redundancy to realise the controller.

3. Computational Experiments

To investigate the feasibility and the characteristics of the QNN trained using the learning algorithm based on the $\mathbb{GHR}$ calculus in control systems application, computational experiments are performed to control a robot manipulator.

3.1. Robot Manipulator

The robot manipulator which has three joints, rotation–pivot–pivot type, shown in Figure 2, is the target plant controlled using the QNN-based feedforward–feedback controller. The motion equation of the robot manipulator is given as:

$$\boldsymbol{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + \boldsymbol{C}\dot{\boldsymbol{\theta}} + \boldsymbol{K}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \boldsymbol{\tau}, \quad (28)$$

where $\boldsymbol{\theta} \in \mathbb{R}^3$ is the joint angle vector, $\boldsymbol{\tau} \in \mathbb{R}^3$ is the control torque vector, $\boldsymbol{M}(\boldsymbol{\theta}) \in \mathbb{R}^{3\times3}$ is the inertia matrix, $\boldsymbol{C} = \mathtt{diag}(c_1, c_2, c_3) \in \mathbb{R}^{3\times3}$ is the viscous matrix, and $\boldsymbol{K}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) \in \mathbb{R}^3$ is the nonlinear force vector including the Coriolis force, centrifugal force, gravitational force and solid frictional force at the joints.

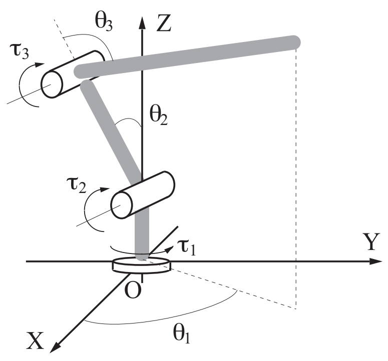

Figure 2. Schematic of a three-link robot manipulator.

The control task is executed in the joint space, although the control objective for the robot manipulator is to ensure that the end-effector tracks the desired trajectory given in the workspace (Cartesian space). The desired position on the trajectory is transformed into the desired joint angles in advance through the robot manipulator's inverse kinematics model. The movable range of each joint angle is restricted to avoid singular configurations and multiple solutions of the robot manipulator.

3.2. Controller Condition

In the QNN-based feedforward–feedback controller, the QNN has nine units in the input layer and one unit in the output layer. As described in Section 2.3, the input to the QNN $s_i(k)$ $(i = 1, 2, \cdots, 9)$ consists of the tapped–delay–line input–output sets of the robot manipulator as follows:

$$\begin{cases} s_1(k) = r_1(k+1)\imath + r_2(k+1)\jmath + r_3(k+1)\kappa \\ s_2(k) = \theta_1(k)\imath + \theta_2(k)\jmath + \theta_3(k)\kappa \\ s_3(k) = \theta_1(k-1)\imath + \theta_2(k-1)\jmath + \theta_3(k-1)\kappa \\ s_4(k) = \theta_1(k-2)\imath + \theta_2(k-2)\jmath + \theta_3(k-2)\kappa \\ s_5(k) = \theta_1(k-3)\imath + \theta_2(k-3)\jmath + \theta_3(k-3)\kappa \\ s_6(k) = u_1(k-1)\imath + u_2(k-1)\jmath + u_3(k-1)\kappa \\ s_7(k) = u_1(k-2)\imath + u_2(k-2)\jmath + u_3(k-2)\kappa \\ s_8(k) = u_1(k-3)\imath + u_2(k-3)\jmath + u_3(k-3)\kappa \\ s_9(k) = v_1(k)\imath + v_2(k)\jmath + v_3(k)\kappa \end{cases}, \tag{29}$$

where $r_i(k)$ is the reference signal to the joint angle $\theta_i(k)$, $u_i(k)$ is the control input, and $v_i(k)$ is the feedback controller output $(i = 1, 2, 3)$. Using the real variables of the imaginary part of the QNN output $\sigma(k) = \sigma_0(k) + \sigma_1(k)\imath + \sigma_2(k)\jmath + \sigma_3(k)\kappa$, the control input is synthesised by $u_i(k) = \sigma_i(k) + v_i(k)$ $(i = 1, 2, 3)$. The feedback controller output is given by $v_i(k) = \gamma \boldsymbol{h}_i^{\mathrm{T}} \boldsymbol{e}_i(k)$ where γ is the scale factor, $\boldsymbol{h}_i \in \mathbb{R}^2$ is the feedback gain vector, $\boldsymbol{e}_i(k) = [\ e_i^{\theta}(k) \quad \Delta_i(k)\ ]^{\mathrm{T}}$ is the error vector, $e_i^{\theta}(k)$ is the control error in the joint space, and $\Delta_i(k) = e_i^{\theta}(k) - e_i^{\theta}(k-1)$ is the difference of the control error $(i = 1, 2, 3)$. Consequently, the robot manipulator's control torque is given by $\tau_i(k) = T_i u_i(k)$, where T_i is the torque constant $(i = 1, 2, 3)$.

3.3. Numerical Simulations

The robot manipulator's trajectory-tracking control experiments were conducted using the QNN-based feedforward–feedback controller. In the QNN, the unit's activation function in the hidden layer is the split-type function using a component-wise sigmoid function. Each real variable of the quaternion numbers representing the connection weights was initialised randomly from the interval $[-1, 1]$. The FEL of the QNN was conducted in real-time. It was terminated when the normalised cost function, defined as the mean of the cost function, Equation (10), within one iteration of the desired trajectory, was less than the threshold value 0.0005 after five continuous iterations. The network training was restarted using the new connection weight's initial values if the normalised cost function did not satisfy the termination condition after 100 iterations or if it tended to exceed a threshold and diverge during the training process. When the number of resets exceeded 10,000, the network training was defined as a failure. In the training process, the end-effector's desired trajectory was a circle of radius 0.2 in the vertical plane parallel to the O–YZ plane. The trajectory consisted of 1000 data points per round, and one round was defined as one iteration. The feedback gains were $\boldsymbol{h}_1 = [\ 4.0 \quad 0.04\ ]^{\mathrm{T}}$, $\boldsymbol{h}_2 = \boldsymbol{h}_3 = [\ 2.0 \quad 0.02\ ]^{\mathrm{T}}$, and the scale factor was $\gamma = 1$. The torque constants was $T_i = 10$ ($i = 1,2,3$). The motion equation, Equation (28), was solved by the Runge–Kutta method with a 0.01 s step size in the numerical simulations. In the robot manipulator, the links 1, 2 and 3 were 0.05 m long and 0.4 kg mass, 0.25 m long and 0.2 kg mass, and 0.25 m long and 0.2 kg mass, respectively. The damping factor was $c_i = 0.005$ ($i = 1,2,3$), and the solid frictional forces were 0.2 Nm, 0.1 Nm and 0.1 Nm at the joints 1, 2 and 3, respectively. The robot manipulator's initial condition was $\boldsymbol{\theta}(0) = [\ 0 \quad -\frac{\pi}{12} \quad \frac{5\pi}{12}\ ]^{\mathrm{T}}$.

First, the feasibility of training the QNN with the proposed learning algorithm derived using the $\mathbb{GHR}$ calculus and controlling the robot manipulator using the QNN-based feedforward–feedback controller was investigated through numerical simulations. Figure 3 shows the change of the normalised cost function where the number of units in the hidden layer is two and the learning factor is 0.012. The normalised cost function decreases rapidly, and the network training is completed when the training's terminating condition is satisfied. The normalised cost function decreases as the number of iterations progress and converge, indicating that it is possible to train the QNN in the control system. Figure 4 shows the response of the robot manipulator's end-effector in the workspace and the variation of the cost function at each sampling time, indicating the responses at the first and termination of the training process shown in Figure 3. At the beginning of the training process, the end-effector does not follow the desired trajectory due to a lack of enough adaptation using the QNN. However, after the training converges, the end-effector could follow the desired trajectory; namely, the QNN-based feedforward–feedback controller achieves the control task. When the network training is terminated, the cost function at each sampling time is sufficiently small. This means that the major role of controlling the robot manipulator has shifted from the feedback controller to the QNN-based feedforward–feedback controller. These results confirmed that the QNN learning algorithm derived from the $\mathbb{GHR}$ calculus is functioning well. Therefore, it is feasible to apply the QNN-based controller for controlling the robot manipulator.

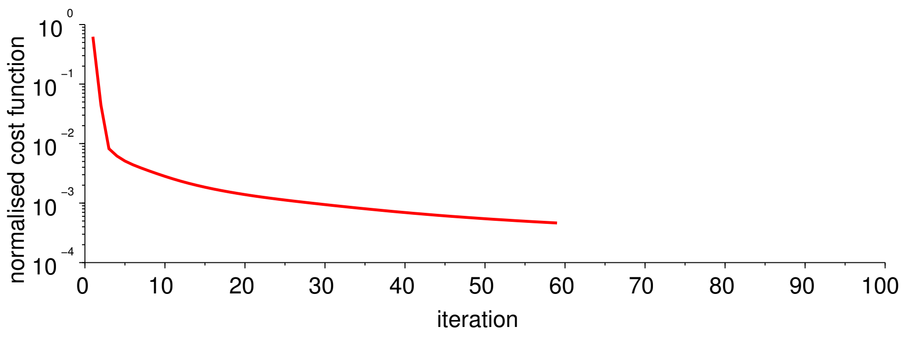

Figure 3. An example of the QNN's training process.

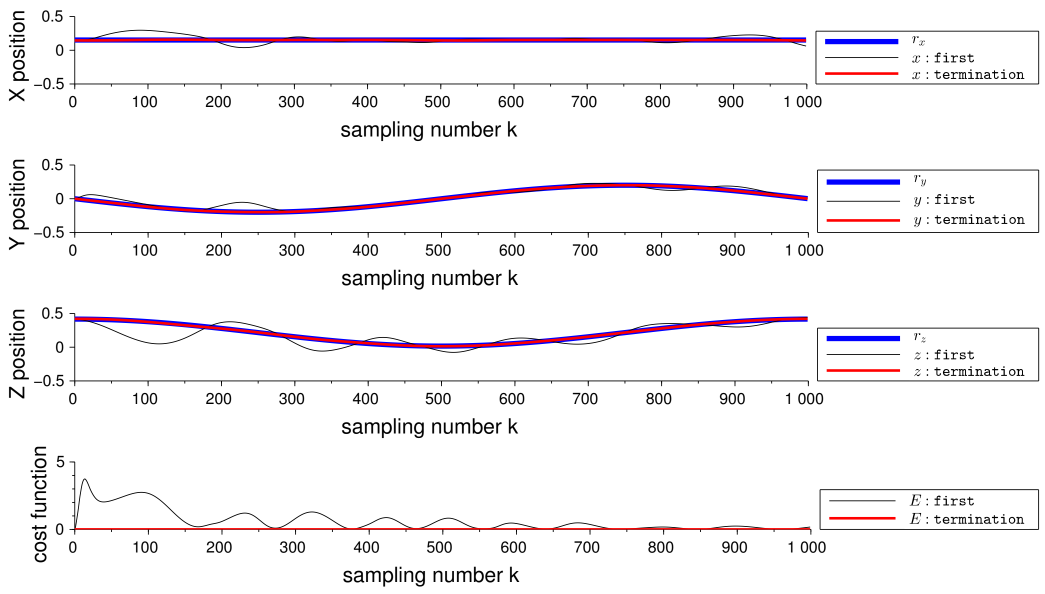

Figure 4. An example of system responses in the workspace for controlling the robot manipulator using the QNN-based feedforward–feedback controller and the variation of the cost function. In the top three figures, the desired trajectory's reference signal is drawn with a thick line, and the system response within the first iteration and the response that terminates the network training are drawn with thin and medium thick lines, respectively. In the bottom figure, the cost function within the first iteration and that which terminates the network training are drawn with thin and medium thick lines, respectively.

Next, the QNN's learning characteristics were evaluated using two indexes: the number of iterations required to satisfy the network training termination condition and the number of resets during the training process. Here, 50 experimental results with different initial network weights were used to calculate these indexes. Table 1 summarises the number of iterations and resets when the number of units in the hidden layer is varied from one to four with a 0.012 learning factor. The index's median and interquartile range are used instead of the mean and standard deviation because normal distribution is unobserved in these indexes according to the Shapiro–Wilk test with a 5% significant level. Additionally, multiple comparisons are conducted using the Steel–Dwass test. This result indicates that the number of units in the hidden layer has less effect on the learning speed if the network learning could progress, but the dependency on the connection weight's initial values to proceed with the network training increases as the number of units increases. Table 2 summarises the number of iterations and resets when the learning factor is changed, where the number of units in the hidden layer is two. Increasing the learning factor improves the learning speed; however, the number of reset increases, namely, the dependency on the connection weight's initial values to proceed with the network training increases. Furthermore, using a small learning factor increases the number of resets. This is because the first-order approximation of the steepest descent method makes the network training unstable with a large learning factor, whereas it is easy to reach the upper limit of the number of iterations with a small learning factor. Table 3 summarises the number of iterations and resets when the feedback gains are changed by the scale factor, where the number of units in the hidden layer is two and the learning factor is 0.012. Using smaller or larger feedback gains increases the number of resets although there is no statistical difference in the number of iterations. The network training did not converge when the scale factor was $\gamma = 0.1$. Since the QNN output at the beginning of the network training affects the feedback controller as a disturbance, it is difficult to keep the control system stable with the smaller feedback gains. The control system becomes easily unstable with larger feedback gains due to the nonlinear characteristics of the robot manipulator because the feedback controller uses a simple error feedback. This result shows that an appropriate value of the feedback gain is required to proceed with the network training while ensuring the stability of the control system.

Table 1. Learning characteristics of the QNN concerning the number of units in the hidden layer.

		Steel–Dwass Test				Iteration
		1	2	3	4	Median (IQR)
Number of units	1	–				42.5 (32.5–59.5)
	2	*	–			60.5 (38.5–81)
	3	**	ns	–		63 (44.75–82.25)
	4	**	ns	ns	–	75 (60.75–87)
		Steel–Dwass Test				**Reset**
		1	2	3	4	Median (IQR)
Number of units	1	–				48.5 (20.75–83.75)
	2	ns	–			75 (33.75–143.25)
	3	**	**	–		256.5 (92.25–634.5)
	4	**	**	**	–	1006.5 (317.75–2378.75)

(ns: not significant, * $p < 0.05$, ** $p < 0.01$).

Table 2. Learning characteristics of the QNN concerning the learning factor.

		Steel–Dwass Test					Iteration
		0.006	0.012	0.018	0.024	0.030	Median (IQR)
Learning factor η	0.006	–					72 (55.25–87.25)
	0.012	ns	–				60.5 (38.5–81)
	0.018	**	ns	–			48.5 (37–67.5)
	0.024	**	*	ns	–		45 (29–66.25)
	0.030	**	†	ns	ns	–	43.5 (29.25–66)
		Steel–Dwass Test					**Reset**
		0.006	0.012	0.018	0.024	0.030	Median (IQR)
Learning factor η	0.006	–					219.5 (91–681.5)
	0.012	**	–				75 (33.75–143.25)
	0.018	**	ns	–			75.5 (31.75–141.25)
	0.024	†	ns	†	–		129 (47.75–270.5)
	0.030	ns	*	**	ns	–	193.5 (64.5–392.25)

(ns: not significant, † $p < 0.1$, * $p < 0.05$, ** $p < 0.01$).

Table 3. Learning characteristics of the QNN concerning the feedback gain.

		Steel–Dwass Test				Iteration
		0.5	1	2	10	Median (IQR)
Scale factor γ	0.5	–				62.5 (45.25–78.5)
	1	ns	–			60.5 (38.5–81)
	2	ns	ns	–		71 (53–87.75)
	10	ns	ns	ns	–	71 (54.75–92)
		Steel–Dwass Test				**Reset**
		0.5	1	2	10	Median (IQR)
Scale factor γ	0.5	–				287.5 (124.5–432)
	1	**	–			75 (33.75–143.25)
	2	**	ns	–		72 (27.75–119.25)
	10	*	**	**	–	483.5 (236.25–758.5)

(ns: not significant, * $p < 0.05$, ** $p < 0.01$).

The control ability of the QNN-based feedforward–feedback controller was tested against untrained conditions after the QNN training process with the circular trajectory was completed. Figure 5 illustrates examples of the system response in the workspace where the desired trajectory and the initial condition of the joint angle were different from those set in the training process. Here, the number of units in the hidden layer was two, and the learning factor was 0.012. The FEL of the QNN was continued in real-time during the test. Although control errors can be observed partially around the curvature of the desired trajectory changes, the end-effector can follow the desired trajectory. The cost function sometimes increased with the control error but decreased soon as the sampling number progressed because the FEL can be performed stably in real-time after the pretraining with the circular trajectory. These results show that the QNN's learning and generalisation capabilities are effective to compensate for the robustness of the control system against varying the robot manipulator's experimental conditions.

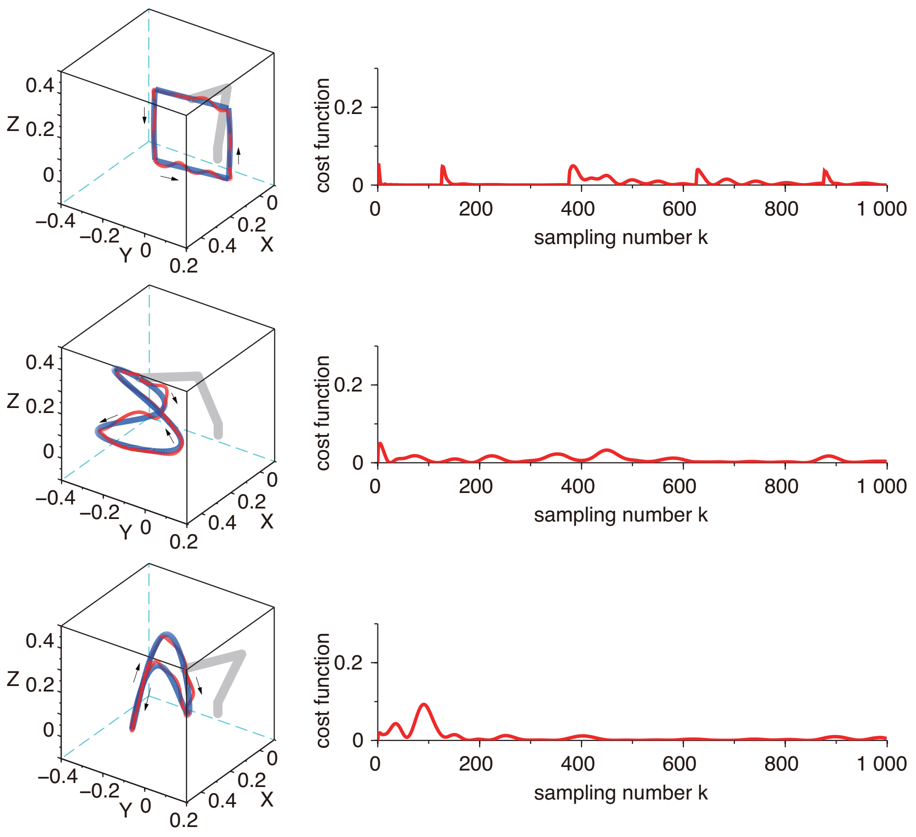

Figure 5. Examples of responses for controlling the robot manipulator using the QNN-based feedforward–feedback controller against untrained conditions, where the desired trajectory and the robot manipulator's initial condition are as follows: in the top figure, a square and $\boldsymbol{\theta}(0) = [\ 0 \quad -\frac{\pi}{12} \quad \frac{\pi}{2}\ ]^{\mathrm{T}}$; in the middle figure, a Lissajous curve and $\boldsymbol{\theta}(0) = [\ -\frac{3\pi}{16} \quad \frac{\pi}{9} \quad \frac{\pi}{3}\ ]^{\mathrm{T}}$; in the bottom figure, a Lissajous curve and $\boldsymbol{\theta}(0) = [\ -\frac{3\pi}{8} \quad -\frac{\pi}{8} \quad \frac{3\pi}{4}\ ]^{\mathrm{T}}$. In each figure, the left figure illustrates the system response in the workspace where the desired trajectory and the response of the end-effector are drawn with thick and medium thick lines, respectively. The arrow indicates the direction of the end-effector's movement and the stick figure illustrates the initial posture of the robot manipulator, whereas the left figure shows the variation of the cost function.

Finally, the learning algorithm derived using the pseudo-derivative to calculate the cost function derivative to the network parameters was applied to train the QNN as a reference. Table 4 compares the number of iterations and resets the learning algorithm obtained using the $\mathbb{GHR}$ calculus with those obtained by the learning algorithm using the pseudo-derivative. A total of 100 experimental results were used to calculate these indexes. Here, there were two units in the hidden layer, and the learning factor was 0.012. To compare the learning characteristics under more severe conditions, the threshold value for terminating the network training was 0.0002. The learning algorithm using the pseudo-derivative attains a little faster learning speed, and the Mann–Whitney *U* test shows a statistical difference in the number of iterations. However, the learning algorithm using the pseudo-derivative requires many resets to proceed with the network training. Additionally, the network training by the learning algorithm using the pseudo-derivative sometimes failed to converge with the same value of the learning factor used in the learning algorithm using the $\mathbb{GHR}$ calculus. These results indicate that the learning algorithm using the $\mathbb{GHR}$ calculus can perform the network training stably with relatively less dependence on the connection weight's initial values.

Table 4. Comparison of learning characteristics between the connection weight's learning algorithms.

	Iteration	**Reset**	**Success Rate [%]**
	Median (IQR)	**Median (IQR)**	
$\mathbb{GHR}$ calculus	69.5 (51.25–85.75)	874.5 (460.5–2038.5)	100
Pseudo–derivative	45.5 (29.75–75.25)	2106 (935–4656.5)	94
Mann–Whitney *U* test	$**$	$**$	

(** $p < 0.01$).

4. Conclusions

This study investigated a feedforward–feedback controller based on the QNN trained by the learning algorithm derived using the $\mathbb{GHR}$ calculus. In the QNN-based feedforward–feedback controller, the tapped–delay–line input–output sets of a plant were input to the QNN, and the control input to the plant was synthesised using the QNN output. A FEL framework performed at every sampling time was introduced to train the controller's QNN to ensure that the plant output followed the desired output in real-time. Additionally, the local stability condition of the controller was analysed under linear assumptions of the QNN and plant. The result clarified how the learning factor, the feedback gain and the QNN's connection weights affect the stability of the parameter error. Computational experiments on the trajectory-tracking control of a three-link robot manipulator using the QNN-based feedforward–feedback controller were conducted. Simulation results confirmed the feasibility of the learning algorithm derived using the $\mathbb{GHR}$ calculus to train the QNN, and the effectiveness of the QNN's learning and generalisation capabilities to maintain the robot manipulator's control performance. The experimentation of the QNN-based feedforward–feedback controller on an actual robot manipulator in real-world scenarios will be discussed in future work.

Author Contributions: Conceptualization, K.T.; methodology, K.T.; software, K.T. and E.T.; validation, K.T., E.T. and M.H.; formal analysis, K.T. and E.T.; investigation, K.T. and E.T.; resources, K.T. and M.H.; data curation, K.T. and E.T.; writing—original draft preparation, K.T.; writing—review and editing, K.T.; visualization, K.T. and E.T.; supervision, K.T. and M.H.; project administration, K.T.; funding acquisition, K.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by JSPS KAKENHI Grant Number JP20K11980.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lu, Y. Artificial Intelligence: A survey on evolution, models, applications and future trends. *J. Manag. Anal.* **2019**, *6*, 1–29. [CrossRef]
2. Ingrand, F.; Ghallab, M. Robotics and artificial intelligence: A perspective on deliberation functions. *AI Commun.* **2014**, *27*, 63–80. [CrossRef]

3. Prieto, A.; Prieto, B.; Ortigosa, E.M.; Ros, E.; Pelayo, F.; Ortega, J.; Rojas, I. Neural networks: An overview of early research, current frameworks and new challenges. *Neurocomputing* **2016**, *214*, 242–268. [CrossRef]
4. Dargan, S.; Kumar, M.; Ayyagari, M.R.; Kumar, G. A survey of deep learning and its applications: A new paradigm to machine learning. *Arch. Comput. Methods Eng.* **2020**, *27*, 1071–1092. [CrossRef]
5. Wang, H.; Liu, N.; Zhang, Y.; Feng, D.; Huang, F.; Li, D.; Zhang, Y. Deep reinforcement learning: A survey. *Front. Inf. Technol. Electron. Eng.* **2020**, *21*, 1726–1744. [CrossRef]
6. Alfsmann, D.; Göckler, H.G.; Sangwine, S.J.; Ell, T.A. Hypercomplex algebras in digital signal processing: Benefits and drawbacks. In Proceedings of the 15th European Signal Processing Conference, Poznan, Poland, 3–7 September 2007; pp. 1322–1326.
7. Nitta, T. (Ed.) *Complex–Valued Neural Networks—Utilizing High–Dimensional Parameters*; Information Science Reference: Hershey, PA, USA, 2009.
8. Hirose, A. (Ed.) *Complex–Valued Neural Networks—Advances and Applications*; Wiley-IEEE Press: Hoboken, NJ, USA, 2013.
9. Tripathi, B.K. (Ed.) *High Dimensional Neurocomputing—Growth, Appraisal and Applications*; Springer: New Delhi, India, 2015.
10. Parcollet, T.; Morchid, M.; Linares, G. A survey of quaternion neural networks. *Artif. Intell. Rev.* **2020**, *53*, 2957–2982. [CrossRef]
11. García–Retuerta, D.; Casado–Vara, R.; Rey, A.M.; Prieta, F.D.; Prieto, J.; Corchado, J.M. Quaternion neural networks: State–of–the–art and research challenges. In Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning, Guimaraes, Portugal, 4–5 November 2020; pp. 456–467.
12. Bayro-Corrochano, E. A survey on quaternion algebra and geometric algebra applications in engineering and computer science 1995–2020. *IEEE Access* **2021**, *9*, 104326–104355. [CrossRef]
13. Takahashi, K.; Isaka, A.; Fudaba, T.; Hashimoto, M. Remarks on quaternion neural network–based controller trained by feedback error learning. In Proceedings of the 2017 IEEE/SICE International Symposium on System Integration, Taipei, Taiwan, 11–14 December 2017; pp. 875–880.
14. Takahashi, K. Comparison of neural network–based adaptive controllers using hypercomplex numbers for controlling robot manipulator. In Proceedings of the 13th IFAC Workshop on Adaptive and Learning Control Systems, Winchester, UK, 4–6 December 2019; pp. 67–72.
15. Takahashi, K.; Watanabe, L.; Yamasaki, H.; Hiraoka, S.; Hashimoto, M. Remarks on control of a robot manipulator using a quaternion recurrent neural–network–based compensator. In Proceedings of the 2020 Australian & New Zealand Control Conference, Gold Coast, Australia, 25–26 November 2020; pp. 244–247.
16. Takahashi, K. Comparison of high–dimensional neural networks using hypercomplex numbers in a robot manipulator control. *Artif. Life Robot.* **2021**, *26*, 367–377. [CrossRef]
17. Parcollet, T.; Ravanelli, M.; Morchid, M.; Linarés, G.; Trabelsi, C.; Mori, R.D.; Bengio, Y. Quaternion recurrent neural networks. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019; pp. 1–9.
18. Mandic, D.P.; Jahanchahi, C.; Took, C.C. A quaternion gradient operator and its applications. *IEEE Signal Process. Lett.* **2011**, *18*, 47–50. [CrossRef]
19. Xu, D.; Jahanchahi, C.; Took, C.C.; Mandic, D.P. Enabling quaternion derivatives: The generalized HR calculus. *R. Soc. Open Sci.* **2015**, *2*, 150255. [CrossRef] [PubMed]
20. Xu, D.; Mandic, D.P. The theory of quaternion matrix derivatives. *IEEE Trans. Signal Process.* **2015**, *63*, 1543–1556. [CrossRef]
21. Xu, D.; Gao, H.; Mandic, D.P. A new proof of the generalized Hamiltonian–Real calculus. *R. Soc. Open Sci.* **2016**, *3*, 160211. [CrossRef]
22. Xu, D.; Xia, Y.; Mandic, D.P. Optimization in quaternion dynamic systems: Gradient, hessian, and learning algorithms. *IEEE Trans. Neural Netw. Learn. Syst.* **2016**, *27*, 249–261. [CrossRef] [PubMed]
23. Popa, C.A. Learning algorithms for quaternion–valued neural networks. *Neural Process. Lett.* **2018**, *47*, 949–973. [CrossRef]
24. Takahashi, K.; Tano, E.; Hashimoto, M. Remarks on feedforward-feedback controller using a trained quaternion neural network based on generalised HR calculus and its application to controlling a robot manipulator. In Proceedings of the 2021 (11th) International Conference on Advanced Mechatronic Systems, Tokyo, Japan, 9–12 December 2021; pp. 81–86.
25. Kawato, M.; Furukawa, K.; Suzuki, R. A hierarchical neural-network model for control and learning of voluntary movement. *Biol. Cybern.* **1987**, *57*, 169–185. [CrossRef] [PubMed]
26. Yamada, T. Remarks on feedback loop gain characteristics of adaptive type neural network feedforward feedback controller. In Proceedings of the SICE Annual Conference 2008, Tokyo, Japan, 20–22 August 2008; pp. 2244–2249.