Journal of MECHANICAL and ENERGY ENGINEERING

ISSN: 2544-0780 | e-ISSN: 2544-1671
DOI: 10.30464/jmee.00300

FORWARD AND INVERSE KINEMATICS SOLUTION OF A ROBOTIC MANIPULATOR USING A MULTILAYER FEEDFORWARD NEURAL NETWORK

Abdel-Nasser SHARKAWY[1*, 2]

[1*]Mechatronics Engineering, Mechanical Engineering Department, Faculty of Engineering, South Valley University, Qena, 83523, Egypt, e-mail: abdelnassersharkawy@eng.svu.edu.eg
[2]Mechanical Engineering Department, College of Engineering, Fahad Bin Sultan University, Tabuk 47721, Saudi Arabia

(Received 06 July 2022, Accepted 12 September 2022)

Abstract: In this paper, a multilayer feedforward neural network (MLFFNN) is proposed for solving the problem of the forward and inverse kinematics of a robotic manipulator. For the forward kinematics solution, two cases are presented. The first case is that one MLFFNN is designed and trained to find solely the position of the robot end-effector. In the second case, another MLFFNN is designed and trained to find both the position and the orientation of the robot end-effector. Both MLFFNNs are designed considering the joints' positions as the inputs. For the inverse kinematics solution, a MLFFNN is designed and trained to find the joints' positions considering the position and the orientation of the robot end-effector as the inputs. For training any of the proposed MLFFNNs, data is generated in MATLAB using two different cases. The first case is that data is generated assuming an incremental motion of the robot's joints, whereas the second case is that data is obtained with a real robot considering a sinusoidal joint motion. The MLFFNN training is executed using the Levenberg-Marquardt algorithm. This method is designed to be used and generalized to any DOF manipulator, particularly more complex robots such as 6-DOF and 7-DOF robots. However, for simplicity, this is applied in this paper using a 2-DOF planar robot. The results show that the approximation error between the desired output and the estimated one by the MLFFNN is very low and it is approximately equal to zero. In other words, the MLFFNN is efficient enough to solve the problem of the forward and inverse kinematics, regardless of the joint motion type.

Keywords: multilayer feedforward neural network, forward kinematics, inverse kinematics, 2-DOF planar robot, Levenberg-Marquardt algorithm, generated data.

1. INTRODUCTION

Forward kinematics [1–3] refers to determining the position and the orientation of the robot end-effector (Cartesian space) based on the joints' variables. The forward kinematics problem is always straightforward. In addition, deriving the equation is easy and no complexity is found. Inverse kinematics refers to determining the joints' variables based on a given position and orientation for the robot end-effector. A solution to the problem of inverse kinematics is complex and difficult. In addition, it is computationally expensive. Forward kinematics vs. inverse kinematics is presented in Fig. 1.

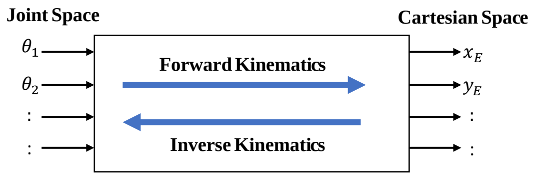


Fig. 1. The relationship between forward and inverse kinematics, [1]. The left side of the figure represents the joint variable of the manipulator. The right side represents the position and orientation of the robot end-effector.

Different methods have been proposed for solving the forward kinematics problem of the manipulator such as Denavit–Hartenberg (DH), the product of exponential, trigonometric, dual-quaternion, support vector regression, and neural networks (NNs). In [4], Denavit and Hartenberg demonstrated that four parameters were required for the general transformation between two joints. These four parameters were known as DH parameters. In addition, these parameters became a standard to describe robot kinematics. The DH method is the most consistent and the most concise method of all. However, it has some limitations [5]. In [6], Farah and LIU presented forward kinematics using DH parameters for a 6-DOF surgical robot. The product of exponential [7] and unit dual quaternions [8] were based on the screw theory, and they were proposed in robot kinematics. In [9], Sharkawy and Aspragathos presented a comparative study between the product of exponentials formula and the unit dual quaternion algebra for determining the forward kinematics of a serial manipulator (i.e. the 7-DOF KUKA LWR robot). The results showed that the unit dual quaternions are characterised by higher compactness, and it facilitates the understanding of the joint axes geometrical meaning compared to the product of exponential method. The geometric method [10] was also proposed to obtain forward kinematics. This method uses the geometric properties of specific mechanisms for transforming and reducing the problem. After that, analytical geometric means are used to obtain the solutions. An example of this method was presented in [10], where it was used to find the closed-form forward kinematics of an H4 parallel robot. Support vector regression [11] is a supervised machine learning technique and it is widely used for classification and regression tasks. NN has the properties that it can approximate any function and its ability of generalization under different conditions [12], [13]. Support vector regression was used to develop forward kinematics for parallel manipulator robots and the NN method was used with cable-driven robots [14–17].

A solution of the inverse kinematics problem of the manipulator as stated in the beginning of this section is a difficult and quite challenging task [3]. Different methods were proposed for solving the inverse kinematics problem such as the closed-form solution method, numerical methods, evolutionary computing, and NNs. The closed-form solution method depends on an analytic expression or any polynomial having a DOF less than four [2]. This solution takes the advantage of the robot's specific geometry for formulating the mathematical model. This method is divided into types, algebraic and geometric methods. Examples of closed-form solution methods were presented in these references [18–19]. Numerical methods were used to solve the inverse kinematic problem when the obtained polynomial in the closed-form solution with more than four DOF; therefore, the robot did not have a close-form solution, [20]. Numerical methods are classified into the following types [2]: the symbolic elimination method, continuation methods, Iterative Methods, and methods based on optimization techniques. Examples of using the numerical methods to solve the inverse kinematics problem were presented in [21–22]. The numerical methods lack accuracy when compared with closed-form methods, [2]. Evolutionary computing was also proposed for solving the inverse kinematics problem as in [23–24]. NNs were presented for the inverse kinematic solution. In [25], Tejomurtula and Kak proposed structured neural networks to solve inverse kinematics. They used conventional backpropagation learning for the training process which led to several difficulties related to accuracy. To overcome the problems of accuracy and training time, they devised a variant of the conventional error backpropagation learning. NNs based methods were also proposed in references [3, 26–27].

From this discussion, we can conclude that further investigation using soft computing-based methods such as NNs for the forward and particularly inverse kinematics solution is required. The main issue with the previous researchers was a low accuracy of NN. Therefore, improving the NN performance by achieving a very small (close to zero) mean squared error (MSE) and an approximation error and optimizing the NN architecture should be considered and performed.

The main contribution and aim of this paper cover the following three issues:

1) The first issue is developing a MLFFNN that can be generalized and used for solving the problem of forward kinematics and inverse kinematics of any DOF manipulator, particularly for complex robots that have 6-DOF and 7-DOF. For simplicity, the method is applied with a 2-DOF planar manipulator in this paper. In addition, the 2-DOF robot is often used in robotics as testbeds for various algorithms and theories. A similar example of this issue is presented in our previous works in [28]–[32], where we implemented a collision detection method for human-robot collaboration. The method in these references was applied and investigated with 1-DOF and 2-DOF and 3-DOF robots as simple cases to minimize the calculations and the human effort. Although the method was applied with these simple cases, we concluded that it can be applied with any complex robot such as 6- or 7- DOF as the same methodology can be followed with these complex robots.
2) The second issue is seeking to obtain the best performance of the designed MLFFNN by achieving MSE and an approximation error that

are close to zero . In other words, this means high accuracy.

3) The third issue is executing a new methodology in the case of forward kinematics to investigate the approximation error between the desired and the estimated outputs. This methodology considers two cases:

– The first case is designing and training a MLFFNN for estimating the robot end-effector position only.
– The second case is designing and training another MLFFNN for estimating both the robot end-effector position and orientation.

The main purpose of this methodology is to demonstrate whether implementing a MLFFNN to predict only the position of the robot end-effector and another MLFFNN to predict only the orientation of the robot end-effector is better or a MLFFNN to predict both the position and the orientation of the robot end-effector simultaneously.

For the inverse kinematics solution, a MLFFNN is designed and trained to estimate the joints' positions based on both the position and the orientation of the robot end-effector.

The Levenberg-Marquardt algorithm is used to train any of the proposed MLFFNNs. The training is executed using two types of data: the first data is generated considering the incremental joint's motion, whereas the second data is obtained with a real robot (i.e., a KUKA LWR robot) considering the sinusoidal joint's motion. The simulation work is presented in detail together with the trained MLFFNN verification. All this work is executed in MATLAB and using an Intel®Core™ i5-8250U CPU @ 1.60 GHz processor.

The rest of the paper is divided as follows. Section 2 presents the equations of the forward and the inverse kinematics of the 2-DOF planar robot using the geometric solution approach. In Section 3, the collected data which is used for training and testing the MLFFNN is discussed. In Section 4, the design, training, and verification of the MLFFNN for the forward kinematics solution are presented in detail. Two cases are shown clearly in this section. Section 5 shows the inverse kinematics solution based on the proposed MLFFNN. The design, training, verification of this MLFFNN are discussed in detail. Finally, Section 6 summarizes the main points of this paper, and it offers some future works.

2. FORWARD AND INVERSE KINEMATICS OF 2-DOF ROBOT

In this section, the forward and the inverse kinematics analysis of the SCARA type robot (2-DoF for planar horizontal motions) are presented. The 2-DOF planar robot is often used in robotics as a testbed for various algorithms and theories, [3, 33]. This robot is shown in Fig. 2.

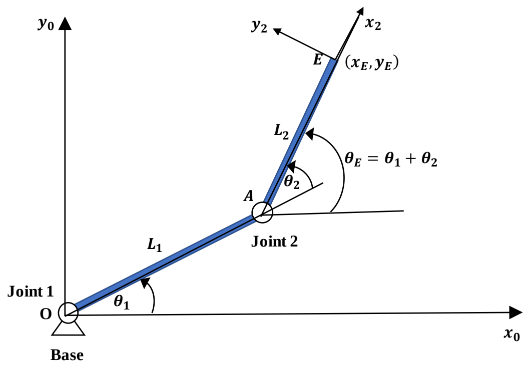


Fig. 2. A 2-DOF planar robot.

2.1. Forward Kinematics

Using the geometric solution approach, the forward kinematics equations of the 2-DOF planar robot are as follows [1], [9], [34]:

$$x_E = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2), \qquad (1)$$

$$y_E = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2). \qquad (2)$$

Equations (1) and (2) present the coordinates (position) of the robot end-effecter in the frame attached to the base of the robot O (x_0, y_0). The orientation of the robot end-effecter is described by the angle of rotation of the frame attached to the end-effecter relative to the fixed frame attached to the base of the robot. The orientation of the end-effecter θ_E is related to the actual joint displacements as follows:

$$\theta_E = \theta_1 + \theta_2. \qquad (3)$$

Therefore, equations (1) to (3) present the end-effector position and orientation viewed from the fixed coordinate system attached to the base of the robot in relation to the joint variables θ_1 and θ_2, [35].

2.2. Inverse Kinematics

Using the geometric solution approach, inverse kinematics (θ_1, θ_2) are presented as follows [1], [36]:

$$\theta_2 = arctan2(\pm\sin\theta_2\,, \cos\theta_2), \qquad (4)$$

where: $\cos\theta_2 = \frac{x_E^2+y_E^2-L_1^2-L_2^2}{2L_1L_2}$ and $\sin\theta_2 = \pm\sqrt{1-\left(\frac{x_E^2+y_E^2-L_1^2-L_2^2}{2L_1L_2}\right)^2}$.

$$\theta_1 = arctan2(\pm\sin\theta_1\,, \cos\theta_1), \qquad (5)$$

where: $\cos\theta_1 = \frac{x_E(L_1+L_2\cos\theta_2)+y_E L_2 \sin\theta_2}{x_E^2+y_E^2}$ and $\sin\theta_1 = \pm\sqrt{1-\left(\frac{x_E(L_1+L_2\cos\theta_2)+y_E L_2\sin\theta_2}{x_E^2+y_E^2}\right)^2}$.

The 2-DOF planar robot, as presented, is a very simple structure. However, the inverse kinematics solution is considered to be very cumbersome. Therefore, a MLFFNN is designed and trained for solving forward and inverse kinematics. The MLFFNN is a very simple structure compared to the other types of NNs [12, 37, 38]. In addition, it can be easily and successfully applied in various problem domains [32, 39–43]. The MLFFNN has the properties of adaptivity, parallelism, and generalization that it presents [44–46]. Furthermore, it can be linear or nonlinear. The MLFFNN requires a large number of pairs of input and target for the training process [47, 48]. However, this disadvantage is considered in the current work.

In the current paper, the designed MLFFNN is trained using Levenberg-Marquardt (LM) learning. The LM algorithm can implement the work in a fast manner. This algorithm is a type of second-order optimization techniques that have a strong theoretical basis and provide significantly fast convergence and this is considered as an approximation to the Newton's Method [49, 50]. Compared with other learning algorithms, LM learning is used because offers a trade-off between the fast learning speed of the classical Newton's method and the guaranteed convergence of the gradient descent [49, 51]. This learning is suitable for larger datasets as well as converges in less iterations and in shorter time than other training methods.

The next three sections present the collected data that is used for training the MLFFNN as well as the MLFFNN design, training, and testing for solving the forward kinematics and the inverse kinematics of the 2-DOF planar robot.

3. COLLECTED DATA

In this paper, the MLFFNN is trained and tested considering two cases of the data generated using MATLAB. The first case is that the data is generated considering the joints' incremental motion. The second case is that data is obtained with a real robot considering the joints' sinusoidal motion. The main aim of using these types of motions is to show the effectiveness of the designed MLFFNN under different motion types of the robot whether simple or complex. However, any other type of motion can be used and considered. Both cases are presented in detail in the following two subsections.

3.1. Generated Data Using Joint's Incremental Motion

In this case, data is generated in MATLAB assuming the following: $L_1 = 0.39\,m$, $L_2 = 0.156\,m$, $\theta_1 \in [0, 90]$ deg or $\in [0, 1.57]$ rad, $\theta_2 \in [0, 150]$ deg or $\in [0, 2.62]$ rad. Based on Equations (1) to (3), the position (x_E, y_E) and the orientation θ_E of the robot end-effector are determined. The number of the samples generated is 10000. In MATLAB, θ_1 and θ_2 are defined as:

```
theta1 = (linspace(0,90,10000)*pi/180)',
theta2 = (linspace(0,150,10000)*pi/180)'.
```

All of the data generated is presented in Fig. 3, assuming that the incremental motion of the two joints takes 10 seconds.

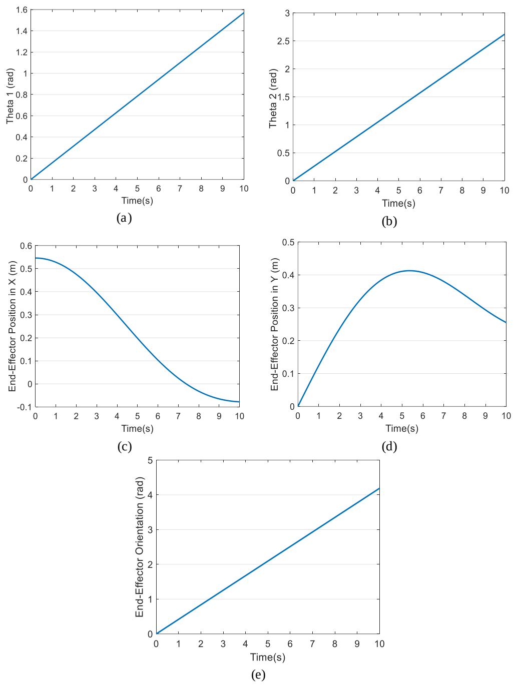


Fig. 3. The data generated in MATLAB used for training and testing the MLFFNN. (a) variable θ_1 in radians, (b) variable θ_2 in radians, (c) the position of the robot end-effector in $x -$ direction (x_E), (d) the position of the robot end-effector in $y -$ direction (y_E), (e) the orientation of the robot end-effector (θ_E) in radians.

3.2. Generated Data Using Joint's Sinusoidal Motion

In this case, data is generated using the KUKA LWR IV robot, as presented in Fig. 4. The manipulator is configured to be as a SCARA type robot (2-DoF for planar horizontal motions). In this configuration, Joint 1 represents KUKA's A3 (4th joint) and joint 2 represents KUKA's A5 (6th joint).

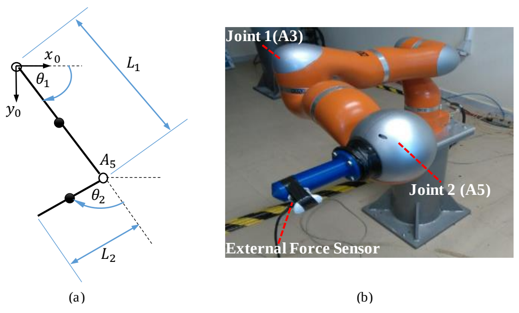


Fig. 4. (a) Motion of joint 1 (A3) and joint 2 (A5). (b) Kuka LWR IV manipulator. The figure is taken from ref. [29–30].

A sinusoidal motion with a variable frequency is commanded on both joints of the manipulator. For joint 1, the frequency is linearly increasing from 0.05 Hz to 0.17 Hz. For joint 2, the frequency is linearly increasing from 0.05 Hz to 0.23 Hz. The range of the joints' motion is $[-80, 10]\ deg$ for each joint. This motion is the same that was considered in our previous work in ref. [29], [30], and defined by the following equation:

$$\theta_{1,2}(t) = \frac{\pi}{4}\left(1 - cos(2\pi f t)\right), \tag{6}$$

where f is the frequency of the joint's sinusoidal motion.

Based on Equations (1) to (3), the position (x_E, y_E) and the orientation θ_E of the robot end-effector are determined considering $L_1 = 0.39\,m$ and $L_2 = 0.156\,m$. The number of the samples obtained is 64862. The time range is $[0, 53]$ seconds. All of the data generated is presented in Fig. 5.

4. MLFFNN FOR FORWARD KINEMATICS SOLUTION

In this section, an MLFFNN is designed and trained to solve the problem of the forward kinematics of the 2-DOF planar robot. Two cases are presented: the first case is that an MLFFNN is implemented to find the position of the robot end-effector only. The second case is that an MLFFNN is implemented to find the position and the orientation of the robot end-effector. Both implemented MLFFNNs are based on the joint variables θ_1 and θ_2. The following Subsections (4.1) and (4.2) present that in detail.

4.1. MLFFNN For Finding End-Effector Position Only

In this subsection, an MLFFNN is designed and trained to find the robot end-effector position only.

The main inputs for the designed MLFFNN are variables θ_1 and θ_2. The architecture of this NN is composed of three layers as follows: 1) the input layer which contains the two inputs, 2) the non-linear (hyperbolic tangent activation function) hidden layer, and 3) the output layer which estimates the position of the robot end-effector (x'_E, y'_E). This estimated position is compared with the desired position of the robot end-effector (x_E, y_E) which is presented in Fig. 3 (c) and (d). This MLFFNN architecture is presented in Fig. 6.

The equations of the feedforward part of the designed MLFFNN are given as follows:

$$y_j = \varphi_j(h_j) = \varphi_j\left(\sum_{i=0}^{2} w_{ji}\, x_i\right) \tag{7}$$

where, x_i are the inputs to the designed MLFFNN. $x_0 = 1, x_1 = \theta_1(k)$, and $x_2 = \theta_2(k)$.

$$\varphi_j(h_j) = \tanh(h_j) \tag{8}$$

$$out_k = \psi_k(O_k) = \psi_k\left(\sum_{j=0}^{n} b_{kj}\, y_j\right) = \left(\sum_{j=0}^{n} b_{kj}\, y_j\right) \tag{9}$$

where $k = 1, 2$. $out_1 = x'_E$ and $out_2 = y'_E$.

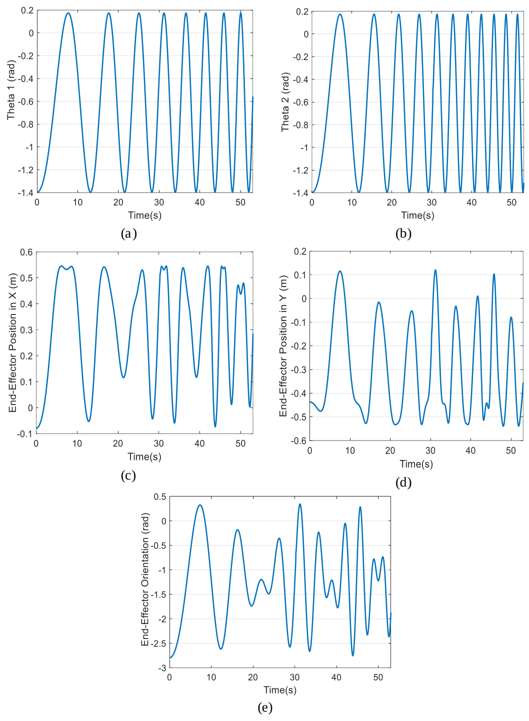


Fig. 5. Data generated using the KUKA LWR robot and considering sinusoidal motion, used for training and testing the MLFFNN. (a) variable θ_1 in radians, (b) variable θ_2 in radians, (c) the position of the robot end-effector in $x-$ direction (x_E), (d) the position of the robot end-effector in $y-$ direction (y_E), (e) the orientation of the robot end-effector (θ_E) in radians.

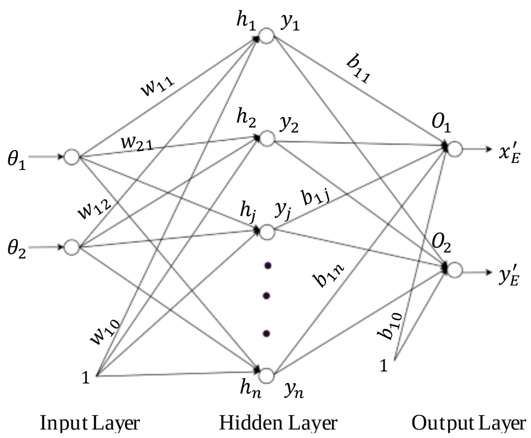


Fig. 6. The designed MLFFNN architecture for finding the end-effector position only. The drawing of this architecture is carried out using the following website: https://app.diagrams.net/.

The desired position of the robot end-effector (x_E, y_E) is used only for training the designed MLFFNN and the training errors $e_1(t)$ and $e_2(t)$ should be as close to zero as possible and given by the following equation:

$$e_1(t) = x_E - x'_E\,, \tag{10}$$

$$e_2(t) = y_E - y'_E. \tag{11}$$

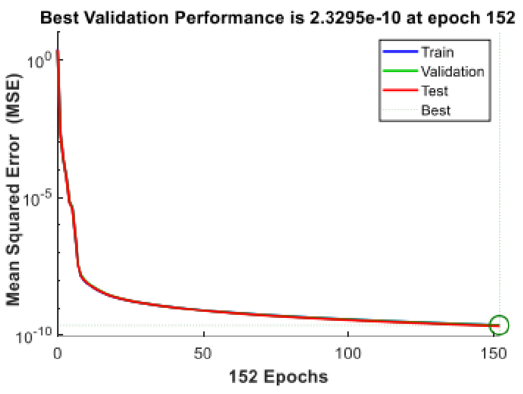
The data generated in MATLAB (presented in Fig. 3) is used for training the designed MLFFNN. From the data, 80% is used for training, 10% for validation, and 10% for testing. The main criterion in training the MLFFNN is obtaining high performance which is the lowest mean squared error (MSE) and the lowest training error. After trying and testing many different weights' initializations and a number of hidden neurons, the best parameters of the MLFFNN that achieve high performance are as follows:

– The number of hidden neurons is 90,
– The number of iterations is 152,
– The lowest MSE is 2.3295×10^{-10}.
– The training time is 46 seconds. The training time is not very important because the main purpose is to obtain a well-trained MLFFNN that can estimate the robot end-effector position correctly. However, this training time is very low.

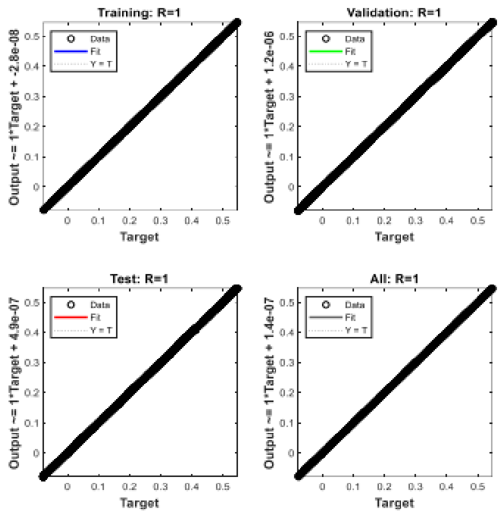
The results obtained from the training process are presented in Fig. 7.

As shown from the results. the MSE obtained is very low and is approximately equal to zero (Fig. 7(a)). In addition, regression (Fig. 7(b)) is equal to 1, which means that the convergence/approximation between the desired robot end-effector position (x_E, y_E) and the estimated one by the MLFFNN (x'_E, y'_E) is very good or approximately ideal. This proves that the designed MLFFNN is trained very well, and it is qualified to estimate the position of the end-effector correctly.

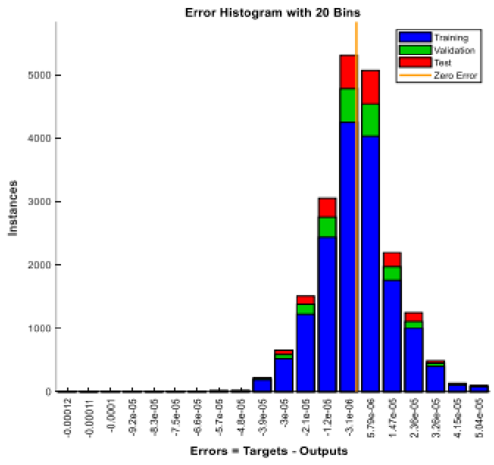
Once the MLFFNN training is finished completely, the trained MLFFNN designed is tested and investigated using the same dataset that was used for the training to obtain an insight about the approximation. The comparison between the desired end-effector position (x_E, y_E) and the estimated one (x'_E, y'_E) by the trained MLFFNN is presented in Fig. 8 and Fig. 9.



(a) The lowest MSE

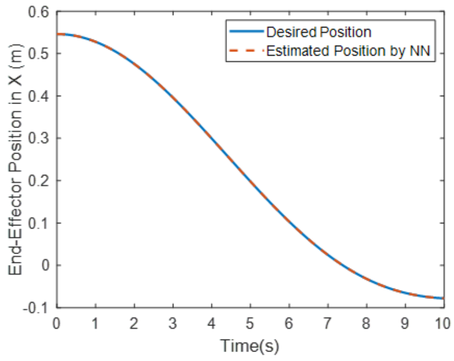


(b) The regression obtained

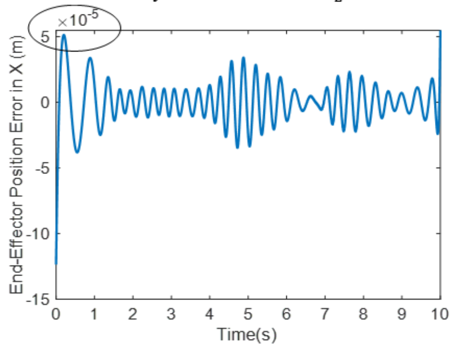


(c) The error histogram

Fig. 7. The results obtained from the training process of the designed MLFFNN.



(a) The desired end-effector position x_E and the estimated one by trained MLFFNN x'_E.



(b) The error $e_1(t) = x_E - x'_E$.

Fig. 8. Comparison between the desired position of the end-effector and the estimated position by the trained MLFFNN, in x −direction.

As shown in Fig. 8 and Fig. 9, the approximation/convergence between the desired position of the robot end-effector and the estimated position by the trained NN is very good (approximately ideal) whether in $x-$ or $y-$ direction. The approximation error between these positions is very low and it is almost zero. Indeed, this proves that the proposed and the designed MLFFNN is trained very well and is qualified to estimate the position of the robot end-effector correctly. We may conclude from these results that the MLFFNN is able to find the position of the robot end-effector efficiently.

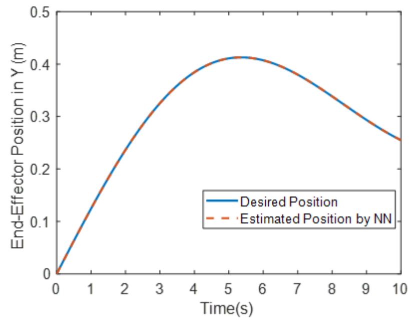
In the next subsection (4.2), the MLFFNN is used to find both the position and the orientation of the robot end-effector simultaneously.

4.2. MLFFNN For Finding Both End-Effector Position and Orientation

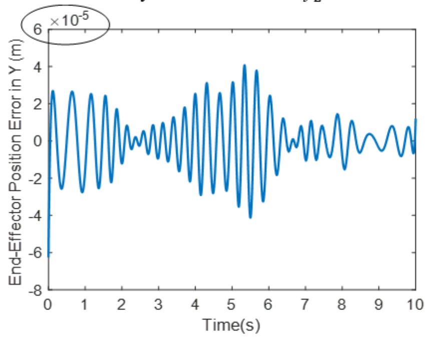
In this subsection, an MLFFNN is designed and trained to find both the position and the orientation of the robot end-effector simultaneously. The same protocol presented in Subsection 4.1 is followed here in this subsection.

A. The MLFFNN Design

The main inputs for this designed MLFFNN are the variables θ_1 and θ_2. The architecture of this NN is composed of three layers as follows: 1) the input layer which contains the two inputs: θ_1 and θ_2, 2) the non-linear (hyperbolic tangent activation function) hidden layer, and 3) the output layer, which estimates the position (x'_E, y'_E) and the orientation θ'_E of the robot end-effector. The estimated position and orientation are compared with the desired position (x_E, y_E) and orientation θ_E of the robot end-effector. This MLFFNN architecture is presented in Fig. 10.



(a) The desired end-effector position y_E and the estimated one by trained MLFFNN y'_E.



(b) Error $e_2(t) = y_E - y'_E$.

Fig. 9. Comparison between the desired position of the end-effector and the estimated position by the trained MLFFNN, in y −direction.

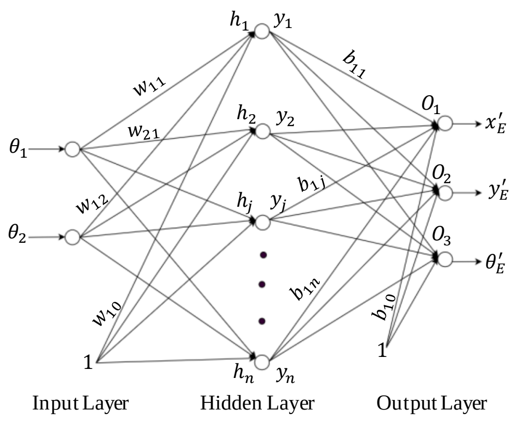


Fig. 10. The designed MLFFNN architecture for finding both the position and the orientation of the robot end-effector. The same symbols of the weights are used as in Fig. 6, but their values are different. The drawing of this architecture is carried out using the following website: https://app.diagrams.net/.

B. MLFFNN Training and Verification using Data Generated from Joint's Incremental Motion

The data generated from the joints' incremental motion, which is presented in Fig. 3, is used for training the designed MLFFNN. From the data, 80% is used for the training, 10% for validation and 10% for testing. After trying and testing many different weights' initializations and a number of hidden neurons, the best parameters of the MLFFNN that achieve high performance (the lowest MSE and the lowest training error) are as follows:

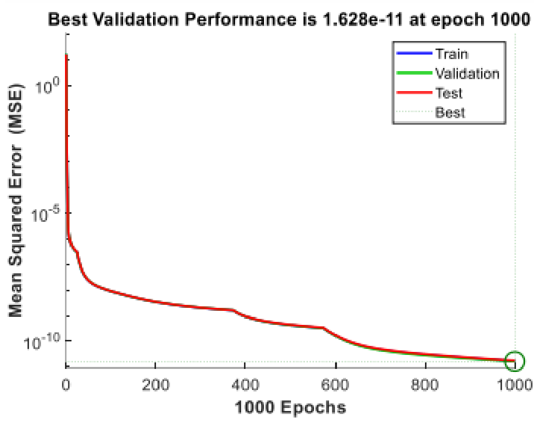
– The number of hidden neurons is 70,
– The number of iterations is 1000,
– The lowest MSE is 1.628×10^{-11}.
– The training time is 5 minutes and 21 seconds. This time is not very important as discussed before in Subsection 4.1.

The results obtained from the training process are presented in Fig. 11.

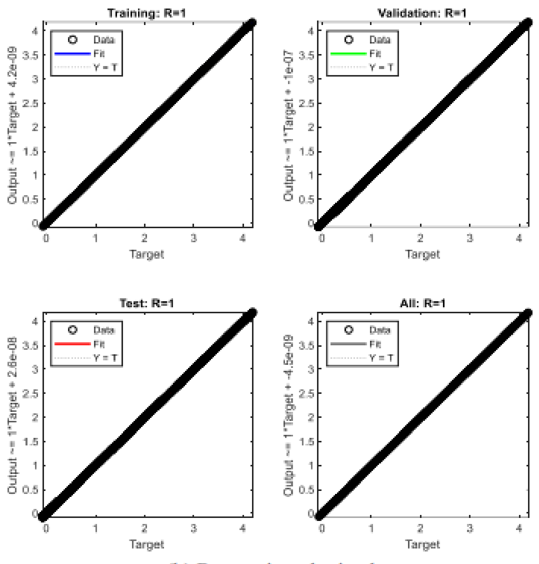
As shown from the results presented in Fig. 11, the obtained MSE is very low (Fig. 11(a)), and it is approximately equal to zero. Furthermore, the regression (Fig. 11(b)) is equal to 1, which means that the convergence/approximation between the desired position (x_E, y_E) and orientation θ_E of the robot end-effector position and the estimated position (x'_E, y'_E) and orientation θ'_E by the designed MLFFNN is very good. These results prove that the designed MLFFNN is trained very well, and it is qualified to correctly estimate both the position and the orientation of the end-effector, simultaneously.

Once the MLFFNN training is finished completely, the trained MLFFNN designed is tested and investigated using the same dataset that was used for the training to obtain an insight about the approximation. The comparison between the desired end-effector position (x_E, y_E) and orientation θ_E and the estimated ones (x'_E, y'_E), θ'_E by the trained MLFFNN is presented in Fig. 12 to Fig. 14.

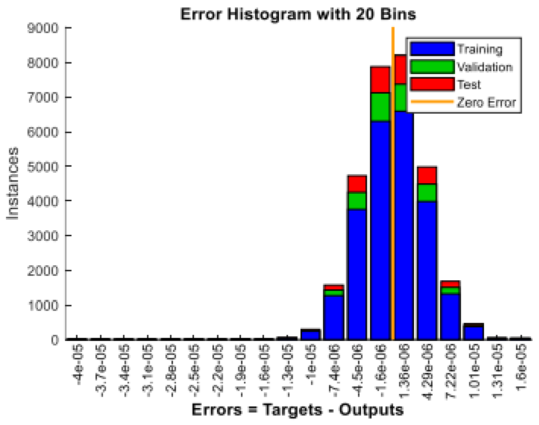
As shown from the results presented from Fig. 12 to Fig. 14, the approximation/convergence between the desired position of the robot end-effector and the estimated position by the trained MLFFNN is very good and it is approximately ideal, whether in $x-$ or $y-$ direction. The approximation error between these positions is approximately zero. In addition, the approximation between the desired orientation and the estimated one by the trained MLFFNN is very good and the error between them is very low and it is almost zero. These results prove that the trained MLFFNN is working well and efficiently to estimate correctly both the position and the orientation of the robot end-effector, simultaneously.



(a) The lowest MSE.

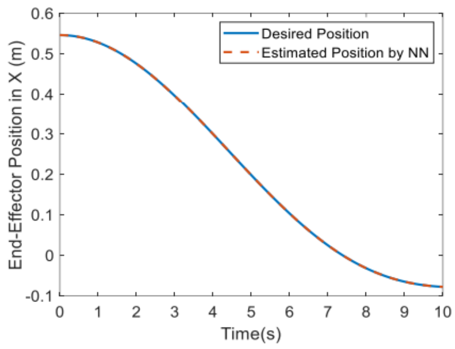


(b) Regression obtained

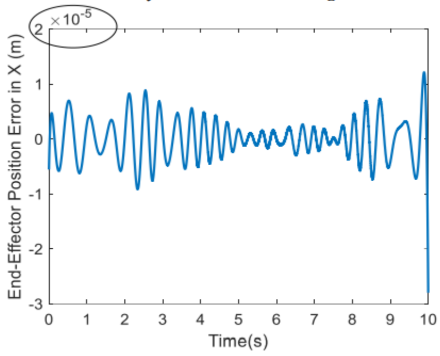


(c) The error histogram.

Fig. 11. The results obtained from the training process of the designed MLFFNN using the data from the joint's incremental motion

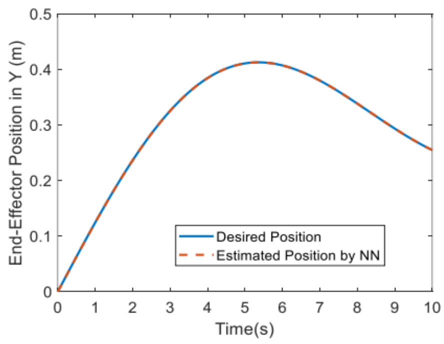


(a) The desired end-effector position x_E and the estimated one by trained MLFFNN x'_E.

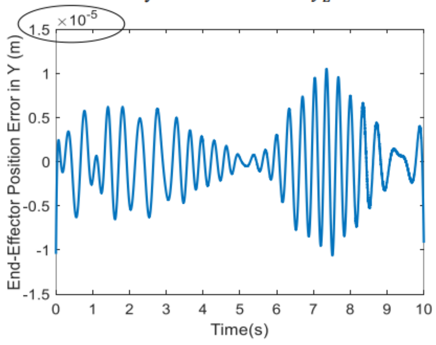


(b) Error $e_1(t) = x_E - x'_E$.

Fig. 12. Comparison between the desired position of the end-effector and the estimated position by the trained MLFFNN, in x −direction.

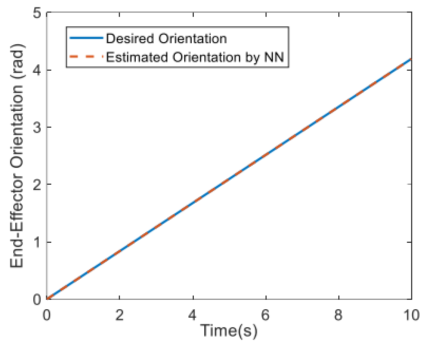


(a) The desired end-effector position y_E and the estimated one by trained MLFFNN y'_E.

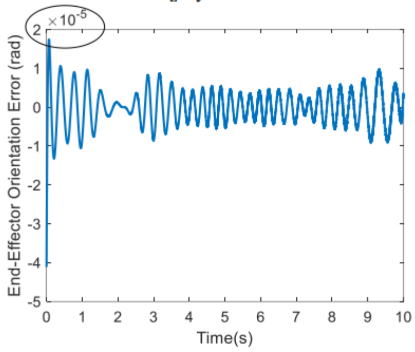


(b) Error $e_2(t) = y_E - y'_E$.

Fig. 13. Comparison between the desired position of the end-effector and the estimated position by the trained MLFFNN, in y −direction.



(a) The desired end-effector orientation θ_E and the estimated orientation θ'_E by trained MLFFNN



(b) Error $e_3(t) = \theta_E - \theta'_E$

Fig. 14. Comparison between the desired orientation of the end-effector and the estimated one by the trained MLFFNN.

The comparison between this trained MLFFNN and the one presented in Subsection 4.1, where the position of the end-effector is only estimated, is discussed as follows. The approximation error between the desired and the estimated position in the current case or in the previous case presented in Subsection 4.1 is very low and close to zero. Also, the approximation error between the desired and the estimated orientation is very low, and it is close to zero in the current case. We may conclude from that whether implementing a MLFFNN to estimate only the position of the robot end-effector, implementing another MLFFNN to estimate the orientation only, or implementing a MLFFNN to estimate both the position and the orientation, leads to a very low approximation error and MSE (being approximately equal to zero). However, implementing one MLFFNN to estimate both the position and the orientation of the robot end-effector will minimize the effort and also the time. This point needs further investigation by applying other data. For this purpose, the data generated from the joint's sinusoidal motion of the KUKA LWR robot is used. This is presented in the next subsection.

C. MLFFNN Training and Verification using Data from Joint's Sinusoidal Motion

The data obtained from the sinusoidal motion with the KUKA LWR robot (presented in Fig. 5) is used for training the designed MLFFNN presented in Fig. 10 to estimate both the position and the orientation of the robot end-effector. From this data, 80% is used for the training, 10% for validation, and 10% for testing. After trying and testing many different weights' initializations and a number of hidden neurons, the best parameters of the MLFFNN that achieve high performance (the lowest MSE and the lowest training error) are as follows:

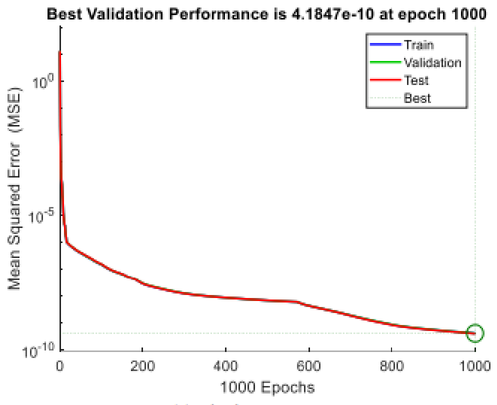
– the number of hidden neurons is 90,
– the number of iterations is 1000,
– the lowest MSE is 4.1847×10^{-10},
– the training time is 40 minutes and 52 seconds. This time is higher compared to the corresponding one in the previous case presented in the subsection because the data is higher in the current case.

The results obtained from the training process are presented in Fig. 15.

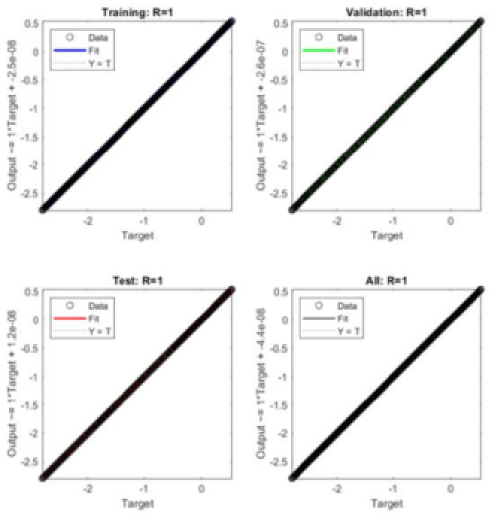
As shown from the results presented in Fig. 15, the MSE obtained is very low (Fig. 15(a)), and it is approximately equal to zero. Furthermore, the regression (Fig. 15(b)) is equal to 1, which means that the convergence/approximation between the desired position (x_E, y_E) and orientation θ_E of the robot end-effector position and the estimated position (x_E', y_E') and orientation θ_E' by the designed MLFFNN is very good or approximately ideal. These results prove that the MLFFNN designed is trained very well, and it is qualified to correctly estimate both the position and the orientation of the end-effector, simultaneously.

Once the MLFFNN training is finished completely, the trained MLFFNN designed is tested and investigated using the same dataset that was used for the training to obtain an insight about the approximation. The comparison between the desired end-effector position (x_E, y_E) and orientation θ_E and the estimated ones (x_E', y_E'), θ_E' by the trained MLFFNN is presented in Fig. 16 to Fig. 18.

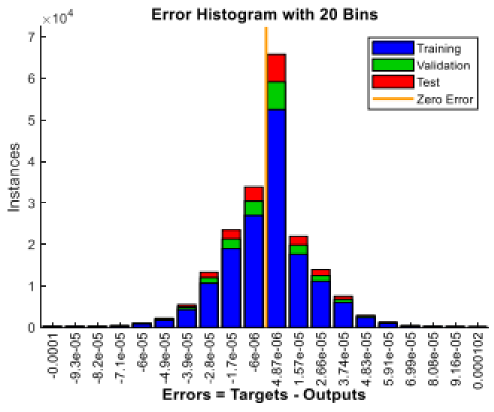
As shown from the results presented in Fig. 16 to Fig. 18, the approximation/convergence between the desired position of the robot end-effector and the estimated position by the trained MLFFNN is very good and approximately ideal, whether in $x-$ or $y-$ direction. The approximation error between these positions is approximately zero. In addition, the approximation between the desired orientation and the estimated one by the trained MLFFNN is very good and the error between them is very low and close to zero. These results prove that the trained MLFFNN is working well and efficiently to estimate correctly both the position and the orientation of the robot end-effector, simultaneously.



(a) The lowest MSE

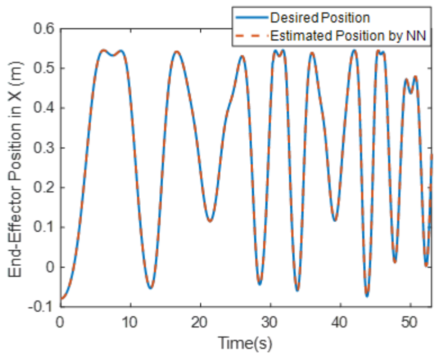


(b) The regression obtained

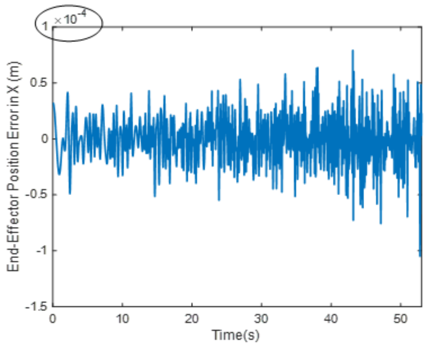


(c) Error histogram

Fig. 15. The results obtained from the training process of the designed MLFFNN using the data obtained from the joint's sinusoidal motion.

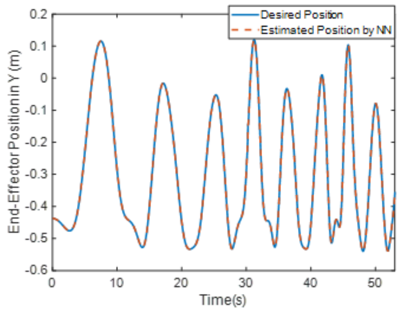


(a) The desired end-effector position x_E and the estimated one by trained MLFFNN x_E'

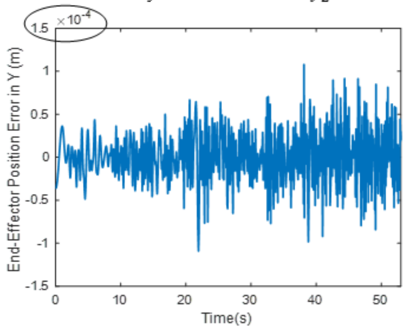


(b) Error $e_1(t) = x_E - x_E'$

Fig. 16. Comparison between the desired position of the end-effector and the estimated position by the trained MLFFNN, in x −direction.

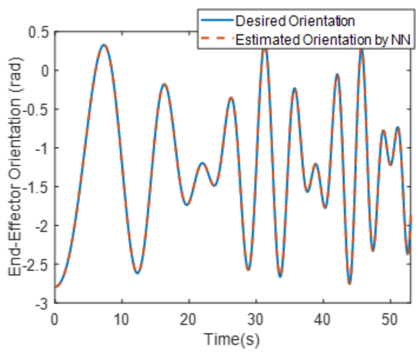


(a) The desired end-effector position y_E and the estimated one by trained MLFFNN y_E'

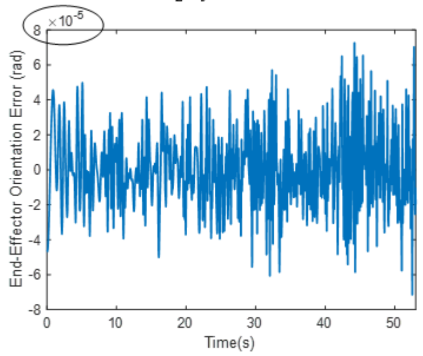


(b) Error $e_2(t) = y_E - y_E'$

Fig. 17. Comparison between the desired position of the end-effector and the estimated position by the trained MLFFNN, in y −direction.



(a) The desired end-effector orientation θ_E and the estimated orientation θ_E' by trained MLFFNN



(b) Error $e_3(t) = \theta_E - \theta_E'$

Fig. 18. Comparison between the desired orientation of the end-effector and the estimated one by the trained MLFFNN.

Indeed, the results obtained in this subsection support the results obtained in the previous subsection, namely that the MLFFNN has an ability to correctly estimate the position and the orientation of the robot end-effector, regardless the type of the joints' motion. In addition, these results prove that using one MLFFNN for estimating both the position and the orientation of the robot end-effector simultaneously is a better solution since this minimizes the effort and the time and, at the same time, very good results are obtained, compared with using one MLFFNN for estimating the position only and another MLFFNN for estimating the orientation only.

5. MLFFNN FOR INVERSE KINEMATICS SOLUTION

In this section, an MLFFNN is designed and trained to solve the problem of the inverse kinematics of the 2-DOF planar robot. Therefore, the MLFFNN will be used to find the joints' positions based on the position and the orientation of the robot end-effector. The same steps presented in Section 4 are followed here in this section.

5.1. MLFFNN Design

In this case, the main inputs for the designed MLFFNN are the position (x_E, y_E) and the orientation

θ_E of the robot end-effector, whereas its outputs are the joint variables θ_1' and θ_2'. The architecture of this MLFFNN is composed of three layers as follows: 1) the input layer which contains three inputs, 2) the non-linear (hyperbolic tangent activation function) hidden layer, and 3) the output layer which estimates joint variables θ_1' and θ_2'. The estimated joints' positions are compared with the desired ones (θ_1 and θ_2). This MLFFNN architecture is presented in Fig. 19.

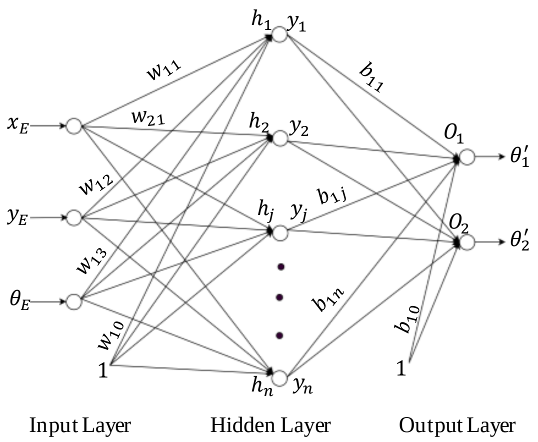


Fig. 19. The designed MLFFNN architecture for finding the joints' positions. The same symbols of the weights are used as in Fig. 6 and Fig. 10; however, their values are different. The drawing of this architecture is carried out using the following website: https://app.diagrams.net/.

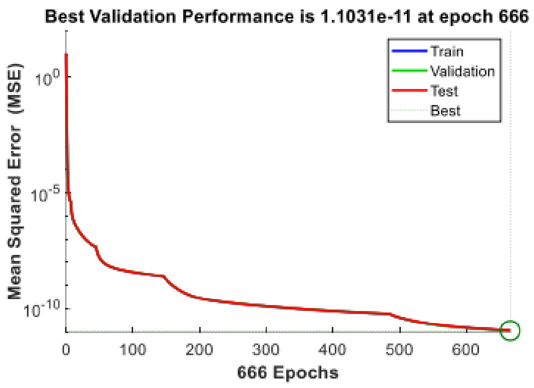
5.2. MLFFNN Training and Verification using Data Generated from Joint's Incremental Motion

The data generated from the joint's incremental motion, which is presented in Fig. 3, is used for training the designed MLFFNN. From this data, 80% is used for training, 10% for validation, and 10% for testing. After trying and testing many different weights' initializations and a number of hidden neurons, the best parameters of the MLFFNN that achieve high performance (the lowest MSE and the lowest training error) are as follows:

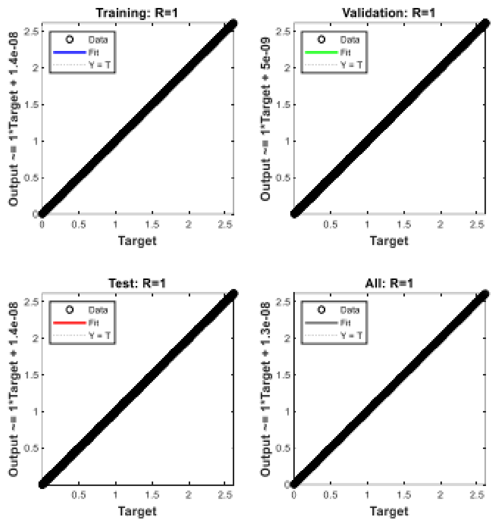
- The number of hidden neurons is 70,
- The number of iterations is 666,
- The lowest MSE is 1.1031×10^{-11}.
- The training time is 2 minutes and 44 seconds.

The results obtained from the training process are presented in Fig. 20.

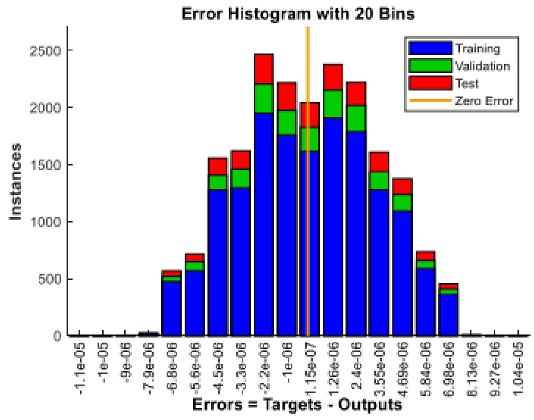
As shown from the results presented in Fig. 20, the MSE obtained is very low and it is approximately equal to zero (Fig. 20(a)). Furthermore, the regression obtained (Fig. 20(b)) is equal to 1, which means that the convergence/approximation between the desired joints' positions (θ_1, θ_2) and the estimated ones (θ_1', θ_2') by the designed MLFFNN is very good. These results prove that the designed MLFFNN is trained very well, and it has an ability to correctly estimate the joints positions/variables.



(a) The lowest MSE.

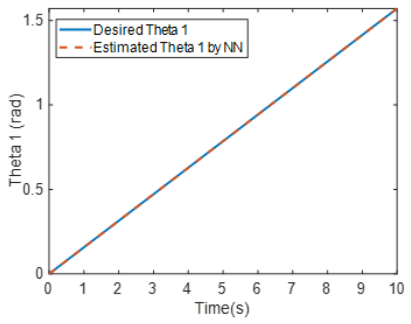


(b) The regression obtained

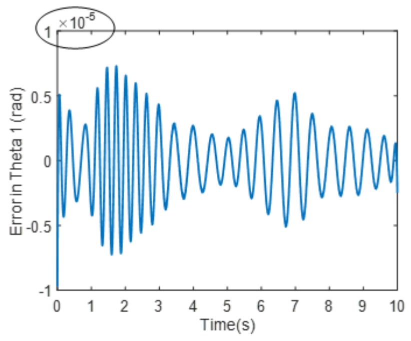


(c) Error histogram

Fig. 20. The obtained results from the training process of the designed MLFFNN used to find the joints' positions.

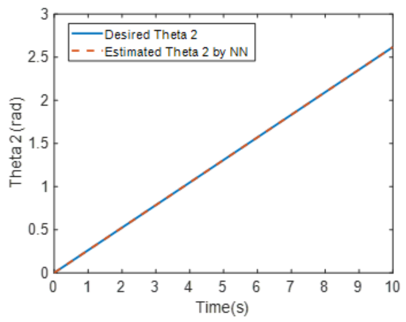


(a) The desired joint variable θ_1 and the estimated one θ_1' by the trained MLFFNN

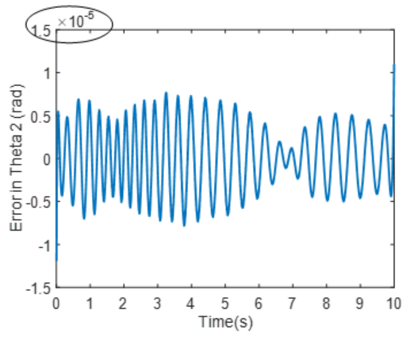


(b) Error $e_1(t) = \theta_1 - \theta_1'$

Fig. 21. Comparison between the desired joint position θ_1 and the estimated one θ_1' by the trained MLFFNN.



(a) The desired joint variable θ_2 and the estimated one θ_2' by the trained MLFFNN



(b) Error $e_2(t) = \theta_2 - \theta_2'$

Fig. 22. Comparison between the desired joint position θ_2 and the estimated one θ_2' by the trained MLFFNN.

Once the MLFFNN training is finished completely, the trained MLFFNN designed is tested and investigated using the same dataset that was used for the training to obtain an insight about the approximation. The comparison between the desired joint variables (θ_1, θ_2) and the estimated ones (θ_1', θ_2') by the trained MLFFNN is presented in Fig. 21 and Fig. 22.

As shown from the results obtained in Fig. 21 and Fig. 22, the desired joint position is coinciding with the estimated one by the trained MLFFNN. In other words, the approximation between them is approximately ideal. The approximation error is very low, and it is approximately equal to zero. From these results, we conclude that the MLFFNN is trained very well and is able to correctly find the joints' positions of the robot. In the next subsection, the data obtained from the joint's sinusoidal motion of the KUKA LWR robot is used to train and verify the MLFFNN designed.

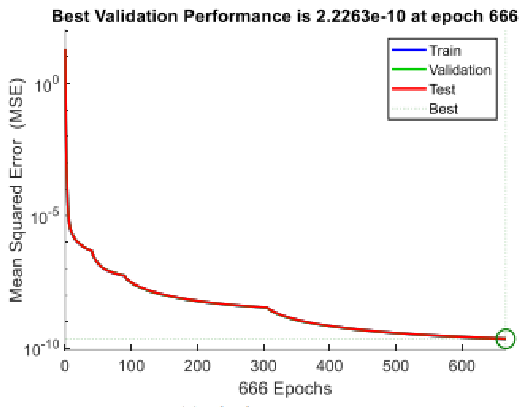
5.3. MLFFNN Training and Verification using Data from Joint's Sinusoidal Motion

The data generated from the sinusoidal joint's motion with KUKA robot (presented in Fig. 5) is used for training the designed MLFFNN presented in Fig. 19. From this data, 80% is used for training, 10% for validation, and 10% for testing. After trying and testing many different weights' initializations and a number of hidden neurons, the best parameters of the MLFFNN that achieve high performance (the lowest MSE and the lowest training error) are as follows:

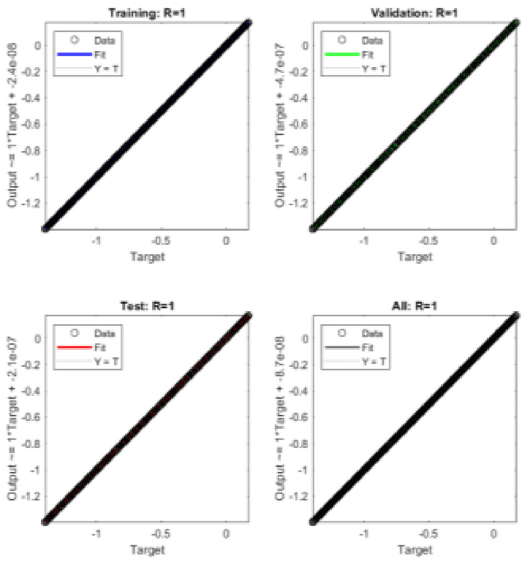
- the number of hidden neurons is 80,
- the number of iterations is 666,
- the lowest MSE is 2.2263×10^{-10},
- the training time is 19 minutes and 51 seconds,
- the results obtained from the training process are presented in Fig. 23.

As shown from the results presented in Fig. 23, the MSE obtained is very low and it is approximately equal to zero (Fig. 23(a)). Furthermore, the regression obtained (Fig. 23(b)) is equal to 1, which means that the convergence/approximation between the desired joints' positions (θ_1, θ_2) and the estimated ones (θ_1', θ_2') by the MLFFNN designed is very good. These results prove that the MLFFNN designed is trained very well, and it has an ability to correctly estimate the joints' positions/variables.

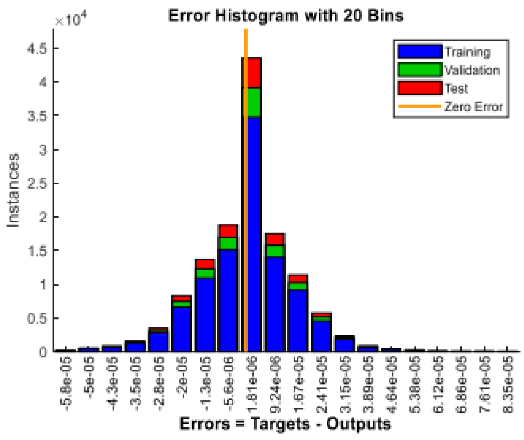
Once the MLFFNN training is finished completely, the trained MLFFNN designed is tested and investigated using the same dataset that was used for the training to obtain an insight about the approximation. The comparison between the desired joint variables (θ_1, θ_2) and the estimated ones (θ_1', θ_2') by the MLFFNN trained is presented in Fig. 24 and Fig. 25.



(a) The lowest MSE

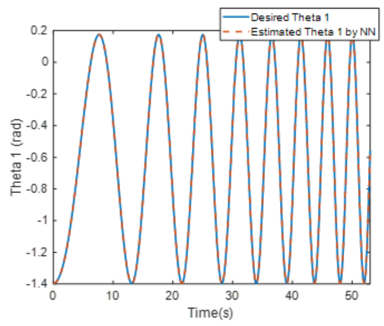


(b) The regression obtained

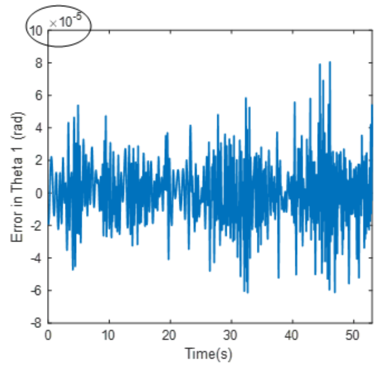


(c) Error histogram.

Fig. 23. The results obtained from the training process of the MLFFNN designed used to find the joints' positions.

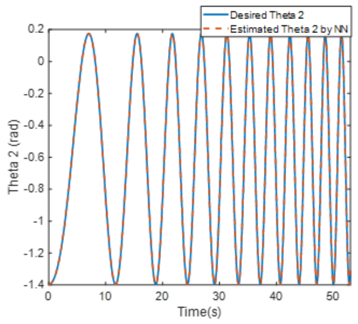


(a) The desired joint variable θ_1 and the estimated one θ_1' by the MLFFNN trained

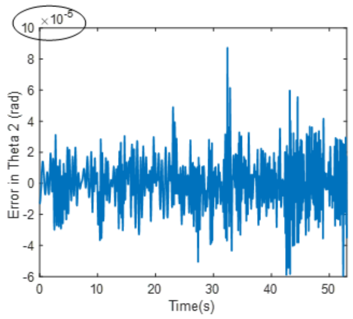


(b) Error $e_1(t) = \theta_1 - \theta_1'$

Fig. 24. Comparison between the desired joint position θ_1 and the estimated one θ_1' by the trained MLFFNN.



(a) The desired joint variable θ_2 and the estimated one θ_2' by the MLFFNN trained



(b) Error $e_2(t) = \theta_2 - \theta_2'$

Fig. 25. Comparison between the desired joint position θ_2 and the estimated one θ_2' by the trained MLFFNN.

As shown from the obtained results in Fig. 24 and Fig. 25, the desired joint position is coinciding with the estimated one by the MLFFNN trained. In other words, the approximation between them is approximately ideal. The approximation error is very low, and it is approximately equal to zero. From these results, we conclude that the MLFFNN is trained very well and is able to correctly find the joints' positions of the robot.

These results support the corresponding ones presented in the previous subsection (5.2) and prove that the MLFFNN designed can estimate the joints' variables correctly, regardless of the type of joints' motion.

The results, presented in the current section and the previous one, prove that the MLFFNN is efficient to solve the problems of the forward and inverse kinematics of the 2-DOF SCARA robot, regardless of the type of joints' motion. However, this MLFFNN could be applied to any DOF robot.

6. CONCLUSIONS AND FUTURE WORKS

In this paper, a MLFFNN is proposed for solving the problem of the forward and inverse kinematics of the robotic manipulator. For simplicity, the method proposed is applied to a 2-DOF planar robot.

For the forward kinematics solution, two cases are presented. The first case is designing and training an MLFFNN to find only the position of the robot end-effector. The second case is designing and training another MLFFNN to estimate both the position and the orientation of the robot end-effector. The results show that the MLFFNN is efficient enough to solve the forward kinematics of the manipulator, regardless of the type of the joint motion. The approximation error between the desired and the estimated position, whether in the first case or in second case, is very low and it is approximately equal to zero. The approximation error between the desired and the estimated orientation is also very low and close to zero, in the second case. This proves that whether one MLFFNN is used to find the position of the robot end-effector only and another MLFFNN is used to find only the orientation, or one MLFFNN is used to find both the robot end-effector position and orientation simultaneously, the approximation error and the MSE are very low (being approximately equal to zero). However, implementing one MLFFNN to estimate both the position and the orientation simultaneously is a better solution to minimize the effort and the time and to make the method more compact.

For the inverse kinematics solution, a MLFFNN is designed and trained to estimate the joints' positions of the manipulator. The inputs of the designed MLFFNN are the position and the orientation of the robot end-effector. The results show that the approximation error between the desired and estimated joints' positions is very low, and it is approximately equal to zero. Therefore, the MLFFNN trained is able to correctly estimate the joints positions of the manipulator.

The training of any of the MLFFNNs proposed is executed in MATLAB using the Levenberg-Marquardt algorithm. Two types of data are used for training and testing the designed MLFFNN: the first data is generated from incremental joint's motion, whereas the second data is obtained during the joint's sinusoidal motion of a real robot.

The results obtained in this paper motivate us, in near future, to use different types of NNs such as recurrent NN, cascaded forward NN, a radial basis function, etc. for solving the forward and the inverse kinematics of the manipulator. In addition, an application of the method using more complex robots such as the 7-DOF robot will be considered. The whole workspace of the robot joints will be used. More investigations will be performed for using the artificial NN to select a more desirable configuration during the solution of the inverse kinematics problem, particularly with the 6-DOF or 7-DOF manipulator.

Funding Acknowledgements and Conflict of interest

The author states that there is no conflict of interest. In addition, the author declares that no financial support has been received for this manuscript.

References

1. S. Kucuk and Z. Bingul, "Robot Kinematics: Forward and Inverse Kinematics," in *Industrial Robotics: Theory, Modelling and Control*, no. December, S. Cubero, Ed. Germany, 2006, p. 964.
2. R. Singh, V. Kukshal, and V. S. Yadav, "A review on forward and inverse kinematics of classical serial manipulators," in *Lecture Notes in Mechanical Engineering*, Springer Singapore, 2021, pp. 417–428.
3. A.-V. Duka, "Neural Network based Inverse Kinematics Solution for Trajectory Tracking of a Robotic Arm," *Procedia Technol.*, vol. 12, pp. 20–27, 2014.
4. J. Denavit and R. S. Hartenberg, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," *J. Appl. Mech. Am. Soc. Mech. Eng.*, vol. 22, no. 2, pp. 215–221, 1955.
5. M. Himanth and L. V. Bharath, "Forward Kinematics Analysis of Robot Manipulator Using Different Screw Operators," *Int. J. Robot. Autom.*, vol. 3, no. 2, pp. 21–28, 2018.
6. E. Farah and S. G. Liu, "D-H parameters and forward kinematics solution for 6dof surgical robot," *Appl. Mech. Mater.*, vol. 415, no. May, pp. 18–22, 2013.
7. R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.
8. J. -H Kim and V. R. Kumar, "Kinematics of robot manipulators via line transformations," *J. Robot. Syst.*, vol. 7, no. 4, pp. 649–674, 1990.
9. A. N. Sharkawy and N. Aspragathos, "A comparative study of two methods for forward kinematics and Jacobian matrix determination," *2017 Int. Conf. Mech. Syst. Control Eng. ICMSC 2017*, no. May, pp. 179–183, 2017.
10. Y. Liu, M. Kong, N. Wan, and P. Ben-Tzvi, "A geometric approach to obtain the closed-form forward kinematics of

H4 parallel robot," *J. Mech. Robot.*, vol. 10, no. 5, pp. 1–9, 2018.

11. K. R. Müller, A. J. Smoła, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik, "Predicting time series with support vector machines," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1997, vol. 1327, pp. 999–1004.
12. S. Haykin, *Neural Networks and Learning Machines*, Third Edit. Pearson, 2009.
13. M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.
14. A. Morell, M. Tarokh, and L. Acosta, "Solving the forward kinematics problem in parallel robots using Support Vector Regression," *Eng. Appl. Artif. Intell.*, vol. 26, no. 7, pp. 1698–1706, 2013.
15. H. Sadjadian and H. Taghirad, "Comparison of different methods for computing the forward kinematics of a redundant parallel manipulator," *J. Intell. Robot. Syst. Theory Appl.*, vol. 44, no. 3, pp. 225–246, 2005.
16. A. Ghasemi, M. Eghtesad, and M. Farid, "Neural network solution for forward kinematics problem of cable robots," *J. Intell. Robot. Syst. Theory Appl.*, vol. 60, no. 2, pp. 201–215, 2010.
17. A. A. Canutescu and R. L. Dunbrack, "Cyclic coordinate descent: A robotics algorithm for protein loop closure," *Protein Sci.*, vol. 12, no. 5, pp. 963–972, 2003.
18. T. Asfour and R. Dillmann, "Human-like Motion of a Humanoid Robot Arm Based on a Closed-Form Solution of the Inverse Kinematics Problem," in *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, 2003, no. October, pp. 1407–1412.
19. T. Ho, C. G. Kang, and S. Lee, "Efficient closed-form solution of inverse kinematics for a specific six-DOF arm," *Int. J. Control. Autom. Syst.*, vol. 10, no. 3, pp. 567–573, 2012.
20. W. K., W. K., S. J., and S. J., "Chapter 1: Kinematics," in *Handbook of robotics*, 2008, pp. 9–33.
21. N. Courty and E. Arnaud, "Inverse kinematics using sequential Monte Carlo methods," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5098 LNCS, pp. 1–10, 2008.
22. M. Sekiguchi and N. Takesue, "Fast and robust numerical method for inverse kinematics with prioritized multiple targets for redundant robots," *Adv. Robot.*, vol. 34, no. 16, pp. 1068–1078, 2020.
23. P. Kalra, P. B. Mahapatra, and D. K. Aggarwal, "An evolutionary approach for solving the multimodal inverse kinematics problem of industrial robots," *Mech. Mach. Theory*, vol. 41, no. 10, pp. 1213–1229, 2006.
24. R. Köker, "A neuro-genetic approach to the inverse kinematics solution of robotic manipulators," *Sci. Res. Essays*, vol. 6, no. 13, pp. 2784–2794, 2011.
25. S. Tejomurtula and S. Kak, "Inverse kinematics in robotics using neural networks," *Inf. Sci. (Ny).*, vol. 116, no. 2, pp. 147–164, 1999.
26. A. R. J. Almusawi, L. C. Dülger, and S. Kapucu, "A New Artificial Neural Network Approach in Solving Inverse Kinematics of Robotic Arm (Denso VP6242)," *Comput. Intell. Neurosci.*, vol. 2016, pp. 1–10, 2016.
27. A. Csiszar, J. Eilers, and A. Verl, "On solving the inverse kinematics problem using neural networks," *2017 24th Int. Conf. Mechatronics Mach. Vis. Pract. M2VIP 2017*, vol. 2017-Decem, pp. 1–6, 2017.
28. A.-N. Sharkawy and N. Aspragathos, "Human-Robot Collision Detection Based on Neural Networks," *Int. J. Mech. Eng. Robot. Res.*, vol. 7, no. 2, pp. 150–157, 2018.
29. A.-N. Sharkawy, P. N. Koustoumpardis, and N. Aspragathos, "Manipulator Collision Detection and Collided Link Identification based on Neural Networks," in *Advances in Service and Industrial Robotics. RAAD 2018. Mechanisms and Machine Science*, A. Nikos, K. Panagiotis, and M. Vassilis, Eds. Springer, Cham, 2018, pp. 3–12.
30. A. N. Sharkawy, P. N. Koustoumpardis, and N. Aspragathos, "Neural Network Design for Manipulator Collision Detection Based only on the Joint Position Sensors," *Robotica*, vol. 38, no. Special Issue 10: Human–Robot Interaction (HRI), pp. 1737–1755, 2020.
31. A. N. Sharkawy, P. N. Koustoumpardis, and N. Aspragathos, "Human–robot collisions detection for safe human–robot interaction using one multi-input–output neural network," *Soft Comput.*, vol. 24, no. 9, pp. 6687–6719, 2020.
32. A.-N. Sharkawy and A. A. Mostfa, "Neural Networks' Design and Training for Safe Human-Robot Cooperation," *J. King Saud Univ. - Eng. Sci.*, 2021.
33. A.-N. Sharkawy and P. N. Koustoumpardis, "Dynamics and computed-torque control of a 2-DOF manipulator: Mathematical analysis," *Int. J. Adv. Sci. Technol.*, vol. 28, no. 12, pp. 201–212, 2019.
34. Y. D. Patel and P. M. George, "Performance Measurement and Dynamic Analysis of Two Dof Robotic Arm Manipulator," *Int. J. Res. Eng. Technol.*, vol. 02, no. 09, pp. 77–84, 2013.
35. H. H. Asada, *Introduction to Robotics*. Department of Mechanical Engineering, Massachusetts Institute of Technology, 2005.
36. R. V. Neeraj Kumar and R. Sreenivasulu, "Inverse Kinematics (IK) Solution of a Robotic Manipulator using PYTHON," *J. Mechatronics Robot.*, vol. 3, no. 1, pp. 542–551, 2019.
37. J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.
38. A.-N. Sharkawy, "Principle of Neural Network and Its Main Types: Review," *J. Adv. Appl. Comput. Math.*, vol. 7, pp. 8–19, 2020.
39. S. C. Chen, S. W. Lin, T. Y. Tseng, and H. C. Lin, "Optimization of back-propagation network using simulated annealing approach," in *2006 IEEE International Conference on Systems, Man and Cybernetics*, 2006, pp. 2819–2824.
40. M. A. Sassi, M. J. D. Otis, and A. Campeau-Lecours, "Active stability observer using artificial neural network for intuitive physical human–robot interaction," *Int. J. Adv. Robot. Syst.*, vol. 14, no. 4, pp. 1–16, 2017.
41. E. De Momi, L. Kranendonk, M. Valenti, N. Enayati, and G. Ferrigno, "A Neural Network-Based Approach for Trajectory Planning in Robot–Human Handover Tasks," *Front. Robot. AI*, vol. 3, no. June, pp. 1–10, 2016.
42. A. N. Sharkawy, P. N. Koustoumpardis, and N. Aspragathos, "A neural network-based approach for variable admittance control in human–robot cooperation: online adjustment of the virtual inertia," *Intell. Serv. Robot.*, vol. 13, no. 4, pp. 495–519, 2020.
43. A.-N. Sharkawy, P. N. Koustoumpardis, and N. Aspragathos, "A recurrent neural network for variable admittance control in human–robot cooperation: simultaneously and online adjustment of the virtual damping and Inertia parameters," *Int. J. Intell. Robot. Appl.*, vol. 4, no. 4, pp. 441–464, 2020.
44. A. B. Rad, T. W. Bui, V. Li, and Y. K. Wong, "A NEW ON-LINE PID TUNING METHOD USING NEURAL NETWORKS," *IFAC Proc. Vol. IFAC Work. Digit. Control Past, Present Future. PID Control*, vol. 33, no. 4, pp. 443–448, 2000.
45. S. A. Elbelady, H. E. Fawaz, and A. M. A. Aziz, "Online Self Tuning PID Control Using Neural Network for Tracking Control of a Pneumatic Cylinder Using Pulse Width Modulation Piloted Digital Valves," *Int. J. Mech.*

Mechatronics Eng. IJMME-IJENS, vol. 16, no. 3, pp. 123–136, 2016.
46. R. Hernández-Alvarado, L. G. García-Valdovinos, T. Salgado-Jiménez, A. Gómez-Espinosa, and F. Fonseca-Navarro, “Neural Network-Based Self-Tuning PID Control for Underwater Vehicles,” *sensors*, vol. 16, no. 9: 1429, pp. 1–18, 2016.
47. P. Jeatrakul and K. W. Wong, “Comparing the performance of different neural networks for binary classification problems,” in *2009 8th International Symposium on Natural Language Processing, SNLP ’09*, 2009, pp. 111–115.
48. D. Anderson and G. McNeill, “Artificial neural networks technology: A DACS state-of-the-art report,” Utica, New York, 1992.
49. K. Du and M. N. S. Swamy, *Neural Networks and Statistical Learning*. Springer, 2014.
50. D. W. Marquardt, “An Algorithm for Least-Squares Estimation of Nonlinear Parameters,” *J. Soc. Ind. Appl. Math.*, vol. 11, no. 2, pp. 431–441, 1963.
51. M. T. Hagan and M. B. Menhaj, “Training Feedforward Networks with the Marquardt Algorithm,” *IEEE Trans. NEURAL NETWORKS*, vol. 5, no. 6, pp. 2–6, 1994.

Biographical note

Abdel-Nasser Sharkawy graduated with a first-class honors degree (B.Sc.) in May 2013 from Mechatronics Engineering, Mechanical Engineering Department, South Valley University (SVU) and followed this nomination as a demonstrator (a teaching assistant) at the same university. Sharkawy received his M.Sc. degree in Mechatronics Engineering in April 2016 and followed this nomination as an assistant lecturer at the same university (SVU). In March 2020, Sharkawy received his Ph.D. degree from Robotics Group, Department of Mechanical Engineering and Aeronautics, University of Patras, Patras, Greece. His PhD was about “Intelligent Control and Impedance Adjustment for Efficient Human-Robot Cooperation”. Dr. Sharkawy has been a lecturer (Assistant Professor) at Mechatronics Engineering, Mechanical Engineering Department, SVU, Egypt from June 2020 to present as a permanent Job. During this period, Sharkawy had a good experience for teaching the following under-graduate courses: Automatic Control, Theory of Machines and Mechanisms, Measurements and Instrumentations, and Engineering Mathematics, and the following postgraduate courses: Robotics Dynamics and Advanced Control Theory and Its Applications I. From (15 April to 30 November), 2021, Sharkawy was a Postdoctoral Researcher at Humanoid and Cognitive Robotics Group, Department of Cybernetics, Faculty of Electrical Engineering, the Czech Technical University in Prague, Prague, the Czech Republic. From September 2022, Sharkawy is an assistant professor at Mechanical Engineering Department, College of Engineering, Fahad Bin Sultan University, Tabuk, Saudi Arabia. Sharkawy has published more than 30 papers in international scientific journals, book chapters and international scientific conferences. His published work has attracted more than 375 citations (h-index: 10). He is a member of the editorial board of the International Journal of Robotics and Control Systems (IJRCS), SVU-International Journal of Engineering Sciences and Applications (SVU-IJESA), and Journal of Advances in Applied & Computational Mathematics (2020-2022). He serves as a reviewer for ca. 28 journals and 10 conferences. He was a student member of IEEE RAS (2018-2019). His research areas of interest include robotics, human-robot interaction, mechatronic systems, neural networks as well as control and automation.