JACET

J. ADV COMP ENG TECHNOL, 6(2) Spring 2020

Using a Neural Network instead of IKM in 2R Planar Robot to follow a rectangular path

First A. Ali Ahmad Ali[1]
1- Mechatronics , Mechanical and Electrical Engineering , Tishreen University , Lattakia , Syria.
(ali.ali.journals@gmail.com)

Received (2019-09-23) Accepted (2020-04-12)

Abstract: *An educational platform is presented here for the beginner students in the Simulation and Artificial Intelligence sciences. It provides with a start point of building and simulation of the manipulators, especially of 2R planar Robot. It also displays a method to replace the inverse kinematic model (IKM) of the Robot with a simpler one, by using a Multi-Layer Perceptron Neural Network (MLP-NN), to make the end-effector able to track a specific path, which has a rectangular shape (in this article), and allocated in the robot's workspace. The method is based on Back-Propagation Levenberg Marquardt algorithm. This paper also suggests a good strategy for the simulation of the robot's motion in Matlab to tell the beginners how the operation could be done quite closely to the built-in Matlab functions. The control part was ignored here for the simplicity. So we can classify this paper as a manual in the robotic world.*

Keywords: *Back-Propagation(BP), Denavit Hartenberg (DH), Direct Kinematic Model (DKM), Inverse Kinematic Model (IKM), Neural Network (NN).*

How to cite this article:
First A. Ali Ahmad Ali. Using a Neural Network instead of IKM in 2R Planar Robot to follow a rectangular path. J. ADV COMP ENG TECHNOL, 6(2) Spring 2020 : 71-78

I. INTRODUCTION

With the development of the robotic sciences, the importance and exciting of robots have grown up day by day. We can provide them with an Artificial Intelligence techniques to be able to mimic the human's performance in many fields such as industry, agriculture, domestic robots, etc. [2, 1, 3]. They can be a big assistance for the person in his daily life. One common type of those manipulators is 2R planar robot. It has two links (i.e. two degrees of freedom). Actually it is very similar to the human arm, but this robotic arm works only in 2D plane, it can also be used as a plotter machine in some cases [4]. This work splits into two main sections: Simulation, and the robot's IKM solution based on a Neural Network. The results of, and the used figures and tables in this manual are exited in sections five and six respectively. The seventh section is specified for conclusions. In other words this article could be a useful tool especially for the students in mechatronics and robotic departments.

II. RELATED WORKS

In April 1991 , Stuart Kieffer, Vassilios Morellas and Max Donath published their paper : "*Neural Network Learning of the Inverse Kinematics Relationships for a Robot Arm*", in which a Widrow-Hoff based Kohonen's (SOM) neural network is learnt the inverse kinematic model of a 2 DOF robotic arm , and the results show that the suggested approximation is indeed successful [6].

In 2007 "*Simulation of Robotic Arm using*

Genetic Algorithm & AHP" article suggested a method to reduce the cost of a 3 DOF robotic arm's motion using Genetic Algorithm and Analytical Hierarchy Process by V. K. Banga, Y. Singh, and R. Kumar. This paper simulated and tested the inverse kinematics, fitness value evaluation and binary encoding like tasks . Movement, friction and least settling time (or min. vibration) are used for finding the fitness function / fitness values [7].

Wong Guan Hao, Yap Yee Leck and Lim Chot Hun edited their paper "*6-Dof pc-based robotic arm (PC-ROBOARM) with efficient trajectory planning and speed control*" in 2011 , which introduces the design and development of 6-DOF (degree of freedom) PC-Based Robotic Arm (PC-ROBOARM) . The robotic arm design and control solution is implemented by self-developed computer software which is called *SMART ARM* . It allows user to model or design virtual robotic arm before building the real one. Therefore, the user can estimate the optimum size of actual robotic arm at the beginning so as to minimize the building cost and suite the practical environment. Furthermore, once the actual robotic arm has been built, the user can reuse the software to control the actual robotic arm in an effortless way without wasting time in constructing new control solution [8].

A novel learning-based multiscale modelling approach is proposed in [5] to address the efficiency and accuracy issues through a combination of models with different levels of fidelity. Specifically, low-fidelity models are formulated using the Kernel Ridge Regression (KRR), which has a much lower computational cost compared with kinematic approximation models. Additionally, in order to eliminate position singularity of a serial manipulator, high-fidelity models are based on the Long Short Term Memory (LSTM) neural network are also created to calibrate fidelity by training the significant fidelity samples.

III. SIMULATION OF THE ROBOTIC ARM

1. The Model of The Arm

In this section, the model of the robot is built. *Firstly*, we establish the joint coordinates frames. Frame(**0**) is for the first link and it is allocated exactly at the center of the joint one. Frame(**1**) is of the second link, and it is centered exactly at the joint two. Frame(**2**) is centered at the grip joint. According to the right-hand rule we can conclude that Z_0, Z_1 and Z_2 axes are perpendicular on (X_0-Y_0), (X_1-Y_1) and (X_2-Y_2) planes, outside of the paper. So the frames of the robot will be like those in figure (1).

Secondly, We need to fill the DH Parameters Table (Look at table 1), Where:

$\mathbf{a_i}$: is the distance between Z_{i-1} and Z_i axes along X_i axes.

$\boldsymbol{\alpha_i}$: is the angle between Z_{i-1} and Z_i axes along X_i axes.

$\mathbf{d_i}$: is the distance between X_{i-1} and X_i axes along Z_{i-1} axes.

$\boldsymbol{\Theta_i}$: is the angle between X_{i-1} and X_i axes along Z_{i-1} axes.

Thirdly, we write DH Transformation Matrix:

$$\mathbf{T^i_{i-1} = Rot_{z,\Theta i}.Trans_{z,di}.Trans_{x,ai}.Rot_{x,\alpha i}} \quad [9]$$

$$T^i_{i-1} = \begin{bmatrix} C_{\Theta_i} & -S_{\Theta_i}.C_{\alpha_i} & S_{\Theta_i}.S_{\alpha_i} & a_i.C_{\Theta_i} \\ S_{\Theta_i} & C_{\Theta_i}.C_{\alpha} & -C_{\Theta_i}.S_{\alpha_i} & a_i.S_{\Theta_i} \\ 0 & S_{\alpha_i} & C_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

$$T_0^{\,1} = \begin{bmatrix} C_1 & -S_1 & 0 & L_1.C_1 \\ S_1 & C_1 & 0 & L_1.S_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

$$T_1^{\,2} = \begin{bmatrix} C_2 & -S_2 & 0 & L_2.C_2 \\ S_2 & C_2 & 0 & L_2.S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

$$T_0^{\,2} = T_0^{\,1} \times T_1^{\,2} \quad (4)$$

$$T_0^2 = \begin{bmatrix} C_{12} & -S_{12} & 0 & L_2.C_{12}+L_1.C_1 \\ S_{12} & C_{12} & 0 & L_2.S_{12}+L_1.S_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

Where:
$C_1 = \cos(\Theta_1)$, $S_2 = \sin(\Theta_2)$,
$C_{12} = \cos(\Theta_1+\Theta_2)$, $S_{12} = \sin(\Theta_1+\Theta_2)$.

2. The Direct Kinematic Model of The Robot

DKM is used to calculate the (X-Y-Z) coordinates of the grip depending on the values of the links angles. From the last column in (5), we obtain:

$$X = L1.\cos(\theta_1) + L2.\cos(\theta_1+\theta_2) \quad (6)$$
$$Y = L1.\sin(\theta_1) + L2.\sin(\theta_1+\theta_2) \quad (7)$$
$$Z = 0 \quad (8)$$

Those three equations form " Direct Kinematic Model of Two-link planar manipulator (DKM) ".

From (5) , The Rotation Matrix is:

$$R = \begin{bmatrix} C_{12} & -S_{12} & 0 \\ S_{12} & C_{12} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} Xx & Yx & Zx \\ Xy & Yy & Zy \\ Xz & Yz & Zz \end{bmatrix} \quad (9)$$

Euler Parameters [9]:

If $Zz \neq \pm 1$, then

$$\Theta = \text{acos}(Zz) \quad (10)$$
$$\Phi = \text{atan2}(Zx,-Zy) \quad (11)$$
$$\Psi = \text{atan2}(Xz,Yz) \quad (12)$$

Roll – Pitch – Yaw Parameters [9] :

$$\beta = \text{atan2}(-Xz, \pm\sqrt{Xx^2 + Xy^2}) \quad (13)$$

If $\beta \neq \pm \pi/2$, then

$$\alpha = \text{atan2}(Xy,Xx) \quad (14)$$
$$\gamma = \text{atan2}(Yz,Zz) \quad (15)$$

3. The Inverse Kinematic Model of The Robot

IKM is used to calculate the links angles depending on the values of (X-Y-Z) coordinates. From (6) and (7), we can generate Θ_1 and Θ_2 equations:

$(6)^2 + (7)^2 =$

$$X2 + Y2 = L1^2 + L2^2 + 2.L1.L2 . \cos(\theta 2)$$

So:

$$\cos(\theta 2) = \frac{X^2 + Y^2 - L1^2 - L2^2}{2L1.L2} \quad (16)$$

It is a standard formula of a trigonometric equation of the form:

$\cos(\theta 2) = a$

We can solve it as the follow:

$$\theta 2 = atan2\left(\pm\sqrt{1-a^2}, a\right) \quad (17)$$

After calculating Θ_2 values, we replace Θ_2 in (7):

$$Y = L2.\sin(\theta 2).\cos(\theta 1) + (l1+L2.\cos(\theta 2)).\sin(\theta 1)$$
$$Y = a . \cos(\theta 1) + b . \sin(\theta 1) \quad (18)$$

We can solve it as the following:

$$\theta 1 = atan2\left(Y, \pm\sqrt{a^2+b^2-Y^2}\right) - atan2(a,b) \quad (19)$$

That was the basic model of the 2R planar robot. It contains DKM in which we know the values of the links' angles and then we conclude the (X-Y-Z) Coordinates of the grip, and there is IKM in which we know the (X-Y-Z) Coordinates and we want to calculate the angles of the two links.

4. Simulation Algorithm

In this section the simulation of the 2R Planar Robot is presented using basic programming instructions. I need to design a function to determine the workspace of the robot. It just takes the lengths (l_1 and l_2) of the links, and returns two circles where the workspace of the robot is between them, Fig (3). Next there is a function used to calculate the $T0^2$ matrix called "total_Tran". Another one called "Tran" is used to calculate $T0^1$ and $T1^2$ matrices respectively.

Next, two functions called: "Euler" and "R_P_Y" are designed to calculate Euler and Roll-Pitch-Yaw parameters. From (9) we can note that Zz equals to (1). So (Theta = 0) and

(Phi=Psi=Nan) in all cases.

Now it is necessary to show the motion of the robot. Hence I suggest a method to animate this motion Depending on the successive image display (image by image). The animation steps of the link one will be explained reaching to a defined point in the workspace, and the other link uses the same steps. *First*, the whole rotation angle of the link one is acquired (Θ_1) in radian, then it is segmented into smaller angles each segment is equaled to (t_1) *therefore*, a step value for the motion should be defined in advance – the step value will be better if it is as small as possible to make the animation process smooth – . At each segment/frame the position of the second joint should be calculated using "Tran" function which has two parameters (the updated frame angle, and the fixed length of the first link), Note that "Tran" function gets the updated angle (angle of previous frame plus the step t_1) towards the goal point.

This method is good if the arm moves from the origin point to another one counterclockwise, but if we want to move the arm in a clockwise direction, then we should subtract the step (t_1) from the previous frame angle. So, for one link there are two cases of motion (CW and CCW), but there will be four possible cases for two links: (Look at table 2).

The second link animation is done by the same algorithm. It should be mentioned that the number of segments/ frames of the total animation is determined by the maximum integer number of the frames of both links. You can see the animation algorithm box diagram in figure (2) .

In this section the algorithm used to simulate the 2R Planar Robot is designed and explained.

IV. THE ARCHITECTURE OF THE NEURAL NETWORK

Now, I will illustrate how to build the neural network used in the application of this article. I suppose the path we want to follow has a rectangular shape. we should mention that the path must be allocated in the workspace of the Robot, else the Robot may face singularity cases. I use a Multi-Layer Perceptron (MLP), and Back Propagation algorithm (BP) to train the network. The main purpose of using the Neural Network in this manual is: controlling of the end effector to follow a specific path without using IKM or even DKM. This thing is important for the Robot controlling process, because of the grip can follow any path regardless of the mathematical formula of that path, it just needs to detect the sample points from the path using some sensors for one time. And the most exciting thing is: anyone can control the grip even if he/ she does not have any mathematical background of trigonometric and mathematical equations, he just should move the end effector on the specific path for the first one, and lets the rest of the operation to the neural network.

Designing steps of the NN are:

1) Choosing of training patterns.

2) Determining the number of Inputs and outputs of the network and Identifying them.

3) Specifying the number of hidden layers neurons.

4) Choosing of the transfer functions used in the NN.

1. Choosing of training patterns

Training samples are some points acquired from the rectangular path. The whole sample points number is equaled to (QW = 148 points), I will separate them into two groups, the first one is called "Training Group" , it contains (80.405%) of QW (i.e. QT = 119 points) and the second one is called "Generalization Group", it contains (19.595%) of QW (i.e. QG = 29 points). This group is used to test the NN's ability to generalize. We can choose the elements of QT and QG groups as the following: (starting from the first element in QW Group, *then* counting four elements of it. Put all of the last elements in the QT Group, then Put the fifth element in the QG Group. Repeat the same steps until reaching to the final element in the QW).

2. Determining the number of inputs and outputs of the NN and identifying them

Since we want to replace the IKM of the Robot by NN, so the inputs of the network are (X ,Y ,Z) coordinates of the sample points, and the outputs are the rotation angles of both links (Θ_1, Θ_2). I will ignore Z coordinates, because it is always equaled to zero. Hence the number of the NN's inputs is two, and the number of the NN's outputs is also

two.

3. Specifying the number of hidden layer neurons

Actually it is an experimental issue, depending on trial and error, but there is a constraint should not be exceeded. It is about the Generalization problem which is: "the parameters of the NN should be equal to or even smaller than the training patterns Q_T" [10] .

I depend on this rule to limit the hidden layer neurons number. In this article, I use three layers (Input – Hidden – Output) for the NN. The parameters number of the NN is distributed as the following: In the first layer there are two inputs (i.e. R=2). In the Hidden layer there are S1 neurons and b1 biases (one bias for each neuron). Finally in the output layer there are two neurons ($S_2 = 2$), and also two bias ($b_2 = 2$). The parameters of the NN involves the weights and biases used in the network. The weight matrix between the input and hidden layers has a [S_1. R] size. The biases of the hidden layer has a [S_1.1] size. The weight matrix between the hidden and output layers has a [S_2.S_1] size. The biases of the output layer has a [S_2.1] size,

Here is the previous constraint mathematical expression:

$$W_1 + b_1 + W_2 + b_2 \leq Q_T \quad (20)$$

$$[S_1.R] + [S_1.1] + [S_2.S_1] + [S2.1] \leq 119$$

$$2.S_1 + S_1 + 2.S_1 + 2 \leq 119$$

$$So : S_1 \leq 23 \quad (21)$$

I tried multiple scenarios by changing S_1 and epoch values, and the results are illustrated in the figure (4) representing the relationship between Θ_1 and Θ_2 at the same inputs. The blue color in that figure refers to desired output of the NN, the red one refers to the actual output of the network, and the black one represents the NN response to the generalization group elements (testing points).

We can note in figure (4), that all the scenarios have a hidden neurons number less than (23). It is also obvious that the best result from the same figure is (d). which have only one big error.

At the upper left corner of figure (4) we have scenario(a), in which (S_1=10 neurons, epochs=1,000). On the right of (a) sub-figure there is the scenario(b), in which (S_1 = 10 neurons, epochs = 50,000); (c): (S_1 = 10 neurons, epochs = 1,000,000); and so on.

We continue the same method until we reach to an acceptable result. I choose this one: (S_1 = 11, and epochs = 50,000).

4. Choosing the transfer functions used in the NN

We can conclude that our problem is a function approximation matter. In figure (4) we have a blue curve presenting one function (the relationship between Θ_1 and Θ_2), and we want to make the neural network converge it. *Actually*, "we can converge any function using *Back Propagated based Neural Network* contains one hidden layer with **sigmoid** transfer function, and one output layer with **linear** transfer function, if there are enough neurons in the hidden layer" [10] . So that, the hidden transfer function is **sigmoid**:

$$f(n) = \frac{1}{1 + e^{-n}} \quad (22)$$

And the output transfer function is linear:

$$f(n) = n \quad (23)$$

V. THE RESULTS

To speed up the convergence process I used *Levenberg Marquardt* method by using "trainlm" function in Matlab with initial value of $\mu = 0.001$, and the last value of μ is 1.00e-07 [10]. The performance function is *Mean Squared Error* (the goal error=0, the actual performance= 1.90e-05), train time = 0:00:01, iterations = 244, Gradient =0.00221, cost of the NN-based solution of Robot's IKM is 2.047888s. We should tell about one more thing: since the robot displays only one solution for IKM in all cases, so if Θ_1 is equal to or even smaller than (180o), the link (1) will rotate counter clockwise CCS, but if Θ_1 is bigger than (180o), the link will rotate with clock wise CS. To avoid that reflection especially when the robot follows a continues path, I add (+360o) to the value of Θ_1 angle if it is more than (180o); (i.e. I use the complementary of Θ_1 to 3600 when the angle/ rotation is negative).

System setup: (Intel® Core™2 Duo CPU T6500 @ 2.10 GHZ – RAM 4GB – Windows 64 bits), Matlab version: (R2016a).

The training process produced these values for the parameters of the NN:

$$W^1 = \begin{bmatrix} 44.7945 & 2.4519 \\ -19.8769 & -3.86 \\ 0.56371 & 1.605 \\ -10.9511 & -2.2423 \\ 1.4341 & 0.27999 \\ 4.6931 & 6.5116 \\ -2.8344 & 2.526 \\ 4.1951 & 5.6292 \\ -2.6133 & 1.7312 \\ -3.9916 & 3.5954 \\ 1.7696 & -1.5949 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} 20.4074 & 0.12878 \\ 7.6798 & 0.28748 \\ -0.075889 & 9.3342 \\ 0.39188 & -0.082484 \\ 4.9961 & -11.7411 \\ 1.4487 & 0.064866 \\ -4.875 & -2.5575 \\ -2.0416 & -0.064457 \\ 6.1304 & 1.1664 \\ -1.0744 & -1.2515 \\ -1.1154 & 7.382 \end{bmatrix}^T$$

$$b^1 = \begin{bmatrix} -48.9365 \\ 25.0821 \\ 2.0273 \\ 10.0613 \\ -1.9259 \\ 1.4264 \\ 0.29026 \\ 1.3632 \\ -0.45523 \\ -2.7754 \\ 3.5222 \end{bmatrix}$$

$$b^2 = \begin{bmatrix} -7.5819 \\ -10.8979 \end{bmatrix}$$

VI. FIGURES AND TABLES

Figure (3) shows the final application Graphical User Interface and DKM of the two-link Planar manipulator. You can see the workspace of the Robot between two red circles. We type the values of Θ_1 and Θ_2, then the application calculates the X-Y-Z coordinates. You can also note that the application calculates both of Euler and R-P-Y parameters, as expected (Teta = 0) and (Phi = Psi = NaN) in all cases.

Figure (5) shows the grip of the Robot following a rectangular blue path; We can note how accurate the result is. The approximation result of the training process with (S_1 = 11 neurons in the hidden layer, and epochs = 50,000) is illustrated in figure (6). We can note that the convergence is acceptable. Figure (7) shows the end effector of the Robot following the same specific rectangular path using the neural network.

VII. CONCLUSIONS

This work is a manual for the students. It shows the IKM solution using a neural network trained by Levenberg Marquardt algorithm. It also suggests a practical method to simulate the motion of the 2R Planar Robot with Programming from scratch. The controlling results of the grip with and without IKM are largely Identical. We saw some acceptable aliasing in NN result in fig. (7), while the IKM result is so straight fig. (5). Using NN instead of IKM makes the following task easier regardless the mathematical formula of the path, or even the student's mathematical background, but it costs more time than IKM (2.047888s for NN vs. 0.676140s for IKM). The used NN is valid only for the last rectangular path, but if we want to use another path, we should take a new sample points from it, and repeat the same stages to train the NN again.

ACKNOWLEDGMENT

"F. A. Ali thanks both of Dr. Tammam haydar (Tishreen university) for his supervision to create this work, and Dr. Fadi Motawej (Tishreen university) for his supportive efforts to review this work before submission."

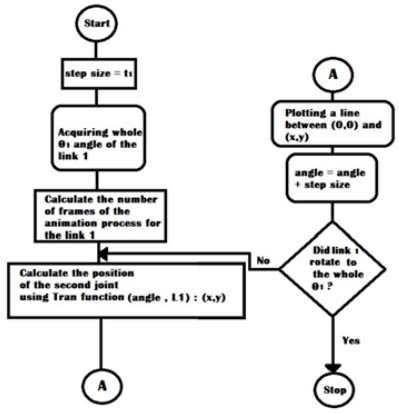

Fig. 2. The animation algorithm.

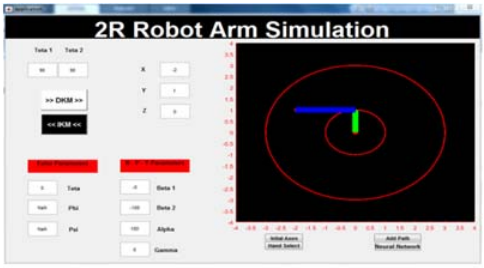

Fig. 3. The DKM of the 2R Robot in the final application.

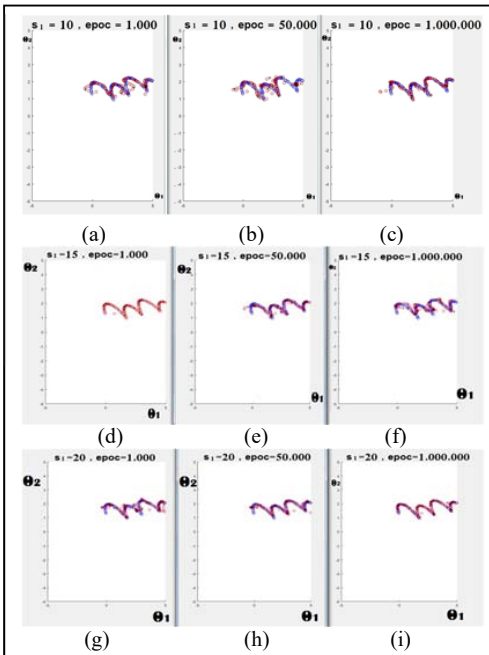

Fig . 4. Using trial and error to detect the number of the hidden neurons. [o: training, +: generalization, o: target] patterns.

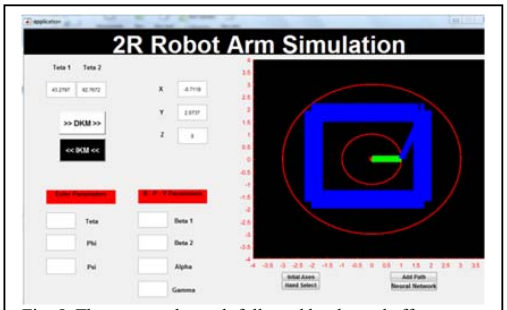

Fig. 5. The rectangular path followed by the end effector of the manipulator using IKM.

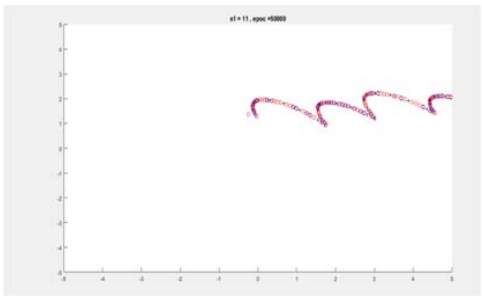

Fig. 6. The Convergence final result (the relationship between Θ_1 and Θ_2).
* Horizontal axes is Θ_1, and Vertical axes is Θ_2.

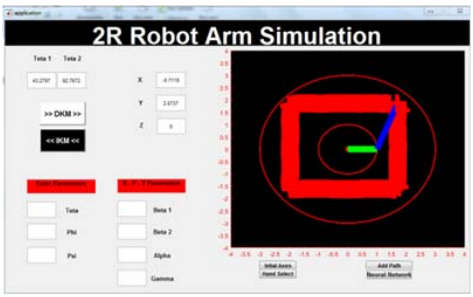

Fig. 7. Tracking a rectangular Path using the NN.

TABLE I
DENAVIT HARTENBERG PARAMETERS

Link	a_i	α_i	d_i	Θi
1	L1	0	0	Θ_1^*
2	L2	0	0	Θ_2^*

TABLE II
POSSIBLE ROTATION DIRECTIONS FOR TWO LINKS

L_1 's motion	L_2 's motion
CW	CW
CW	CCW
CCW	CW
CCW	CCW

VIII. REFERENCES

1. Dorsey, N. Top 5 Robotic Systems to Watch in Agriculture. 2018 [cited 2019 19 September]; Available from: https://www.precisionag.com/in-field-technologies/top-5-robotic-systems-to-watch-in-agriculture/.
2. Bouchard, S. Industrial robots: What are the different types? 2014 [cited 2019 19 September]; Available from: https://blog.robotiq.com/bid/63528/what-are-the-different-types-of-industrial-robots.
3. Household robots. [cited 2019 19 September]; Available from: http://www.allonrobots.com/household-robots.html.
4. [cited 2019 19 September]; Available from: https://www.youtube.com/watch?v=ZWZGfWU8_p0.
5. Zhao, J., et al., A learning-based multiscale modelling approach to real-time serial manipulator kinematics simulation. Neurocomputing, 2019; Available from: https://www.sciencedirect.com/science/article/abs/pii/S0925231219314456.
6. Kieffer, S., V. Morellas, and M. Donath. Neural network learning of the inverse kinematic relationships for a robot arm. in Proceedings. 1991 IEEE International Conference on Robotics and Automation. 1991. IEEE.
7. Banga, V., Y. Singh, and R. Kumar, Simulation of robotic arm using genetic algorithm & AHP. World Academy of Science, Engineering and Technology, 2007. 25(1): p. 95-101; Available from: https://pdfs.semanticscholar.org/cbef/ee29e571ec3a50637e8de3c6f6ce56d00338.pdf.
8. Hao, W.G., Y.Y. Leck, and L.C. Hun. 6-DOF PC-Based Robotic Arm (PC-ROBOARM) with efficient trajectory planning and speed control. in 2011 4th International Conference on Mechatronics (ICOM). 2011. IEEE.
9. Spong, M.W., S. Hutchinson, and M. Vidyasagar, Robot modeling and control. 2020: John Wiley & Sons.
10. Beale, H.D., H.B. Demuth, and M. Hagan, Neural network design. Pws, Boston, 1996; Available from: https://pdfs.semanticscholar.org/6a81/22861b80842ee6f406acdeec35aec913f1b8.pdf.