A Two-Axis Robotic Arm Controller Design Using Exact Radial Bases And General Regression Neural Networks With FPGA Technique

Lecturer. Mohannad Abid Shehab Ahmed *Lecturer. Emad Ahmed Hussien*
Asst. Lecturer. Haithem Abd Al-Raheem Taha
Electrical Department, Engineering College, Al-Mustansiriyah University.

Abstract *:*

Although neural networks applicable to the solution of robotics control problems are, in fact, neurocontrollers, their function is specialized mainly to provide solutions to robot arm motion problems. In this paper, two types of neural network models are applied for solving a number of robot kinematics problems these types are: Exact Radial Bases Neural Network (RBENN) and General Regression Neural Networks (GRNN). Learning the robot arms motions to achieve the desired final position are the main kinematics tasks covered here. Hardware realization of a Neural Network, to a large extent depends on the efficient implementation of a single neuron. Field Programmable Gate Array FPGA-based reconfigurable computing architectures are suitable for hardware implementation of neural networks. FPGA realization of NNs with a large number of neurons is still a challenging task. This paper also discusses the issues involved in implementation of a multi-input neuron with linear/nonlinear excitation functions using VHDL programming language for FPGA.

Key words: *Two-Axis Robot, Exact Radial Bases Neural Network (RBE-NN), General Regression Neural Networks (GRNN), FPGA and VHDL.*

تصميم مسيطر ذراع الي من محورين باستخدام الشبكات العصبية ذات الدالة النصف قطرية والارتداد العام مطبقة على مصفوفة البوابات المنطقية المبرمجة

م. مهند عبد شهاب م. عماد احمد حسين م. هيثم عبد الرحيم طه
الجامعة المستنصرية، كلية الهندسة، قسم الهندسة الكهربائية

الخلاصة :

بالرغم من أن الشبكات العصبية قابلة للتطبيق لحلمشاكلِ السيطرة على الإنسان الآلي، في الحقيقة ، "المسيطرات العصبية"، وظيفتهم متخصصة بشكل رئيسي لتزويد الحلول إلى مشاكل حركة الذراع الآلية .في بحثنا هذا، نوعان من أنواع الشبكات العصبية ستطرح لحل عدد من المشاكل الحركية المرافقة لعمل الإنسان الآلي هذه الأنواع هي: الشبكة

العصبية ذات الدالة النصف قطرية الملابوطة (RBENN) والشبكات العصبية ذات الارتداد العام (GRNN). تتضمن حركة الإنسان الآلي دراسة هندسة معالج حركات الذراع بتعلّم حركات الذراع الآلية لإعطاء الموقع النهائي المطلوب هي من المهام الحركية الرئيسية التي تمت تغطيتها هنا. الإدراك المادي للشبكة العصبية، لدرجة كبيرة يعتمد على التطبيق الكفوءِ لخلية عصبية مفردة. تقنية إل (FPGA) المعتمدة على إعادة تشكيل الحسابات البنوية المناسبة لبناء وتصميم الشبكة العصبية. إن الإدراك المادي لتقنية إل(FPGA) لتكوين عدد كبير من الخلايا العصبيةِ ما زال تحديا مهما. يناقش هذا البحث أيضا القضايا المعقدة التي اشتركت في تطبيق خلية عصبية متعددة المداخل ذات دوال تغذية خطية / لا خطية باستعمال لغة برمجة (VHDL) لبرمجة إل (FPGA).

1. Introduction

For a manipulator with n degree of freedom, at any instant of time joint variables is denoted by θi = θ (t), i =1, 2, 3,..., n and position variables x_j = x(t), j = 1, 2, 3,...,m. The relations between the end-effector position x(t) and joint angle θ(t) can be represented by the following equation [1][2] :

$$x(t) = f(\theta(t)) \qquad .....(1)$$

where f is a nonlinear, continuous and differentiable function. In a two-axis robotic arm, given the angles of the joints, so the two forward kinematics equations (Eq.(2) and Eq. (3)) give the location of the tip of the arm.

$$x = l_1 \cos(\theta_1) + l_2 \cos(\theta 1 + \theta 2) \qquad .....(2)$$

$$y = l_1 \sin(\theta_1) + l_2 \sin(\theta 1 + \theta 2) \qquad .....(3)$$

Given a desired location for the tip of the robotic arm, what should the angles of the joints be so as to locate the tip of the arm at the desired location (as shown in **Figure. (1)**.

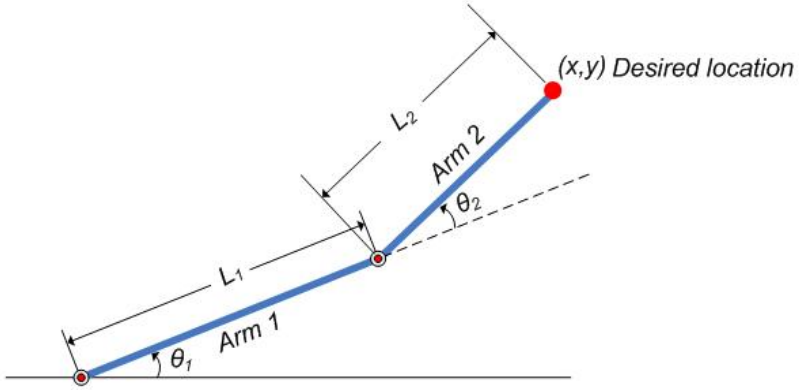

Fig .(1): The two-axis robotic arm with the two angles, θ_1 and θ_2

On the other hand, with the given desired end effector position, the problem of finding the values of the joint variables is inverse equation(4), which can be solved by:

$$\theta(t) = f'(x(t)) \qquad .....(4)$$

There is usually more than one solution and can at times be a difficult problem to solve. This is a typical problem in robotics that needs to be solved to control a robotic arm to perform tasks it is designated to do. In a 2-dimensional input space, with a two-axis robotic arm and given the desired co-ordinate, the problem reduces to finding the two angles involved. The first angle is between the first arm and the ground (or whatever it is attached to). The second angle is between the first arm and the second arm. So in our case the inverse kinematics equation are :

$$\theta_1 = tan^{-1}(y/x) - tan^{-1}(k_2/k_1) \qquad .....(5)$$
$$\theta_2 = tan^{-1}(sin(\theta_2) / cos(\theta_2)) \qquad .....(6)$$

Where:

$$k_1 = l_1 + l_2 cos(\theta_2) \qquad .....(7)$$
$$k_2 = l_2 \, sin(\theta_2) \, cos(\theta_2) \qquad .....(8)$$

Since the two formulas for the two-axis robotic arm are known, x and y co-ordinates of the tip of the arm are deduced for the entire range of angles of rotation of the two joints. The co-ordinates and the angles are saved to be used as training data to train the two Neural Networks (RBENN and GRNN)for the two angles ($\theta_1$and θ_2) respectively.

During training, the two NNs learns to map the co-ordinates (x,y) to the angles (θ_1, θ_2). The trained NNs are then used as a part of a larger control system to control the robotic arm. Knowing the desired location of the robotic arm, the control system uses the trained NNs to deduce the angular positions of the joints and applies force to the joints of the robotic arm accordingly to move it to the desired location.

After training these NNs we will implement this network by using new hardware implementation methods. Hardware realization of a Neural Network (NN), to a large extent depends on the efficient implementation of a single neuron. Field Programmable Gate Array FPGA-based reconfigurable computing architectures are suitable for hardware implementation of neural networks. FPGA realization of NNs with a large number of neurons is still a challenging task[3][4][5].

The new proposed method of implementation is the FPGA, which is suitable for fast implementation and quick hardware verification. FPGA based systems are flexible and can be reprogrammed unlimited number of times.

2. Exact Radial Bases (ERB) and General Regression Neural Networks (GRNN)

Exact Radial Bases (ERB) neural network is a two-layer network as shown as a blocks in **Figure(2).** The first layer has neurons with radial bases transfer function, and calculates its weighted inputs with Euclidean distance weight function and uses the network production for the input function. The second layer has linear transfer function neurons, and calculates its weighted input with dot product weight function and its inputs with network sum input function. Both layers have biases see **Figure.(3).**

RBENN sets the first-layer weights to transpose of the input vector (P'), and the first-layer biases are all set to 0.8326/S, resulting in radial basis functions that cross 0.5 at weighted inputs of +/- S. Where S is the spread of radial basis functions [1][6][7].

The second-layer weights LW{2,1} and biases b{2} are found by simulating the first-layer outputs a{1} and then solving the following linear expression:

$$[LW\{2,1\}\ b\{2\}] * [a\{1\}; ones] = T \qquad .....(9)$$

Where T is the target vector and (ones) is a square matrix of ones elements. We know the inputs to the second layer a{1} and the target (T), and the layer is linear. We can use the following expression to calculate the weights and biases of the second layer to minimize the sum-squared error.

$$Wb = T/[P; ones(1,Q)] \qquad .....(10)$$

Here Wb contains both weights and biases, with the biases in the last column.

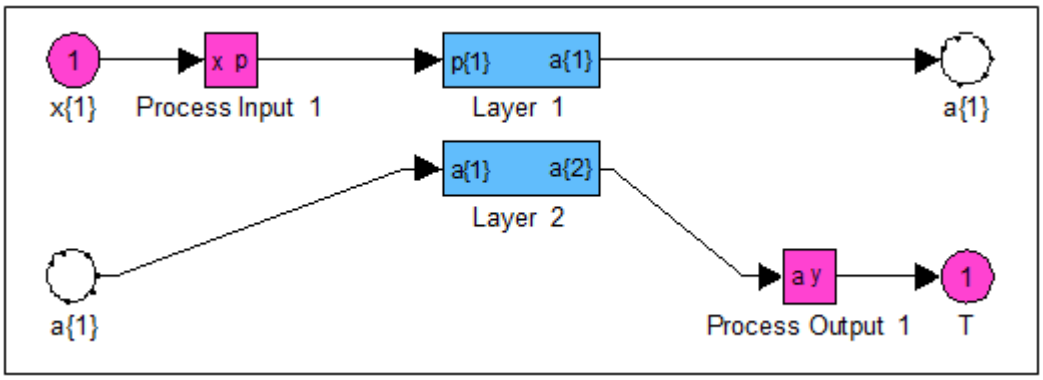

Fig .(2): The two layers of ERB Neural Network

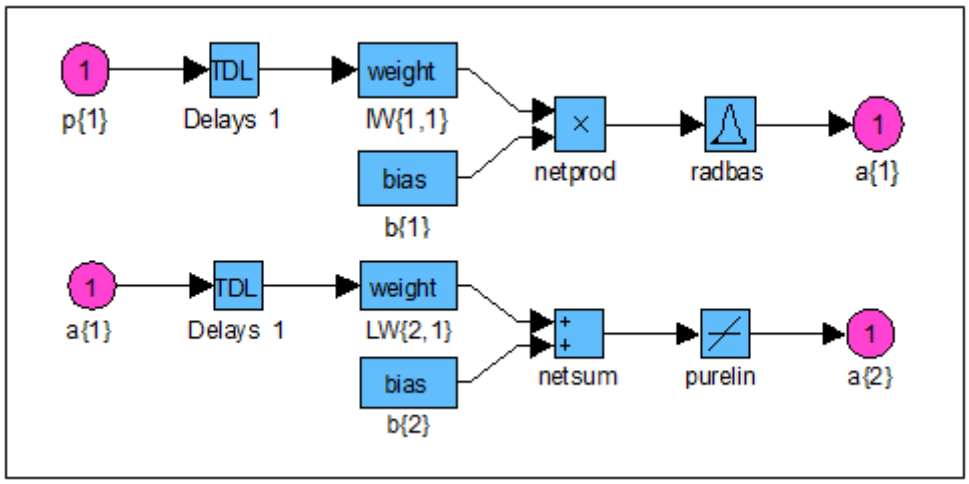

Fig .(3): Inside the two layers for the ERB Neural Network

General Regression neural networks (GRNN) are widely used for solving prediction problems. GRNN is one of the type neural network that can be used for prediction. This type of neural networks a kind of radial basis network that is often used for function approximation.

As in RBENN, GRNN contain two-layers the first layer has radial bases neurons, and calculates weighted inputs with Euclidean distance function and network input with production function. The second layer has linear transfer function neurons but calculates weighted input with normalized dot product weight function, and net inputs with network sum. Only the first layer has biases.

GRNN sets the first layer weights to P', and the first layer biases are all set to 0.8326/S, resulting in radial basis functions that cross 0.5 at weighted inputs of +/- S. The second layer weights LW{2,1} are set to T as shown in **Figure(4).**

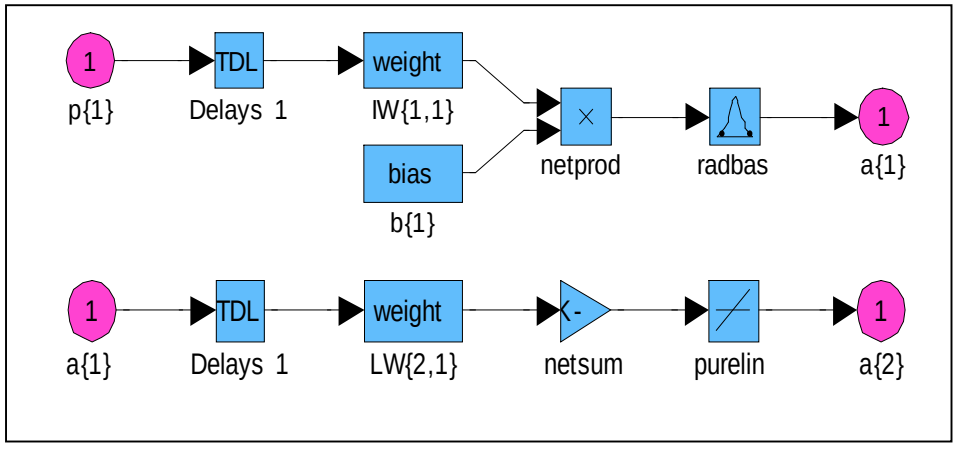

Fig .(4): The two layers for the GRNN

3. Neural Network implementation using FPGA technique

During the 1980s and early 1990s there was significant work in the design and implementation of hardware neurocomputers. Nevertheless, most of these efforts may be judged to have been unsuccessful: at no time have hardware neurocomputers been in wide use. This lack of success may be largely attributed to the fact that earlier work was almost entirely aimed at developing custom neurocomputers, based on ASIC technology, but for such niche areas this technology was never sufficiently developed or competitive enough to justify large-scale adoption. On the other hand, gate-arrays of the period mentioned were never large enough nor fast enough for serious artificial-neural network (ANN) applications. But technology has now improved: the capacity and performance of current FPGAs are such that they present a much more realistic alternative. Consequently neurocomputers based on FPGAs are now a much more practical proposition than they have been in the past [4].

So, neural networks can be implemented using analog or digital systems. The digital implementation is more popular as it has the advantage of higher accuracy, better repeatability, lower noise sensitivity, better testability, higher flexibility, and compatibility with other types of preprocessors. The digital NN hardware implementations are further classified as [8][9]:

i. FPGA-based implementations
ii. DSP-based implementations
iii. ASIC-based implementations.

DSP based implementation is sequential and hence does not preserve the parallel architecture of the neurons in a layer. ASIC implementations do not offer re-configurability by the user. FPGA is a suitable hardware for neural network implementation as it preserves the parallel architecture of the neurons in a layer and offers flexibility in reconfiguration.[3][5][10]

4. Solution of the Forward and Inverse Kinematics Problem Using Our method

In order for the two networks to be able to predict the angles (θ_1 and θ_2) they have to be trained with sample input-output data. The first RBE network will be trained with X and Y coordinates as input and corresponding θ_1 values as output (see Figure.5(a)). Similarly, the second network (GRNN) will be trained with X and Y coordinates as input and corresponding θ_2 values as output (Figure.5(b)). Each of the neural networks involved performs static mapping of two two-variable functions as discussed earlier in Eq.(2) and Eq.(3)

Once the training is complete, the two networks would have learned to approximate the angles (θ_1 and θ_2) as a function of the coordinates (X, Y). One advantage of using the two approaches is that the RBE network would now approximate the angles for coordinates that are similar but not exactly the same as it was trained with GRNN. For example, the trained

networks are now capable of approximating the angles for coordinates that lie between two points that were included in the training dataset. This will allow the final controller to move the arm smoothly in the input space.

We now have two trained networks which are ready to be deployed into the larger system that will utilize these networks to control the robotic arms.

Having trained the networks, an important follow up step is to validate the networks to determine how well the networks would perform inside the larger control system. Since this paper deals with a two-joint robotic arm whose inverse kinematics formulae derived in Eq.(5) and Eq.(6), it is possible to test the answers that the networks produce with the answers from the derived formulae.

Now given a specific task, such as robots picking up an object in an assembly line, the larger control system will use the trained networks as a reference, much like a lookup table, to determine what the angles of the arms must be, given a desired location for the tip of the arm. Knowing the desired angles and the current angles of the joints, the system will apply force appropriately on the joints of the arms to move them towards the desired location.

After all this completed the program will be translated to a Simulink diagram using MATLAB. With the use of Simulink® HDL Coder™ software [6] that lets us generate hardware description language (VHDL) code based on Simulink® models the coder brings the Model-Based Design approach into the domain of FPGA development. Using the coder, we can spend more time on fine-tuning algorithms and models through rapid prototyping and experimentation and less time on VHDL coding that Xilinx's core generator can read. All parts in the two neural networks will be compiled to VHDL program then all parts will be collected to represent the whole network structure that we use it in our work. The flowchart of our procedure is shown in **Figure.(6).**

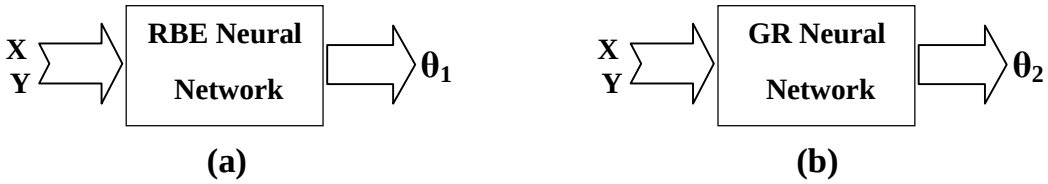

Fig .(5): RBE and GR Neural Network for robot kinematics transformation: (a) forward kinematics problem and (b) inverse kinematics problem.

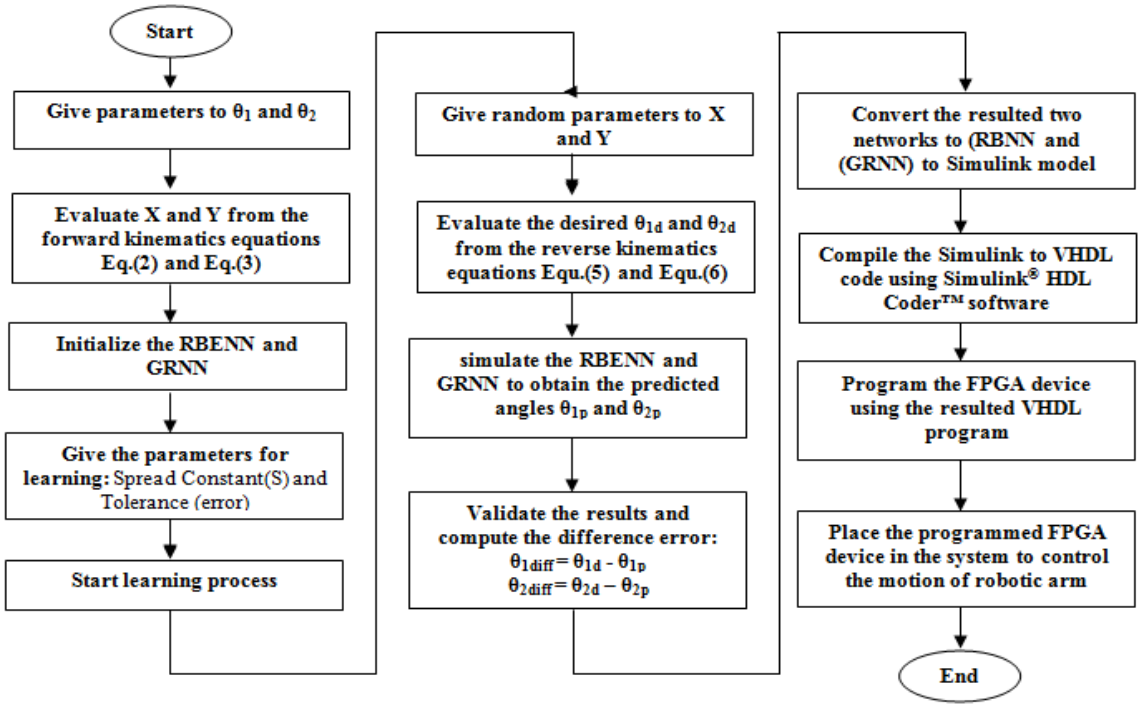

Fig .(6): A flow chart of the whole process for the proposal

5. Simulated Example

Let us assume that length of first arm (l_1) = 10 and length of second arm (l_2) = 7. The first joint has limited freedom to rotate and it can rotate between 0 and 90 degrees. Similarly, assume that the second joint has limited freedom to rotate and can rotate between 0 and 180 degrees. Hence, $0 <= \theta_1 <= \pi/2$ and $0 <= \theta_2 <= \pi$ (see **Figure .7**).

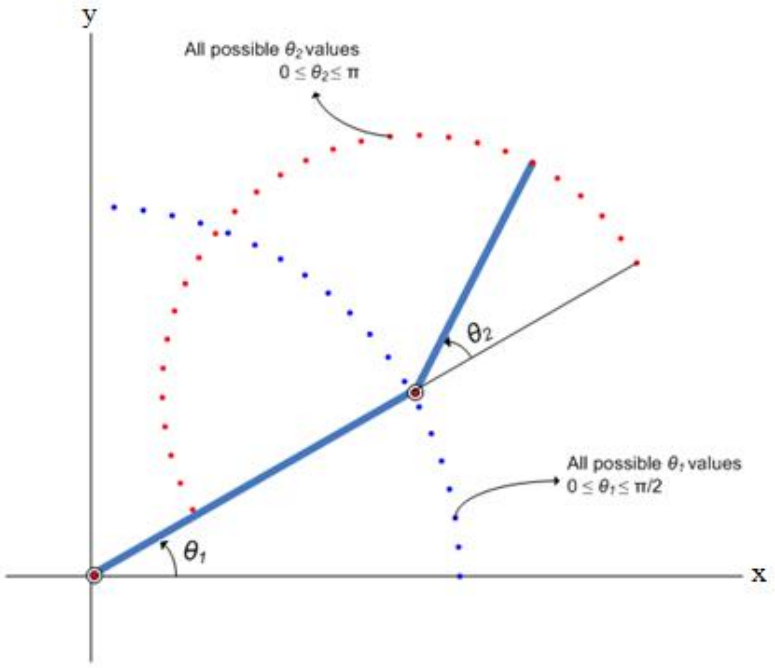

Fig .(7): Illustration of all possible θ1 and θ2 values.

Now, for every combination of θ_1 and θ_2 values the X and Y coordinates are deduced using forward kinematics formulae (Eq.2 and Eq.3).The data is generated for all combination of θ_1 and θ_2 values and saved into a matrix to be used as training data for RBENN and RGNN. Fig.8 shows all the X-Y data points generated by cycling through different combinations of θ_1 and θ_2 and deducing X and Y co-ordinates for each.

RBENN and GRNN represent the two trained neural networks that will be deployed in the larger control system. With Spread constant (S) = 1 and 5 for RBENN and GRNN respectively we can train the two networks.

Let's assume that it is important for the networks to have low errors within the operating range $0<x<2$ and $8<y<10$. The θ_1 and θ_2 values are deduced mathematically from the x and y coordinates using inverse kinematics formulae (Eq.5 and Eq.6) to get θ_{1d} and θ_{2d}.

theta1 and theta2 values predicted by the trained RBENN and GRNN networks respectively to obtain the predicted angles θ_{1P} and θ_{2P}.

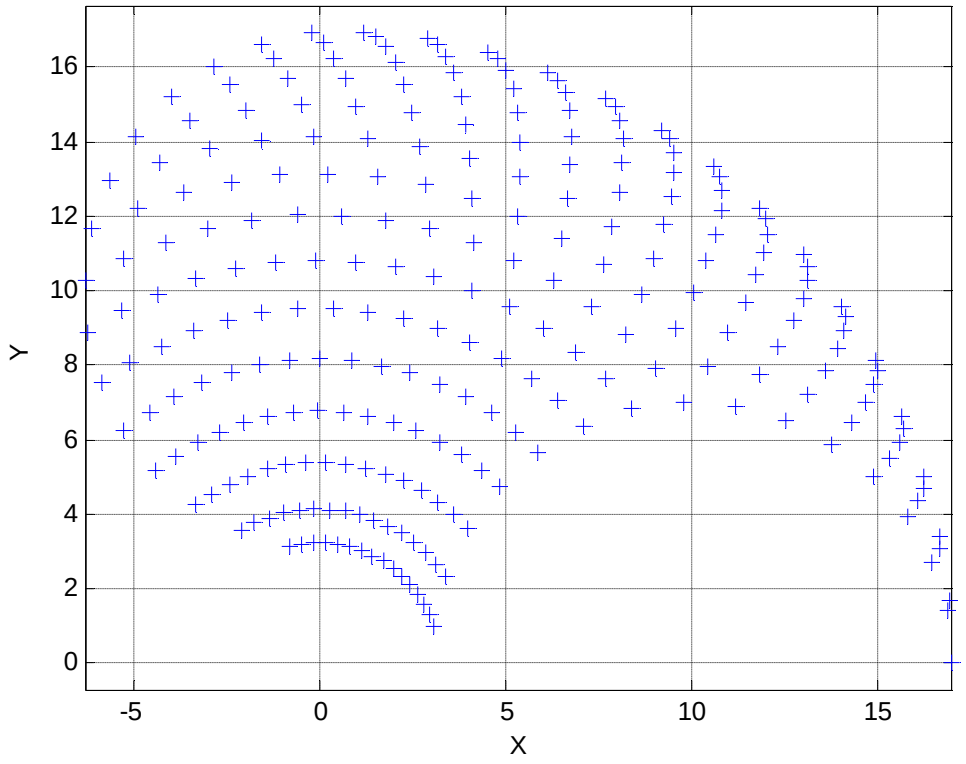

Fig .(8): X-Y coordinates generated for all θ1 and θ2 combinations

Now, we can see how close the networks outputs are with respect to the deduced values by using the following formulas (see **Figure.9** and **Figure.10**):

$\theta_1 diff = \theta_{1d} - \theta_{1P}$

$\theta_2 diff = \theta_{2d} - \theta_{2P}$

The resulted Mean square error (MSE) for the two networks is: 0.0085 for RBENN and 0.0037 for GRNN which is a fairly good number for the application it is being used in. However this may not be acceptable for another application, in which case the parameters to

the networks may be tweaked until an acceptable solution is arrived at. Also, other techniques like input selection and alternate ways to model the problem may be explored.

With the use of MATLAB Ver.7.6.0 (R2008a) we can convert our program to Simulink model as shown previously in **Figure.2, Figure.3 and Figure.4.**

The final step is to program the Xilinx FPGA instrument depending on Simulink® HDL Coder™ software that comes with the MATLAB package. The HDL coder can compile the resulted Simulink to VHDL program. Now the FPGA device was ready to program using the resulted VHDL program with the use of the board (Spartan-3 Startup Kit Demo Board) interfaced with Pentium-4 PC of speed 3.02 GHz and RAM 2.0 GB.

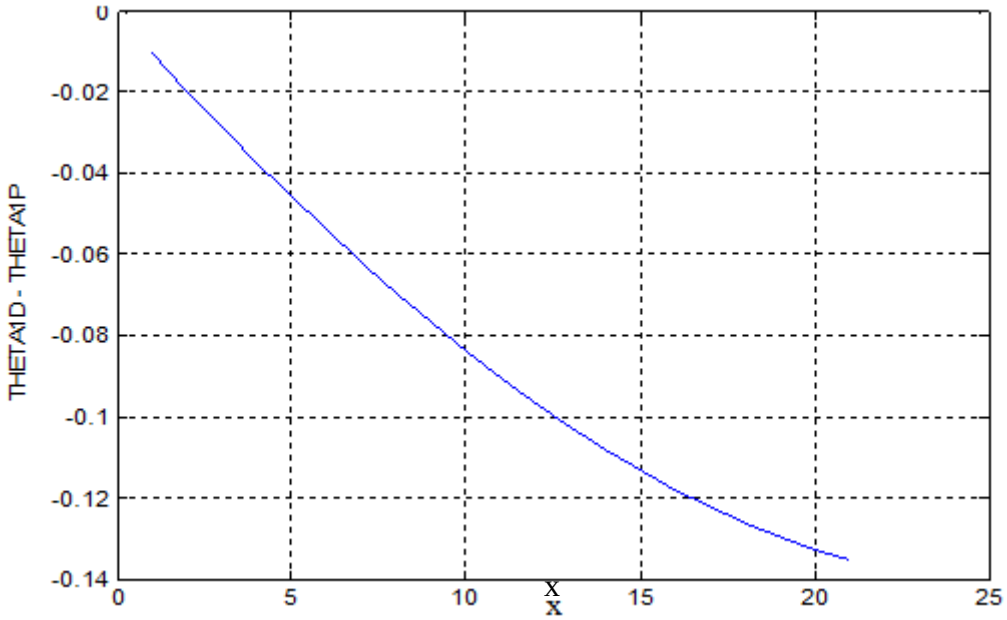

Fig .(9): Deduced θ_1-Predected θ_1

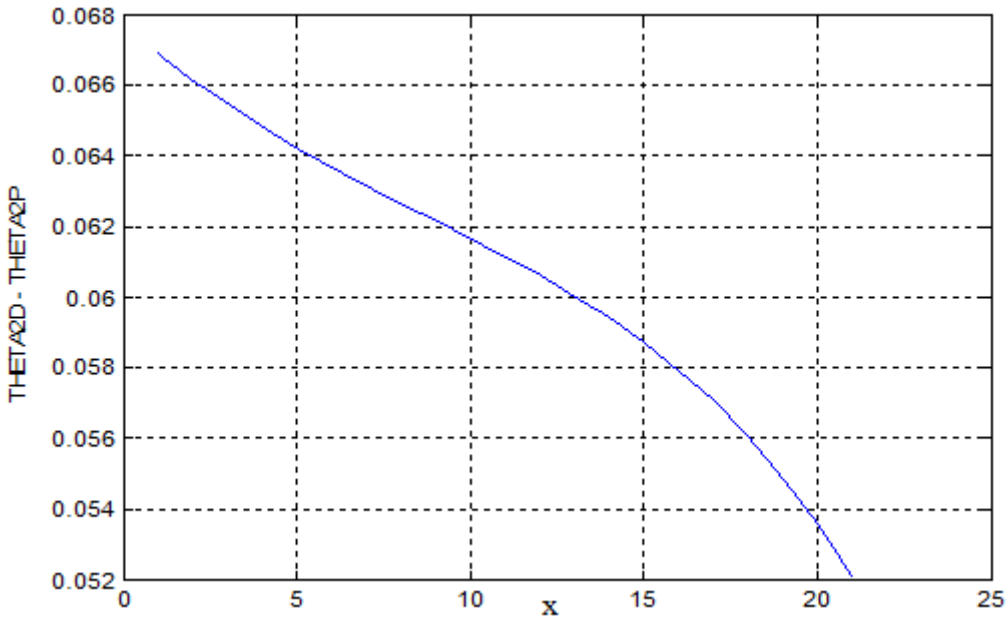

Fig .(10): Deduced θ_2-Predected θ_2

Here is a samples for some VHDL code in some parts in our network. **Figure.11** shows the VHDL code for the first layer IW{1,1} and **Figure.12** for the second layer LW{2,1}.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.nono1_pkg.ALL;

ENTITY IW_1_1 IS
  PORT( pd_1_1           :  IN    vector_of_real(0 TO 1);  -- double [2]
        iz_1_1           :  OUT   vector_of_real(0 TO 15)  -- double [16]
        );
-- Component Configuration Statements
  FOR ALL : dist1
    USE ENTITY work.dist1(rtl);

  FOR ALL : dist2
    USE ENTITY work.dist2(rtl);
-- Signals
  SIGNAL IW_1_1_1_out1    : vector_of_real(0 TO 1) := (OTHERS => 0.0);  -
- double [2]
  SIGNAL dist1_out1       : real := 0.0;  -- double
  SIGNAL IW_1_1_2_out1    : vector_of_real(0 TO 1) := (OTHERS => 0.0);  -
- double [2]
  SIGNAL dist2 out1
```

Fig .(11): A sample of VHDL code for the input layer IW{1,1}

```
-- Component Configuration Statements
  FOR ALL : dotprod1
    USE ENTITY work.dotprod1(rtl);

  -- Signals
  SIGNAL IW_2_1_1_out1         : vector_of_real(0 TO 15) := (OTHERS =>
0.0);  -- double [16]
  SIGNAL dotprod1_out1         : real := 0.0;  -- double
  SIGNAL Mux_out1              : real := 0.0;  -- double
IW_2_1_1_out1(0) <= -7.9778031686867990E-001;
  IW_2_1_1_out1(1) <= -6.9674375155472545E-001;
  IW_2_1_1_out1(2) <= -5.9672714420956308E-001;
  IW_2_1_1_out1(3) <= -4.9655040051601318E-001;
IW_2_1_1_out1(4) <= -3.9624066449480905E-001;
  IW_2_1_1_out1(5) <= -2.9587876808554481E-001;
  IW_2_1_1_out1(6) <= -1.9573471749315374E-001;
  IW_2_1_1_out1(7) <= -9.6547649553977863E-002;
  IW_2_1_1_out1(8) <= 0.0000000000000000E+000;
  IW_2_1_1_out1(9) <= 9.0761446381729324E-002;
  IW_2_1_1_out1(10) <= 1.7096104800836659E-001;
  IW_2_1_1_out1(11) <= 2.3679591998538135E-001;
```

Fig .(12): A sample of VHDL code for the input layer LW{2,1}

6. Conclusions

We can conclude the following points from our work:

- This paper takes two types of neural networks to control a two axes robotic arm and to solve the inverse kinematics problem.
- The use of different types of neural network depending on the preferred of some networks in a specific application on other networks. This depends on our sense and the type of the used application also depends on trial and error.
- As we can see from the previous figures (Fig.(9) and Fig.(10)) that the difference between the deduced and the predicted angles can decrease as the iterations (X,Y samples) increased.
- The difference in theta deduced and the data predicted clearly depicts that the proposed method results in an acceptable mean square error.
- Our trained networks can be utilized to provide fast and acceptable solutions of the inverse kinematics problem.
- This paper also presents design solution to eliminate the FPGA design for a neural network. The motivation for this study stems form the fact that an FPGA coprocessor with limited logic density and capabilities can used in building many complex, flexible, and faster design systems.

References

1. **Jacek M. Zurada, "Introduction to Artificial Neural Systems", WEST PUBLISHING COMPANY, COPYRIGHT © 1992.**
2. **Srinivasan Alavandar, M.J. Nigam, "Inverse Kinematics Solution of 3DOF Planar Robot using ANFIS", Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844 Vol. III (2008), Suppl. issue: Proceedings of ICCCC 2008, pp. 150-155**
3. **AMOS R. OMONDI and JAGATH C. RAJAPAKSE, "FPGA Implementations of Neural Networks", All Rights Reserved © 2006 Springer, Printed in the Netherlands.**
4. **Haitham Kareem Ali and Esraa Zeki Mohammed, "Design Artificial Neural Network Using FPGA", IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.8, August 2010.**
5. **A. Muthuramalingam, S. Himavathi, E. Srinivasan, "Neural Network Implementation Using FPGA: Issues and Application", International Journal of Information and Communication Engineering 4:6 2008.**
6. **Howard Demuth and Mark Beale, "Neural Network Toolbox For Use with MATLAB®", copyrighted© 2002 by The MathWorks, Inc.**

7. **Martin T., Howard B, and Mark B", "Neural Network Design, Original copyrighted© 1996 by PWS publishing Company. All right reserved.**
8. **Y.J.Chen, Du Plessis, "Neural Network Implementation on a FPGA", Proceedings of IEEE Africon, vol.1, pp. 337-342, 2002.**
9. **Sund Su Kim, Seul Jung, "Hardware Implementation of Real Time Neural Network Controller with a DSP and an FPGA", IEEE International Conference on Robotics and Automation, vol. 5, pp. 3161-3165, April 2004.**
10. **Aydoğan Savran, Serkan Ünsal, "Hardware Implementation of a Feed forward Neural Network Using FPGAs", Ege University, Department of Electrical and Electronics Engineering, 2003.**