Neural Network Based Inverse Kinematic Solution of a 5 DOF Manipulator for Industrial Application

S. Aravinddhakshan, Sudhanv Apte, and Akash S M
SRM Institute of Science and Technology, Kattankulattur, Tamil Nadu, 603203, India
aravinddhakshan@gmail.com

Abstract: This paper showcases the application of Neural Networks for manipulation. Algorithm based approaches work well although are limited in their ability to find solutions sometimes. The tedious task of programming a manipulator can be replaced with Neural Networks which can learn how to solve Inverse Kinematics. We present a pick and place scenario using Neural Network to solve Inverse Kinematics. Their ability to learn from examples make them a good candidate to solve the inverse kinematics problem. For this purpose, a unique configuration of 5 DOF arm is designed to suit industrial needs. To train the network, a dataset of random joint positions is created and forward kinematics is derived for the corresponding joint angles. The joint variables are then fed to a path planner in Simulink and then the process is simulated.

Keywords: Artificial Neural Networks; Regression; Mean Square Error; Forward kinematics; Inverse kinematics; Path Planner; DOF; Manipulator; Robotic arm.

1. Introduction
As the automation in industry increases, it is required to transform the machines to learn and function with minimum effort. This has been the focus of robotics and automation engineers for several years. Manipulators have found their way into industrial applications. Manipulators can efficiently carry out repetitive tasks with great accuracy and speed. Pick and place operation is one of the most common applications in many manufacturing systems. The manipulator is designed to pick an object from a part delivery mechanism and place it onto the other so as to reduce the manpower and improve the production rate of an industry to the maximum. It is imperative for the arm to get an accurate inverse kinematic solution as quickly as possible so that the overall operation of planning and execution is complete within a short period of time. However, inverse kinematics is a challenging enterprise. Various ways such as geometrical, numerical and iterative methods are used, but solving inverse kinematics becomes computationally complex for higher DOF robotic arm and sometimes we don’t even get a solution. Closed form solutions are also equally difficult to work with. This is where Neural Networks come into picture. Many studies were done on the implementation of ANN on a robotic arm to overcome the singular configuration problem of robotic arm. A network inversion method was proposed for solving inverse kinematic problem of a 3 DOF manipulator using multilayer neural network where the NN was trained to predict joint angles giving the end-effector pose as input [1]. An ANN based solution was introduced for singular configuration of 6 DOF robotic arm to estimate the angular position and velocities of each joint by giving end-effector pose and linear velocity as input to the network [2]. A hybrid approach was proposed to minimize end-effector error and improve precision of a nonsurgical robotic arm by combining characteristics of neural networks and genetic algorithms. [3].
NN was used to obtain inverse kinematics solution in generating trajectories for a 3DOF planar manipulator [4]. A non-conventional NN solution was introduced for a PUMA 560 robot by using robot’s end-effector cartesian coordinates as input to estimate its corresponding joint parameters. The Training results showed a regression value of 0.87527 indicating 87.5% fitness [5]. An analytical inverse kinematic solution was presented for 5DOF spatial micromanipulator in analyzing the effective use of micro-mechanism workspace [6]. We now propose a neural network-based solution for pick and place operation of a 5 DOF manipulator having RRRPR configuration.

2. Manipulator Design
The 3D design is carried out in Solidworks, carefully considering the work envelope of all joints, links and gripper. RRRPR configuration is chosen because it gains large work volume and provides high mobility, giving substantial rigidity for the robot in the vertical direction and flexibility in horizontal plane making it ideal for applications such as material handling, spray-painting and assembly tasks in automotive and electronics industries.

Figure 1: Robotic arm model

![Robotic arm model](image-url)
Table 1: Part specifications

| Specification       | Value                                      |
|---------------------|--------------------------------------------|
| Number of axes      | 6 (Position & Orientation along x,y,z axis) |
| Number of DOF       | 5                                          |
| Horizontal reach    | 1306 mm                                    |
| Vertical reach      | 1130 mm                                    |
| Configuration       | RRRPR (4 rotary, 1 prismatic joint)        |
| Waist/Jo1nt rotation | -0 - 180°                                  |
| Shoulder/Jo1nt2 rotation | -0 - 90°                                  |
| Elbow/Jo1nt3 rotation | -90 - 90°                                  |
| Forearm/Link4 actuation | 0 - 260 mm                                 |
| Wrist/Jo1nt5        | -90 - 90°                                  |
| Gripper actuation   | 0 - 77mm                                    |

3. **Forward Kinematic modeling**

The kinematics of a manipulator describes the relationship between the joint space and the cartesian space. Computing the kinematics is necessary for motion control of the robot and its trajectory generation. Therefore, transformation matrices have been used for control.

Forward Kinematics calculates the position and orientation of end-effector while the inverse Kinematics calculate the joint variables based on the known position and orientation of the end-effector.

Our RRRPR manipulator has 4 revolute joints and 1 prismatic joint. System coordinate frames have been assigned before proceeding with kinematic analysis (Figure 2). The coordinates O0, X0, Y0, Z0 are fixed to the base frame while the other coordinate frames are attached to the corresponding links.

Here, the homogeneous transformation matrix is used for calculating the position and orientation of end effector with respect to base coordinate frame. The transformation matrix of the five jointed robot manipulator is given by:

\[
T_{\text{home \ end-effector}} = \begin{bmatrix}
R_{1x} & R_{2x} & R_{3x} & P_x \\
R_{1y} & R_{2y} & R_{3y} & P_y \\
R_{1z} & R_{2z} & R_{3z} & P_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]  

(1)

where \([R]\) is the 3 × 3 rotation matrix and \([P]\) is the position vector of end-effector with respect to home position. The Denavit - Hartenberg (D-H) parameters are derived to understand the forward kinematics of the manipulator designed. The Table 2 shows the D-H parameters derived for the RRRPR manipulator:
Table 2: DH parameters of RRRPR

| Links (i) | Joint angle (θ) (deg) | Link offset (d) (mm) | Twist angle (α) (deg) | Link Length (a) (mm) |
|----------|-----------------------|--------------------|---------------------|---------------------|
| 1        | Q1                    | D1 = 225           | 90                  | 0                   |
| 2        | Q2                    | 0                  | 0                   | L2 = 470            |
| 3        | Q3                    | 0                  | 90                  | 0                   |
| 4        | 0                     | 0                  | -90                 | D4 = 400            |
| 5        | Q5                    | 0                  | 0                   | L5 = 176            |

The above parameters are used in defining the individual transformation matrices \(^{i-1}T_i\) that helps in formulation of forward kinematics of manipulator:

\[
^{i-1}T_i = \begin{bmatrix}
\cos\theta & -\sin\theta\cos\alpha & \sin\theta\sin\alpha & a\cos\theta \\
\sin\theta & \cos\theta\cos\alpha & -\cos\theta\sin\alpha & a\sin\theta \\
0 & \sin\alpha & \cos\alpha & d \\
0 & 0 & 0 & 1
\end{bmatrix}
\quad (2)
\]

\[
^0T_5 = ^0T_1 \times ^1T_2 \times ^2T_3 \times ^3T_4 \times ^4T_5
\quad (3)
\]

Given the joint angles and gripper transformation matrix, the transformation matrix of the gripper (\(T_{end-effector}\)) with respect to the home position (\(T_{home}\)) is calculated:

\[
T_{end-effector}^{home} = \begin{bmatrix}
1 & 0 & 0 & 1046 \\
0 & 0 & -1 & 0 \\
0 & -1 & 0 & 225 \\
0 & 0 & 0 & 1
\end{bmatrix}
\quad (4)
\]

A graphical representation of home configuration is shown below in Figure 3.

![Figure 3: Robot’s home position](image-url)
4. Artificial Neural Network

4.1 Background

Neural Networks are learning systems which employ neurons to learn and predict patterns. They are sets of interconnected neurons and consist of a number of layers which can be varied depending on the complexity of data. The first layer is called the Input layer. We feed in our input vector to the network, the output layer is the number of predictions we want to make. In between we have the hidden layer where the network adjusts the neurons to predict data. In order to facilitate learning we have back propagation algorithm which is used to find the difference between predicted value and target value, making it a closed loop system. The back-propagation algorithm changes the neuron’s weights using MSE value (mean squared error):

\[ \text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (Y_i - Y_i')^2 \] (5)

Where, (n) being total number of samples, (\(Y_i\)) the target value and (\(Y_i'\)) the predicted value to \(i\)th input data.

Here, supervised learning technique is carried where the target values are given as input to the neural network and predictions or outputs are then generated in accordance with the regression value. The weights (w) and bias (b) values for the network are updated as follows:

\[ S = \sum_{i=1}^{n} (w_i \times x_i) + b \] (6)

where, (\(x_i\)) is input, (\(w_i\)) is the weight of each input value, (b) is bias, and (n) is total number of variables.

During the training of network, weights and the bias values are updated with each epoch until a lowest MSE value is reached. After this, the training stage of the neural network is terminated.

4.2 NN Design & Setup

To train the NN, we created train and test data sets using Forward Kinematics (FK). We computed the FK using random joint angles to get the transformation matrix. The values are then used to train the NN, taking the manipulator orientation and end-effector position as input data and the random joint angles as the target. Since the pick and place operation is highly dependent on the designed structure of the robotic arm, two ANNs are designed for this study. For the first NN we have considered only the end effector position as input, and for the second NN have considered both end-effector position and orientation.

\[
\begin{bmatrix}
Q_1 \\
Q_2 \\
Q_3 \\
Q_4 \\
Q_5 \\
\end{bmatrix} = \text{NN1} \begin{bmatrix}
P_x \\
P_y \\
\end{bmatrix} \] (7)

\[
\begin{bmatrix}
Q_1 \\
Q_2 \\
Q_3 \\
Q_4 \\
Q_5 \\
\end{bmatrix} = \text{NN2} \begin{bmatrix}
P_x \\
P_y \\
R_x \\
R_y \\
R_z \\
\end{bmatrix} \] (8)

The neural network is initially tested with Levenberg-Marquardt, Bayesian-Regularization and Scaled Conjugate Gradient (SCG) algorithms for deciding the best suitable/most compatible one.

After evaluating all the algorithms, it is found that Levenberg-Marquardt generalizes quickly, meaning training has stopped due to it being stuck at local minima. Bayesian-Regularization has huge MSE values for train and test data leading to a poor performance, whereas the Scaled Conjugate Gradient algorithm has given the satisfactory results. The MSE difference between test and train data was good enough. Therefore, the entire Neural network analysis for both approaches is done with SCG. The network is designed with following criteria

- Input: [End-effector pose]
- Output: [Joint variables]
- Hidden layer: Tan-Sigmoid activation function
Output layer: Linear activation function

For our training, 70% of the samples are chosen for actual training of the network, 20% for validation, and the rest 10% for testing. The training of the artificial neural network commences once the network architecture is well established and the raw data sets are divided into the three following categories:

- Training Set: The main data set utilized to train the network.
- Validation Set: This data set is used to provide unbiased evaluation of the training model.
- Testing Set: This data set is used to evaluate the network fitness.

4.3 Training the Network

The experiment is divided into two parts. Neural Network training and inference. To train the NN the inputs are first selected from two different input data sets, which have 3 and 12 input parameters respectively. The data set is initially normalized by standardizing the inputs for faster training of network and reduce the chances of getting stuck in local optima. Normalizing the inputs enables our network to effectively learn the parameters in the first layer. As the second layer of network accepts activations as input from previous layer, normalizing these values ensures that our network can learn the second layer parameters more effectively and the overall loss function is reduced.

For the first approach, the ‘input’ is a 1001x3 matrix consisting of end-effector positions (Px, Py, Pz) and ‘target’ is a 1001x5 matrix consisting of joint variables (Q1, Q2, Q3, D4, Q5). The networks consist of one hidden layer and 10 neurons, 700 samples used for training, 100 samples for testing and 200 samples for validation. The selection of number of neurons is done carefully and is varied from 10 to 32.

![NN1 Architecture](image)

Figure 4: NN1 architecture

The performance of a NN is determined by the mean square error (MSE) between the NN’s target output and the actual output. Here, the training terminates when the validation performance is stopped at epoch 23 with MSE value of 1547.55 showing good performance (Figure 5). The number of neurons is then increased significantly until a better performance level is achieved. It is observed that for 27 hidden neurons, the MSE drops rapidly through the learning process; and is decreased until it is stopped at epoch 112 with MSE of 977.72 (Figure.6).
Proceeding with the second approach, where the ‘input’ is 1001x12 matrix consisting both position and orientation of the end effector (Px, Py, Pz, Rx, Ry, Rz) and ‘target’ is a 1001x5 matrix consisting of joint variables (Q1, Q2, Q3, D4, Q5).

The training terminates when the validation performance is stopped at epoch 5 with MSE of 3916.58, showing a very poor performance (Figure 10). The number of neurons is further varied till 32 neurons to see any drop in MSE, but no reduction in MSE is seen and the training performance deteriorates drastically giving a rise of MSE of 7781.5508 (Figure 11).
The regression plot of NN1 (Figure 7) shows close relationship between targets and outputs for training, testing and validation data when compared to NN2 (Figure 12). From Table 3, it is observed that for NN1 the test score (MSE) is close to the training score (MSE) showing that the model has generalized. Whereas, for NN2 the MSE Value is extremely high than train score showing overfitting. Generalization in an NN is necessary because it shows how good our model learns from the given data and how well it applies the learnt information elsewhere. If the NN performs well on the untrained data, we can say that our network has generalized well for the given data.

![Figure 10: Training with 10 neurons (NN2)](image1)

![Figure 11: Training with 32 neurons (NN2)](image2)

![Figure 12: Regression plot (NN2)](image3)

![Figure 13: Error histogram (NN2)](image4)

**Table 3: Training results**

| Performance               | NN 1         | NN 2         |
|---------------------------|--------------|--------------|
| Training MSE              | 954.2141     | 447.9859     |
| Validation MSE            | 977.7223     | 7781.5508    |
| Testing MSE               | 940.1255     | 7564.87      |
| Overall Regression        | 0.98873      | 0.92133      |
| Best validation epoch     | 112          | 6            |
| Error Histogram           | 0.1006       | -3.864       |
| Total epoch runs          | 118          | 12           |
5. Simulation

After the training of NN we move on to Simulink where the robotic arm motion is carried out. The robot’s Simscape model is created in a multibody simulation environment with an object and a floor. The contact force between the gripper and object is modeled using spatial contact force block. A path planner is designed with a set of connected waypoints for generating a trajectory between a start point and goal point. For this purpose, a trapezoidal velocity trajectory is designed which can generates a smooth and shortest path. The trajectory is created with waypoints A, B and C having cartesian coordinates (X, Y, Z). The position coordinates are given as input to the NN-1 for predicting the joint variables required for each path point (Table.4). These values are finally fed to the path planner for actuating joints.

The manipulator picks the object positioned at point A, accelerates all the way to point B with maximum velocity and then decelerates towards point C where it places the object. Figure 15-17 illustrates path configuration of the end effector and motion of the robotic arm in Cartesian space.

| Coordinates | Position (x,y,z) (mm) | Predicted joint values |
|-------------|------------------------|------------------------|
| A           | (857,0, 433)           | 65.960 42.776 -47.6843 407.322 -44.468 |
| B           | (400, 0, 519)          | 0 91.0567 -87.988 401 -90 |
| C           | (870, 0, 49)           | 121.187 0 -17.576 403.0028 -88.752 |

Figure 14: Process

Table 4: NN1 predictions

Figure 15: Start point (A)  
Figure 16: Mid-point (B)  
Figure 17: Goal point (C)
6. Results & discussion

After training of both networks, two samples of End-effector pose from each data set are introduced our neural network. This data set has a known set of joint variables (target) calculated from the section 3. At this stage, the data undergoes a simulation run through the previously trained network and generate results as network outputs. The difference between the predicted value and the target value from the NN’s is shown below and a comparison is drawn between NN1 and NN2 predictions. The NN1 predicted the actual joint angles, successfully at most target values. It is seen that the prediction error in NN1 is very minimal as compared to NN2.

Table 5: Error calculation for NN1 outputs

| Known input (End-effector position) | Joint variables | Target [T] | Predicted [P] | Error [T – P] |
|------------------------------------|----------------|------------|--------------|--------------|
| (Px, Py, Pz)                       | Q1             | 30         | 30.68054     | -0.68054     |
|                                    | Q2             | 59         | 60.51092     | -1.51092     |
| (30.36, 17.53, 1274.06)            | Q3             | 42         | 42.58868     | -0.58868     |
|                                    | D4             | 523        | 520.476      | 2.524        |
|                                    | Q5             | 27         | 28.5938      | -1.5938      |
| (Px, Py, Pz)                       | Q1             | 55         | 56.28757     | -1.28757     |
|                                    | Q2             | 46         | 46.29584     | -0.29584     |
| (394.10, 562.84, 1184.18)          | Q3             | 2          | 0.898731     | 1.101269     |
|                                    | D4             | 607        | 610.8413     | -3.84134     |
|                                    | Q5             | 57         | 57.94988     | -0.94988     |

Table 6: Error calculation for NN1 outputs

| Known input (End-effector Orientation & Position) | Joint variables | Target [T] | Predicted [P] | Error [T – P] |
|-------------------------------------------------|----------------|------------|--------------|--------------|
| (RX, RY, RZ, PX, PY, PZ, )                       | Q1             | 62         | 58.31893     | -3.68107     |
|                                                 | Q2             | 13         | 17.23059     | -4.23059     |
| (0.45, 0.11, 0.8, 0.85, 0.21, -0.47,           | Q3             | 16         | 8.60745      | 7.39255      |
| -0.24,0.97,0,463, 0,                           | D4             | 511        | 525.3929     | -14.3929     |
| 0.94,872.52,487.40)                             | Q5             | -43        | -37.6786     | -5.3214      |
| (RX, RY, RZ, PX, PY, PZ)                       | Q1             | 4          | -2.71732     | 6.71732      |
|                                                 | Q2             | 61         | 68.5568      | -7.5568      |
| (-0.88, 0.45, 0.069, -0.062, 0.031,            | Q3             | 61         | 54.55283     | 6.44717      |
| -0.997, -0.453, -0.89, 0,                      | D4             | 414        | 396.9698     | 17.03023     |
| -147.98, -10.35, 907.26)                      | Q5             | 85         | 76.44982     | 8.55018      |

To obtain a better understanding of NN prediction, the network is tested on the yet unseen test set and comparison is drawn based on Regression value. The regression value [R] measures the correlation between targets and outputs. Where [R] value near to 1 means close relationship and closer to 0 means random relationship. It is found that the R value for NN-1 is 0.98784 (Figure 18), indicating a good fitness of 98.78% and value of 0.83205 for NN-2 (Figure 19) showing a poor fit of 83.205%.
In a neural network, the number of parameters means number of weights which is $\propto$ to number of layers and the number of neurons in each layer. If there are more number of layers, more are the number of activation functions disrupting the linearity between the layers. Further, a huge weight in the network leads to increased input-output variance, causing incorrect predictions in test data and decreasing generalization of the NN. Therefore, it can be said that, more the number of input parameters more are the chances of overfitting. Hence, it can be concluded that NN1 has the best prediction over NN2 and that’s the reason NN1 model have been used for simulating robotic arm.

7. Conclusion
Our work introduced a very accurate solution for inverse kinematic problem. By using artificial neural network, the drawbacks in solving inverse kinematics such as nonlinear trigonometric equations and multiple solutions are overcome. When compared to the traditional methods such as geometric, iterative, and algebraic, ANNs are computationally inexpensive and can easily identify, map, and predict complex non-linear behaviour in data sets. The NN1 developed here is confidently able to predict the inverse kinematic solution for RRRPR manipulator with high accuracy and low MSE. The pick & place simulation of robotic arm has been successfully executed and is within the constraint of designed structure of manipulator. Robot kinematics are thoroughly analyzed and suitable path planner is designed, achieving a shortest path from point A to point C. In future, this NN can be extended for performing intricate operations such as sorting of 3D primitive objects with different shapes in real time, where Convolutional NN architectures can be used for shape classification.

Conflicts of Interest: The authors declare no conflicts between them regarding the present work.

References
[1] T. Ogawa and H. Kanada (2010), “Solution for Ill-posed inverse kinematics of robot arm by network inversion,” Journal of Robotics, vol. 2010, Article ID 870923, 9 pages.
[2] T. Hasan, N. Ismail, A.M.S. Hamouda, I. Aris, M. H.Marhaban, and H.M.A.A. Al-Assadi (2010), “Artificial neural network-based kinematics Jacobian solution for serial manipulator passing through singular configurations,” Advances in Engineering Software, vol. 41, no. 2, pp. 359–367.
[3] R. Koker (2013), “A genetic algorithm approach to a neural-network based inverse kinematics solution of robotic manipulators based on error minimization,” Information Sciences, vol. 222, pp. 528–543.
[4] V. Duka (2014), “Neural network based inverse kinematics solution for trajectory tracking of a robotic arm,” Procedia Technology, vol. 12, pp. 20–27.
[5] Luv, A. Kush, and J. Ruth (2014), “Use of artificial neural networks for the development of an inverse kinematic solution and visual identification of singularity zone(s),” Procedia CIRP, vol. 17, pp. 812–817.
[6] D. Prusak, K. Kobus, and G. Karpiel (2014), “Kinematics analysis of a novel five-degree-of-freedom spatial parallel micromanipulator,” Journal of Robotics, vol. 2014, Article ID 806294, 13 pages.