http://doi.org/10.5281/zenodo.13763695

Inverse kinematics using neural networks and random forests for trajectory tracking of a three-degree-of-freedom robotic arm

Jose Francisco Ramírez-Vasquez[1], Emanuel de Jesus Carlock-Acevedo[2], Joel Castro-Ramírez[1]* iD

[1]Department of Mechatronic Engineering, Universidad Politécnica de Tlaxcala, A. Universidad Politécnica, No.1, San Pedro Xalcaltzinco, 90180, Tlaxcala, México. [2]Facultad de Ciencias Básicas, Ingeniería y Tecnología, de la Universidad Autónoma de Tlaxcala, De Apizaquito, 20 de Noviembre, 90401 Cdad. de Apizaco, Tlaxcala, México.

Email de autor para correspondencia: *joel.castro@uptlax.edu.mx

Recibido: 3 junio 2024. **Aceptado:** 02 septiembre 2024

ABSTRACT

Direct and inverse kinematics are crucial in the operation of manipulator robots to achieve desired positions and orientations and execute specific tasks through precise trajectory tracking in the workspace. This study addresses the challenge of inverse kinematics for a three-degree-of-freedom robot, highlighting its complexity due to the nonlinear nature of the trigonometric equations and the existence of multiple solutions for a given end-effector position.

To solve this problem, two machine learning approaches were implemented: artificial neural networks and random forests. First, the direct kinematics model was obtained using the Denavit-Hartenberg method. With these equations, a training dataset was generated by positioning the robot at various points within the workspace using MATLAB and the Robotics Toolbox by Peter Corke.

The models were developed in Python using TensorFlow, Keras, and Scikit-learn. The neural network was adjusted by increasing the number of neurons in the hidden layers until a satisfactory response was obtained, while the random forest model was optimized by varying the number of decision trees. Both models were evaluated with previously unseen trajectories, and their performance was compared using graphs generated in Python and trajectory tracking simulations in MATLAB.

The results demonstrated that both approaches can effectively adapt to multidimensional regression problems when provided with an appropriate dataset. In this context, the inverse kinematics problem for a three-degree-of-freedom robotic manipulator was posed as a regression problem, using neural network and random forest models to map a set of numerical inputs to a set of numerical targets, thereby demonstrating the feasibility of these approaches in solving complex robotics problems.

Keywords: Robotic arm; inverse kinematics; neural networks; random forests; trajectories.

RESUMEN

La cinemática directa e inversa son cruciales en el manejo de robots manipuladores para lograr las posiciones y orientaciones deseadas y ejecutar tareas específicas a través del seguimiento preciso de trayectorias en el espacio de trabajo. Este estudio aborda el desafío de la cinemática inversa de un robot de tres grados de libertad, destacando su complejidad debido a la naturaleza no lineal de las ecuaciones trigonométricas y la existencia de múltiples soluciones para una ubicación dada del efector final.

Para resolver este problema, se implementaron dos enfoques de aprendizaje automático: redes neuronales artificiales y bosques aleatorios. Primero, se obtuvo el modelo de cinemática directa mediante el método de Denavit-Hartenberg, con estas ecuaciones, se generó un conjunto de datos de entrenamiento, posicionando el robot en diversos puntos dentro del área de trabajo utilizando MATLAB y la librería Robotics Toolbox de Peter Corke.

Los modelos se desarrollaron en Python utilizando TensorFlow, Keras y Scikit-learn. La red neuronal se ajustó incrementando el número de neuronas en las capas ocultas hasta obtener una respuesta satisfactoria, mientras que el modelo de bosque aleatorio se optimizó variando el número de árboles de decisión, ambos modelos se evaluaron con trayectorias no vistas previamente, y su rendimiento se comparó mediante gráficos obtenidos en Python y simulaciones de seguimiento de trayectorias en MATLAB.

Los resultados demostraron que ambos enfoques pueden adaptarse eficazmente a problemas de regresión multidimensional cuando se les proporciona un conjunto de datos adecuado. En este contexto, el problema de la cinemática inversa para un manipulador robótico de tres grados de libertad se planteó como un problema de regresión, utilizando los modelos de redes neuronales y bosques aleatorios para mapear un conjunto de datos de entradas numéricas a un conjunto de objetivos numéricos, mostrando así la viabilidad de estos enfoques en la resolución de problemas complejos de robótica.

Palabras clave: Brazo robótico; cinemática inversa; redes neuronales; bosques aleatorios; trayectorias.

INTRODUCCIÓN

The relevance of calculating direct and inverse kinematics is of great importance in the handling of manipulator robots to achieve the desired position and orientation, or to perform specific tasks based on proper trajectory tracking in a workspace.

On one hand, we have direct kinematics, which indicates, from a joint space—meaning angular positions in revolute joints and distances in prismatic joints—the position of the end effector in a Cartesian space. On the other hand, we have inverse kinematics, where from the knowledge of coordinates in a Cartesian space

(position and orientation), we calculate the necessary values for each of the joints of a manipulator robot. These relationships can be observed in Figure 1.

The solution to the inverse kinematics problem in robotics poses significant challenges due to the nonlinear nature of the trigonometric equations linking the position and orientation of the end effector in Cartesian space with the joint variables of the manipulator [1, 2]. This complexity is exacerbated by the existence of multiple solutions for the same end effector location, necessitating a meticulous approach to determine the optimal joint configuration.

In the face of the difficulty of finding analytical solutions directly, various alternative strategies have emerged, such as geometric methods [3-8], numerical algorithms based on optimization techniques [9, 10], evolutionary computing, and the application of neural networks [11, 12]. Notably, the latter stand out for their ability to learn from examples and represent complex relationships between input and output data, making them a promising tool to address the challenge of mapping between Cartesian space and joint space in the inverse kinematics of robotic manipulators.

In [13], various neural network structures are tested for solving the inverse kinematics problem. These structures include neural networks trained by backpropagation or neural networks whose weights are defined in terms of sine and cosine functions to fit the representation of the robot's direct kinematics.

In [14], the focus is on the use of a neural network to determine the joint configuration corresponding to a specific location of the end effector in a three-link robotic manipulator, training the network with direct kinematics data to learn the inverse mapping of the configuration space.

This paper investigates the application of a neural network to solve the inverse kinematics problem for a three-degree-of-freedom robotic manipulator arm, in addition to comparing the results with the use of a random forest model, a machine learning tool. A random forest is a supervised learning algorithm used in the field

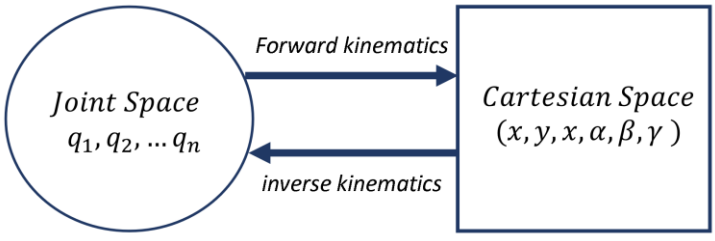


Figure 1. Forward and inverse kinematics.

of machine learning and belongs to the category of ensemble methods, which means it combines multiple learning models to improve predictive performance [15, 16], making it a robust and flexible model that can handle a variety of machine learning problems, including classification and regression, furthermore, it tends to generalize well to previously unseen data.

The mentioned machine learning models can learn complex and nonlinear patterns in the training data, which can lead to higher accuracy in predicting the robot joint configurations for a given task. Moreover, these models can generalize well to new data, meaning they can perform well in scenarios not encountered during training. They can adapt to different types of robots and work environments, making them useful in applications where robots may have different joint configurations or where the work environment may vary [17, 18]. Additionally, they can be computationally efficient once trained, allowing for real-time inverse kinematics calculations, which are crucial for real-time robot control applications. Traditionally, solving the inverse kinematics problem involved deriving complex mathematical equations for each specific robot configuration, and the use of machine learning approaches can reduce the need for this manual engineering [19, 20, 21], enabling faster and scalable development of inverse kinematics solutions.

Furthermore, more recent research has addressed this problem from various perspectives. For example, Duka [17] proposed a neural network-based solution for redundant manipulators with joint constraints. On the other hand, Ren & Ben-Tzvi [18] improved the inverse kinematics solution using data augmentation techniques and transfer learning. Zhang [22] explored how deep learning techniques can further enhance inverse kinematics solutions in robotic manipulators. These studies provide a broad perspective on the use of neural networks in the inverse kinematics of manipulator robots, highlighting different approaches and considerations.

Problem statement

In modern robotics, optimizing algorithms for the control and manipulation of robotic arms is crucial for improving efficiency and precision in various industrial and research applications. A robotic arm with multiple degrees of freedom and revolute joints can perform complex tasks that require precise and controlled movements. However, developing and evaluating effective algorithms for the control of these robotic arms remains a significant challenge due to the complexity of their movements and the dynamic interactions between their components. Identifying algorithms that optimize the performance of these systems is fundamental to advancing the field of robotics.

The central problem of this work is to evaluate the effectiveness of different proposed algorithms for the control of a fictional three-degree-of-freedom robotic arm, with revolute joints and link lengths of $\boldsymbol{l_1 = 0.1}$; $\boldsymbol{l_2 =}$

$\mathbf{0.5}$; $\boldsymbol{l_3 = 0.5}$

In Figure 2 shows a representation of the three-degree-of-freedom robotic arm in a three-dimensional Cartesian space, with the links connected by rotational joints.

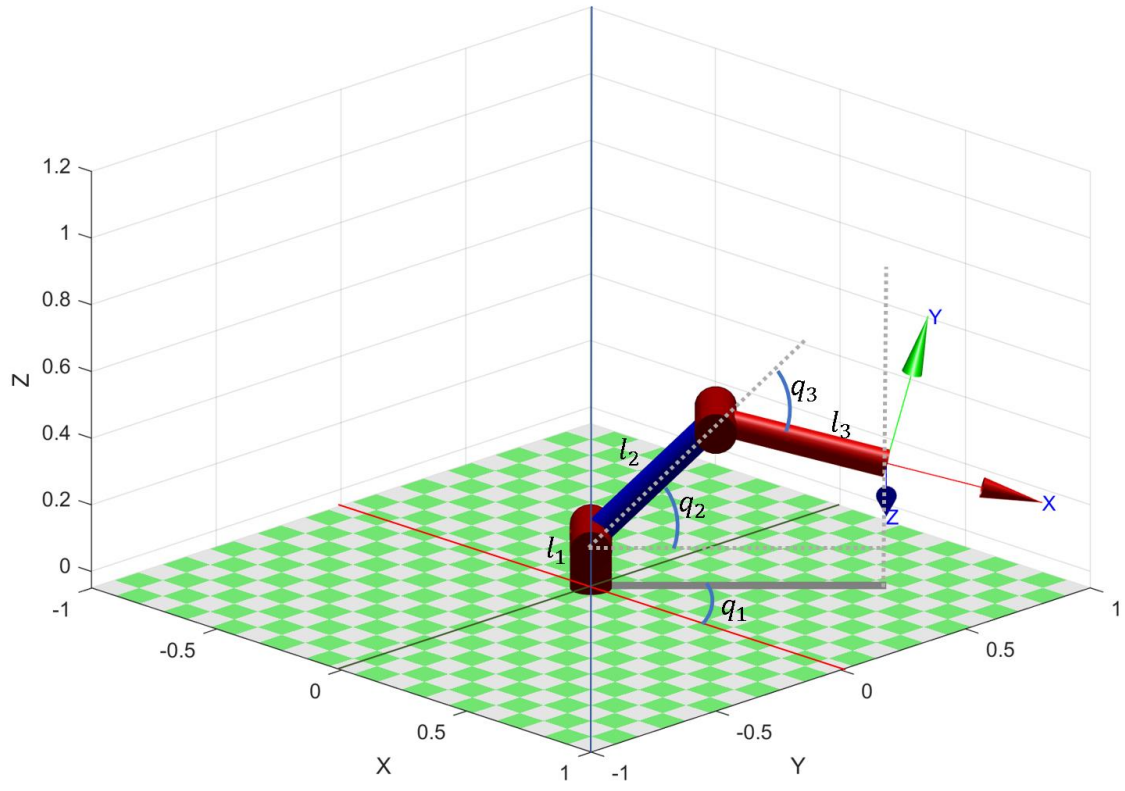


Figure 2. Three-link manipulator created with MATLAB and Robotics Toolbox.

The equations for forward kinematics are given by (1), which define the coordinates of the end effector of the robotic arm in the attached three-dimensional reference frame. On the other hand, the orientation of the end effector is represented by rotations with respect to the reference frame attached to the end effector, as shown in (2), relative to the robot's base reference frame.

$$\begin{aligned} x &= [\, l_2 \cos(q_2) + l_3 \cos(q_2 + q_3) \,] \cos(q_1) \\ y &= [\, l_2 \cos(q_2) + l_3 \cos(q_2 + q_3) \,] \operatorname{sen}(q_1) \\ z &= l_1 + l_2 sen(q_2) + l_3 sen(q_2 + q_3) \end{aligned} \tag{1}$$

$$R_{zyx} = R_z(q_1) R_y(-q_2 - q_3) R_x(\pi) \tag{2}$$

In equations (3, 4, 5), the homogeneous transformation matrices of each frame in the robot are expressed by the Denavit-Hartenberg matrix [23], and equation (6) provides the forward kinematics of our three-degree-of-freedom manipulator involving the position and orientation of the robot. The Denavit-Hartenberg parameters are shown in Table 1.

Table. 1. Denavit-Hartenberg parameters for the 3-degree-of-freedom robot.

Joint (i)	θ_i	d_i	a_i	α_i
1	q_1	l_1	0	$\pi/2$
2	q_2	0	l_2	0
3	q_3	0	l_3	0

$$H_1^0 = \begin{bmatrix} \cos(q_1), & 0 & \sin(q_1) & 0 \\ \sin(q_1) & 0 & -\cos(q_1) & 0 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

$$H_2^1 = \begin{bmatrix} \cos(q_2) & -\sin(q_2) & 0 & l_2\cos(q_2) \\ \sin(q_2) & \cos(q_2) & 0 & l_2\sin(q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

$$H_3^2 = \begin{bmatrix} \cos(q_3) & -\sin(q_3) & 0 & l_3\cos(q_3) \\ \sin(q_3) & \cos(q_3) & 0 & l_3\sin(q_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

$$H_3^0 = \begin{bmatrix} \cos(q_2 + q_3)\cos(q_1) & -\sin(q_2 + q_3)\cos(q_1) & \sin(q1) & \cos(q_1)(l_3\cos(q_2 + q_3) + l_2\cos(q_2)) \\ \cos(q_2 + q_3)\sin(q_1) & -\sin(q_2 + q_3)\sin(q_1) & -\cos(q_1) & \sin(q_1)(l_3\cos(q_2 + q_3) + l_2\cos(q_2)) \\ \sin(q_2 + q_3) & \cos(q_2 + q_3) & 0 & l_1 + l_3\sin(q_2 + q_3) + l_2\sin(q_2) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

Similarly, the equations for inverse kinematics are obtained in (3, 4, 5), with which a comparison is made with the performance of the neural network algorithm and the random forest.

$$q_1 = \operatorname{atan}\left(\frac{p_y}{p_x}\right) \quad (3)$$

$$q_2 = \operatorname{atan}\left(\frac{p_z - l_1}{\sqrt{p_x^2 + p_y^2}}\right) - \operatorname{atan}\left(\frac{l_3\sqrt{1 - \left(\frac{p_x^2 + p_y^2 + (p_z - l_1)^2 - l_2^2 - l_3^2}{2\,l_2 l_3}\right)^2}}{l_2 + l_3\left(\frac{p_x^2 + p_y^2 + (p_z - l_1)^2 - l_2^2 - l_3^2}{2\,l_2 l_3}\right)}\right) \quad (4)$$

$$q_3 = \text{atan}\left(\frac{\sqrt{1-\left(\frac{p_x^2+p_y^2+(p_z-l_1)^2-l_2^2-l_3^2}{2\,l_2 l_3}\right)^2}}{\frac{p_x^2+p_y^2+(p_z-l_1)^2-l_2^2-l_3^2}{2\,l_2 l_3}}\right) \quad (5)$$

To solve the inverse kinematics problem of the robotic manipulator arm, the application of two machine learning methods is proposed. The first method involves a sequential neural network for each joint of the manipulator arm, and the second method involves a random forest. For the training of both the neural network and the random forest, a point cloud in Cartesian coordinates is used as learning data for the computational algorithms, and the joint coordinates of each link of the robot serve as targets. For each point in the Cartesian space, there are two joint solutions for the positioning of the robot's end-effector, referred to as elbow up and elbow down. In this work, only the elbow up configuration is used for study purposes, without considering the robot's mechanical constraints, however, the mechanical constraints could be defined to be the same as those of a physical manipulator robot with the same morphology, or by modifying the values of the links, in the replication, transfer, and implementation of the model. A dataset consisting of 600,000 training data points is utilized, as shown in Fig. 4. The robotic arm can move within a workspace depicted in Figure 3. Using the forward kinematics equations (1, 6), the corresponding location of the end effector is calculated. The resulting values are stored in a vector in the form $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ as input or training data, and another vector $(\boldsymbol{q_1}, \boldsymbol{q_2}, \boldsymbol{q_3})$ as output or targets for the machine learning algorithms.

The neural network

In the context of solving the inverse kinematics problem, it can be approached as a regression problem suitable for both neural networks and the random forest algorithm. This involves determining the values or parameters of the joints that produce a desired output, which is the final position of the end effector. To do this, a sequential neural network is trained for each joint of the robot using the obtained data, which have the same structure, the same inputs $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$, and different outputs $\boldsymbol{q_1}, \boldsymbol{q_2}$ and $\boldsymbol{q_3}$ respectively. The training was conducted by varying the number of neurons in the hidden layers and the number of hidden layers, while maintaining three neurons in both the input and output layers, the Mean Squared Error (MSE) value was observed until a desirable error close to 1e-05 was achieved in the tests, a total of 100 training epochs were established, with early stopping applied to prevent overfitting, each test took between 80 and 95 epochs until early stopping was triggered.

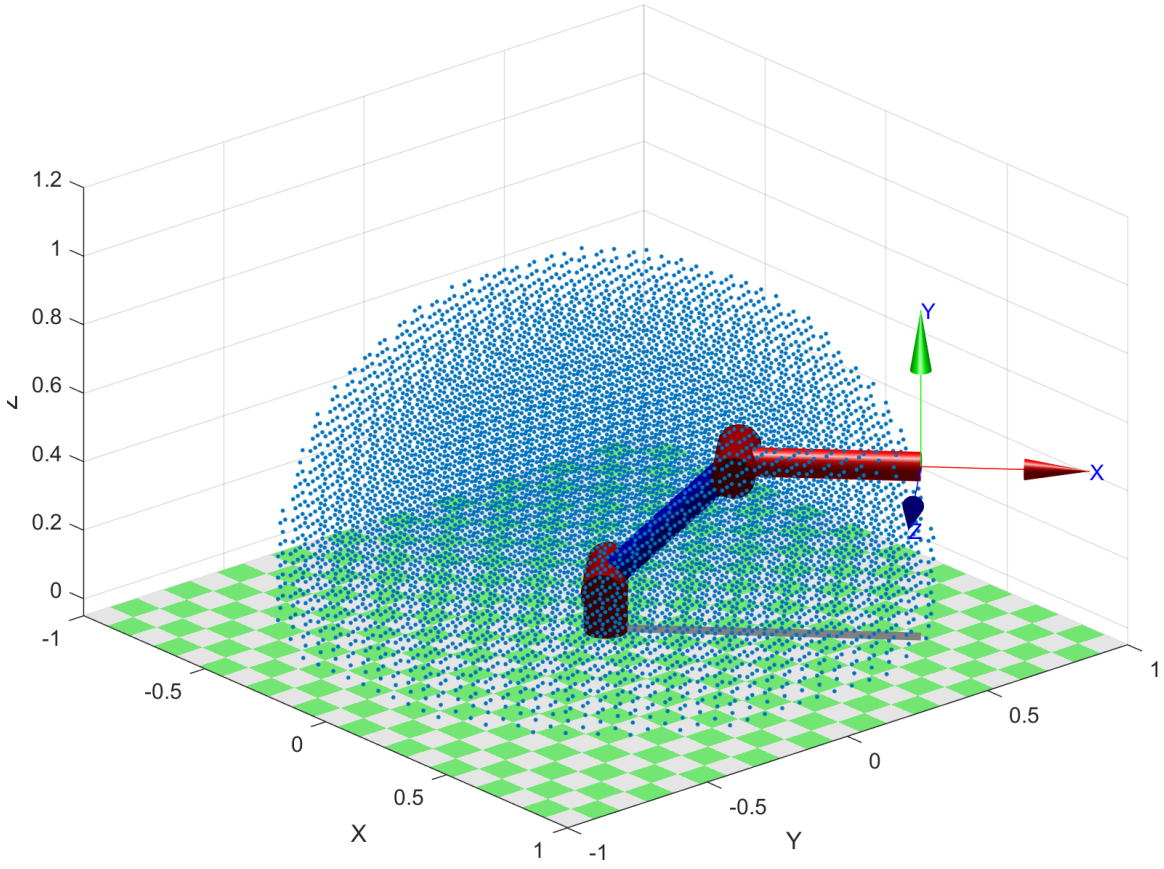


Figure 3. Location of the end effector to generate joint values.

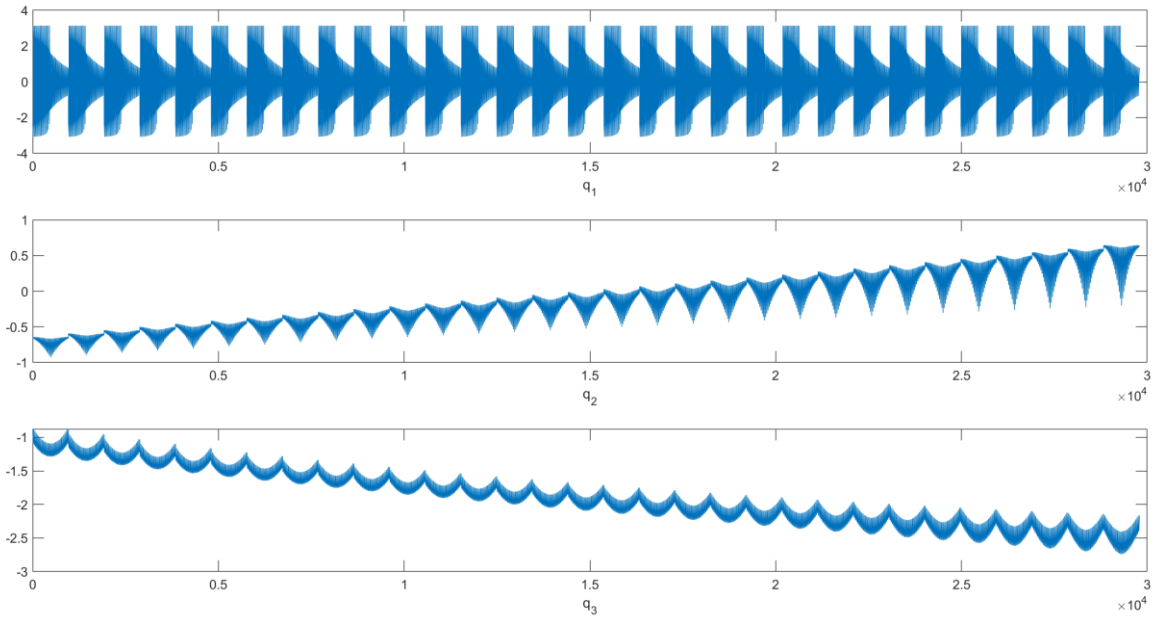


Figure 4. Coordinates of each joint locating positions $(\boldsymbol{q_1}, \boldsymbol{q_2}, \boldsymbol{q_3})$.

The Table 2. shows some of the tests conducted and the mean squared error obtained by each neural network model, highlighting the selected model. During training, the input data consist of 3-dimensional location vectors, resulting from direct kinematics, while the target data are the corresponding sets of 3-dimensional joint parameters, generated previously.

Table 2. Number of layers and neurons in the tests conducted with their obtained Mean Squared Error (MSE).

Proof	neurons in input layer	Hidden layers and neurons in the hidden layers	neurons in output layer	Mean Square Error (MSE)
1	3	Layer 1:16 Layer 2: 4	3	0.0029
2	3	Layer 1:32 Layer 2: 8	3	3.8575e-04
3	3	Layer 1:32 Layer 2: 16	3	1.8189e-04
4	3	Layer 1:64 Layer 2: 16	3	7.4771e-05
5	**3**	**Layer 1:64 Layer 2: 32**	**3**	**5.5989e-05**
6	3	Layer 1:64 Layer 2: 64	3	5.0762e-05
7	3	Layer 1:128 Layer 2: 64	3	3.3594e-05
8	3	Layer 1:128 Layer 2: 128	3	3.7288e-05
9	3	Layer 1:64 Layer 2: 32 Layer 3:16	3	3.5500e-05

Given that, for the same location in the end effector space, the inverse kinematics problem has multiple solutions, duplicates are identified in the input data of the neural network, and a unique training set is established. This way, each configuration that the neural network is learning refers to a unique mapping from Cartesian space to joint space.

The training set is not subjected to a preprocessing stage before being used for actual training; the values provided to the neural network algorithm are directly the obtained data.

The structure of the neural network used to learn the inverse kinematics of the robot is shown in Figure 5.

This backpropagation neural network consists of 3 inputs, two hidden layers: the first with 64 neurons and the second with 32 neurons, and one neuron in the output layer. The transfer function for the neurons in the hidden layers is the ReLU (Rectified Linear Unit) function, as shown in equation (6) and its response in Figure 6. For the output neuron, the transfer function is the linear function, as shown in equation (7) and its response in Figure 7.

The output of the neural network is a scalar representing the angle of each corresponding joint, and this same neural network configuration is used for each joint of the robotic arm. The output is determined based on the desired position of the end effector, according to equation (8).

$$q_i = W_k \cdot (ReLU(W_j \cdot (ReLU(W_i \cdot p + b_i)) + b_j)) + b_k \quad (8)$$

where:
p: the input vector (x, y, z),
W_i: weights of the first layer,

b_i : biases of the first layer,
W_j: weights of the second layer,
b_j: biases of the second layer,
W_k: weights of the third layer,
b_k: biases of the third layer,
q_i: is the output of the last neuron.

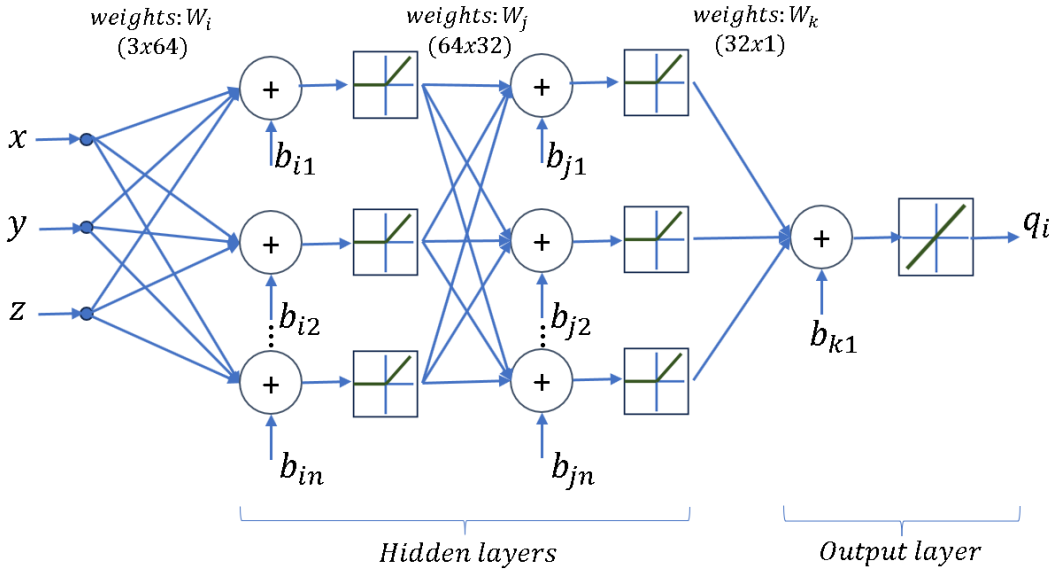


Figure 5. The structure of the neural network.

$$ReLu: \max(0, x) \qquad f(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & other\ case \end{cases} \tag{6}$$

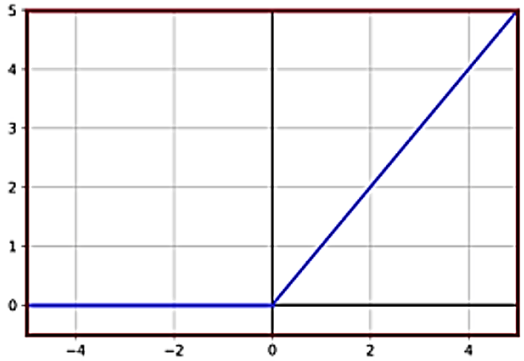


Figure 6. ReLu activation function.

Linear
$$f(x) = x \tag{7}$$

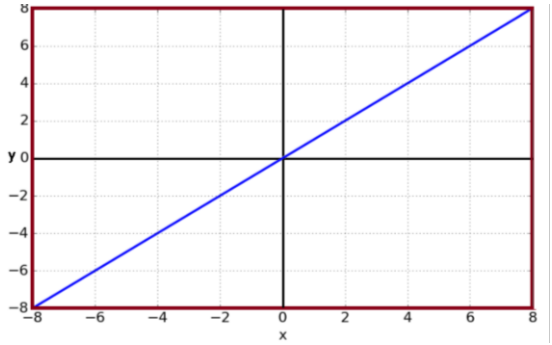


Figure 7. Linear activation function.

The training algorithm used is the Adam (Adaptive Moment Estimation) algorithm, which is an efficient optimization algorithm for training neural networks and combines ideas from the Momentum and RMSProp (Root Mean Square Propagation) optimization algorithms [24].

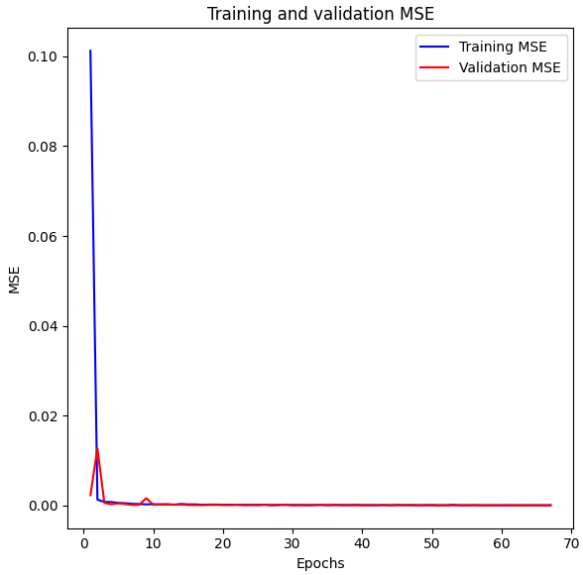
The model training is conducted in Python using the TensorFlow and Keras libraries. Mean squared error is specified as the loss function to optimize, and the Adam optimizer is used to adjust the neural network weights. Additionally, EarlyStopping is employed as a regularization technique to prevent overfitting. With this, training is automatically stopped if the loss on the validation set does not improve after 10 epochs, and monitoring is performed on the validation set loss.

From the set of 600,000 generated samples, 20% is used for validation, and the remaining 80% of the samples are used for training the neural network, in addition to testing with new generated trajectories. The performance of the neural network was determined based on the mean squared error between the neural network outputs, the actual output of the network, and the desired output. Additionally, the possibility of controlled overfitting is considered to potentially improve the performance of the neural network in specific tasks that require high precision and repeatability, such as robotic surgery, semiconductor manufacturing, high-precision 3D printing, automated assemblies, micromanipulation, inspection and quality control, and scientific research.

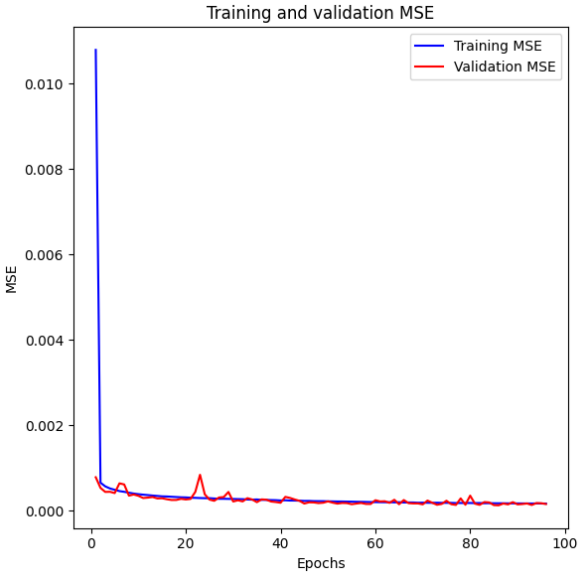
However, there are limitations associated with controlled overfitting. Allowing overfitting can reduce the model's ability to generalize to new, unseen data, which can be a significant drawback if the model needs to handle a variety of tasks or adapt to changing conditions. Furthermore, the model's performance becomes heavily dependent on the quality and representativeness of the training data, and any biases or errors in this data can be amplified, leading to suboptimal performance in real-world applications. There is also the risk that controlled overfitting can lead to overly complex models that are difficult to interpret

and maintain, as well as increase computational requirements, making the model less efficient.

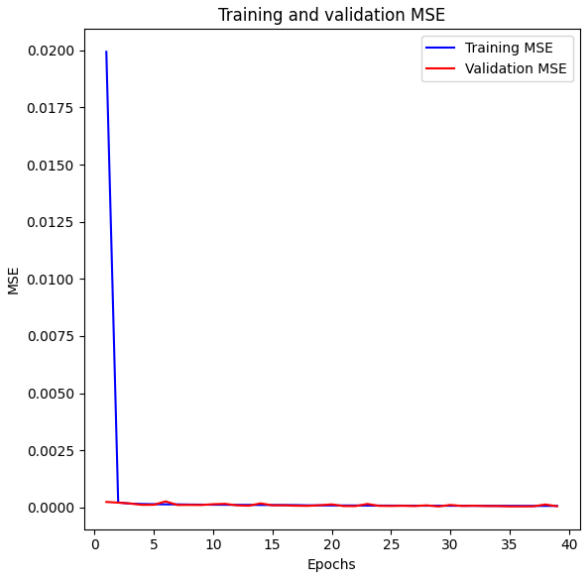
Figure 8 shows the training evolution with the aforementioned test set.



a) Training of q1
loss: 9.9219e-05 - mse: 9.9219e-05



b) Training of q2
loss: 1.4919e-04 - mse: 1.4919e-04



c) Training of q3
loss: 5.5989e-05 - mse: 5.5989e-05

Figure 8. Evolution of neural network training with mean squared error for **a)** joint q_1, **b)** joint q_2 and **c)** joint q_3

The Table 3 show the hyperparameters of the neural network used for each joint of the robotic manipulator arm.

The random forest

Like neural networks, the random forest method or algorithm is suitable for addressing regression problems.

Random Forests are a set of decision trees trained independently and then combined to obtain a more robust and accurate prediction [15, 16, 25], in the context of inverse kinematics, each tree in the forest can be considered as an estimator of the relationship between the Cartesian positions of the end effector and the joint parameters [26].

Table 3. Hyperparameters of the neural network used.

Number of hidden layers:	2 hidden layers.
Number of neurons in each hidden layer:	Layer 1: 64 neurons. Layer 2: 32 neurons.
Activation functions in hidden layers:	ReLU (Rectified Linear Activation)
Activation function of the output layer:	No activation function specified, indicating it is a linear regression.
Optimizer for training:	Adam optimizer.
Learning rate:	Default value of the Adam optimizer.
Loss function:	Mean Squared Error (MSE)
Batch size:	Batch size of 64.
Number of epochs:	100 epochs
Training set size:	80% of the dataset.
Validation set size:	20% of the dataset.
Test set size:	The size of the test dataset varies since tests are conducted with new data by generating new trajectories unknown to the trained neural network.
Early Stopping:	EarlyStopping is used as a technique to stop training if the loss on the validation set does not improve after 10 epochs.

In this work, we explore how the Random Forest method can be an effective alternative to address this problem, comparing it with the traditional neural network approach. We test the performance of a random forest by varying the number of decision trees.

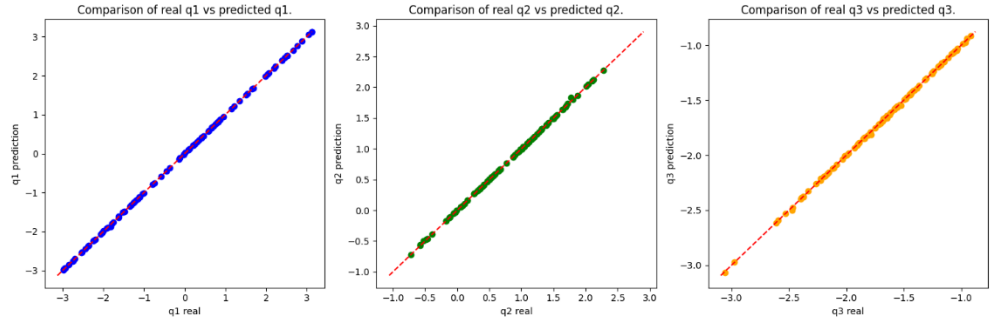
To assess the performance of the Random Forest in solving inverse kinematics, we conducted comparative experiments using the same synthetically generated training data as the neural network. Our dataset comprised 600,000 samples of Cartesian positions of the end effector and their corresponding joint parameters. We split this dataset into 80% for training and 20% for validation, and additionally, we generated new trajectories comprising new datasets for testing.

We create different Random Forest models with 6, 10, 12, 15, 18, 21, 50, and 100 estimators (trees), and a random seed (50) to ensure that the model training process generates the same results each time the code is run, thus ensuring reproducibility of results. This means that other researchers or individuals using this code will obtain the same results if they follow the same instructions and conditions.

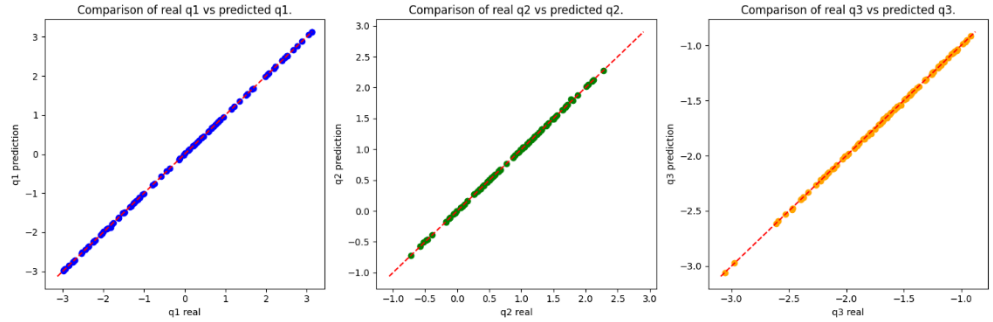
In Table 4, the mean squared error for some of the random forest tests with different decision trees (6, 18, 50, 100) is shown, along with their scatter plot representing the relationship between the actual labels and the predictions generated by the model.

Table 4. Scatter plot for random forest with different numbers of estimators.

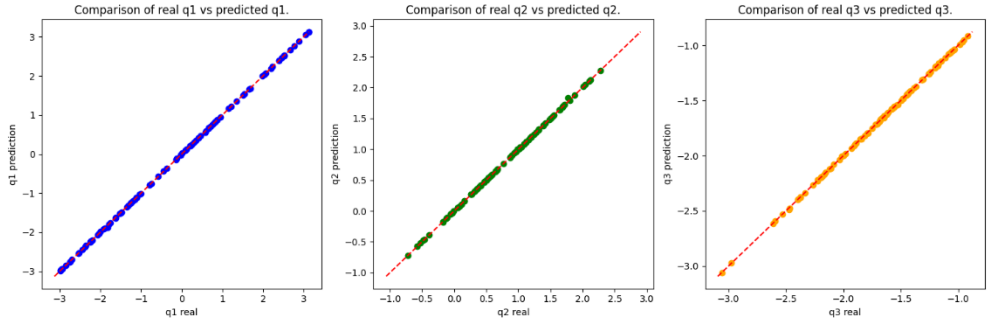
Forest with 6 trees with an inference time of 0.002304 seconds.
Mean squared error on the test set: 3.1888755937091065e-05
Disk space of the model: 261.31 Mb



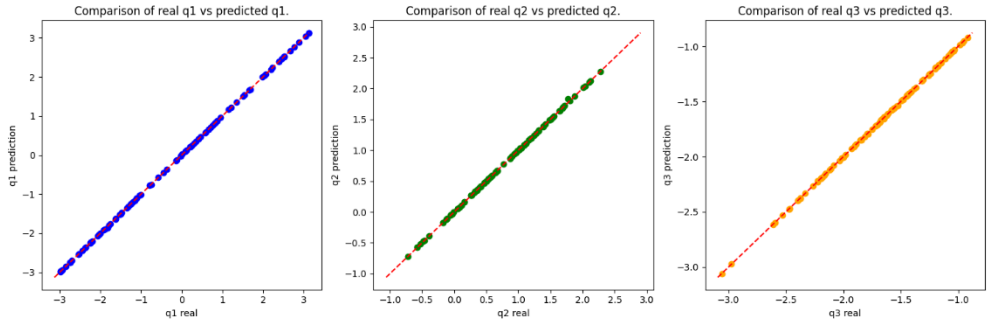
Forest with 12 trees with an inference time of 0.003442 seconds
Mean squared error on the test set: 1.9986268385181817e-05
Disk space of the model: 784.32 Mb



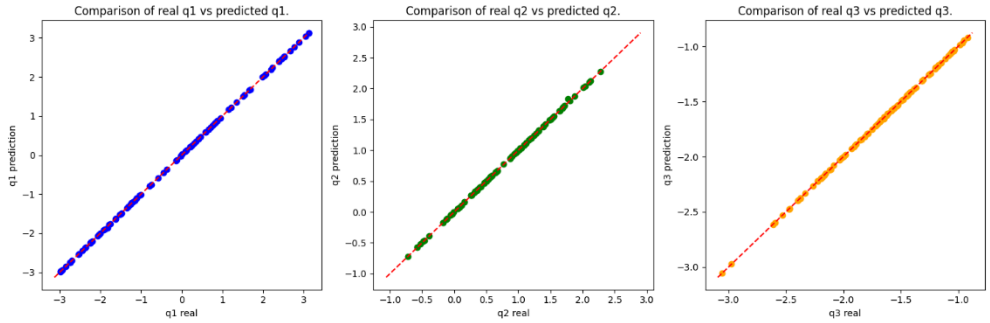
Forest with 18 trees with an inference time of 0.003648 seconds
Mean squared error on the test set: 1.600484020736282e-05
Disk space of the model: 1.15 Gb



Forest with 50 trees with an inference time of 0.014571 seconds
Mean squared error on the test set: 1.0995831374541944e-05
Disk space of the model: 3.19 Gb



Forest with 100 trees with an inference time of 0.035438 seconds
Mean squared error on the test set: 9.583877415837402e-06
Disk space of the model: 6.38 Gb



It is observed that as the random forest model includes more decision trees, the mean squared error decreases. However, the disk space required by the model and the training time increase, therefore, a random forest model with 12 estimators (trees) is chosen, with a disk space of 784.32 MB and a mean squared error of 1.998e-05, this allows the model to run on low-resource machines without compromising performance or the accuracy of the estimated values.

Therefore, the Random Forest Regressor is configured with the following hyperparameters defined by the training command line (12 estimators or trees and a random seed of 50):

'RandomForestRegressor(n_estimators=12, random_state=50)'

RESULTS

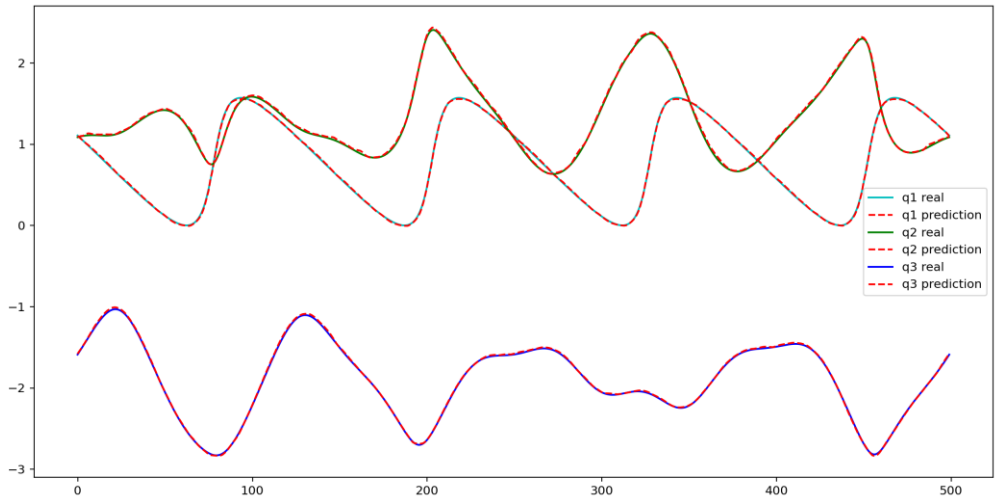
Once the training is completed, new coordinates are generated to form the desired trajectory in the Cartesian system. These new trajectories serve as inputs to the neural network and the random forest in the form (x, y, z), the models will produce parameters as a result that position the end effector on the required trajectory.

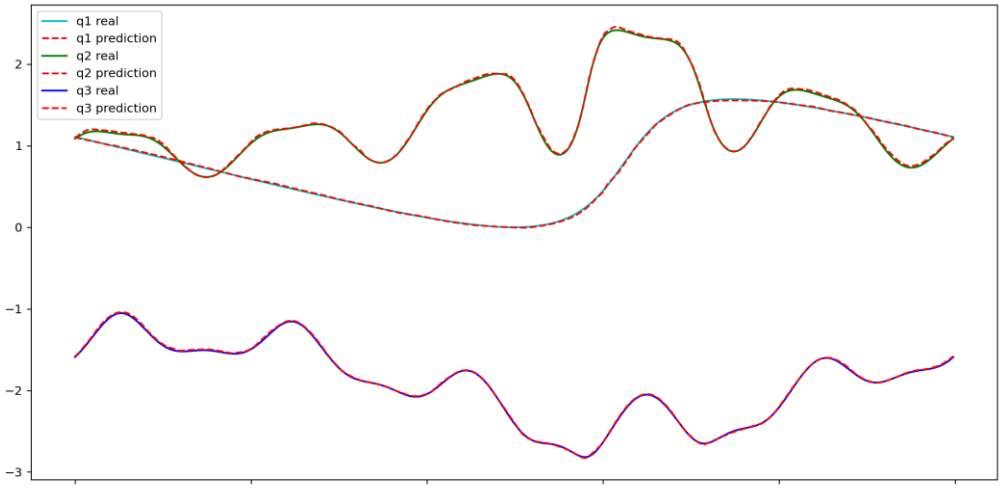
Table 5 shows the numerical results of the Cartesian coordinates obtained by each model, compared to a desired trajectory of 50 data points, similar to the first graph in Figure 9, and the error generated by each model in each of the axes.

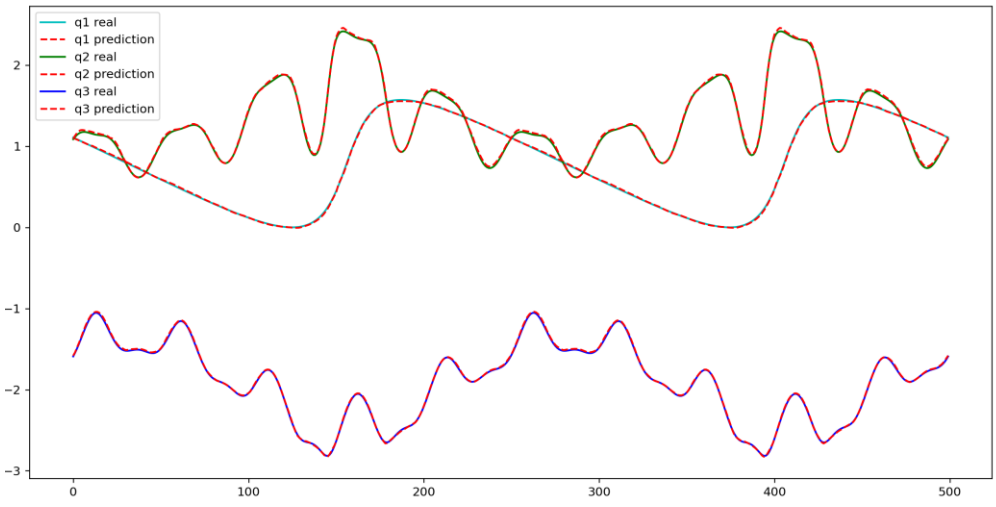
Table 5. Real, estimated, and error coordinates for x, y, z.

				Neural Network							Random Forest					
Real coordinates				Predicted coordinates			Error				Predicted coordinates			Error		
x	y	z		x	y	z	x	y	z		x	y	z	x	y	z
0.3	0.6	0.3		0.3039	0.5995	0.3075	-0.004	0.0005	-0.007		0.3111	0.5995	0.3117	-0.011	0.0005	-0.0117
0.3384	0.5975	0.4794		0.3349	0.5889	0.5042	0.0034	0.0086	-0.025		0.34	0.5991	0.4692	-0.001	-0.0015	0.0102
0.3761	0.5902	0.5876		0.3689	0.5806	0.6107	0.0072	0.0096	-0.023		0.3677	0.5911	0.5906	0.0084	-0.0009	-0.003
0.4126	0.5781	0.5815		0.4023	0.5753	0.6016	0.0103	0.0027	-0.02		0.4201	0.5705	0.5904	-0.007	0.0076	-0.0089
0.4472	0.5614	0.4637		0.4393	0.5652	0.4765	0.0079	-0.004	-0.013		0.4496	0.5697	0.4612	-0.002	-0.0083	0.0024
0.4794	0.5404	0.2808		0.47	0.5451	0.3074	0.0094	-0.005	-0.027		0.4701	0.5417	0.2881	0.0094	-0.0013	-0.0073
0.5087	0.5155	0.1055		0.5052	0.5232	0.1189	0.0035	-0.008	-0.013		0.4988	0.5201	0.111	0.0099	-0.0046	-0.0055
0.5345	0.487	0.0075		0.5334	0.4936	0.0081	0.0012	-0.007	-6E-04		0.5407	0.4921	0.0113	-0.006	-0.0051	-0.0038
0.5565	0.4555	0.0257		0.5579	0.4613	0.0291	-0.001	-0.006	-0.003		0.5495	0.4499	0.0318	0.007	0.0056	-0.0061
0.5743	0.4214	0.1528		0.5771	0.423	0.1681	-0.003	-0.002	-0.015		0.5701	0.4197	0.1619	0.0042	0.0018	-0.0091
0.5876	0.3854	0.3384		0.5813	0.3851	0.3602	0.0063	0.0003	-0.022		0.5907	0.3911	0.3418	-0.003	-0.0058	-0.0034
0.5962	0.3479	0.5087		0.5877	0.3536	0.5261	0.0084	-0.006	-0.017		0.5993	0.3404	0.4986	-0.003	0.0074	0.0101
0.5998	0.3096	0.5962		0.5942	0.3154	0.6036	0.0056	-0.006	-0.007		0.599	0.3114	0.5989	0.0008	-0.0018	-0.0028
0.5986	0.2712	0.566		0.5927	0.2757	0.5755	0.0059	-0.004	-0.01		0.5993	0.2697	0.5698	-0.001	0.0015	-0.0038
0.5925	0.2332	0.4302		0.5894	0.2391	0.4389	0.0031	-0.006	-0.009		0.5918	0.2404	0.4392	0.0007	-0.0072	-0.009
0.5815	0.1964	0.2427		0.5803	0.1984	0.25	0.0013	-0.002	-0.007		0.5913	0.1902	0.2416	-0.009	0.0062	0.0011
0.566	0.1612	0.0779		0.5654	0.1638	0.0883	0.0005	-0.003	-0.01		0.5703	0.1609	0.0825	-0.004	0.0003	-0.0046
0.5461	0.1284	0.0014		0.546	0.1293	0.0013	8E-05	-1E-03	7E-05		0.5491	0.1191	0.0088	-0.003	0.0093	-0.0074
0.5221	0.0983	0.0435		0.5234	0.096	0.0435	-0.001	0.0023	-3E-05		0.5201	0.0916	0.0387	0.002	0.0067	0.0048
0.4945	0.0716	0.1874		0.4962	0.0731	0.1943	-0.002	-0.001	-0.007		0.4913	0.068	0.1894	0.0031	0.0036	-0.002
0.4637	0.0486	0.3761		0.4626	0.0485	0.3809	0.001	1E-04	-0.005		0.4701	0.0401	0.3698	-0.006	0.0085	0.0063
0.4302	0.0297	0.5345		0.4332	0.0279	0.5332	-0.003	0.0018	0.0013		0.44	0.0191	0.5405	-0.009	0.0106	-0.006
0.3945	0.0153	0.5998		0.3839	0.0128	0.6102	0.0107	0.0025	-0.01		0.3911	0.0186	0.5989	0.0034	-0.0033	0.0009
0.3573	0.0055	0.5461		0.3484	0.0051	0.5558	0.009	0.0005	-0.01		0.3607	0.0104	0.5478	-0.003	-0.0049	-0.0018
0.3192	0.0006	0.3945		0.3183	0.0008	0.403	0.001	-2E-04	-0.008		0.3191	0.0118	0.3913	0.0001	-0.0112	0.0032
0.2808	0.0006	0.2055		0.2751	-0.001	0.2056	0.0057	0.0021	-9E-05		0.2904	0.0116	0.2114	-0.009	-0.011	-0.0059
0.2427	0.0055	0.0539		0.2476	0.0043	0.0606	-0.005	0.0012	-0.007		0.2419	0.0115	0.0599	0.0007	-0.006	-0.0059
0.2055	0.0153	0.0002		0.2028	0.0132	0.0036	0.0027	0.0021	-0.003		0.21	0.0188	0.0099	-0.004	-0.0035	-0.0097
0.1698	0.0297	0.0655		0.1694	0.0262	0.0671	0.0004	0.0036	-0.002		0.1686	0.0191	0.0599	0.0012	0.0106	0.0056
0.1363	0.0486	0.2239		0.1434	0.0496	0.2284	-0.007	-0.001	-0.004		0.1394	0.0401	0.218	-0.003	0.0084	0.0059
0.1055	0.0716	0.4126		0.0957	0.065	0.4211	0.0099	0.0066	-0.009		0.1107	0.07	0.4122	-0.005	0.0016	0.0004
0.0779	0.0983	0.5565		0.0737	0.0907	0.563	0.0042	0.0077	-0.006		0.0693	0.0901	0.5546	0.0086	0.0082	0.0019
0.0539	0.1284	0.5986		0.0533	0.1215	0.6038	0.0007	0.0069	-0.005		0.0606	0.1192	0.5983	-0.006	0.0091	0.0003
0.034	0.1612	0.5221		0.035	0.1566	0.529	-0.001	0.0046	-0.007		0.04	0.1627	0.5185	-0.006	-0.0015	0.0036
0.0185	0.1964	0.3573		0.0178	0.1933	0.3573	0.0007	0.0031	1E-05		0.0189	0.1901	0.3615	-0.001	0.0063	-0.0041
0.0075	0.2332	0.1698		0.0091	0.2254	0.1704	-0.002	0.0078	-5E-04		0.0108	0.2399	0.1678	-0.003	-0.0066	0.002
0.0014	0.2712	0.034		0.0053	0.2703	0.0341	-0.004	0.0008	-1E-04		0.0116	0.2683	0.0305	-0.010	0.0029	0.0035
0.0002	0.3096	0.0038		0.0052	0.3037	0.004	-0.005	0.0059	-1E-04		0.0113	0.3116	0.0096	-0.011	-0.002	-0.0057
0.0038	0.3479	0.0913		0.0068	0.3507	0.0937	-0.003	-0.003	-0.002		0.0123	0.3394	0.0884	-0.008	0.0084	0.0029
0.0124	0.3854	0.2616		0.0125	0.3853	0.2651	-1E-04	0.0001	-0.003		0.0112	0.3907	0.2616	0.0012	-0.0054	0.0001
0.0257	0.4214	0.4472		0.0245	0.4155	0.4599	0.0012	0.0059	-0.013		0.0182	0.4203	0.4404	0.0075	0.0011	0.0068
0.0435	0.4555	0.5743		0.0389	0.445	0.5852	0.0046	0.0105	-0.011		0.0398	0.4493	0.5699	0.0037	0.0062	0.0044
0.0655	0.487	0.5925		0.0655	0.4784	0.6028	-2E-05	0.0087	-0.01		0.0689	0.4907	0.5919	-0.003	-0.003	0.0006
0.0913	0.5155	0.4945		0.0905	0.5127	0.5085	0.0008	0.0028	-0.014		0.0908	0.5203	0.4935	0.0005	-0.004	0.001
0.1206	0.5404	0.3192		0.1197	0.5408	0.3256	0.0009	-3E-04	-0.006		0.1185	0.5405	0.3192	0.0021	-0.001	0.0001
0.1528	0.5614	0.1363		0.1537	0.5591	0.1463	-9E-04	0.0023	-0.01		0.1606	0.5699	0.1391	-0.007	-0.008	-0.0028
0.1874	0.5781	0.0185		0.1868	0.5808	0.0349	0.0006	-0.003	-0.016		0.1904	0.5702	0.0105	-0.003	0.0079	0.008
0.2239	0.5902	0.0124		0.2272	0.5945	0.0293	-0.003	-0.004	-0.017		0.2185	0.5919	0.0095	0.0054	-0.001	0.0029
0.2616	0.5975	0.1206		0.2619	0.601	0.1307	-3E-04	-0.003	-0.01		0.2612	0.5996	0.1186	0.0004	-0.002	0.002
0.3	0.6	0.3		0.3039	0.5995	0.3075	-0.004	0.0005	-0.007		0.3111	0.5995	0.3117	-0.011	0.0005	-0.0117

Figure 9 illustrates the resulting joint space trajectories corresponding to the new trajectories defined in the Cartesian system, estimated by the neural network model and Figure 10 shows the comparison of new trajectories estimated by the chosen random forest model.


q1 real
q1 prediction
q2 real
q2 prediction
q3 real
q3 prediction


q1 real
q1 prediction
q2 real
q2 prediction
q3 real
q3 prediction


q1 real
q1 prediction
q2 real
q2 prediction
q3 real
q3 prediction

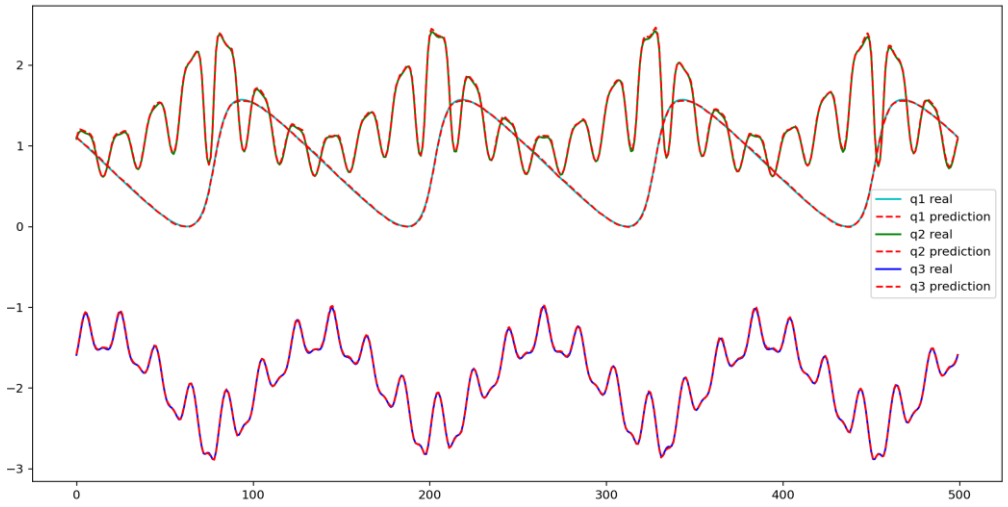
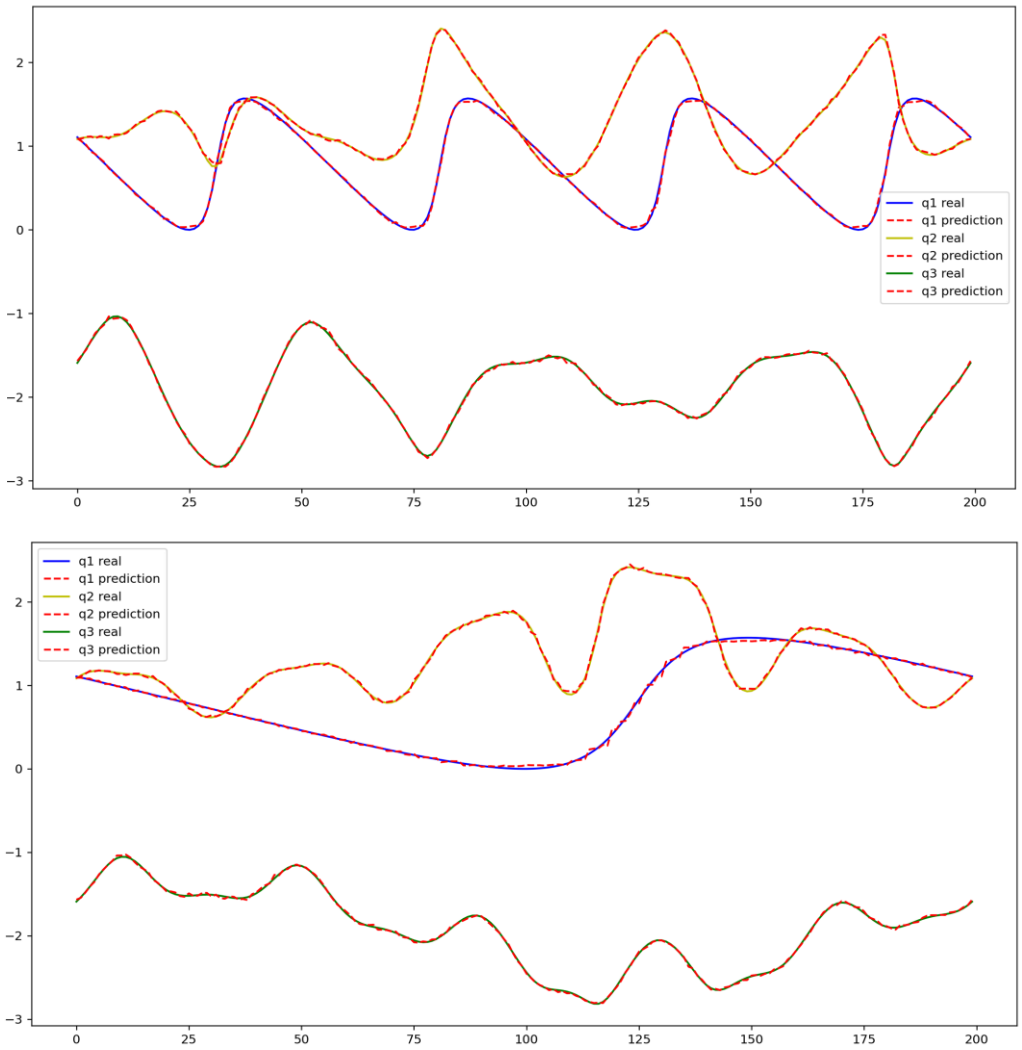


Figure 9. Comparison of the real and estimated trajectories by the neural network in the joint space (q_1, q_2, q_3).



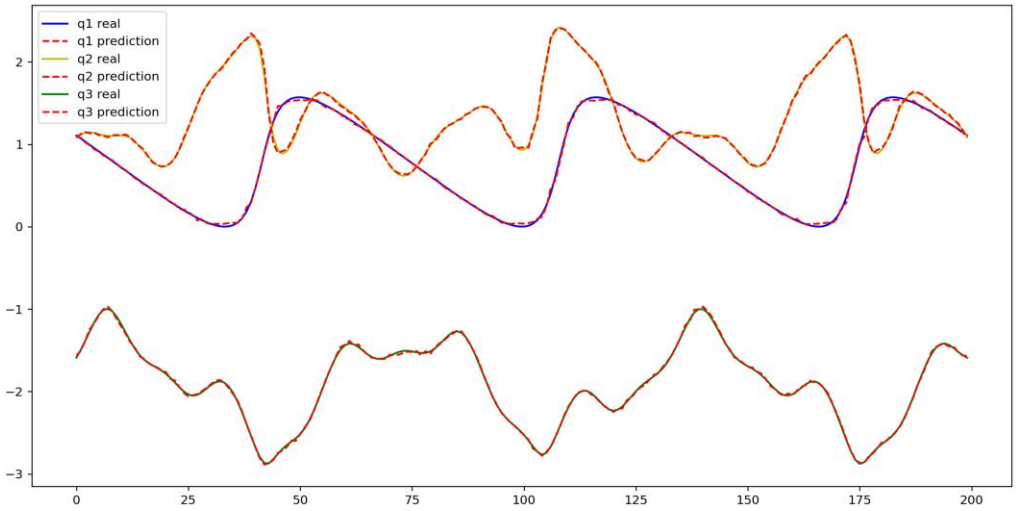


Figure 10. Comparison of the real and estimated trajectories by the random forest model in the joint space (q_1, q_2, q_3).

The points in the generated new trajectories can be easily interpolated by fifth-degree polynomials, and the repeatability of the trajectory without having to resort each time to the computational requirements of the neural network and random forest model demonstrates the tracking capabilities of the robot, whose joint parameters are generated by the neural network and random forest models.

The Figure 11 illustrates how the end effector follows the desired position of the trajectory in the Cartesian space calculated by the neural network, and the Figure 12 shows the tracking capabilities of the end effector generated by the random forest.

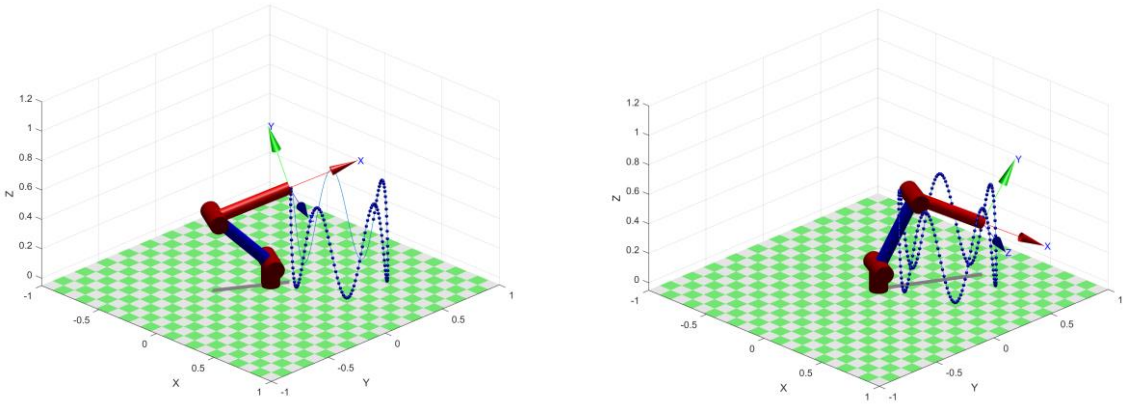


Figure 11. Trajectory tracking using the neural network model.

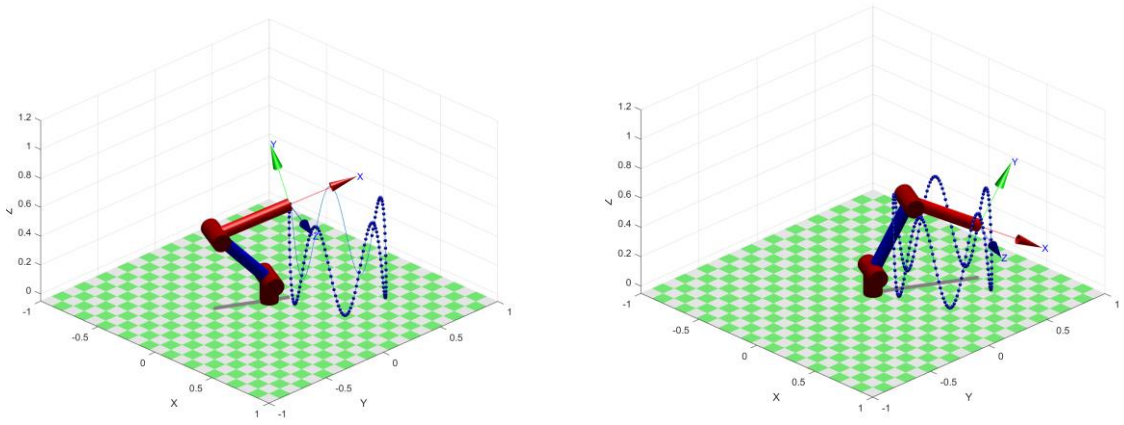


Figure12. Trajectory tracking using the random forest model.

CONCLUSION AND FUTURE WORK

The study demonstrates that both neural networks and random forests can effectively address the inverse kinematics problem for a three-degree-of-freedom robotic manipulator, the successful adaptation of these models to multidimensional mapping tasks, given the appropriate data and network configuration, indicates their potential utility in solving complex kinematic problems. The trajectory tracking results achieved in this study were satisfactory, highlighting the models' ability to handle such problems in a controlled environment.

The proposed methodology has significant potential to advance the field of robotics by providing robust solutions for inverse kinematics problems. The ability to apply machine learning algorithms like neural networks and random forests can improve the efficiency and accuracy of robotic manipulators. This impact is particularly notable in tasks requiring precise and repeatable movements, potentially enhancing capabilities in various industrial and research applications.

Future research could explore several promising directions to build on the findings of this study, one area of focus could be the exploration of additional machine learning algorithms beyond neural networks and random forests to address the inverse kinematics problem, incorporating real-world data into the training process could also provide more practical insights and improve model performance in real robotic systems. Additionally, extending the approach to more complex robotic systems, such as those with six degrees of freedom or different robot configurations, would further test the generalizability and robustness of the proposed methods. Finally, optimizing the size and structure of the models used, as well as applying these algorithms to real robotic manipulators, would be crucial steps in advancing the practical applicability of the research.

CONFLICT OF INTEREST

The authors confirm that the content of this article has no conflicts of interest.

REFERENCES

[1]. Featherstone R. Position and velocity transformation between robot end-effector coordinate and joint angle. The International Journal of Robotics Research. 1983; 2(2): 35-45. Available from: https://journals.sagepub.com/doi/10.1177/027836498300200203

[2]. Lee GCS. Robot Arm Kinematics, Dynamics and Control. Computer. 1982; 15(12): 62-79. Available from: https://ieeexplore.ieee.org/document/1653917

[3]. Duffy J. Analysis of Mechanisms and Robot Manipulators, Wiley, New York, 1980. Available from: https://doi.org/10.1115/1.3157585

[4]. Synder WE. Industrial Robots: Computer Interfacing and Control, Prentice-Hall, New York, 1985. Available from: https://archive.org/details/industrialrobots0000snyd/mode/2up

[5]. Paul RP, Shimano B, Mayer GE. Kinematic control equations for simple manipulators. IEEE Transactions on Systems, Man, and Cybernetics SMC-II. 1981; 6: 66-72. Available from: https://ieeexplore.ieee.org/document/4046335

[6]. Craig JJ. Introduction to Robotics: Mechanisms and Controls, Addison-Wesley, Reading, MA, 1989. Available from: https://api.pageplace.de/preview/DT0400.9781292164953_A42125292/preview-9781292164953_A42125292.pdf

[7]. Manocha D, Canny JF. Efficient inverse kinematics for general 6r manipulators. IEEE Transactions on Robotics and Automation. 1994; 10(5): 648-657. Available from: https://ieeexplore.ieee.org/document/326569

[8]. Manocha D, Zhu Y. A fast algorithm and system for the inverse kinematics of general serial manipulators. IEEE Conference on Robotics and Automation. 1994; 94: 3348-3354. Available from: https://ieeexplore.ieee.org/document/351055

[9]. Köker R. A genetic algorithm approach to a neural-network-based inverse kinematics solution of robotic manipulators based on error minimization. Information Sciences. 2013; 222: 528-543. Available from: https://www.sciencedirect.com/science/article/abs/pii/S0020025512005233

[10]. Olague G, Dunn E. Development of a practical photogrammetric network design using evolutionary computing. The Photogrammetric Record. 2007; 22(117): 22-38. Available from: https://onlinelibrary.wiley.com/doi/10.1111/j.1477-9730.2007.00403.x

[11]. Köker R, Öz C, Çakar T, Ekiz H. A study of neural network based inverse kinematics solution for a three-joint robot. Robotics and Autonomous Systems. 2004; 49(3–4): 227-234. Available from: https://www.sciencedirect.com/science/article/abs/pii/S0921889004001666

[12]. Srisuk P, Sento A, Kitjaidure Y. Inverse kinematics solution using neural networks from forward kinematics equations. In 2017 9th international conference on Knowledge and Smart Technology (KST). Chonburi, Thailand. IEEE. 2017 (pp. 61-65). Available from: https://ieeexplore.ieee.org/document/7886084

[13]. Tejomurtula S, Kak SC. Inverse Kinematics in Robotics using Neural Networks. Inf. Sci. 1999; 116: 147-164. Available from: https://www.sciencedirect.com/science/article/abs/pii/S0020025598100981?via%3Dihub

[14]. Adrian-Vasile D. Neural network based inverse kinematics solution for trajectory tracking of a robotic arm. Procedia Technology. 2014; 12: 20-27. Available from: https://www.sciencedirect.com/science/article/pii/S2212017313006361

[15]. Breiman L. Random forests. Machine learning. 2001; 45: 5-32. Available from: https://link.springer.com/article/10.1023/A:1010933404324

[16]. Segal MR. Machine learning benchmarks and random forest regression. Center for Bioinformatics and Molecular Biostatistics. 2004; UCSF: 1-14. Available from: https://escholarship.org/uc/item/35x3v9t4

[17]. Duka AV. Neural network based inverse kinematics solution for trajectory tracking of a robotic arm. Procedia Technology. 2014; 12: 20-27. Available from: https://www.sciencedirect.com/science/article/pii/S2212017313006361

[18]. Ren H, Ben-Tzvi P. Learning inverse kinematics and dynamics of a robotic manipulator using generative adversarial networks. Robotics and Autonomous Systems. 2020; 124: 103386. Available from: https://www.sciencedirect.com/science/article/abs/pii/S0921889019303501

[19]. Tejomurtula S, Kak S. Inverse kinematics in robotics using neural networks. Information Sciences. 1999; 116(2-4): 147-164. Available from: https://www.sciencedirect.com/science/article/abs/pii/S0020025598100981?via%3Dihub

[20]. Duffy J. Analysis of mechanisms and robot manipulators. London: Edward Arnold. Pp. 156-120.

[21]. Manocha D, Canny JF. Efficient inverse kinematics for general 6R manipulators. IEEE transactions on robotics and automation. 1994; 10(5): 648-657. Available from: https://ieeexplore.ieee.org/document/326569

[22]. Zhang B. Inverse Kinematics Implementation Techniques in Robotics. Highlights in Science, Engineering and Technology. 2024; 81: 109-120. Available from: https://drpress.org/ojs/index.php/HSET/article/view/15925

[23]. Denavit J, Hartenberg RS. A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. Journal of Applied Mechanics. 1955; 22(2): 215-221. Available from: https://doi.org/10.1115/1.4011045

[24]. Dogo EM, Afolabi OJ, Nwulu NI, Twala B, Aigbavboa CO. A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks. In 2018

international conference on computational techniques, electronics and mechanical systems (CTEMS) (pp. 92-99). IEEE. Available from: https://ieeexplore.ieee.org/document/8769211

[25]. Smith PF, Ganesh S, Liu P. A comparison of random forest regression and multiple linear regression for prediction in neuroscience. Journal of neuroscience methods, 2013; 220(1), 85-91. Available from: https://www.sciencedirect.com/science/article/abs/pii/S0165027013003026

[26]. Sanjeev MM, Thomas MJ, Kumar TS, Sudheer AP, Joy ML. Determination of inverse kinematic solutions for a 3 Degree of Freedom Parallel Manipulator using Machine Learning. In 2020 IEEE Students Conference on Engineering & Systems (SCES). 2020; Prayagraj, India: 1-6. Available from: https://ieeexplore.ieee.org/abstract/document/9236725