

Motion Planning of a Flexible Robotic Arm Based on Neural Network Under Simulated Conditions

Xin Li *

Department of Intelligent Science and Technology, Sun Yat-sen University, Shenzhen, 518107, China

* Corresponding Author Email: lixin323@mail2.sysu.edu.cn

Abstract. Flexible robotic systems are gaining significant attention due to their ability to adapt to complex environments, offering advantages in fields such as medical robotics and autonomous manipulation. This study focuses on developing a neural network-based control framework to enhance the motion planning and tracking capabilities of a flexible robotic manipulator. The proposed controller utilizes simulated datasets to train a neural network that maps the current state of the manipulator's actuators and the target end-effector position to the next state, enabling precise single-point positioning and trajectory tracking. Experimental results demonstrate minimal tracking errors, with the mean squared error (MSE) converging close to zero, indicating high accuracy across training, validation, and test phases. The system successfully follows both static targets and dynamic paths, as confirmed by the alignment between actual and target trajectories. Additionally, the controller shows stable performance, ensuring smooth motion without oscillations or delays. This research highlights the potential of neural networks to improve control precision in flexible robotic systems, bridging the gap between simulation and real-world applications. The study lays a solid foundation for further advancements in intelligent control frameworks for flexible robotic systems, contributing to the development of adaptive and autonomous robotic platforms.

Keywords: Flexible robotic arm; Motion planning; neural network.

1. Introduction

Flexible robotic manipulators, inspired by biological systems, have garnered significant attention in recent years due to their ability to adapt to complex environments. Unlike traditional rigid robotic arms, flexible manipulators can deform and adjust their structure, allowing them to operate in confined spaces and interact safely with their surroundings. This makes them highly useful in applications such as medical surgery, rescue operations, and manufacturing. In the medical field, flexible manipulators have shown great potential in minimally invasive surgeries, offering enhanced precision and safety compared to rigid counterparts. Additionally, flexible robots in manufacturing provide versatility in handling intricate assembly and manipulation tasks due to their higher degrees of freedom and soft structure.

Despite their advantages, the dynamic behavior of flexible manipulators is highly nonlinear and complex due to their infinite degrees of freedom and elastic nature. This complexity poses significant challenges in accurately modeling and controlling their movements. Efficient control of flexible manipulators remains an active area of research, as precise control is necessary for their safe and effective operation in real-world scenarios.

The modeling of flexible manipulators has evolved over time, with constant curvature models becoming a widely adopted approach. This model simplifies the mathematical representation of a manipulator by assuming that each segment maintains a constant curvature during movement, which reduces the complexity of kinematic equations. Webster and Jone provided a comprehensive review of constant curvature models, highlighting both their advantages and limitations [1]. Although effective in many scenarios, these models often struggle to capture the full range of dynamic behaviors, especially when external disturbances are present or in complex tasks. Walke and Burgner-Kahrs et al. further emphasized the need for alternative methods that better address the control challenges of flexible robots [2, 3].

Recent research focuses on data-driven approaches to overcome the limitations of traditional models. Neural networks are effective for learning complex, nonlinear behaviors of flexible robots. Abe proposed a trajectory planning method using neural networks and particle swarm optimization to improve vibration control in Cartesian robots, demonstrating the practical value of neural networks in flexible manipulators [4].

In addition to data-driven models, finite element modeling (FEM) has been employed to increase the precision of simulations. Duriez applied FEM for real-time control of soft robots, while Wang et al. explored FEM techniques to bridge the gap between simulation and real-world conditions [5, 6]. These studies confirm the value of combining neural networks and FEM to enhance control and motion planning.

Although significant progress has been made in flexible manipulator modeling and control, challenges remain. The constant curvature model, while simplifying the kinematic equations, does not fully capture the manipulator's complex dynamic properties, particularly in high-dynamic scenarios. Similarly, while neural networks provide a robust framework for modeling nonlinearity, their integration with physics-based models has not been fully realized in practical applications.

This study aims to develop a hybrid control framework that combines the constant curvature model with neural network-based control strategies. The goal is to create a precise and adaptive motion planning system for flexible manipulators, particularly in environments where external disturbances are present. The proposed research framework includes the following steps: first, a hybrid modeling framework that integrates neural network control with the constant curvature model is proposed; and then, simulation experiments are carried out to validate the effectiveness and robustness of the proposed method.

2. Kinematical Modeling

2.1. Section Heading Segment Model and Geometric Parameters

Each segment of the three-segment flexible robotic arm can be described geometrically by its length and bending radius. Suppose the lengths of three flexible actuators for each segment are (l_1, l_2, l_3) , respectively. Through geometric derivation, the bending angle θ and the angle between the robot's bending direction and the location of actuator 1 ϕ for each segment are obtained as [1].

$$\theta = \frac{2\sqrt{l_1^2 + l_2^2 + l_3^2 - (l_1 l_2 + l_1 l_3 + l_2 l_3)}}{l_1 + l_2 + l_3} \quad (1)$$

$$\phi = \tan^{-1} \left(\frac{\sqrt{3}(l_3 - l_2)}{l_2 + l_3 - 2l_1} \right) \quad (2)$$

2.2. Coordinate Transformation Matrix

To implement forward kinematics for the flexible arm, the coordinate transformation matrix for each segment is defined. For the bending motion of each segment, the transformation can be represented by a rotation matrix. The formula for the rotation matrix is:

$$R(\theta, \phi) = \begin{bmatrix} c\phi \cdot c\theta \cdot c\phi + s\phi \cdot s\phi & c\phi \cdot c\theta \cdot s\phi - s\phi \cdot c\phi & -c\phi \cdot s\theta \\ s\phi \cdot c\theta \cdot c\phi - c\phi \cdot s\phi & s\phi \cdot c\theta \cdot s\phi + c\phi \cdot c\phi & -s\phi \cdot s\theta \\ s\theta \cdot c\phi & s\theta \cdot s\phi & c\theta \end{bmatrix} \quad (3)$$

where c is $\cos$, s is $\sin$.

Using this rotation matrix, the coordinates of the endpoint of a segment p_i to the coordinate system of the previous segment p_{i-1} are as follows:

$$\mathbf{p}_{i-1} = \text{Rot}(\varphi, \theta) \cdot \mathbf{p}_i + \begin{bmatrix} l \cdot \sin X\left(\frac{\theta}{2}\right) \cdot \cos\left(\frac{\pi - \theta}{2}\right) \cdot \sin\left(\varphi - \frac{\pi}{2}\right) \\ -l \cdot \sin X\left(\frac{\theta}{2}\right) \cdot \cos\left(\frac{\pi - \theta}{2}\right) \cdot \cos\left(\varphi - \frac{\pi}{2}\right) \\ l \cdot \sin X\left(\frac{\theta}{2}\right) \cdot \sin\left(\frac{\pi - \theta}{2}\right) \end{bmatrix} \quad (4)$$

Where l is the average length from the segment's center to the bending endpoint?

$$\sin X(\alpha) = \begin{cases} 1, & \text{if } \sin(\alpha) = 0 \text{ and } |\alpha| < \pi \\ \frac{\sin(\alpha)}{\alpha}, & \text{otherwise} \end{cases} \quad (5)$$

This definition ensures that when α approaches 0, the function remains differentiable, thereby avoiding numerical instability in calculations. This is crucial for handling small-angle approximations in physical models.

3. Neural Network Construction

In this section, what is presented is the neural network-based control approach applied to a flexible manipulator system. The goal is to utilize the network to map the current state of the manipulator's actuators and the target end-effector position to the required adjustments in actuator lengths for the next time step. This approach leverages the ability of neural networks to model highly non-linear systems effectively, enabling precise control of a complex, highly dynamic robotic structure.

3.1. Dataset Generation

To train the neural network for controlling the flexible manipulator, a dataset of actuator states and corresponding end-effector positions is generated. This process involves simulating the manipulator's behavior using forward kinematics and collecting various configurations.

Initially, the home posture of the manipulator is set with actuator lengths initialized to specific values. The forward kinematics model computes the corresponding end-effector position. Then, to generate more data, the actuator lengths are randomly perturbed over several iterations. Each perturbation is followed by computing the updated end-effector position, which forms the target for the neural network training.

In total, 10,000 samples are generated to ensure a wide range of manipulator behaviors is captured, covering both small and large perturbations of the actuator lengths.

3.2. Neural Network Architecture

A Feedforward Neural Network (FNN) architecture is employed with the following structure:

- 1) Input Layer: Takes in the current lengths of the flexible manipulator's actuating tendons and the desired position of the end-effector.
- 2) Hidden Layer: Contains 50 neurons, utilizing the ReLU activation function to capture the non-linear relationships between the input and output.
- 3) Output Layer: Predicts the next actuator lengths for each tendon in the manipulator.

The architecture is created using the `fitnet` function in MATLAB. The Levenberg-Marquardt algorithm is selected for training due to its efficiency in handling non-linear regression problems [7, 8].

The neural network structure can be visualized in Fig. 1

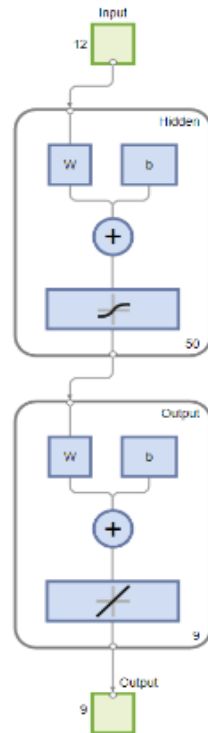


Fig 1. Neural network architecture. (Photo/Picture credit: Original).

3.3. Data Preprocessing and Training

Before training the neural network, the input and output data undergo a preprocessing step to improve training efficiency and performance. Specifically, both the actuator states and the corresponding end-effector positions are normalized. This normalization ensures that all data points fall within a defined range, facilitating faster convergence during training and preventing potential issues caused by varying data scales. The normalization process also helps the network handle non-linearities more effectively.

After preprocessing, the dataset is divided into three parts: 70% of the data is used for training the network, 15% is reserved for validation, and the remaining 15% is set aside for testing. A random split of the data ensures that the network generalizes well to unseen data, reducing the risk of overfitting and improving overall model robustness.

The network is trained using mean squared error (MSE) as the loss function. MSE is commonly used in regression tasks, where it calculates the average squared difference between the predicted actuator adjustments and the actual values, thus serving as an indicator of model accuracy. The neural network undergoes 50 training epochs, allowing the model to refine its predictions gradually and reduce the error over time.

During the training process, the model iteratively adjusts its internal weights based on the error calculated at each step, minimizing the difference between its predictions and the actual target values. This iterative learning helps the network accurately map the complex relationships between the actuator states and the end-effector positions.

3.4. Evaluation and Testing

The training and evaluation results of the neural network are illustrated in Fig. 2(a) and Fig. 2(b), providing essential insights into its convergence, learning stability, and generalization.

Fig. 2(a) displays the gradient descent, learning rate (μ), and validation checks over 50 epochs. The gradient decreases consistently, reaching a very low value of (9.7792×10^{-7}) at the final epoch, indicating that the network's weights have effectively optimized. Simultaneously, the learning rate (μ) quickly stabilizes at (1×10^{-8}) , ensuring that the updates are appropriately small to prevent

oscillations near the optimal solution. Notably, no validation checks were triggered during the training, suggesting stable model behavior with no signs of overfitting or performance degradation.

Fig. 2(b) focuses on the MSE performance across the training, validation, and test phases. The initial MSE is high but rapidly decreases within the first few epochs, reflecting efficient learning. The validation and test curves closely follow the training curve, demonstrating the model's ability to generalize well to unseen data. At the end of 50 epochs, the best validation performance achieved is (1.451×10^{-6}) , confirming that the network provides accurate predictions with minimal error. Overall, the combination of low final gradient, steady Mu, and aligned performance across datasets indicates that the neural network has successfully learned the mapping from actuator states to end-effector positions without overfitting, thereby meeting the design goals.

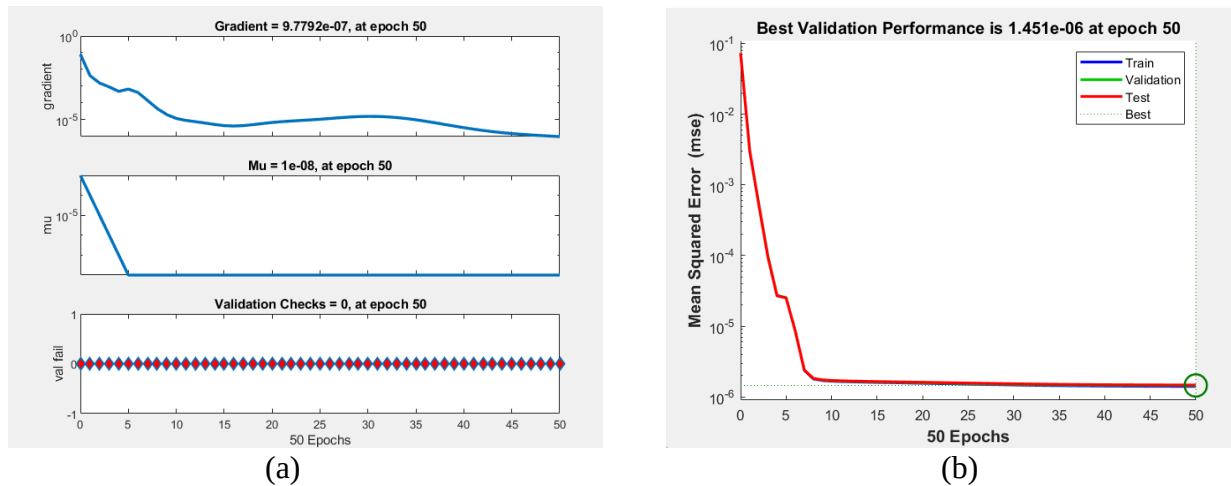


Fig 2. Performance Analysis of Neural Network Training. (a) Gradient and Mu Tracking Across Epochs; (b) MSE Performance Across Epochs. (Photo/Picture credit: Original).

4. Results and Analysis

The neural network-based controller was tested under two scenarios—first, maintaining a static end-effector position at a specific point in 3D space in Fig. 3(a), and second, following a predefined trajectory through multiple waypoints in Fig. 3(b). These scenarios provide insights into the system's ability to achieve precise control in both static and dynamic tasks. The results demonstrate that the controller is capable of adapting actuator lengths effectively to achieve the target positions in the workspace.

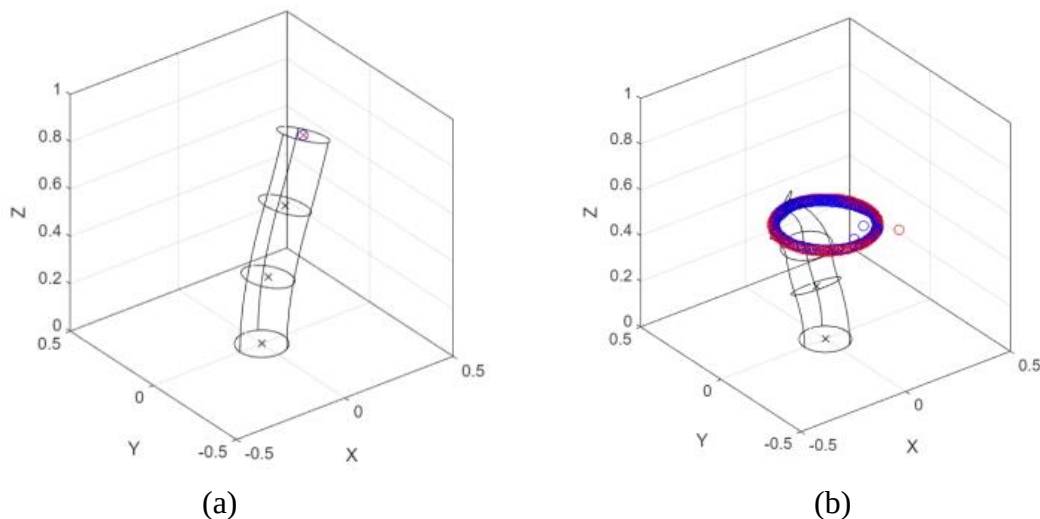


Fig 3. Test of the Controller. (a) Single Point Control; (b) Trajectory Tracking. (Photo/Picture credit: Original).

Fig. 3(a) shows the outcome of the single-point control test, where the end-effector (marked by the blue circle) aligns closely with the desired target position (marked by the X mark). The alignment in the 3D space along the X, Y, and Z axes demonstrates the high positional accuracy of the controller. This result indicates a minimal error between the predicted and actual positions, validating the network's capacity to translate actuator inputs into precise spatial configurations.

Fig. 3(b) illustrates the result of the trajectory tracking test. In this figure, the blue circle represents the actual trajectory followed by the system, while the red circle marks the target trajectory. The close alignment between the two paths demonstrates the controller's precision and stability during the tracking task. Although minor deviations are present, the overall tracking error is minimal, confirming the system's robustness in dynamic path-following scenarios. The controller effectively synchronized the actuator's movements with the desired trajectory, ensuring smooth and accurate control during operation.

Key metrics highlighted by these results include the range of motion achieved in all three spatial dimensions and the minimal deviations observed between theoretical predictions and simulated outcomes. The experiments suggest that the developed neural network effectively maps complex input-output relationships, demonstrating high tracking precision with minimal oscillations.

5. Limitations and Future Outlooks

The neural network-based controller developed in this study, though effective in simulation, has several limitations. One major limitation is the reliance on idealized simulation data, which does not fully account for uncertainties and disturbances that arise in real-world environments. Factors such as sensor noise, actuator delays, and mechanical wear could degrade the model's performance when applied to physical systems. Furthermore, the controller's precision depends on the initial choice of hyper parameters, such as learning rates and training epochs, which may require extensive manual tuning to achieve optimal performance. Another challenge lies in the computational cost of training neural networks, especially when scaled for more complex systems with higher degrees of freedom. Although the model demonstrated stability and convergence in a controlled environment, the transferability of these results to varying conditions remains uncertain. The lack of an explicit feedback mechanism for online adjustments also limits the adaptability of the controller in dynamic environments.

Looking ahead, several areas offer opportunities for future research and improvement. One promising direction is the integration of adaptive learning techniques, such as reinforcement learning, to enable the controller to adjust in real-time to unforeseen changes in its operating environment. Additionally, hybrid approaches combining neural networks with classical control methods may enhance both precision and robustness, leveraging the strengths of both paradigms. Expanding the dataset with real-world samples could improve generalization and minimize the performance gap between simulation and practical deployment. Future work could also explore multi-objective optimization techniques, balancing competing objectives such as energy efficiency and tracking accuracy. Moreover, efforts to reduce the model's computational requirements, possibly through model pruning or lightweight architecture, could make the approach more suitable for resource-constrained robotic platforms. Addressing these areas would enhance the controller's applicability in fields like soft robotics, surgical robotics, and autonomous manipulation tasks, further contributing to the advancement of intelligent control systems.

6. Conclusion

This study focused on developing a neural network-based controller to achieve precise motion control for a flexible robotic system. Using simulated data, the neural network accurately tracked both individual points and complex trajectories, demonstrating its capability to meet the desired control objectives. Results showed that the model maintained minimal tracking errors, with the mean

squared error converging close to zero, indicating high accuracy across training and validation phases. The comparison between actual and target trajectories confirmed the system's ability to generalize well and remain stable throughout operation.

The significance of this research lies in its potential application in real-world robotics, where precise and adaptive control is crucial. The study provides a solid foundation for future exploration of neural network-based control frameworks, particularly in flexible systems that require continuous adjustments to environmental dynamics. Future work will aim to improve noise handling, enhance robustness, and further expand the application scope.

References

- [1] Webster III R J, Jones B A. Design and kinematic modeling of constant curvature continuum robots: A review. *The International Journal of Robotics Research*, 2010, 29(13): 1661-1683.
- [2] Walker I D. Continuous backbone "continuum" robot manipulators. *International Scholarly Research Notices*, 2013, 2013(1): 726506.
- [3] Burgner-Kahrs J, Rucker D C, Choset H. Continuum robots for medical applications: A survey. *IEEE Transactions on Robotics*, 2015, 31(6): 1261-1280.
- [4] Abe A. Trajectory planning for flexible Cartesian robot manipulator by using artificial neural network: numerical simulation and experimental verification. *Robotica*, 2011, 29(5): 797-804.
- [5] Duriez C. Control of elastic soft robots based on real-time finite element method. 2013 IEEE international conference on robotics and automation. *IEEE*, 2013: 3982-3987.
- [6] Dubied M, Michelis M Y, Spielberg A, et al. Sim-to-real for soft robots using differentiable fem: Recipes for meshing, damping, and actuation. *IEEE Robotics and Automation Letters*, 2022, 7(2): 5015-5022.
- [7] Ranganathan A. The levenberg-marquardt algorithm. *Tutorial on LM algorithm*, 2004, 11(1): 101-110.
- [8] Moré J J. The Levenberg-Marquardt algorithm: implementation and theory. *Numerical analysis: proceedings of the biennial Conference held at Dundee, June 28–July 1, 1977*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006: 105-116.