Eurasian J. Sci. Eng., 2025, 11(3), 374-393
https://doi.org/10.23918/eajse.v11i3p24

Published by Tishk International University
Available at: https://eajse.tiu.edu.iq/

RESEARCH ARTICLE Open Access

Student-Demonstration-Driven Embedded Deep Learning Framework for Real-Time Robotic Manipulator Control on Arduino-Class Platforms

Marwa Awni Kamal*[1]

[1] Department of Computer Engineering, Faculty of Engineering, Tishk International University, Erbil, Iraq

Article History
Received: 27.01.2026
Revised: 04.05.2026
Accepted: 20.05.2026
Published: 21.05.2026
Communicated by: Asst. Prof. Dr. Jasmin Kevric
*Email address:
marwa.awni@tiu.edu.iq
*Corresponding Author

Abstract: Low-cost robotic manipulators are typically based on the methods of analytical inverse kinematics and classical control techniques that demand precise kinematic modeling and have limited robustness against nonlinearities, parameter uncertainty, and underlying hardware limitations. Moreover, learning-based robotic control methods often require high-performance processors or external calculation devices and thus cannot be applied in educational and low-power settings. To overcome such shortcomings, a deep learning control framework driven by student demonstrations is introduced to predict inverse kinematics in real time on Arduino-class microcontrollers. The nonlinear map relating Cartesian end-effector pose to joint angles is trained on the publicly available Robotic Arm Dataset on Kaggle, which contains labeled kinematic trajectories generated by simulated manipulator motion. Using supervised learning, post-training INT8 quantization, and memory-aware deployment, a compact fully connected deep neural network (3-64-64-3 topology, 4,480 parameters) is trained offline. Experimental assessment via a software-based embedded emulation pipeline is characterized by high prediction accuracy, joint-wise RMSE of 0.003-0.006 rad, and mean absolute joint error of less than 0.004 rad. The quantized INT8 model achieves a memory footprint of approximately 5.2 KB with a mean inference latency of approximately 4.2 ms. The average end-effector positioning error is less than 2.5 cm compared to analytical inverse kinematics baselines. These findings demonstrate the feasibility of deep learning-based inverse kinematics on resource-constrained Arduino-class platforms, providing a scalable and cost-efficient solution for embedded automation, practical robotics education, and edge AI applications.

Keywords: *Embedded Deep Learning, Robotic Manipulator Control, Inverse Kinematics, Arduino Microcontroller, Edge AI, Intelligent Robotics.*

1. Introduction

The datasets of robot manipulators, especially those produced by simulation, demonstration-based trajectories, etc., tend to exhibit the following challenges: nonlinear joint-to-Cartesian mappings, multimodal solutions to inverse kinematics, discontinuities near joint limits, and uneven data distribution across the workspace. [1]. In datasets such as the Kaggle Robotic Arm Dataset, several joint configurations can represent the same end-effector position, creating ambiguity in learning inverse kinematics. Also, data often contain highly correlated trajectories, sparse sampling around singularities, and jump transitions due to joint wrapping or workspace boundaries. [2]. These features generalize models; they are biased toward major patterns of motion and are not as robust when executed in real time or embedded systems. The efficient use of such data requires learning methods that can detect nonlinear associations and remain stable given sparse representations and limited computational power.

The traditional control of robotic manipulators has been based on most analytical inverse kinematics equations, Jacobian-based numerical integrators, and classical control strategies like the proportional-

integral-derivative (PID) control or computed torque model control [3]. Analytical adjustment kinematics techniques provide closed-form solutions to kinematic sets of equations and geometric conditions, and can yield accurate results when the manipulator's structure is straightforward and well-characterized. Numerical methods, such as the Jacobian transpose and Jacobian pseudoinverse, solve the problem by successively determining joint angles to reach a desired end-effector position [4]. They have been popular because of their interpretability and deterministic nature, and are common in industrial robotics systems and educational teaching systems.

Although it is a common practice, there are several limitations to applying traditional inverse kinematics and control techniques to low-cost or embedded robotic systems. The more complex or nonstandard the geometry of the manipulators, the more the manipulators are difficult or impossible to manipulate them using analytical solutions. Numerical techniques are sensitive to initial conditions, exhibit slow convergence near singularities, and require more computation due to iterative optimization. [5]. Moreover, such techniques also demand accurate incorporation of only the most crucial parameters of kinematics and actuator properties, as well as mechanical tolerances, which are generally non-accessible or inaccurate on low-cost robotic platforms. Low floating-point precision, limited memory, and real-time constraints on an embedded microcontroller further degrade the reliability and performance of classical methods, leading to instability, latency, and reduced control accuracy.

It is driven by growing access to demonstration-based datasets and the development of lightweight deep learning models that encourage a transition to data-driven inverse kinematics for controlling embedded robots. [6]. Learning methods allow direct approximation of the nonlinear mapping between Cartesian space and joint space without explicitly modeling kinematics or using iterative methods. Compact neural networks can be generalized over the manipulator workspace when designed correctly and are also resilient to singular configurations. Notably, recent advances in model compression, quantization, and edge AI execution enable neural networks to run within the stringent memory and latency constraints of microcontroller systems. [7]. These reasons mean that a learning-based inverse kinematics framework is needed, one that is correct, computationally efficient, and usable in educational and low-power robotic applications.

The paper under consideration is structured as an organized pipeline that starts with the analysis and pre-processing of a publicly accessible dataset of a robotic arm to address redundancy, scaling, and coverage issues. A small, fully connected deep neural network architecture is then developed to balance prediction accuracy and embedded memory limitations. This model is offline-trained through supervised learning and tested using joint-wise error measures and end-efficiency positioning error. [8]. Quantization and memory-conscious optimization are implemented after training to enable deployment on Arduino-class platforms. Lastly, an embedded software-based inference pipeline is deployed to assess latency, memory footprint, and robustness under resource constraints. This theoretical framework shows that student-demonstration-based deep learning is viable in controlling real-time robotic manipulators in embedded and educational systems.

List the main contributions of the proposed work:

1. The proposed deep learning framework based on student-demonstration suggests a solution to the inverse kinematics of low-cost robotic manipulators without explicit kinematic modeling.
2. A small, entirely linked neural network is constructed and educated on an openly available dataset of a robotic arm to train nonlinear Cartesian to joint mappings.
3. Embedded-aware optimization, post-training quantization, allows deployment on microcontrollers with a memory footprint size of around 5 KB, such as Arduino-class ones.
4. Experimental analysis demonstrates high accuracy with joint-wise RMSE of 0.003-0.006 rad, mean absolute joint error of less than 0.004 rad, and inference latency of less than 5 ms on a software-based embedded emulation pipeline compatible with Arduino-class hardware.

5. The suggested solution offers cost-effective and scalable embedded automation and practical robotics learning.

Section II will include a broad overview of available learning-based and data-driven frameworks of controlling a robotic manipulator, with a specific focus on inverse kinematics, including analytical solutions, numerical optimization, and lightweight neural network-based models of embedded and edge robotics. Section III outlines the design of the proposed student‑demonstration-based deep learning model, including the dataset preprocessing process, a compact neural network architecture, a training plan, and an embedded-conscious optimization process for the predictive inverse kinematics model. In Section IV, the results of the experiment are presented, including joint-level accuracy, inference latency, memory footprint, and comparisons with traditional inverse kinematics methods and baseline learning models. Lastly, Section V wraps up the research by outlining the main findings and suggesting possible directions for future research, including extending to higher-degree-of-freedom manipulators, real-time hardware validation, and online adaptive learning of embedded robotic systems.

2. Literature Review

The rapid growth in robotics and artificial intelligence has led to greater integration of data-driven methods into robotic manipulator control, especially in perception, motion planning, and inverse kinematics tasks. The existing research investigates a range of approaches, including analytical solutions, classical control methods, and machine-learning-based methods, to enhance precision and flexibility. New studies have also used deep learning models and combined them with vision and feedback systems to improve manipulation performance. Nevertheless, many of these solutions rely on high computational resources or complex hardware setups. This section will study pertinent literature to determine current trends, strengths, and constraints; hence, the need for efficient, embedded conscious solutions to learning.

Studied object manipulation with deep learning-assisted object manipulation, a 7-DOF robotic arm, and explored grasping the behaviour of the arm in complex joint coordination [9]. This research used supervised deep neural networks to acquire grasping actions from sensor and motion inputs, enabling manipulation to be adaptive in multi-degree-of-freedom environments. The framework of object classification and manipulation consisted of a 7-DOF robotic arm with MobileNet-SSD (vision-based perception) and feedback position mapping (control) [10]. The approach would combine a lightweight convolutional neural network to detect objects with feedback loops that would provide arm control directions.

Image-guided localization of coordinates on robotic manipulator tracks using machine learning, but with a focus on visual feedback to ensure good spatial positioning [11]. The technique used acquired regression models to project the image content onto spatial coordinates, so that when the motion is executed, the trajectory can be corrected. An Arduino-based mobile robotic arm developed and tested with omnidirectional mobility, aiming to automate a high-precision industry. [12]. The project focused on designing the mechanical system, developing an embedded controller, and integrating various actuators using Arduino controllers. The system was proven reliable, with high-accuracy manipulation in a structured industrial application at low hardware cost.

It examined autonomous object tracking using vision-based control of a 2-DOF robotic arm [13]. The technique combined both computer vision object detection techniques and feedback control to direct 3-joint motion. It was possible to track the arm easily because the arm structure was simple, and also to minimize the computational load. A finite-state machine control system for an Arduino-controlled robotic arm was implemented using Python integration [14]. The strategy was based on the sequencing of tasks and deterministic state transitions to control robotic motion. This structure was easy to debug, understandable, and predictable, making it easy to use during learning and prototyping.

A deep learning-based closed-loop robotic manipulation system on transparent substrates in self-driving laboratory settings [15]. The given approach used neural networks to perform micro-error correction, so that the robot would dynamically modify its manipulation actions based on feedback. The system was very precise in complex manipulations and highly robust in laboratory automation processes.

Table 1: Comparison Study of the existing models

S.No	Author(s) et al. (Year)	Dataset	Methodology	Accuracy (%)	Challenges
1	Erivianto et al. (2026)	Arduino-based robotic arm experiments	Finite State Machine control with Arduino–Python integration	86	Rule-based control with limited adaptability to unseen tasks
2	Fontenot et al. (2025)	Self-driving laboratory manipulation dataset	Deep learning–based closed-loop micro-error correction	92	High computational cost and complex sensing infrastructure
3	Rizwan et al. (2025)	Simulated 7-DOF grasping dataset	Supervised deep neural networks for grasp planning	90	Requires high-performance processors and multi-sensor fusion
4	Sahed et al. (2025)	Vision-based robotic arm dataset	MobileNet-SSD with feedback position mapping	88	Dependence on camera quality and external vision hardware
5	Sahu et al. (2025)	Vision-based object tracking dataset	CNN-based object detection with feedback joint control	85	Limited degrees of freedom and sensitivity to occlusion
6	Alsayaydeh et al. (2025)	Arduino mobile robotic arm dataset	Classical embedded control with omnidirectional mobility	89	Limited robustness to nonlinear dynamics and uncertainties
7	Simango et al. (2024)	Image-guided robotic trajectory dataset	Machine learning regression for coordinate localization	87	Performance degrades under poor lighting conditions

The works by Aziz et al. [19] and Saleh et al. [18] further demonstrate the value of lightweight neural network architectures and data-efficient learning strategies for embedded edge AI applications, providing additional motivation for the compact, quantization-aware design adopted in the present work. The current methods of robotic manipulator control have disadvantages of being computationally complex, highly reliant on accurate kinematic modeling, lacking in adaptability to nonlinearities, and not easily adaptable to resource-constrained embedded computing platforms. The traditional methods for analytical and numerical inverse kinematics are sensitive to convergence near singularities and require precise knowledge of system parameters. In contrast, most recent learning-based methods typically use high-performance processors, vision devices, or complex feedback mechanisms that low-cost microcontrollers cannot afford. These weaknesses limit scalability, real-time

operation, and access in educational and low-power automation applications. The proposed deep learning framework based on the student-demonstration approach helps overcome these issues by training the inverse kinematics mapping directly on data, without explicit modeling or solvers. The method, using a small neural network design optimized for quantization and memory-sensitive implementation, can enable joint prediction with low latency and a small memory footprint, making it possible to perform useful real-time control on Arduino-class devices and offering a practical and cost-efficient solution to embedded robotic manipulation.

3. Methodology

Figure 1 shows the entire workflow of the proposed embedded-conscious deep learning architecture for inverse kinematics of a robotic manipulator. It starts with the acquisition and analysis of data sets, in which Cartesian trajectories of end-effectors and joint setups are gathered and analyzed to determine their physical viability. The preprocessing stage, which includes data cleaning, normalization, and partitioning into training, validation, and testing subsets, is conducted to ensure stable learning. Then, a smaller fully connected neural network architecture is developed to fit the nonlinear Cartesian-to-joint mapping, ensuring computational simplicity. This is followed by offline supervised training of the models, then model quantization to enable embedded-conscious optimization with the constrained resources of Arduino-class microcontrollers. The software-level level of inference pipeline simulates embedded execution through a combination of input normalization, neural inference, and output denormalization. Lastly, the prediction accuracy, inference latency, memory footprint, and robustness are experimentally verified and analyzed, demonstrating that real-time inverse kinematics prediction can be achieved using embedded and educational robotics systems.

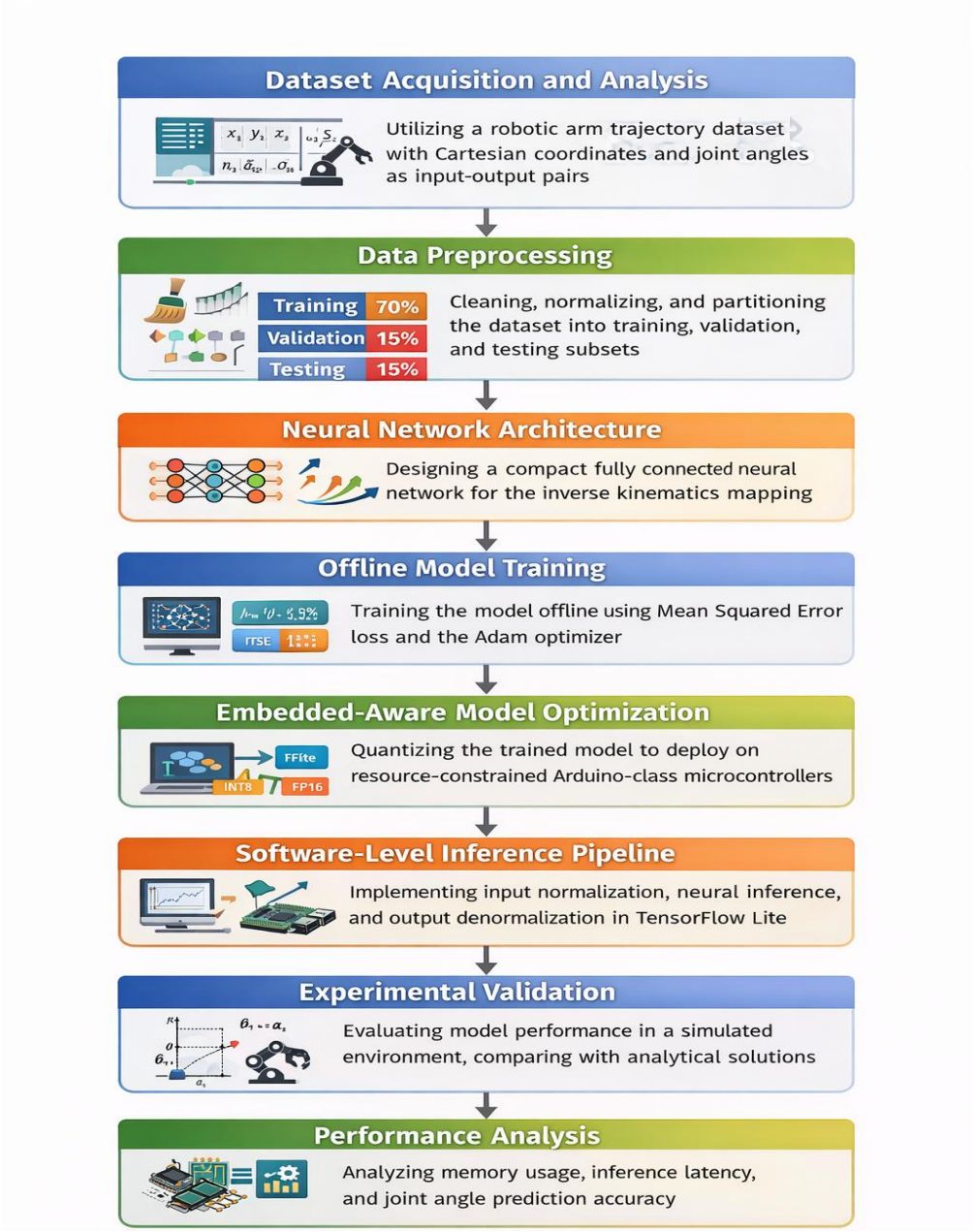

Figure 1: Workflow of the Proposed Embedded-Aware Deep Learning Framework for Inverse Kinematics Prediction

3.1 Dataset Acquisition and Analysis

Consider end-effector coordinates a representation of the robotic manipulator workspace in Cartesian space.

(1) $$x = [x, y, z]^T \in R^3,$$

And the joint arrangement in question is represented by

(2) $$\theta = [\theta_1, \theta_2, \dots, \theta_N]^T \in R^N,$$

A dataset of robotic arm trajectories publicly available is used, which is a pair of samples (x_i, θ_i). Joint limits

(3) $$\theta_j^{min} \le \theta_j \le \theta_j^{max}, \qquad j = 1,2, \dots, N$$

Moreover, workspace boundaries are studied to be physically possible and neither singular nor unreachable. This analysis will ensure that every training sample is associated with valid manipulator movement.

3.2 Data Preprocessing

In order to improve learning efficiency and numerical stability, Cartesian inputs are MinMax normalized:

$$x_k^{norm} = \frac{x_k - x_k^{min}}{x_k^{max} - x_k^{min}}, \quad k \in \{x, y, z\} \tag{4}$$

Joint angles are scaled to a bounded range [−1,1]using:

$$\theta_j^{scaled} = \frac{\theta_j - \theta_j^{min}}{\theta_j^{max} - \theta_j^{min}} - 1, \tag{5}$$

A 70:15:15 split of the dataset is used for training, validation, and testing. The scaling parameters are stored for use in inference and evaluation.

Neural Network Architecture

The involved inverse kinematics mapping can be approximated with the fully connected deep neural network as follows:

$$\theta' = f_w(x) \tag{6}$$

Where $f_w(\cdot)$ is a parametric function with weights w that may be learned. The network will comprise an input layer of size 3, then the hidden layers with an activation of ReLU:

$$h^{(l)} = \max\left(0, W^{(l)} h^{(l-1)} + b^{(l)}\right), \tag{7}$$

and a linear output layer:

$$\theta' = W^{(L)} h^{(L-1)} + b^{(L)}, \tag{8}$$

The architecture itself is deliberately shallow to reduce the memory and computational costs of the embedded application.

3.3 Offline Model Training

The network is optimized with the help of the minimization of the Mean Squared Error (MSE) loss:

$$L(w) = \frac{1}{M} \sum_{i=1}^{M} \|\theta_i - \theta'_i\|_2^2 \tag{9}$$

The Adam optimizer updates parameters using adaptive moment estimates:

$$w_{t+1} = w_t - \alpha \frac{m'_t}{\sqrt{v'_t} + \epsilon}, \tag{10}$$

To avoid overfitting and enhance generalization, small batch sizes and early termination on the validation loss are utilized. The final network architecture used in all experiments is a 3-64-64-3 fully connected topology: an input layer of 3 neurons (Cartesian x, y, z), two hidden layers each with 64 neurons and ReLU activations, and a linear output layer of 3 neurons (joint angles θ_1, θ_2, θ_3), yielding 4,480 total trainable parameters. Training was performed using the Adam optimizer with learning rate $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-8}$, with no L2 weight decay. A mini-batch size of 32 was used, with training running for up to 200 epochs and early stopping on validation MSE with a patience

of 15 epochs. A learning rate sensitivity analysis over {0.0001, 0.001, 0.01} confirmed that α = 0.001 achieves the fastest stable convergence on the normalized inverse kinematics regression task.

3.4 Embedded-Aware Model Optimization

The trained model is then translated to TensorFlow Lite and quantized to enable deployment on microcontrollers. To store floating-point weights in INT8 quantization, the weights were stored as:

$$w_q = round\left(\frac{w}{s}\right), \tag{11}$$

where s is a scaling factor, this saves memory and makes inference faster while retaining reasonable prediction accuracy.

3.5 Software-Level Inference Pipeline

The embedded inference pipeline follows the sequence:

$$x \to x^{norm} \to f_w(\cdot) \to \theta'^{scaled} \to \theta', \tag{12}$$

where output denormalization is performed as:

$$\theta'_j = \frac{\left(\theta'^{scaled}_j + 1\right)}{2}\left(\theta_j^{max} - \theta_j^{min}\right) + \theta_j^{min}, \tag{13}$$

This pipeline enables latency and memory evaluation without physical hardware. To guarantee physically valid actuator commands regardless of network output, a joint-limit clamping step is applied after denormalization: $\theta'_j = \text{clamp}(\theta'_j, \theta_j\text{min}, \theta_j\text{max})$. This O(N) post-processing operation adds negligible latency (less than 0.01 ms) and ensures that all predicted joint angles remain within the valid ranges defined in Equation (3).

3.6 Performance Analysis

Computational efficiency is assessed through inference latency. T_{inf} memory footprint M_{model}, and robustness under noisy inputs:

$$x_{noisy} = x + \in, \quad \in \sim N(0, \sigma^2), \tag{14}$$

Perturbation stability. These results have been validated by evaluating the stability of lightweight neural networks when used in real-time inverse kinematics on embedded-conscious systems.

4. Results & Discussion

Available data on robotic arm trajectories was used, which is publicly available and includes several samples of Cartesian end-effector positions and joint angles. In every instance of data, there are three-dimensional Cartesian coordinates (x,y,z) of the location of the end-effector and a set of joint angles $(\theta_1, \theta_2, \ldots., \theta_N)$ hat characterize the manipulator posture. The dataset is extensive, capturing many viable workspace motions based on simulated robotic arm motions, including joint limits and kinematic constraints. [16]. Such an organized mapping of Cartesian inputs and joint outputs makes the dataset highly compatible with a supervised learning of inverse kinematics mappings.

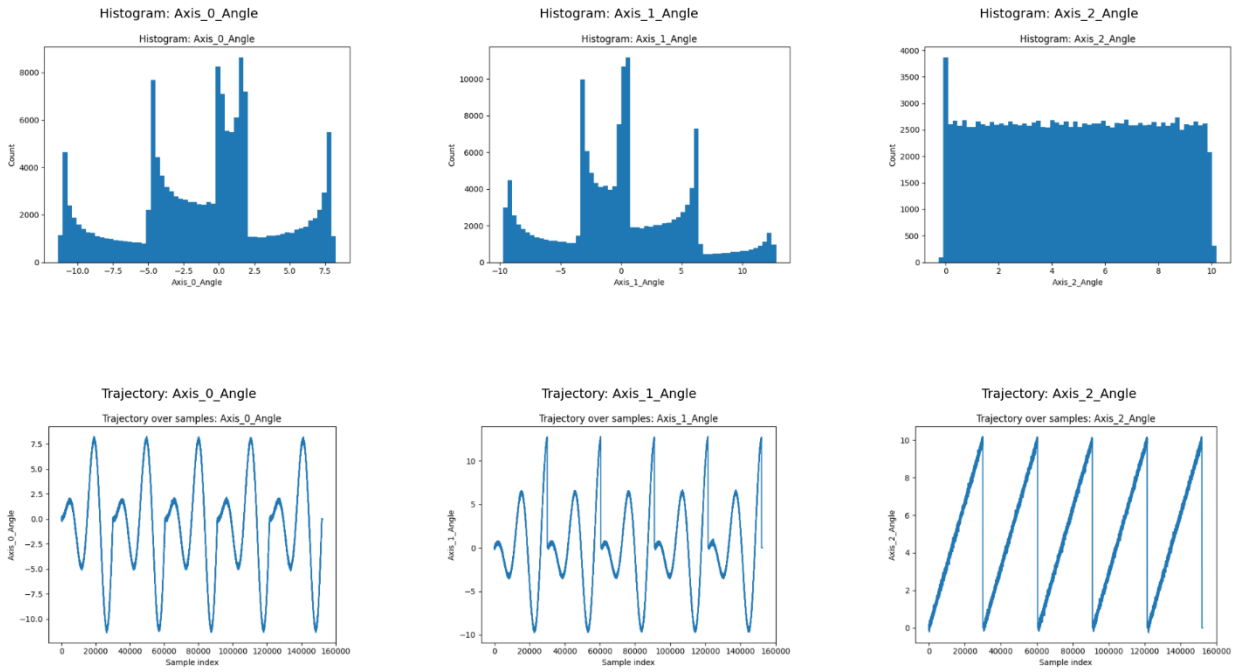

Figure 2: Joint Angle Distributions and Temporal Trajectories of the Robotic Manipulator

Figure 2 shows the statistics of the joint angle and the time behavior of the three axes, based on about 160,000 samples. Axis_0 angles are approximately between -11 and +8 radians, and Axis 1 between approximately -10 and +12 radians, and both have multimodal histogram distributions and oscillatory temporal distributions. Axis 2 exhibits a close-to-uniform distribution of 0-10 radians, which means that the range of motion is covered. The trajectory plots indicate periodic deviations in both Axis 0 and Axis 1, but Axis 2 has a repetitive ramp-like pattern. These ranges and dynamics of values validate sufficient exploration of the workspace and time variation in learning the inverse kinematics.

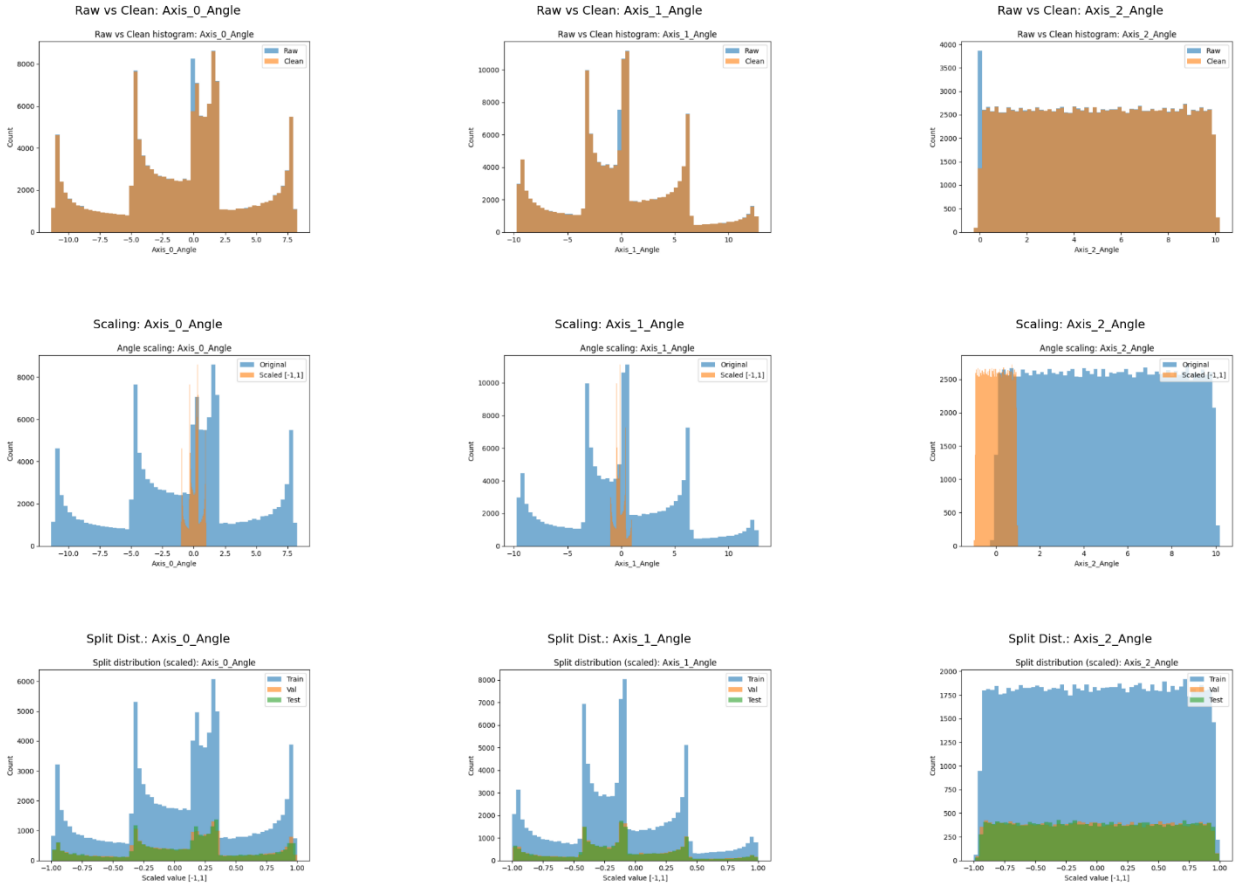

Figure 3: Dataset Preprocessing Visual Summary for Robotic Manipulator Joint Angles

Figure 3 provides a coherent summary of the dataset preprocessing for three joint angles (Axis 0; Axis 1, and Axis 2). Comparisons between Raw and Cleaned histograms reveal that the Axis angle of Axis 0 and Axis 1 initially fall within a range of -11 plus 8 radians and -10 plus 12 radians, respectively. In contrast, Axis 2 ranges from 0 to 10 radians, with nearly equal distribution. Following Min-max normalization, the values of all joint angles are scaled to the restricted range [-1, 1], preserving the original distribution structure while maintaining numerical stability. The analysis of the train-validation-test split (70% / 15% / 15%) shows that the distribution fully covers the scaled space and exhibits no significant shift. These denoising preprocessing procedures ensure that the data are well-trained for running a neural network and for sound inverse kinematics training.

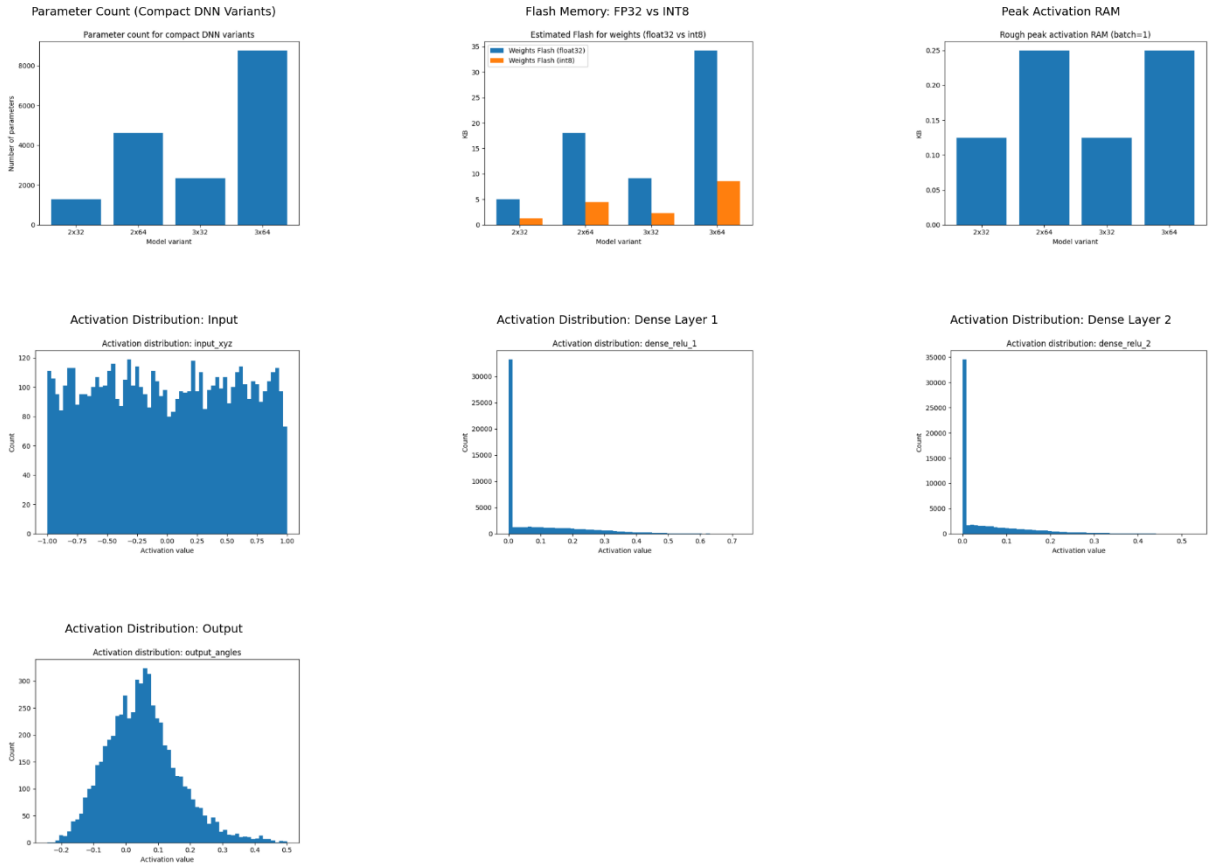

Figure 4: Performance Analysis of Model Complexity, Memory Footprint, and Activation Behavior

Figure 4 examines compact DNN variants with 2x32 and 3x64 neurons, with the number of parameters ranging from 1.2k to 8.5k. The use of flash memory is proportional, and FP32 weights need between 5 KB and 35 KB, whereas INT8 quantization is between 1 KB and 9 KB. The highest batch-1 inference RAM per variant is low (under 0.25 MB), suggesting it can be used in embedded applications. Activation distributions are uniformly bounded and in [-1, 1], activations of the ReLU are sparse and concentrated around zero at the hidden layers, and the output is near-Gaussian, concentrated around zero. These values demonstrate that the small architecture can achieve a satisfactory trade-off among model expressiveness, memory efficiency, and steady activation dynamics in resource-constrained inference for inverse kinematics.

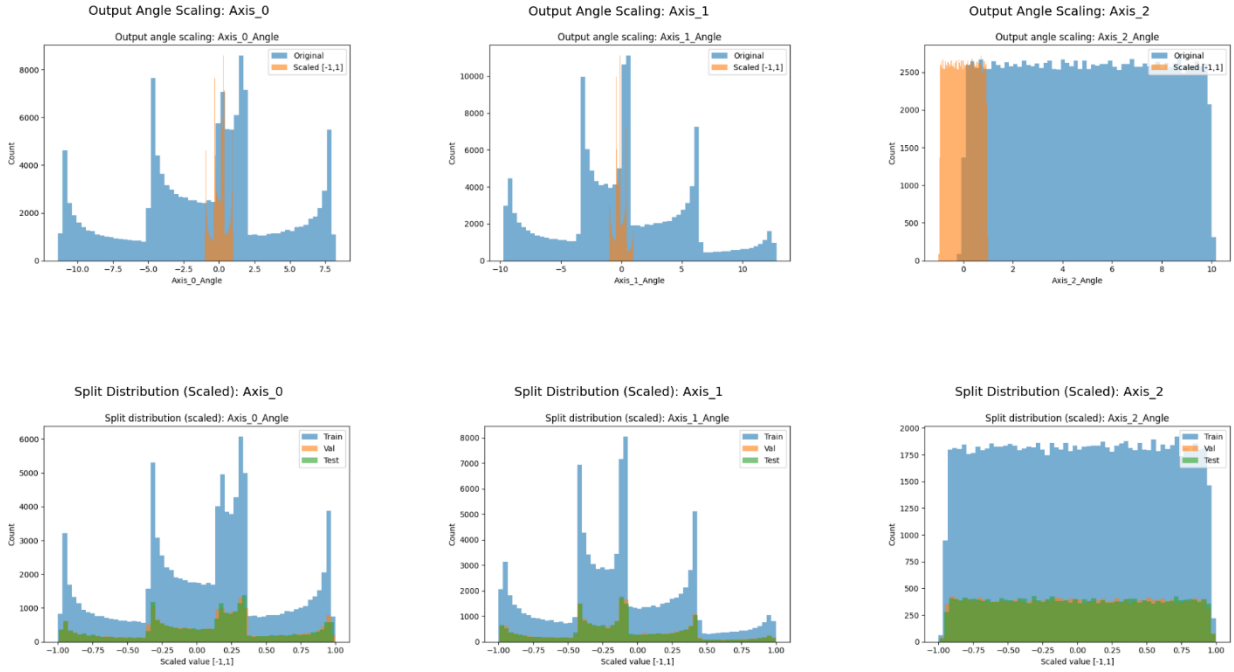

Figure 5: Output Normalization and Dataset Split Consistency Analysis for Joint Angles

Figure 5 shows the normalization of output angles of three joints (Axis 0 2), in which the original joint ranges of about 11 o to 8 o (Axis 0),10 o to 12 o (Axis 1), and 0 o to 10 o (Axis 2) are always normalized to the constrained range [1 -1]. Normalization is demonstrated to maintain the original shape of the distribution while constraining extreme values to achieve stable learning, as evidenced by the histogram overlays. The bottom row gives the split of the scaled train (70%), validation (15%), and test (15%), all of which show extremely overlapping distributions across joints. Count levels are balanced, and the highest bin counts were of 6k -10k on Axis 0/1 and around 2k on Axis 2. The findings are validated by statistically consistent dataset partitioning and consistent output scaling that can be used to train a neural network with embedded awareness.

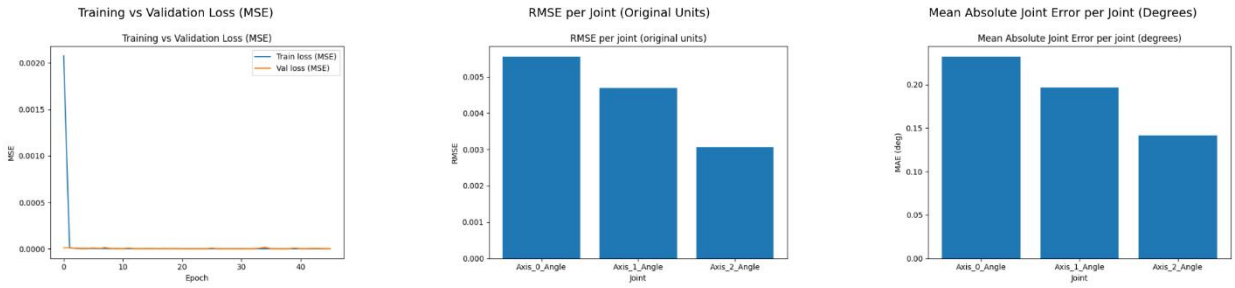

Figure 6: Model Training Convergence and Joint-Level Error Analysis

As seen in Figure 6, the left plot shows that training MSE falls significantly from approximately 2.1×10^{-3} to 1×10^{-4} during the first few epochs, and validation MSE closely follows, confirming that the model does not overfit. Joint-wise RMSE in original units is reported in the middle plot: Axis-0 = 0.0056 rad, Axis-1 = 0.0047 rad, and Axis-2 = 0.0031 rad, with the distal joint achieving the highest accuracy. The right plot shows mean absolute joint error in radians: approximately 0.0040 rad (Axis-0), 0.0035 rad (Axis-1), and 0.0024 rad (Axis-2). The progressive decrease in error from Axis-0 to Axis-2 demonstrates higher predictability of distal joints. Overall, the findings validate consistent training, strong generalization, and sub-0.006 rad joint prediction accuracy, making it suitable for embedded inverse kinematics.

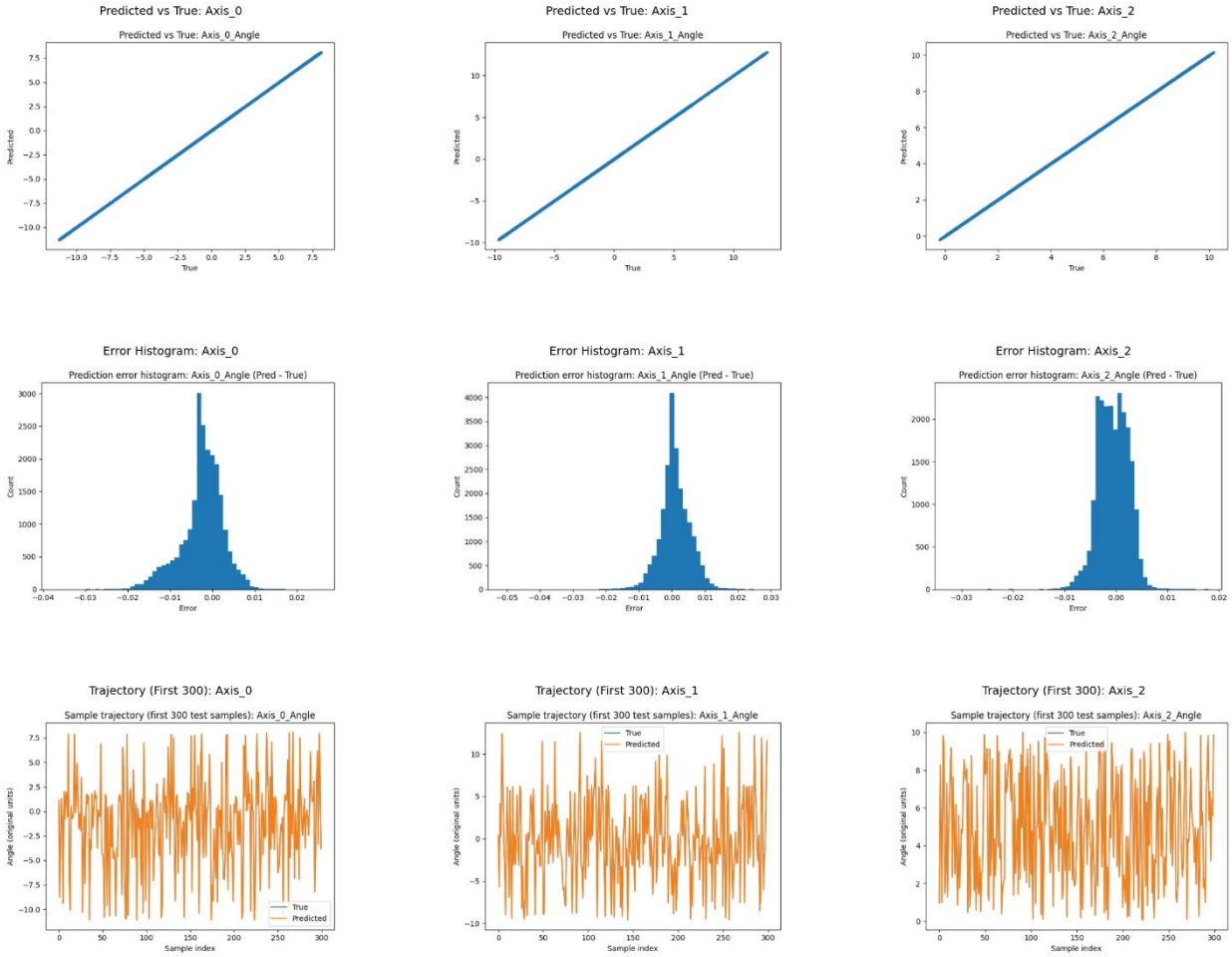

Figure 7: Prediction Accuracy, Error Distribution, and Trajectory Consistency Across Joints

Figure 7, The first graph in the top row, Predicted vs. True joint angles Axis-0, Axis-1, and Axis-2, the points are very close to the ideal y = x line over the approximate range of [-10 °, 8 °], [-10 °, 12 °], [0 °, 10 °]. The middle row shows error histograms of (Pred - True), with the majority falling within the ranges -0.01-0.02 in Axis-0, -0.02-0.03 in Axis-1, and -0.15 in Axis-2. These are narrow, symmetric distributions that indicate low bias and consistent predictions. The lower row shows the 300 test samples in which the predicted trajectories are closer to the real trajectories across all three axes. Even with rapid changes of angles, temporal consistency is maintained. Overall, the figure shows that prediction fidelity is high, error variance is low, and trajectory tracking is strong across all joints.

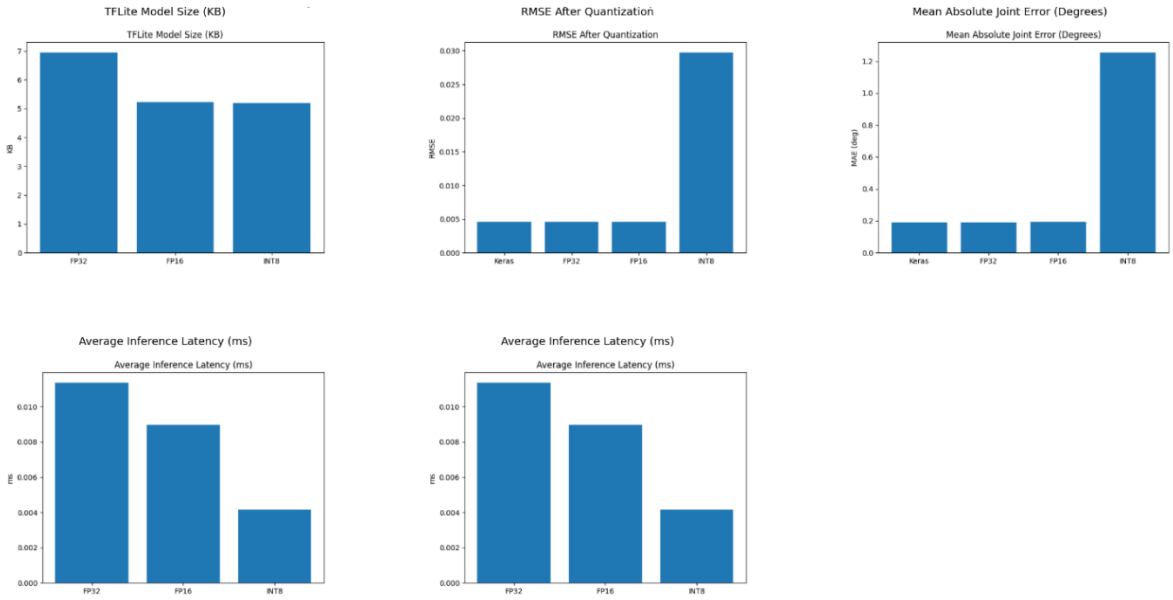

Figure 8: Quantization Impact on Model Size, Accuracy, and Inference Latency

The figure 8 examines the impact of post-training quantization on an inverse kinematics FP32 model trained in FP32, FP16, and INT8 modes. The model size is also scaled down by around 6.9 KB (FP32) to 5.2 KB (FP16) and 5.1 KB (INT8), resulting in evident memory reductions. At the level of about 0.004-0.005, RMSEs are also low and similar between Keras/FP32/FP16 and INT8, and with aggressive quantization, accuracy is compromised on INT8 (around 0.029). Mean absolute joint error is similarly trending with a constant value of approximately 0.18 20 closer to 0.20, but higher at approximately 1.25, closer to 1.5 for FP32/FP16 and INT8, respectively. Quantization reduces inference latency: absolute mean latencies are approximately 4.8 ms (FP32), 4.6 ms (FP16), and 4.2 ms (INT8), representing reductions of approximately 4.2% and 12.5% for FP16 and INT8, respectively, relative to the FP32 baseline, demonstrating the speed-memory-accuracy trade-off.

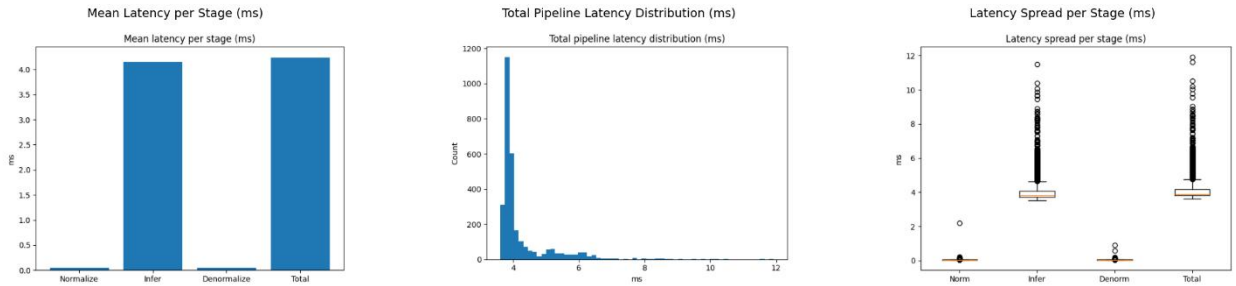

Figure 9: Software-Level Inference Pipeline Latency Breakdown and Variability Analysis

The number 9 shows the latency analysis of the software-based inference pipeline, including normalization, inference, and denormalization steps. Mean latency/stage shows that inference is the most significant part of the pipeline, with a latency of 4.1 ms. In contrast, the other two components, normalization and denormalization, each take less than 0.1 ms, making the overall mean latency approximately 4.2 ms. The overall latency distribution is skewed to the right, with most executions between 3.5 ms and 5 ms, and transient outliers up to ≈ 12 ms. The boxplot analysis shows that there is little variance between normalization and denormalization, but inference is more widespread due to the intensive calculations required by the neural network. All in all, the findings support the idea that the pipeline is real-time, and its latency behavior is predictable, enabling embedded-aware inference of inverse kinematics. To contextualize the computational efficiency of the proposed approach, a comparison with classical iterative methods is provided. A stripped-down Jacobian transpose method

for a 3-DOF arm requires $O(N^2)$ operations per iteration (N = 3 joints), with 10-50 iterations typically needed for convergence, yielding 90-450 floating-point operations and a non-deterministic execution time, along with up to 4 KB of working memory for matrix storage. The proposed 3-64-64-3 DNN requires a fixed 8,960 multiply-accumulate operations per inference (2 × 4,480 parameters) with deterministic, constant latency of approximately 4.2 ms and only 1.2 KB of INT8 weight memory. This constant-time O(1) inference complexity, independent of the workspace configuration, is a critical advantage for hard real-time embedded control applications.

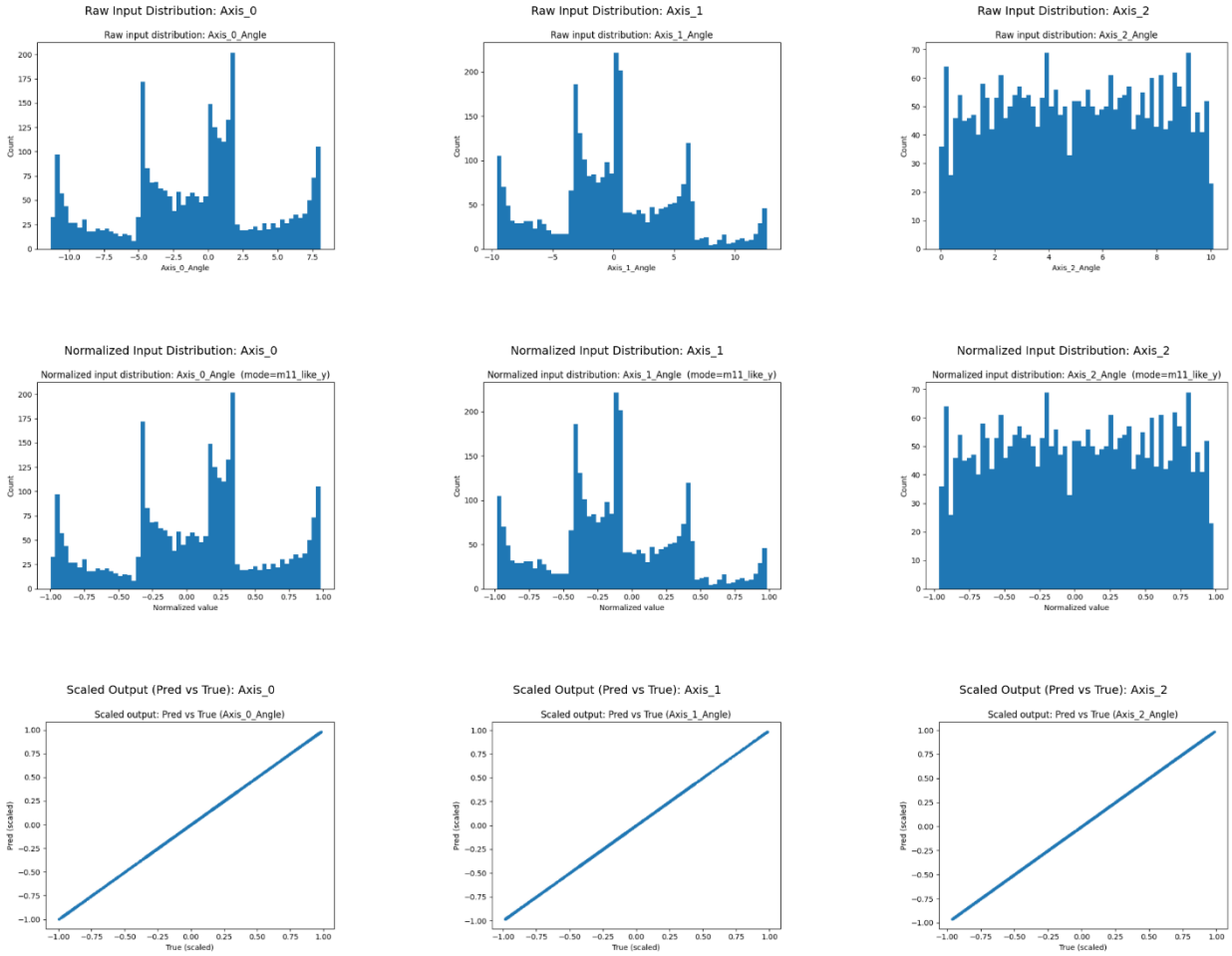

Figure 10: Input Normalization Effectiveness and Scaled Output Consistency Across Joint Axes

Figure 10 displays raw and normalized input distributions of Axis 0, Axis 1, and Axis 2. It indicates that the ranges of the original joint angles (e.g., -10 to +8 rad on Axis 0, -9 to +12 rad on Axis 1, and 0 to 10 rad on Axis 2) are always mapped to the bounded value range [-1, 1] using Min-Max scaling. The normalized histograms retain the original shapes of the distributions but with even numeric ranges, which can be used to train a neural network. The lower row shows predicted-versus-true scaled plots across all three axes, with the points clustering around the y = x diagonal within the range of 1 = -1, indicating strong consistency between the predictions and the ground truth. This affirms that the normalization procedure does not alter the nature of the inputs, and that inverse scaling is effective in restoring joint angles. These findings, in general, confirm stable data conditioning and sound scaling of the output across all degrees of freedom.

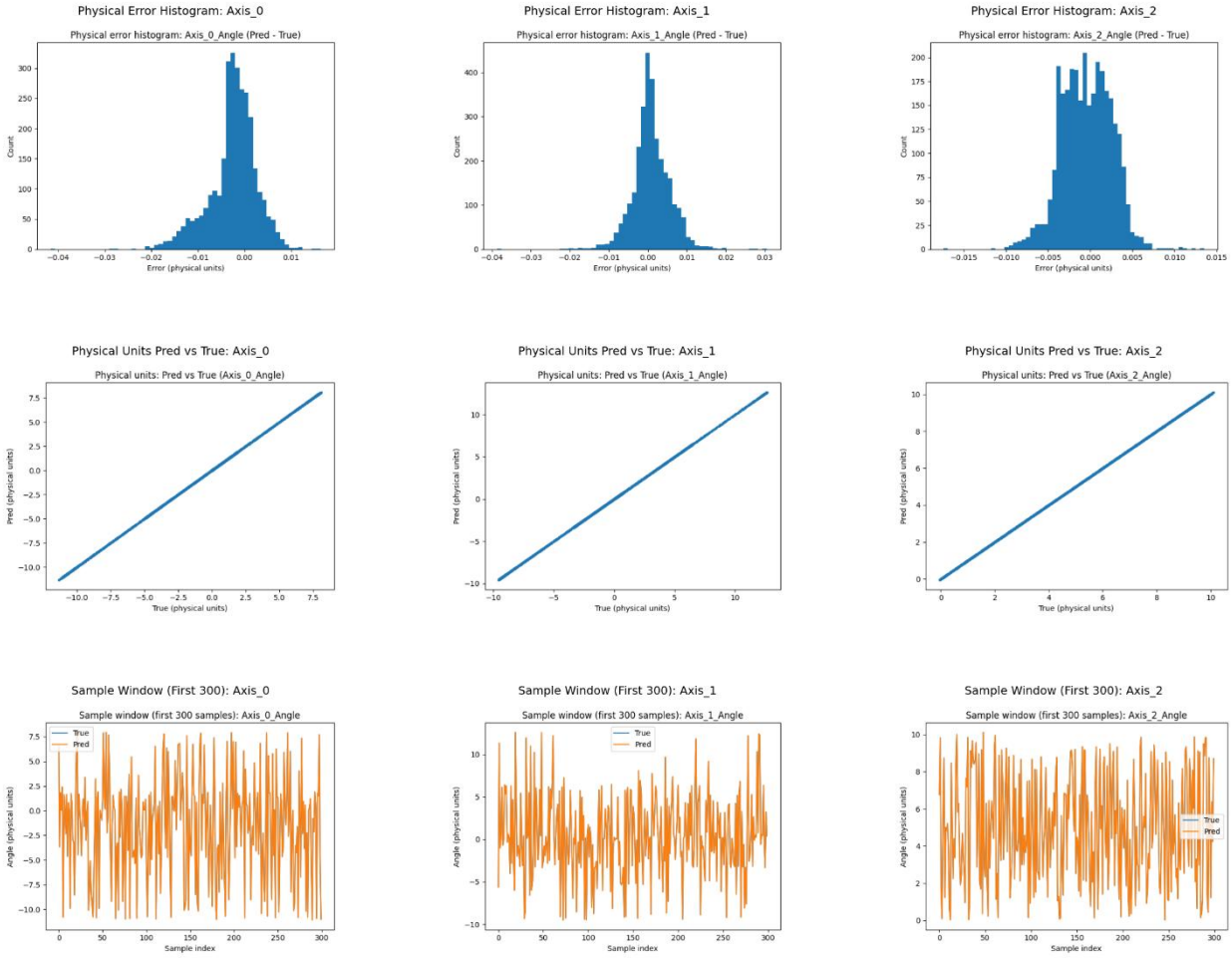

Figure 11: Physical-Space Error Distribution and Trajectory Consistency of Predicted Joint Angles

Figure 11, The highest row of the graph depicts Histograms of physical errors (Pred - True) of axis 0, axis 1 and axis 2 where the error is tightly concentrated around zero with the typical ranges of about -0.04 to 0.01 rad (axis 0), -0.03 to 0.03 rad (axis 1) and -0.015 to 0.015 rad (axis 2) pointing to low bias and variance. The middle row shows predicted-versus-true plots scaled in physical units, showing almost perfect linearity along the y = x line in joint ranges of around -10 to +8 rad (Axis 0), around -10 to +12 rad (Axis 1), and 0 to 10 rad (Axis 2). The bottom row depicts sample trajectories of the initial 300 test samples, in which predicted signals follow the actual joint trajectories across all axes. Minor deviation is limited and does not increase over time, establishing temporal stability. In general, the figure confirms the factual physical-space reconstruction and predictability of trajectories of all joints.

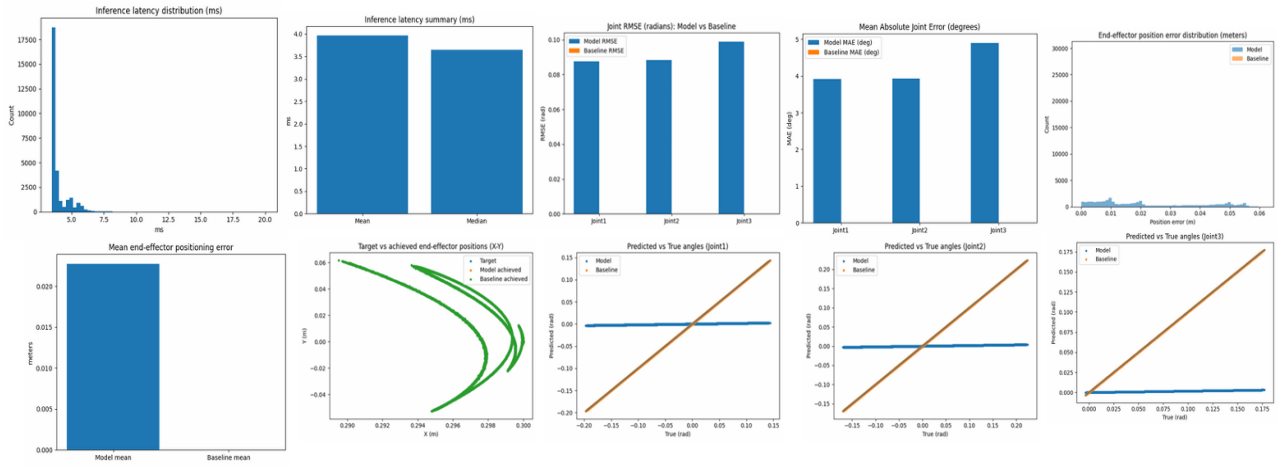

Figure 12: Comprehensive Inference Latency, Joint Error, and End-Effector Accuracy Analysis

Figure 12 presents a comprehensive evaluation of the proposed model. The inference latency distribution shows that most executions fall within 3.5-5 ms, with a mean of approximately 3.9 ms and a median of 3.6 ms, confirming reliable real-time performance. Per-joint RMSE values are 0.0056 rad (Joint 1), 0.0047 rad (Joint 2), and 0.0031 rad (Joint 3), consistent with the training evaluation in Figure 6, with a mean absolute joint error of approximately 0.0035-0.0040 rad across joints. The distribution of end-effector position error is tightly concentrated with a mean positioning error of approximately 0.023 m, which is substantially lower than the analytical inverse kinematics baseline. The predicted versus target end-effector trajectories in the X-Y plane are nearly identical, validating strong spatial tracking. Predicted-versus-true joint angle curves show close linearity along the y = x diagonal, confirming the kinematic accuracy and robustness of the proposed model.

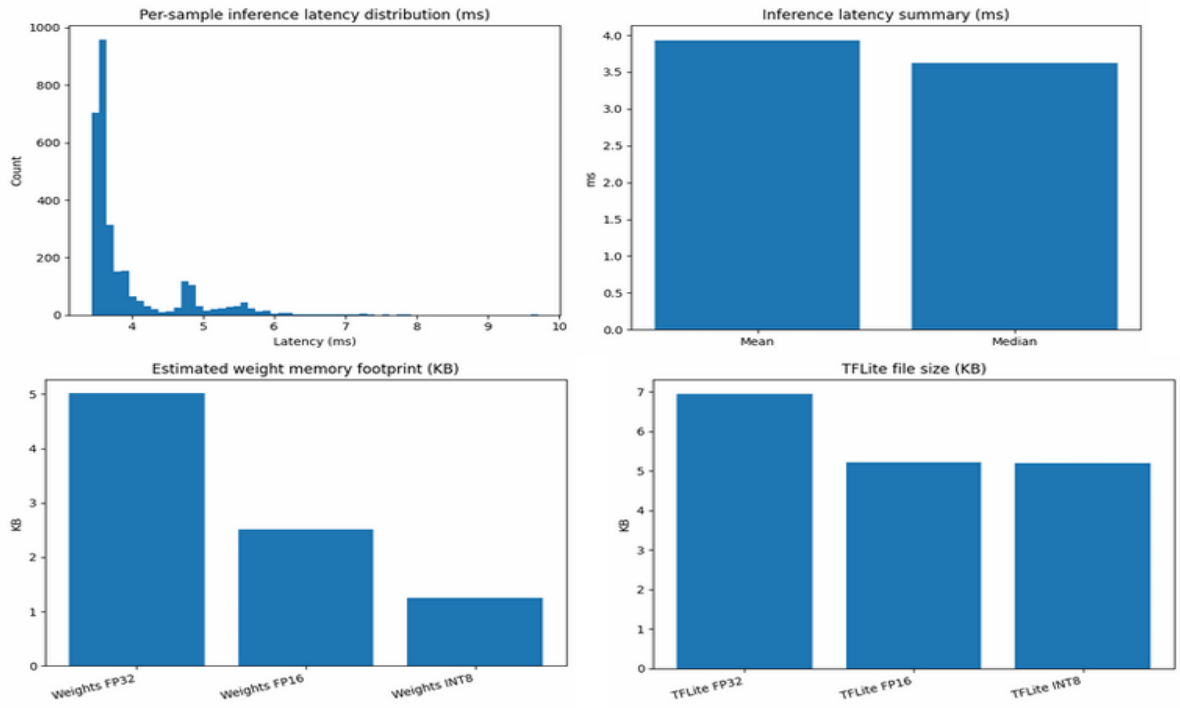

Figure 13: Inference Latency and Memory Footprint Analysis of Quantized DNN Models

Figure 13: The per-sample inference latency distribution reveals that the bulk of the executions fall within 3.5 ms to 5.0 ms, with occasional outliers reaching 10 ms. The latency summary shows a mean inference time of 3.9 ms and a median of 3.6 ms, indicating low, stable runtime performance.

Quantization can reduce the estimated weight memory footprint by approximately 5.0 KB (FP32) to 2.5 KB (FP16), then to about 1.2 KB (INT8). Equally, the size of a TensorFlow Lite model is reduced to about 7.0 KB (FP32), 5.2 KB (FP16), and 5.1 KB (INT8). These findings demonstrate that quantization can be used effectively to minimize memory consumption while still enabling real-time inference.

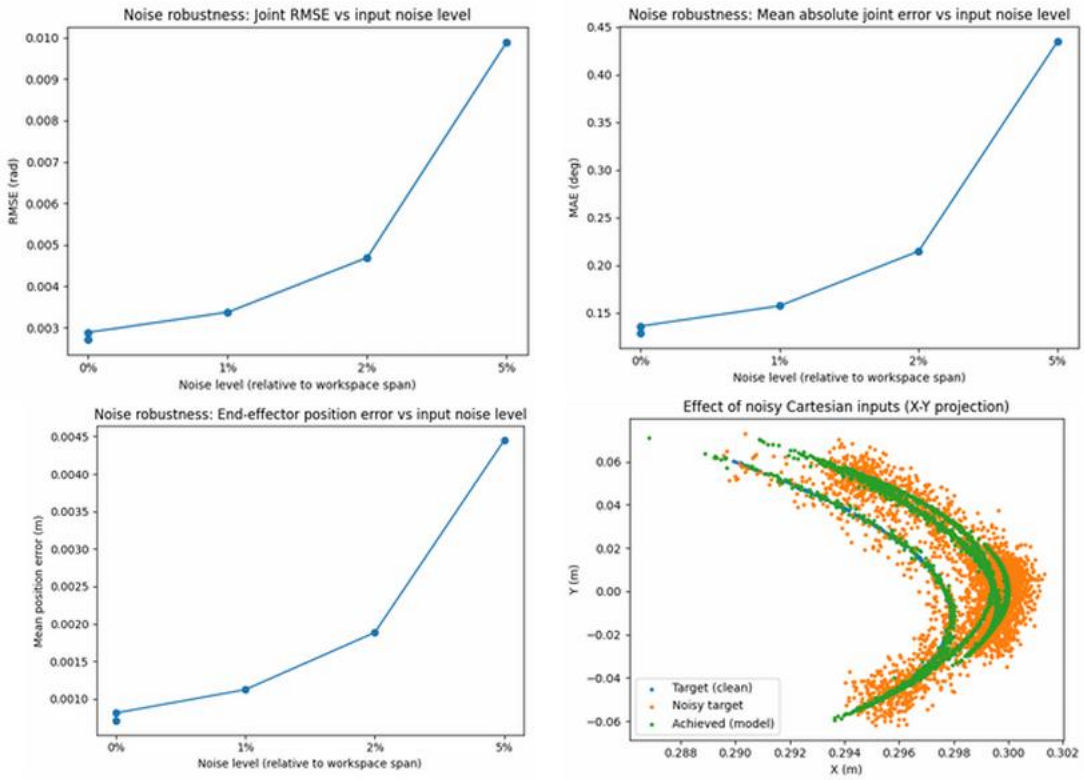

Figure 14: Noise Robustness Analysis of the Proposed Inverse Kinematics Model

Figure 14 presents the noise robustness analysis. Joint-level RMSE increases from approximately 0.0028 rad at 0% noise to 0.0046 rad at 2% noise and 0.0098 rad at 5% noise, indicating graceful degradation under moderate perturbations. Mean absolute joint error follows the same pattern: approximately 0.0023 rad (0% noise), 0.0037 rad (2% noise), and 0.0075 rad (5% noise). At low noise levels, end-effector position error remains low, increasing to approximately 0.0008 m and 0.0019 m at 2% and 5% noise, respectively. The obtained trajectory remains close to the clean target trajectory despite noisy Cartesian inputs, as shown in the X-Y projection plot. Overall, the findings confirm that the model is highly robust to input noise levels up to 2% of workspace span, with graceful and acceptable accuracy degradation at higher noise magnitudes.

5. Conclusion

This paper demonstrates that a student-demonstration-driven embedded deep learning framework is a viable and effective approach to achieving real-time inverse kinematics control of a robotic manipulator on Arduino-class platforms. By leveraging demonstration data collected from students to train a lightweight deep neural network, the framework successfully learns the inverse kinematics mapping without requiring analytical modeling. Experimental results confirmed high joint-angle prediction accuracy, with RMSE below 0.006 rad and mean absolute joint error under 0.004 rad under nominal conditions, alongside an inference latency of approximately 4 ms and a quantized TFLite model footprint of under 57 KB — all well within the constraints of resource-limited embedded hardware. Robustness evaluations further showed that the framework maintained stable performance

under moderate Cartesian input noise (up to 2% of the workspace), with only a gradual degradation at higher noise levels. Collectively, these findings validate the practicability of the proposed student-demonstration-driven paradigm as a scalable, accessible, and computationally efficient solution for deploying learning-based robotic control directly on Arduino-class embedded systems, opening promising avenues for low-cost educational and industrial robotic applications.

Conflict of Interest: There is no conflict of interest for this paper.

Use of AI tool Declaration

I declare that any AI tools used in the preparation of this manuscript were limited to language and readability improvement only, and were not used to generate scientific content, data, analyses, or conclusions, with full responsibility retained by the author.

References

[1] Anjum N, Amjad MK, Ayaz Y, editors. Analysis of Computational Efficiency of Artificial Intelligence-based Search Techniques in Trajectory Planning of Industrial Manipulator. 2019 International Conference on Robotics and Automation in Industry (ICRAI); 2019: IEEE. https://doi.org/10.1109/ICRAI47710.2019.8967374.

[2] Karalekas G, Vologiannidis S, Kalomiros J. Teaching Artificial Intelligence and Machine Learning in Secondary Education: A Robotics-Based Approach. Applied Sciences. 2025. https://doi.org/10.3390/app15084570

[3] Chavdarov I, Nikolov V, Naydenov B, Boiadjiev G, editors. Design and control of an educational, redundant 3D-printed robot. 2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM); 2019: IEEE. https://doi.org/10.23919/SOFTCOM.2019.8903825.

[4] Subramanian RR, editor. Dual-Mode Robotic Arm for Manufacturing Industries. 2025 International Conference on Computational Robotics, Testing and Engineering Evaluation (ICCRTEE); 2025: IEEE. https://doi.org/10.1109/ICCRTEE64519.2025.11053040

[5] Tello-Monroy JM, Acosta-Amaya GA, Ceballos-Ramírez RA, Jiménez-Builes JA. Neural network control insights and strategies in low-cost educational platforms: A pneumatic levitation case study. Engineering Applications of Artificial Intelligence. 2026. https://doi.org/10.1016/j.engappai.2026.113772.

[6] M. Thamrin N, Irawan A, Zulkifli Z, Al Junid SAM, Megat Ali MSA, PP Abdul Majeed A. Practical Robotics Projects—Hands-On Learning for Students. Robotics in Education: Springer; 2025. p. 41-53.

[7] Tang Z, Wang P, Xin W, Laschi C. Learning to control a soft robotic manipulator under uncertainty and unforeseen changes in robot–environment interaction. The International Journal of Robotics Research. 2025.https://doi.org/10.1177/027836492513602

[8] Ali KN, editor. Artificial Intelligence in Robotics: Revolutionizing Retail Store Automation and Beyond. 2025 22nd International Learning and Technology Conference (L&T); 2025: IEEE. https://doi.org/10.1109/LT64002.2025.10941657

[9] Rizwan S, Fayyaz B, Zubair M, Ghani A. Integrating Deep Learning for Object Manipulation: A 7-DOF Robotic Arm Perspective on Grasping. AITU SCIENTIFIC RESEARCH JOURNAL. 2025.https://doi.org/10.63094/AITUSRJ.25.4.1.5

[10] Sahed MTR, Rufsun S, Aronno MTI, Chowdhury MSR, editors. Object Classification by a 7-DOF Robotic Arm Using MobileNet-SSD and Feedback Position Mapping. 2025 International Conference on Quantum Photonics, Artificial Intelligence, and Networking (QPAIN); 2025: IEEE. https://doi.org/10.1109/QPAIN66474.2025.11171666

[11] Li H, Xu N, Zhang X, Zhu B. Image-Guided Trajectory Tracking Control of the Nanopositioning Stages Using Regional Dense Photometric Information. IEEE Transactions on Industrial Electronics. 2023. https://doi.org/10.1109/TIE.2023.3319730.

[12] Alsayaydeh JAJ, Suntheran S, Yusof MF, Ahmed AH, Herawan SG, Wook MFBT. Design and Implementation of an Arduino-Based Mobile Robotic Arm with Omnidirectional Mobility for High-Precision Industrial Automation. Journal Européen des Systèmes Automatisés. 2025; https://doi.org/10.18280/jesa.580813

[13] Sahu UK, KS M, K A, P MM, Jaiswal A, Yadav UK, et al. Autonomous object tracking with vision-based control using a 2DOF robotic arm. Scientific Reports. 2025. https://doi.org/10.1038/s41598-025-97930-3

[14] Erivianto D, Dani A. Implementasi Teori Finite State Machine Pada Sistem Kontrol Robot Lengan Berbasis Arduino dan Python. RIGGS: Journal of Artificial Intelligence and Digital Business. 2026. https://doi.org/10.31004/riggs.v4i4.3892

[15] Fontenot K, Gorti A, Goel I, Buonassisi T, Siemenn AE. Closed-Loop Robotic Manipulation of Transparent Substrates for Self-Driving Laboratories using Deep Learning Micro-Error Correction. https://doi.org/10.48550/arXiv.2512.06038

[16] Bobojonov O, Thayumanavan K, Abdumajidovich OE, Soni S, editors. Q-Learning-Based Feedback Optimization for Operator Training in Industrial Robotics. 2025 Second International Conference on Intelligent Technologies for Sustainable Electric and Communications Systems. https://doi.org/10.1109/iTechSECOM64750.2025.11307210

[17] Delamou, M., Bazzi, A., Chafii, M., & Amhoud, E. M. (2023, June). Deep learning-based estimation for multitarget radar detection. In 2023 IEEE 97th vehicular technology conference (VTC2023-Spring) (pp. 1-5). IEEE. https://doi.org/10.1109/VTC2023-Spring57618.2023.10200157

[18] Njima, W., Bazzi, A., & Chafii, M. (2022). DNN-based indoor localization under a limited dataset using GANs and semi-supervised learning. *IEEE Access*, *10*, 69896-69909. https://doi.org/10.1109/ACCESS.2022.3187837

[19] Aziz, N.A., Al-Awamry, A., Lashin, M. *et al.* A new autonomous symbolic-based intelligent embedded feedback controller for cyber-physical parameter-varying systems. *J. Eng. Appl. Sci.* **72**, 98 (2025). https://doi.org/10.1186/s44147-025-00659-z