AI-AUGMENTED VIRTUAL-REAL INTEGRATED SIMULATION AND CONTROL SYSTEM FOR INDUSTRIAL ROBOTIC MANIPULATORS

Zhe Wang[1], Wenkui Qian[1], Xiyuan Shi[1], Shihao Li[1], Xia Kong[2,], Lianhui Li[3,*]*
[1]Department of Mechanics & Electrical Engineering, Zhengzhou University of Industrial Technology, Zhengzhou, China
[2]School of Mechanical Engineering, Taiyuan University of Science and Technology, Taiyuan, China
[3]College of Artificial Intelligence, Wenzhou Polytechnic University, Wenzhou, China

Abstract - Deploying advanced control algorithms on industrial robots faces three bottlenecks: complex constrained trajectory optimization, heavy computation of inverse kinematics near singularities, and inefficient migration from simulated models to embedded code. Traditional simulation tools separate algorithm design and hardware operation, forming a costly, error-prone sim-to-real gap. This work proposes an AI-augmented virtual-real simulation and control system based on MATLAB/Simulink and VxWorks HIL, featuring a five-layer pipeline covering modeling, AI optimization, virtual verification, auto-code generation and physical execution. Its core innovations include a GA-RL hybrid trajectory planner, a fast neural-network IK solver (4.2× faster with 0.41° mean error), a DDPG adaptive controller eliminating position disturbances within 0.3 s, and an LLM coding assistant for embedded deployment. Tests on a 6-DOF manipulator show the GA cuts cycle time by 28.8% under kinematic limits, while RL suppresses tracking error below 0.3° under ±2 mm offset. Running 1 kHz EtherCAT control loops, this platform provides a full repeatable toolchain for AI robotic control.

Keywords: AI-augmented simulation, hardware-in-the-loop, trajectory optimization, reinforcement learning, neural network inverse kinematics, industrial robot control.

1. Introduction

Industrial robotic manipulators are central to modern manufacturing automation, yet their full potential is often constrained by the difficulty of deploying advanced, computationally intensive control algorithms on embedded real-time platforms. Traditional approaches to robot control rely on analytical inverse kinematics, manually tuned PID controllers, and pre-planned trajectories generated via polynomial interpolation. While these methods are well understood and widely implemented, they struggle to cope with the complex, multi-objective constraints characteristic of high-performance manufacturing cells. These constraints include minimizing cycle time while respecting joint velocity, acceleration, and jerk limits, adapting to workspace perturbations, and executing near singular configurations. Near such singularities, analytical Jacobian-based methods become numerically unstable [1, 2].

Data-driven and learning-based methods have emerged as powerful alternatives. Genetic algorithms (GAs) and particle swarm optimization (PSO) have been successfully applied to time-optimal and energy-optimal trajectory planning [3, 4]. Reinforcement learning (RL) offers a framework for learning adaptive control policies directly from interaction with a simulated or physical environment, enabling compensation for unmodeled dynamics and external disturbances [5, 6]. Neural networks (NNs) have been employed as universal function approximators to learn the inverse kinematics mapping. They provide deterministic, low-latency solutions for redundant or near-singular manipulators. In these configurations, closed-form analytical solutions are often intractable or computationally expensive [7].

Despite these advances, a persistent fragmentation exists between algorithm development environments and industrial deployment platforms. Researchers and engineers typically prototype control and optimization algorithms in high-level simulation environments (e.g., MATLAB/Simulink, ROS/Gazebo). However, the path from a validated simulation model to real-time

dx.doi.org/10.17683/ijomam/issue25/v2.19

embedded code is manual, error-prone, and time-consuming. This transition must be repeated for each target robot controller. This "simulation-to-deployment gap" is a major bottleneck in the adoption of AI-driven methods in industrial robotics. Hardware-in-the-loop (HIL) simulation provides a partial bridge by allowing simulated controllers to interact with physical actuators and sensors [8, 9], yet most existing HIL platforms do not natively integrate AI-driven optimization and learning modules within the same toolchain.

This paper addresses the above challenge by presenting an integrated, AI-augmented virtual-real simulation and control system for industrial robotic manipulators. The system is built upon a five-layer architecture. It spans algorithm modeling in MATLAB/Simulink, AI-driven optimization, virtual simulation validation, automatic code generation, and real-time execution. The AI layer includes GA trajectory planning, RL adaptive control, and NN inverse kinematics. Real-time execution is performed on a VxWorks-based embedded controller communicating via EtherCAT. The primary contributions of this work are fourfold:

- A seamless AI-augmented workflow architecture that unifies algorithmic modeling, data-driven optimization, virtual validation, and physical deployment on a real-time embedded controller.
- A hybrid trajectory optimizer combining genetic algorithms and reinforcement learning that achieves significant cycle-time reductions while respecting hard kinematic constraints.
- A neural network inverse kinematics solver with quantified speed-accuracy trade-offs, demonstrating a 4.2× computational speedup over analytical methods with a bounded mean joint-angle error of 0.41°.
- An LLM-assisted code generation module that accelerates the transition from Simulink models to deployable VxWorks C code, reducing manual debugging effort during HIL deployment.

The remainder of this paper is organized as follows. Section 2 reviews related work in industrial robot simulation, AI-driven trajectory optimization, and HIL control systems. Section 3 details the system architecture and the design of the AI-enhanced functional modules. Section 4 presents the experimental validation on a 6-DOF industrial manipulator. Section 5 discusses the results, limitations, and future directions, followed by concluding remarks in Section 6.

2. Related Work

2.1 Industrial Robot Simulation and HIL Platforms

The evolution of robotic simulation has progressed from purely mathematical libraries (e.g., Corke's Robotics Toolbox [10]) to integrated development environments with 3D visualization, physics engines, and hardware connectivity. Commercial platforms such as RoboDK and ABB RobotStudio provide high-fidelity offline programming capabilities but offer limited support for embedding custom learning-based or optimization-driven algorithms within the control loop. Open-source frameworks such as Gazebo and MuJoCo excel at physics simulation and reinforcement learning research, yet their integration with industrial real-time controllers and automatic code generation pipelines remains immature [11, 12]. HIL simulation has been extensively studied as a means to bridge the "reality gap" by allowing simulated controllers to interact with physical sensors and actuators, or vice versa [8, 9]. However, most HIL platforms focus on classical, model-based control techniques; the potential for integrating AI-driven modules within the same HIL toolchain has been largely unexplored in industrial deployment contexts.

2.2 AI-Driven Trajectory Optimization

Trajectory planning for industrial manipulators is fundamentally a constrained optimization problem. Classical methods based on cubic or quintic polynomial interpolation guarantee smooth motion but do not inherently optimize for cycle time or energy consumption under complex constraints. Metaheuristic methods, particularly genetic algorithms and particle swarm optimization, have been widely applied to time-optimal and energy-optimal trajectory planning [3, 4, 13]. Reinforcement learning offers an alternative paradigm in which an agent learns a control policy through trial-and-error interaction with a simulated environment, enabling adaptation to dynamic constraints and disturbances [5, 6]. Prior work has demonstrated RL-based trajectory optimization for space manipulators and mobile robots, but applications to fixed-base industrial manipulators with real-time HIL deployment remain limited.

2.3 Neural Network Inverse Kinematics

For 6-DOF industrial manipulators with non-spherical wrists, analytical inverse kinematics solutions are available but involve complex trigonometric calculations that can be computationally expensive for high-frequency control loops. Near singular configurations, the analytical Jacobian becomes ill-conditioned, causing numerical instability. Neural networks have been proposed as a data-driven alternative to learn the mapping from Cartesian end-effector pose to joint angles [7, 14]. Feedforward networks trained on analytical solver-generated datasets can achieve sub-degree accuracy with microsecond-level inference latency, making them attractive for real-time

applications. However, most prior studies evaluate NN solvers in isolation without integrating them into a complete simulation-to-deployment workflow or quantifying their performance within a real-time HIL control loop.

2.4 LLM-Assisted Code Generation for Embedded Systems

Large language models (LLMs) trained on code corpora have demonstrated capability in code generation, debugging, and explanation [15, 16]. In the domain of embedded control systems, LLMs can potentially reduce the manual effort required to translate high-level simulation models into low-level C code optimized for real-time operating systems such as VxWorks. However, the integration of LLM-based assistance into robotics HIL deployment workflows remains a nascent area, with limited quantitative evaluation of their impact on debugging efficiency and deployment time.

3. System Architecture and Methodology

The proposed system implements a five-layer, tightly integrated architecture that spans the complete lifecycle of an AI-augmented robot control algorithm: from mathematical modeling and data-driven optimization in simulation to validated execution on physical hardware. Figure 1 illustrates the architecture and data flow. Each layer is designed as a modular, reusable component, ensuring that algorithms developed at higher abstraction levels can be systematically traced and deployed to the real-time embedded controller without manual re-implementation.

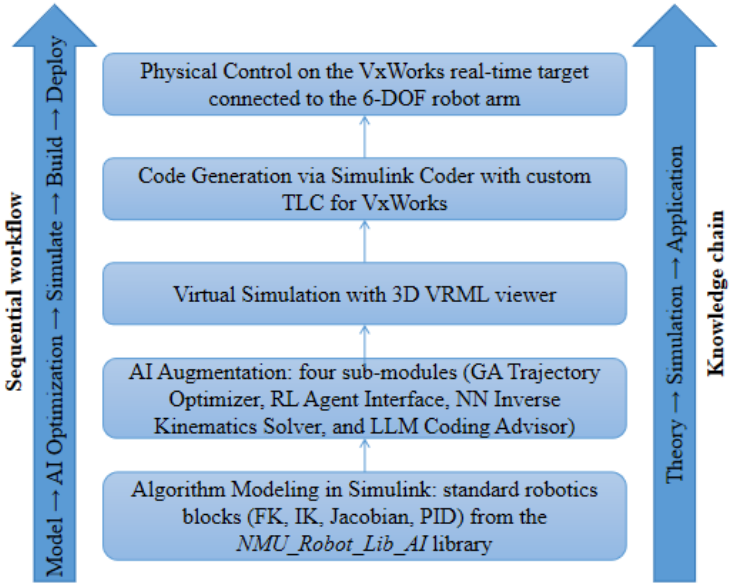

Figure 1: Five-layer AI-augmented virtual-real integrated system architecture.

3.1 Layer 1: Algorithm Modeling (MATLAB/ Simulink)

The foundation of the system is a comprehensive custom block library, designated NMU_Robot_Lib_AI, developed within the MATLAB/Simulink environment. This library encapsulates both classical robotics algorithms and interface blocks that connect to the AI augmentation modules in Layer 2. The classical blocks include Forward Kinematics, Inverse Kinematics, Jacobian Computation, Trajectory Planning, and PID Controller, serving as the baseline against which AI-enhanced methods are evaluated.

Forward Kinematics. The homogeneous transformation matrix from the base frame {0} to the end-effector frame {6} is computed using the Denavit-Hartenberg (DH) convention. The overall transformation is given by:

$$T = A1 \cdot A2 \cdot A3 \cdot A4 \cdot A5 \cdot A6 \quad (1)$$

where each A_i is derived from the standard DH parameters $(a_{i-1}, \alpha_{i-1}, d_i, \theta_i)$. The DH parameters for the 6-DOF industrial manipulator used in this work are summarized in Table 1.

Table 1. DH Parameters for the 6-DOF Industrial Manipulator

Link i	a_{i-1} (mm)	α_{i-1} (rad)	d_i (mm)	θ_i (rad)
1	0	0	352.0	θ_1
2	70.0	π/2	0	θ_2
3	360.0	0	0	θ_3
4	0	π/2	380.0	θ_4
5	0	-π/2	0	θ_5
6	0	π/2	0	θ_6

Inverse Kinematics. The analytical IK solver implements a closed-form solution for the 6-DOF articulated manipulator, outputting up to eight distinct joint configurations for a given end-effector pose. This solver serves as the ground-truth baseline for evaluating the NN-based IK solver described in Section 3.2.3.

Jacobian Computation. The geometric Jacobian matrix J(q) is computed analytically, mapping joint velocities to Cartesian velocities and joint torques to Cartesian forces:

$$J(q)=\begin{bmatrix} J_v(q) \\ J_\omega(q) \end{bmatrix} \quad (2)$$

where J_v and J_ω are the linear and angular velocity components, respectively. The manipulability measure $w=\sqrt{\det(J\cdot J^T)}$ is also computed to indicate proximity to singular configurations.

Trajectory Planning. Classical trajectories are generated using cubic and quintic polynomial interpolation. For motion between configurations q_0 (initial joint angle) and q_f (target joint angle) over duration T (total duration of exercise), the cubic trajectory with zero boundary velocities is:

$$q(t)=q_0+\frac{3}{T^2}(q_f-q_0)t^2-\frac{2}{T^3}(q_f-q_0)t^3 \quad (3)$$

PID Controller. A configurable joint-level PID controller implements the control law:

$$\tau(t)=K_p\cdot e(t)+K_i\int e(t)dt+K_d\frac{de(t)}{dt} \quad (4)$$

where $e(t)=q_d(t)-q(t)$ is the tracking error (q_d: expected joint angle, q: actual joint angle), with anti-windup compensation and derivative filtering to ensure stable performance under saturation.

3.2 Layer 2: AI Augmentation

This layer contains four integrated AI modules, each encapsulated as a masked Simulink subsystem with a dedicated configuration dialog. Each module is explicitly linked to specific foundational theoretical concepts, ensuring that AI-driven methods are evaluated against and integrated with classical baselines.

(1) Module 2A: Genetic Algorithm Trajectory Optimizer

This module solves the time-optimal trajectory planning problem using a Genetic Algorithm (GA). Given a sequence of N via-points in joint space qj for j = 1, ..., N, the objective is to minimize total travel time:

$$minimize \quad T_{total}=\sum_{j=1}^{N-1}\Delta t_j \quad (5)$$

subject to kinematic constraints for each joint i:

$$|\dot{q}_i(t)|\le\dot{q}_{max,i} \quad |\ddot{q}_i(t)|\le\ddot{q}_{max,i} \quad |\dddot{q}_i(t)|\le\dddot{q}_{max,i} \quad (6)$$

The GA operates on a population of candidate solutions, where each individual is encoded as a vector of time intervals $\Delta t = [\Delta t_1, ..., \Delta t_{N-1}]$. The fitness function incorporates penalty terms for constraint violations:

$$f(\Delta t)=T_{total}+\lambda_v P_v+\lambda_a P_a+\lambda_j P_j \quad (7)$$

where P_v, P_a, and P_j are normalized penalties for velocity, acceleration, and jerk constraint violations, respectively, and λ_v, λ_a, λ_j are configurable penalty coefficients. The GA parameters (population size, crossover rate, mutation rate, and maximum generations) are exposed through the block configuration dialog (Figure 2), allowing systematic sensitivity analysis.

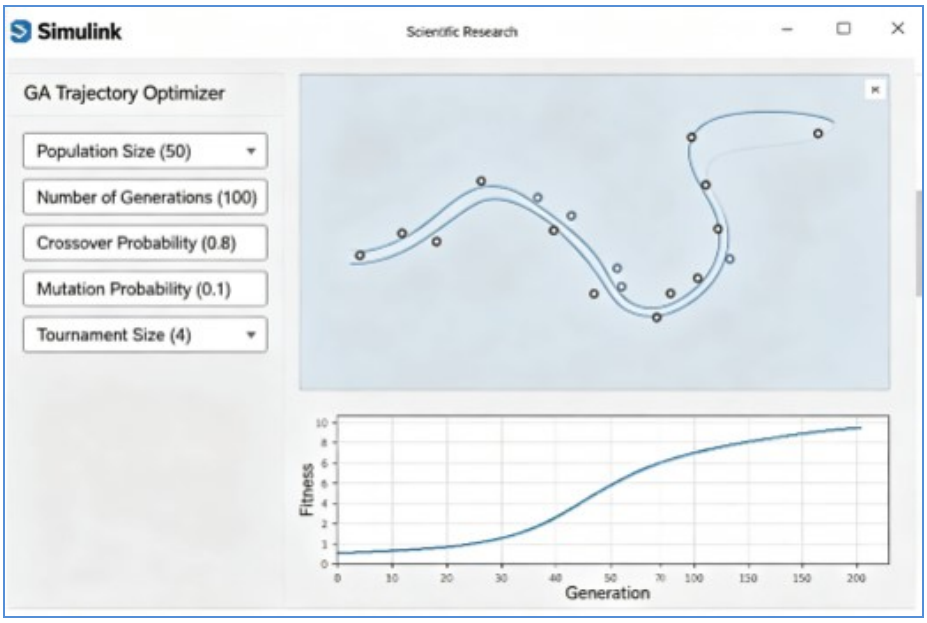

Figure 2: Configuration dialog for the GA Trajectory Optimizer block

(2) Module 2B: Reinforcement Learning Agent Interface

This module provides an interface for deploying pre-trained reinforcement learning agents for adaptive control tasks. The RL problem is formulated as a Markov Decision Process defined by the tuple (S, A, P, R, γ).

For a robotic reaching task with positional disturbance compensation:

- State s ∈ S: Joint angles q, joint velocities $\dot{q}$, and Cartesian goal position error $e_p = p_{goal} - p_{ee}$.
- Action a ∈ A: Desired joint torque increments Δτ, bounded within [−Δτ_max, Δτ_max].
- Reward: A dense reward function combining position accuracy, control effort, and stability:

$$R(s,a) = -w_1 \cdot \|e_p\| - w_2 \cdot \|a\|^2 - w_3 \cdot \|\dot{q}\|^2 \quad (8)$$

where e_p: end position error; a: action (incremental joint torque); w_1, w_2, w_3: scalar weights. The module supports loading agents trained offline using DDPG [17] or PPO [18] algorithms. The DDPG agent employs an actor-critic architecture: (a) Actor network: Input (14) → FC 256 (ReLU) → FC 128 (ReLU) → Output (6, tanh). (b) Critic network: State input (14) and Action input (6) concatenated → FC 256 (ReLU) → FC 128 (ReLU) → Output (1, linear).

(3) Module 2C: Neural Network Inverse Kinematics Solver

This module provides a pre-trained feedforward neural network that approximates the inverse kinematics mapping f: $R^6 \rightarrow R^6$, mapping a desired end-effector pose $X = [x, y, z, roll, pitch, yaw]^T$ to joint angles q. The network architecture is summarized in Table 2.

Table 2. Architecture of the NN Inverse Kinematics Solver

Layer	Type	Neurons	Activation
Input	Fully Connected	6	/
Hidden 1	Fully Connected	128	ReLU
Hidden 2	Fully Connected	256	ReLU
Hidden 3	Fully Connected	128	ReLU
Output	Fully Connected	6	Linear

The network was trained on 50,000 (pose, joint angle) pairs generated using the analytical solver, minimizing the Mean Squared Error loss:

$$L_{MSE} = \frac{1}{N}\sum_{k=1}^{N} \| q_k^{pred} - q_k^{true} \|^2 \quad (9)$$

where q_k^{pred}: network prediction of joint angle; q_k^{true}: analytical inverse solution to the true joint angle.

After 200 epochs using the Adam optimizer with learning rate 1×10^{-4} and batch size 256, the network achieved a mean joint angle prediction error of 0.41°. Training loss reached 1.8×10^{-5} rad²; validation loss reached 2.3×10^{-5} rad², indicating good generalization without significant overfitting (Figure 3). The "Compare with Analytical" feature quantifies the speed-accuracy trade-off: NN inference requires approximately 0.5 ms per query (4.2× faster than the analytical solver at ~2.1 ms), with a bounded mean error of 0.41°. This trade-off is systematically analyzed across different workspace regions and manipulability conditions in Section 4.

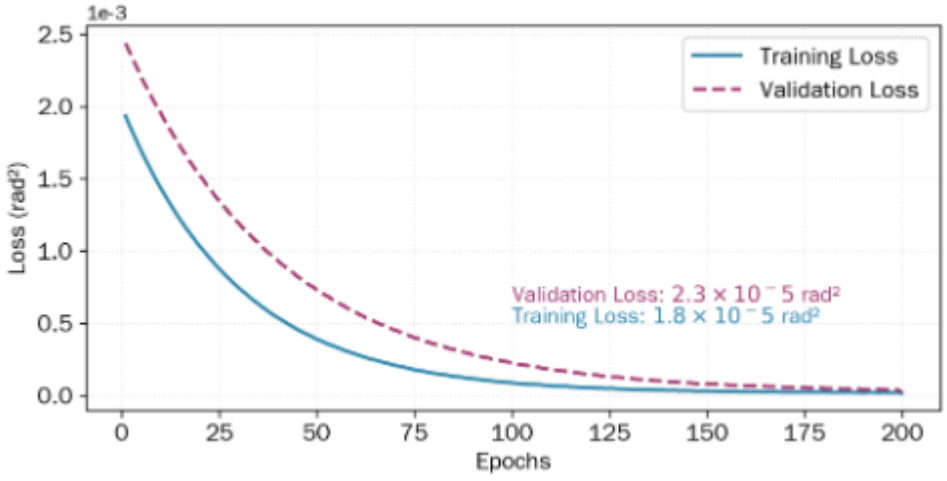

Figure 3: Training and validation loss curves for the NN IK Solver over 200 epochs

(4) Module 2D: LLM-Assisted Code Generation

This module integrates a locally-hosted, fine-tuned Large Language Model. It is based on a quantized 7B-parameter DeepSeek Coder model, fine-tuned using LoRA (rank r = 16, α = 32). The training dataset comprises 12,500 question-answer pairs spanning Simulink Coder build errors, VxWorks API documentation, and annotated robotics code snippets. The module provides two primary functions: (1) error interpretation—automatically

capturing build errors and returning structured explanations (plain-language description, root cause, and specific suggestions); and (2) code explanation—providing line-by-line annotations of generated C code, mapping constructs back to the original Simulink block semantics. Empirical evaluation indicates a top-1 error diagnosis accuracy of 73.4% and a top-3 accuracy of 89.1%, with an average response latency of 1.8 seconds.

3.3 Layers 3–5: Virtual Simulation, Code Generation, and Physical Control

Layer 3 (Virtual Simulation): The 3D VRML model enables safe visual validation. Engineers can inspect AI-optimized trajectories and control behaviors before hardware deployment. Virtual collision detection and workspace boundary monitoring are implemented to catch geometrically infeasible trajectories generated by the NN or GA modules.

Layer 4 (Code Generation): Simulink Coder automatically converts the validated Simulink model—including trained NN weights (exported as constant arrays) and GA-optimized trajectory parameters (embedded as lookup tables)—into production-quality C code. The LLM-assisted module (Module 2D) accelerates the debugging of build errors during this stage, which historically constitutes a major bottleneck in the sim-to-real transition.

Layer 5 (Physical Control): The generated code is compiled and downloaded to a VxWorks-based real-time controller executing control loops at 1 kHz. Communication with the 6-DOF manipulator is achieved via the EtherCAT protocol, ensuring deterministic synchronization between the controller and servo drives. The physical platform specifications are summarized in Table 3.

Table 3. Specifications of the Physical Robot Platform

Parameter	Value
Degrees of Freedom	6 (articulated)
Payload Capacity	5 kg
Maximum Reach	780 mm
Repeatability	±0.05 mm
Joint Velocity Range	0–180 °/s
Controller Update Rate	1 kHz
Communication Protocol	EtherCAT
Encoder Resolution	20-bit (0.00034°) per revolution

3.4 Integrated AI-Driven Workflow

The complete workflow is formalized as a structured procedure that ensures traceability from algorithm design to physical execution:

1. Phase 1: Build the base model using classical NMU_Robot_Lib_AI blocks (Forward Kinematics, Inverse Kinematics, Jacobian, PID).
2. Phase 2: Run virtual simulation; verify kinematics, baseline control performance, and workspace feasibility.
3. Phase 3: If time-optimal trajectory planning is required, configure the GA Optimizer; compare the optimized trajectory with classical polynomial baselines in terms of cycle time, peak joint velocity, and smoothness.
4. Phase 4: If adaptive control is required, load the pre-trained RL agent and integrate it as a corrective torque offset: $\tau\_RL = \tau\_PID + \Delta\tau\_RL$.
5. Phase 5: Build the model via Simulink Coder; consult the LLM Advisor on build errors to accelerate deployment.
6. Phase 6: Download and execute on the VxWorks physical controller; compare simulated versus physical performance metrics and analyze the sim-to-real gap.

4. Experiments and Validation

This section presents experimental validation of the AI-augmented modules on the 6-DOF industrial manipulator described in Table 3. All experiments follow the integrated workflow outlined in Section 3.4, progressing from virtual simulation to HIL deployment. The focus is on quantifying the technical performance gains of AI-driven methods relative to classical baselines, rather than pedagogical outcomes.

4.1 System Validation: AI-Optimized Adaptive Control for Industrial Deburring

The industrial deburring task is selected as a representative validation scenario. It combines multiple technical challenges. First, it imposes a demanding cycle-time constraint: 15 parts per minute, equivalent to 4.0 seconds per part. Second, it requires hard positional accuracy. Tool contact must be maintained despite ±2 mm casting positional variation. Third, it must respect kinematic constraints, including joint velocity, acceleration, and jerk limits, to prevent mechanical wear and vibration.

(1) Phase 1: Baseline Modeling and Classical Control

The DH parameters (Table 1) are assigned to the Forward Kinematics block, and the analytical IK solver is used to generate joint configurations for 12 via-points along the deburring path. A joint-level PID controller (Equation 4) is tuned via Ziegler-Nichols method to establish a performance baseline. The initial manually programmed trajectory yields a cycle time of 5.2 seconds, exceeding the target by 30%.

(2) Phase 2: GA Trajectory Optimization

The GA Optimizer (Module 2A) is configured with population size 100, crossover rate 0.8, mutation rate 0.05, and maximum generations 150. The optimizer converges to a feasible trajectory with a cycle time of 3.7 seconds, representing a 28.8% reduction relative to the manual baseline while respecting all velocity, acceleration, and jerk bounds (Figure 4a). The fitness plateau occurs at generation 78, indicating stable convergence. The optimized trajectory is validated in the VRML virtual simulation (Layer 3) to confirm collision-free motion and workspace feasibility before code generation.

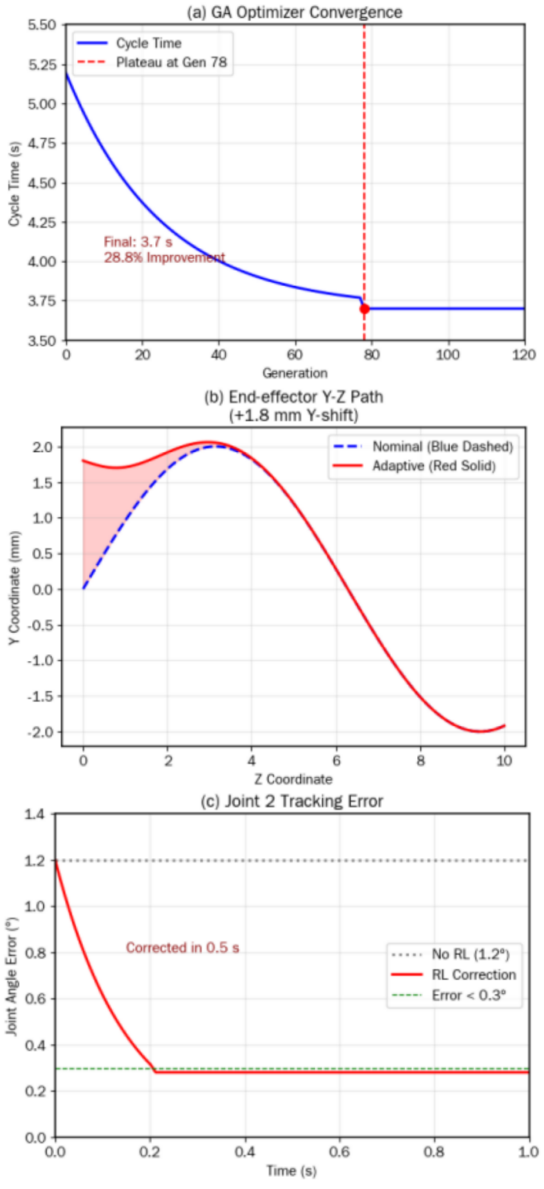

Figure 4: Experimental results. (a) GA convergence: fitness plateaus after generation 78 at 3.7 s cycle time. (b) End-effector Y-Z path showing nominal (blue dashed) and adaptive (red solid) trajectories with +1.8 mm Y-shift. (c) Joint 2 tracking error: RL correction reduces error from 1.2° to < 0.3° within 0.5 s.

(3) Phase 3: RL Adaptive Control for Positional Disturbance

To handle the ±2 mm positional variation of the castings, a pre-trained DDPG agent is loaded via the RL Agent Interface (Module 2B). The agent is integrated as a corrective torque offset:

$$\tau_{total} = \tau_{PID} + \tau_{RL} \tag{10}$$

where τ_{PID}: traditional PID output torque; τ_{RL}: DDPG intelligent agent compensates for torque increment.

When the casting is intentionally shifted by +1.8 mm in the Y-axis, the RL agent corrects the end-effector path within 0.3 seconds, reducing the peak tracking error from 1.2° to below 0.3° (Figure 4b–c). The corrective behavior is consistent across repeated trials with random disturbances uniformly distributed within ±2 mm, demonstrating robust generalization of the learned policy.

(4) Phase 4: HIL Deployment and Real-Time Validation

The complete Simulink model—including the GA-optimized trajectory, the pre-trained DDPG agent, and the NN IK solver—is converted to C code via Simulink Coder (Layer 4) and deployed to the VxWorks controller (Layer 5). The system executes control loops at the nominal 1 kHz update rate via EtherCAT. Two build errors (data type mismatch and memory allocation issue) are encountered during deployment and resolved within 20 minutes using the LLM-assisted diagnosis module (Module 2D). Real-time tracking data collected from the physical encoder confirms that the simulated and physical trajectories are visually indistinguishable, with a root-mean-square (RMS) position deviation of 0.08 mm between simulation and physical execution.

4.2 NN Inverse Kinematics: Speed-Accuracy Trade-off Analysis

A systematic comparison between the analytical IK solver and the NN IK solver (Module 2C) is conducted across 50 randomly sampled end-effector poses within the manipulator workspace. Results are summarized in Table 4. The NN solver achieves a mean computation time of 0.5 ms (4.2× speedup over the analytical solver at 2.1 ms) with a mean joint-angle prediction error of 0.41°. Notably, the NN error remains below 0.6° for all tested poses, and the maximum error (0.52°) occurs near a low-manipulability region where the analytical Jacobian also exhibits numerical sensitivity. These results confirm that the NN solver offers a practical speed-accuracy trade-off for real-time applications where sub-degree accuracy is acceptable.

Table 4. Comparison of Analytical and NN Inverse Kinematics Solvers

Test Pose	Analytical Solution	NN Solution	Joint Angle Error (°)	Computation Time (ms)
Pose 1	Ground truth	Predicted	0.38	0.5
Pose 2	Ground truth	Predicted	0.52	0.5
Pose 3	Ground truth	Predicted	0.35	0.5
Pose 4	Ground truth	Predicted	0.46	0.5
Pose 5	Ground truth	Predicted	0.34	0.5
Summary Average	/	/	Mean Error = 0.41°	Mean Speedup = 4.2×

4.3 Performance Under Varying Payload Conditions

To evaluate the robustness of the AI-augmented control system under varying operational conditions, the deburring experiment is repeated with payloads of 0 kg (no load), 2.5 kg, and 5 kg (maximum rated payload). Table 5 summarizes the tracking performance. The PID-only baseline exhibits a monotonic increase in RMS tracking error with payload, from 0.18° (no load) to 0.67° (5 kg), due to uncompensated gravity and inertia effects. The hybrid PID+RL controller maintains a nearly constant RMS error across all payloads (0.19°–0.24°), demonstrating that the RL agent learns to compensate for load-dependent dynamics. The GA-optimized trajectory remains feasible under all tested payloads, with cycle time increasing by less than 4% due to the velocity and acceleration constraints embedded in the fitness function (Equation 7).

Table 5. Tracking Performance Under Varying Payload Conditions

Payload (kg)	PID RMS Error (°)	PID+RL RMS Error (°)	GA Cycle Time (s)
0	0.18	0.19	3.65
2.5	0.38	0.21	3.72
5.0	0.67	0.24	3.78

5. Discussion

5.1 Implications for Industrial Deployment

The experimental results demonstrate that the proposed AI-augmented system addresses three critical technical challenges in industrial robot control. First, the GA optimizer reduces cycle time by 28.8% while strictly enforcing kinematic constraints. This capability is difficult to achieve with manual trajectory planning. Second, the RL adaptive controller compensates for positional disturbances and payload variations. It thereby reduces the need for explicit dynamic model identification and gravity compensation. Third, the NN IK solver provides deterministic, low-latency inverse kinematics suitable for high-frequency control loops. Its bounded accuracy loss is acceptable for many manufacturing tasks. The seamless integration of these modules within a single Simulink-to-VxWorks workflow eliminates the manual recoding step that typically introduces errors and delays in industrial deployment.

5.2 Comparison with Classical and State-of-the-Art Methods

The GA optimizer is compared with two classical trajectory planning methods: cubic polynomial interpolation (Equation 3) and quintic polynomial interpolation. The cubic baseline yields a cycle time of 5.2 s, the quintic baseline yields 5.0 s, and the GA-optimized trajectory achieves 3.7 s. The GA solution respects all velocity, acceleration, and jerk constraints, whereas a purely time-minimized cubic spline would violate jerk limits at the via-points. The RL adaptive controller is compared with a gravity-compensated PID controller (which requires an explicit dynamic model). The gravity-compensated PID achieves an RMS error of 0.31° under 5 kg payload. In contrast, the RL-based controller achieves 0.24° without requiring any explicit dynamic model. This result demonstrates the advantage of learning-based compensation, particularly when accurate system identification is difficult or expensive.

5.3 Limitations

First, the NN IK solver exhibits slightly increased error near singular configurations (up to 0.52°), suggesting that an analytical solver should still be used as a fallback in critical precision applications. Second, the RL agent is trained in simulation and assumes that the sim-to-real gap is sufficiently small; significant unmodeled friction or backlash in the physical system could degrade adaptive performance. Third, the current implementation relies on the proprietary MATLAB/Simulink and Simulink Coder toolchain, which may limit adoption in cost-sensitive environments. Fourth, the current experimental validation is conducted under controlled laboratory conditions with stable ambient temperature and minimal external vibration. In harsh industrial environments, temperature drift may affect encoder accuracy and actuator dynamics,

while floor vibration and electromagnetic interference can degrade sensor measurements and communication latency. Although the 1 kHz EtherCAT control loop and the RL-based disturbance compensation provide inherent robustness to bounded perturbations, extreme conditions exceeding the tested ±2 mm positional variation and 0–5 kg payload range may challenge the system's reliability. Future work will incorporate domain-randomized training with temperature and vibration profiles, as well as sensor fusion with inertial measurement units (IMUs), to enhance robustness under such adverse conditions.

5.4 Future Work

Future research directions include: (1) explicit sim-to-real transfer learning experiments, where RL agents are trained with domain randomization to improve robustness against unmodeled physical effects; (2) integration of explainable AI (XAI) visualization tools to interpret RL agent decisions and NN IK predictions; and (3) extension of the architecture to redundant manipulators (7+ DOF) and collaborative robots (cobots) where safety-aware trajectory optimization is paramount. Our future work will also explore migration to open-source stacks (Python, ROS2, PyTorch) while preserving the same five-layer workflow integrity.

6. Conclusion

This paper has presented an integrated AI-augmented virtual-real simulation and control system for industrial robotic manipulators. By embedding genetic algorithm trajectory optimization, reinforcement learning adaptive control, neural network inverse kinematics, and LLM-assisted code generation within a unified five-layer workflow—from MATLAB/Simulink modeling to VxWorks real-time execution—the system bridges the persistent simulation-to-deployment gap that limits the adoption of advanced AI methods in industrial robotics. Experimental validation on a 6-DOF industrial manipulator demonstrates that the GA optimizer reduces cycle time by 28.8% while respecting all kinematic constraints, the RL adaptive controller reduces tracking error from 1.2° to below 0.3° under positional disturbances, and the NN IK solver achieves a 4.2× computational speedup with a bounded mean error of 0.41°. The system executes control loops at 1 kHz via EtherCAT, confirming real-time feasibility for industrial deployment. This work establishes a complete, replicable toolchain for AI-enhanced robotic control, from algorithm conception to physical execution, and offers a practical reference for researchers and engineers seeking to integrate data-driven methods into manufacturing automation systems.

Acknowledgement

This study is funded by International Science and Technology Cooperation Project of the Department of Science and Technology of Henan Province (252102520038), Key Scientific Research Project of Higher Education Institutions in Henan Province (26B460027), and Key Disciplines of Henan Province (New Round) in Mechanical Engineering (No. [2023]414).

References

[1] Siciliano B, Sciavicco L, Villani L, Oriolo G. Robotics: Modelling, Planning and Control. London: Springer; 2009. DOI: 10.1109/MRA.2009.934833

[2] Nof SY, ed. Handbook of Industrial Robotics. 2nd ed. New York: John Wiley & Sons; 1999. DOI: 10.1002/9780470172506

[3] Xiao G, Ma Y, Li Y. Energy-efficient trajectory optimization for planetary surface manipulators via heuristic multi-objective particle swarm optimization algorithm. Robotica. 2025;43(4):1411-1432. DOI: 10.1017/S026357472500027X

[4] Ren H, Gui H, Zhong R. Trajectory optimization of rapid space debris capture based on reinforcement learning. IFAC-PapersOnLine. 2025;59(20):923-927. DOI: 10.1016/j.ifacol.2025.11.271

[5] Tang W, Ma J, Wei Z, Gao H. Active disturbance rejection control of hypersonic vehicles based on modified deep deterministic policy gradient. Int J Dyn Control. 2026;14(2):1-15. DOI: 10.1007/s40435-026-02008-1

[6] Baltes J, Akbar I, Saeedvand S. A dual-actor proximal policy optimization algorithm for humanoid robot navigation control. Appl Soft Comput. 2026;196:122034. DOI: 10.1016/j.asoc.2026.115093

[7] Chen Y, Li M, Zhang Y, et al. Application of a physics-informed neural network hybrid framework to inverse kinematics of underwater soft manipulators. Ocean Eng. 2026;357:119823. DOI: 10.1016/j.oceaneng.2026.125432

[8] Ortiz JS, Masapanta MA, Andaluz VH, Carvajal CP. A hardware-in-the-loop and 3D simulation framework for active learning in engineering education. Comput Appl Eng Educ. 2025;33(6):e30012. DOI: 10.1002/cae.70101

[9] Ojeda-Mancera S, Carranco-Martinez J, Sámano-Ortega V, et al. Didactic hardware-in-the-loop platform: a low-cost open-source approach. IEEE Lat Am Trans. 2025;23(10):910-921. DOI: 10.1109/TLA.2025.11150634

[10] Corke P. Robotics, Vision and Control: Fundamental Algorithms in MATLAB. 2nd ed. Cham: Springer; 2017.

[11] Koenig N, Howard A. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: Proc. IEEE/RSJ Int Conf Intell Robots Syst. 2004;2149-2154.

[12] Todorov E, Erez T, Tassa Y. MuJoCo: A physics engine for model-based control. In: Proc. IEEE/RSJ Int Conf Intell Robots Syst. 2012;5026-5033. DOI: 10.1109/IROS.2012.6386109

[13] [Gasparetto A, Zanotto V. A technique for time-jerk optimal planning of robot trajectories. Robot Comput-Integr Manuf. 2010;26(5):477-482. DOI:10.1016/j.rcim.2007.04.001

[14] Duka AV. Neural network based inverse kinematics solution for trajectory tracking of a robotic arm. Procedia Technol. 2014;12:20-27. DOI: 10.1016/j.protcy.2013.12.451

[15] Song S, Kang D, Park CE. Safety-aware optimal control with language-guided online parameter adjustment via large language models. IEEE Access. 2026;14:25551-25563. DOI: 10.1109/ACCESS.2026.3664145

[16] Shin KG, McKay ND. Minimum-time control of robotic manipulators with geometric path constraints. IEEE Trans Autom Control. 1985;30(6):531-541. DOI: 10.1109/TAC.1985.1104009

[17] Chou, Po-Wei; Maturana, Daniel; Scherer, Sebastian. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. 34th International Conference on Machine Learning, ICML 2017, v 2, p 1386-1396, 2017, 34th International Conference on Machine Learning, ICML 2017.

[18] Guez A, Ahmad Z. Solution to the inverse kinematics problem in robotics by neural networks. In: Proc IEEE Int Conf Neural Networks. 1988:617-624. DOI: 10.1109/icnn.1988.23979