IAES International Journal of Robotics and Automation (IJRA)
Vol. 15, No. 4, December 2026, pp. 832~845
ISSN: 2722-2586, DOI: 10.11591/ijra.v15i4.pp832-845

Performance optimization of overclocked Teensy 4.1 for real-time reinforcement learning on OpenManipulator-X

Bakhtiar Alldino Ardi Sumbodo, Jazi Eko Istiyanto, Andi Dharmawan
Department of Computer Science and Electronics, Universitas Gadjah Mada, Yogyakarta, Indonesia

Article Info

Article history:

Received Feb 6, 2026
Revised Aug 13, 2026
Accepted Nov 10, 2026

Keywords:

Embedded reinforcement learning
Industrial robotics
Micro-ROS
On-device inference
Real-time manipulation
Teensy 4.1

ABSTRACT

Real-time reinforcement learning (RL) inference for robotic manipulation requires low-latency policy execution, reliable actuator communication, and sufficient computational headroom close to the sensor–actuator layer. Although RL has advanced robotic manipulation, practical deployment commonly separates computationally intensive training from real-time control and often retains inference on host-class processors. This study investigates a different design point: microcontroller-level performance optimization for on-device RL inference in a physical robotic manipulator. An OpenManipulator-X is controlled by a Teensy 4.1 microcontroller operated at the stock 600 MHz frequency and an actively cooled overclocked frequency of 816 MHz, with bidirectional logic-level conversion and Micro-ROS integration with a host-side ROS 2 environment. A soft actor-critic with conservative Q-learning and synaptic intelligence (SAC-CQL-SI) policy is trained offline on the host workstation, after which the frozen actor is deployed to the Teensy 4.1 for local forward-pass inference, safety-constrained joint-command generation, and direct Dynamixel communication. Under identical manipulation conditions, the 816 MHz configuration shortens task-level response time by approximately 40–43% across the evaluated joint configurations, reduces the number of corrective policy-action steps, and improves manipulation success, with the best measured result reaching 97±2%. The novelty lies not in overclocking alone, but in integrating controlled microcontroller-level optimization with embedded continuous-control RL inference inside the physical sensor–inference–actuator loop. The results provide proof-of-concept evidence that carefully managed microcontroller unit (MCU)-level optimization can extend the practical capability of low-cost embedded controllers for robotic edge control, while thermal margin, communication reliability, and long-term hardware durability remain important deployment constraints.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Jazi Eko Istiyanto
Department of Computer Science and Electronics, Universitas Gadjah Mada
Bulaksumur 21, Yogyakarta 55281, Indonesia
Email: jazi@ugm.ac.id

1. INTRODUCTION

Industrial robotic manipulators are increasingly expected to operate as responsive, sensor-rich, and networked components within modern manufacturing systems. In smart manufacturing and collaborative robotics, robot controllers must integrate sensing, cognition, prediction, and real-time response to support safe and efficient operation in heterogeneous production environments [1]–[3]. These requirements are especially important in flexible assembly, pick-and-place, inspection, sorting, and small-batch production, where robot motion must adapt to changing object positions, operator interaction, and production constraints.

In such systems, the control architecture must not only compute motion commands but also process sensor feedback, maintain actuator communication, and respond within a strict real-time budget.

Reinforcement learning (RL) provides a promising approach for robotic manipulation because a policy can be learned from interaction, demonstrations, simulation, offline data, or sequential task experience [4]–[6]. Continuous-control actor–critic algorithms are especially relevant to manipulators because the commanded variables are inherently continuous. Offline RL is also attractive for physical robotics because unrestricted exploratory interaction can be slow, mechanically costly, and unsafe. Nevertheless, training and real-time deployment impose very different computational requirements.

Modern deep RL has benefited from high-performance parallel computing, including graphics processing units (GPUs), and the prevailing robotic workflow therefore performs training—and often inference—on host-class computers. Hardware-oriented RL and TinyRL/TinyML studies show that inference can be moved toward lower-power edge devices, but microcontrollers remain constrained by memory, arithmetic throughput, and real-time peripheral workloads [7]–[11]. A specific research gap therefore remains at the intersection of robotic manipulation, embedded RL inference, and microcontroller-level performance optimization: comparatively few studies evaluate a continuous-control actor executing directly on a general-purpose microcontroller that simultaneously processes robot state and communicates with physical actuators, and fewer still experimentally isolate how additional microcontroller unit (MCU) computational headroom affects the complete sensor–inference–actuator loop.

The Robotis OpenManipulator-X provides a suitable physical platform for this investigation because it combines multiple Dynamixel actuators, position feedback, serial communication, and compatibility with the robot operating system 2 (ROS 2) ecosystem. Its standard OpenCR controller uses an STM32F746ZGT6 Arm Cortex-M7 device operating at 216 MHz. The Teensy 4.1 instead uses an NXP i.MX RT1062 Arm Cortex-M7 processor operating at 600 MHz in its stock configuration. In this work, the Teensy 4.1 is evaluated at both 600 MHz and an actively cooled 816 MHz configuration while communicating with the Dynamixel network through bidirectional logic-level conversion at 1 Mbps.

The originality of this study does not lie in microcontroller overclocking as an isolated hardware technique. Frequency scaling is already known to influence execution latency, energy consumption, and thermal behavior [12]. The novelty lies in integrating controlled MCU-level optimization with embedded RL inference on a physical manipulator and experimentally relating the additional computational headroom to robot-level response time, corrective policy actions, manipulation success, thermal stability, and actuator communication. This positions the work in the context of real-time robotic control rather than synthetic microcontroller benchmarking alone.

The computational architecture deliberately separates learning from deployment. SAC-CQL-SI training is executed offline on a host workstation, while only the trained actor network is frozen and deployed to the Teensy 4.1. The microcontroller performs state handling, actor forward-pass inference, safety-constrained joint-command generation, and direct communication with the Dynamixel actuator bus. The host remains responsible for training, monitoring, logging, and visualization. Micro-ROS connects the embedded controller to ROS 2 without moving the critical policy-inference path back to the workstation [13], [14]. Robot-centric sensing is central to this architecture because local policy execution is useful only when the controller can obtain sufficiently recent state information and issue actuator commands within the control-loop budget. Heterogeneous sensor fusion is therefore identified as a future extension rather than a capability demonstrated by the current prototype [15], [16].

Accordingly, this study has four objectives: i) to develop a Teensy 4.1-based edge-control architecture that combines local RL inference, Micro-ROS connectivity, and direct Dynamixel communication; ii) to demonstrate an offline-training/on-device-inference pipeline in which the trained SAC-CQL-SI actor executes on the microcontroller without online gradient updates; iii) to quantify the effect of increasing the same Teensy 4.1 controller from 600 to 816 MHz on policy-action steps, task response time, and manipulation success under identical physical conditions; and iv) to assess the engineering trade-offs among computational headroom, thermal stability, actuator communication reliability, and practical deployment constraints. The quantitative claims are restricted to the tested Teensy 4.1 configurations and task conditions. Direct OpenCR/GPU benchmarks, heterogeneous sensors, convergence-speed characterization, dedicated throughput measurements, long-duration thermal qualification, and industrial certification remain outside the current experimental scope.

The remainder of the paper is organized as follows. Section 2 presents the proposed method and RL formulation. Section 3 describes the hardware, communication interface, training/deployment separation, and experimental protocol. Section 4 reports and interprets the experimental results, followed by limitations and future work. Section 5 summarizes the validated contribution and its practical scope.

2. PROPOSED METHOD

2.1. Embedded RL control architecture

The proposed system separates computationally intensive reinforcement learning from time-critical robotic control. The host workstation performs offline policy training, whereas the Teensy 4.1 serves as the embedded inference and actuator-control device. Modern actor–critic training requires replay-buffer management, gradient computation, critic updates, and repeated neural-network optimization, which exceed the practical computational and memory budget of a general-purpose microcontroller. Once training is complete, however, the actor network requires only forward-pass inference and can therefore be deployed on a resource-constrained embedded controller.

The Teensy 4.1 is based on the NXP i.MX RT1062 Arm Cortex-M7 processor and operates at 600 MHz under its stock configuration. In this work, the controller is also evaluated at 816 MHz. Increasing the clock frequency does not modify the RL algorithm or the trained policy parameters. Instead, it increases the computational headroom available for actor inference, state handling, joint-command generation, and surrounding firmware operations. The central hypothesis is that reducing processor-side execution time can shorten the closed-loop response of the sensor–inference–actuator path during robotic manipulation.

The embedded controller remains connected to the ROS 2 environment through Micro-ROS. The host is used for offline training, monitoring, telemetry recording, and visualization, whereas the Teensy 4.1 performs state processing, actor-network inference, safety-constrained action mapping, and direct communication with the Dynamixel actuator bus. The time-critical policy action is therefore generated locally rather than returned to the host. Figure 1 summarizes the resulting architecture.

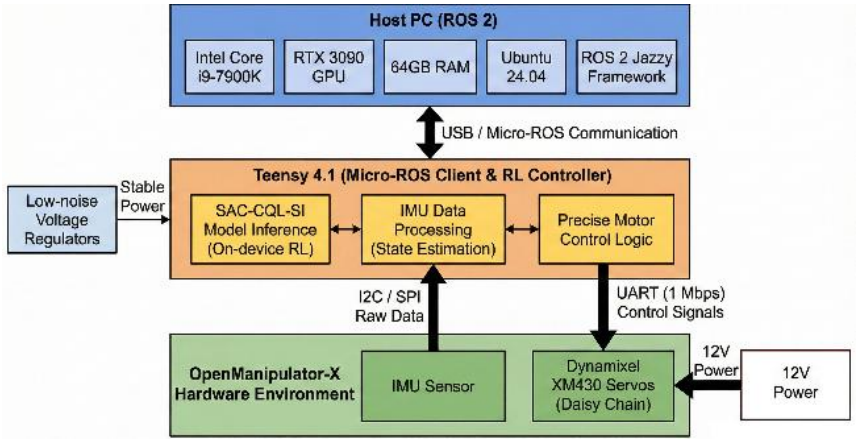

Figure 1. Embedded RL control architecture showing host-side offline training, Teensy 4.1 on-device policy inference, Micro-ROS/ROS 2 integration, and local OpenManipulator-X actuator control

2.2. Reinforcement-learning environment formulation

The manipulation task is formulated as a continuous-state, continuous-action reinforcement-learning problem. Each episode represents a reach-and-approach task in which the OpenManipulator-X end effector moves from a predefined home configuration toward a target position within the robot workspace. The policy receives the current robot configuration and target information and generates continuous commands for the four arm joints involved in positioning the end effector. The gripper is not included in the learned four-dimensional action vector.

a. State space

At time step t, the state is represented by the ten-dimensional continuous vector defined in (1). The vector combines four current joint angles, the current Cartesian end-effector position, and the Cartesian target position.

$$s_t = \left[\theta_1, \theta_2, \theta_3, \theta_4, p_{ee}^x, p_{ee}^y, p_{ee}^z, p_{target}^x, p_{target}^y, p_{target}^z\right]^T \quad (1)$$

In (1), θ_1–θ_4 are the current angles of the four positioning joints; p_{ee}^x, p_{ee}^y, and p_{ee}^z are the current Cartesian coordinates of the end effector; and p_{target}^x, p_{target}^y, and p_{target}^z are the target coordinates. The state therefore contains both proprioceptive configuration information and task-level Cartesian information.

b. Action space and firmware safety mapping

The actor produces the normalized four-dimensional continuous action vector shown in (2).

$$a_t = \left[a_{1,t}, a_{2,t}, a_{3,t}, a_{4,t}\right]^T, \quad a_t \in [-1,1]^4 \tag{2}$$

The normalized action in (2) is not transmitted directly to the Dynamixel servos. Each component is converted by the Teensy 4.1 firmware into a bounded target joint position using (3).

$$q_{i,t}^{target} = q_i^{min} + \left(\frac{a_{i,t}+1}{2}\right)\left(q_i^{max} - q_i^{min}\right) \tag{3}$$

The variables q_i^{min} and q_i^{max} in (3) are the firmware-enforced safety limits for joint i. The specific experimental limits are summarized compactly in (4).

$$q_1 \in [45°, 135°], \quad q_2 \in [0°, 60°], \quad q_3 \in [-30°, 60°], \quad q_4 \in [-120°, 90°] \tag{4}$$

Equation (4) ensures that extreme normalized actor outputs remain inside the experimentally defined motion envelope before position commands reach the actuator bus.

c. Reward function

The reward formulation was adapted to the OpenManipulator-X reach-and-approach task rather than directly inherited from the RoboVerse tasks used by Yadav *et al.* [6], [17]. In the reference SAC-CQL-SI implementation, task rewards are supplied by the environment and are primarily sparse success indicators. For the present reach-and-approach formulation, the instantaneous Cartesian distance is defined in (5).

$$d_t = \|p_{ee,t} - p_{target}\|_2 \tag{5}$$

Using the distance d_t from (5), the reward is defined by (6).

$$R_t = -w_1 d_t + w_2 \mathbb{1}(d_t < 10 \text{ mm}) \tag{6}$$

In (6), w_1 controls the dense distance penalty and w_2 controls the target-reaching incentive. These coefficients are task-specific weights used in this study and are not hyperparameters that can be inferred from the Yadav *et al.* repository [18].

2.3. SAC-CQL-SI learning framework

The learning framework combines soft actor-critic (SAC) [19], conservative Q-learning (CQL) [20], and synaptic intelligence (SI) [21]. The learning algorithm itself is not modified in this study; the contribution is the host-training/microcontroller-inference separation and the experimental characterization of embedded execution. For a replay-buffer transition (s, a, r, s'), the soft Bellman target used by SAC is defined in (7).

$$y = r + \gamma \mathbb{E}_{a' \sim \pi_\phi(\cdot|s')}\left[\min_i \bar{Q}_{\bar{\theta}_i}(s', a') - \alpha \log \pi_\phi\,(a'|s')\right] \tag{7}$$

The discount factor γ and entropy temperature α in (7) determine temporal credit assignment and entropy regularization. Each critic then minimizes the mean-squared Bellman error in (8).

$$L_{\text{critic}}(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_{\theta_i}(s,a) - y\right)^2\right] \tag{8}$$

The critic objective in (8) is optimized jointly with the stochastic actor. The SAC actor objective is given in (9).

$$J_{\text{actor}}(\phi) = \mathbb{E}_{s\sim\mathcal{D}, a\sim\pi_\phi}\left[\min_i Q_{\theta_i}(s,a) - \alpha \log \pi_\phi\,(a|s)\right] \tag{9}$$

Equation (9) encourages high expected value while retaining entropy. For offline training, CQL adds the conservative regularization term shown in (10).

$$L_{\text{CQL}}(\theta_i) = \alpha_{\text{CQL}} \mathbb{E}_{s\sim\mathcal{D}}\left[\log \int \exp\left(Q_{\theta_i}(s,a))\, da - \mathbb{E}_{a\sim\pi_\beta(\cdot|s)} Q_{\theta_i}(s,a)\right)\right] \tag{10}$$

In (10), π_β denotes the behavior policy implicit in the offline dataset and α_{CQL} controls the strength of conservative regularization. For sequential learning, SI first computes the parameter displacement for parameter j after task k according to (11).

$$\Delta\theta_j^{(k)} = \theta_j^{(k,\text{end})} - \theta_j^{(k,\text{start})} \tag{11}$$

The contribution of parameter j to loss reduction is accumulated according to (12).

$$\omega_j^{(k)} = \sum_t \left[-g_j(t)\Delta\theta_j(t)\right] \tag{12}$$

The accumulated contribution in (12) is normalized to obtain the importance value in (13).

$$\Omega_j^{(k)} = \frac{\omega_j^{(k)}}{\left(\Delta\theta_j^{(k)}\right)^2 + \xi} \tag{13}$$

Using the importance score from (13), SI adds the consolidation penalty in (14) during subsequent training tasks.

$$L_{\text{SI}} = c\sum_j \Omega_j \left(\theta_j - \theta_j^*\right)^2 \tag{14}$$

In (14), c is the consolidation strength, ξ in (13) is the damping constant, and θ_j^* is the stored parameter value from previous tasks. Equations (7)–(14) are evaluated only during host-side offline training. After training, critic networks, replay buffers, optimizer states, and gradient computations are excluded from deployment; only the frozen actor parameters are exported to the Teensy 4.1.

2.4. Training configuration and hyperparameters

The SAC-CQL-SI policy was trained on the host workstation using the reference implementation of Yadav *et al.* [6]. The principal hyperparameters were retained to preserve reproducibility and to avoid presenting the contribution as a modification of the RL algorithm. SAC used $\gamma = 0.99$, $\tau = 5\times10^{-3}$, a policy learning rate of 1×10^{-4}, twin Q-function learning rates of 3×10^{-4}, automatic entropy-temperature tuning, and a 10,000-step policy warm-up. CQL used a temperature of 1.0, min-Q weight of 5.0, one random action sample per state, CQL(H) regularization beginning after epoch 5, and the Lagrange variant disabled. SI used $c = 1.0$ and $\xi = 1\times10^{-3}$. Training proceeded for up to 100 epochs per task, with 1,000 gradient-update steps per epoch, mini-batch size 256, and the Adam optimizer. The workstation used an Intel Core i9-7900K, 64 GB RAM, and an NVIDIA RTX 3090 GPU.

2.5. Offline training and hardware-safe deployment

Training and physical testing were strictly separated. Actor and critic parameters were optimized only during offline training. After the training stage, the actor parameters were frozen; no online gradient update, critic optimization, replay-buffer update, CQL regularization, or SI update occurred during physical testing. During deployment, the microcontroller reads the current state, performs one actor forward pass, maps the normalized policy output through the firmware safety limits in (3) and (4), and transmits the resulting position commands to the Dynamixel network. The same frozen actor is used for both 600 MHz and 816 MHz tests.

This separation also reduces hardware risk because unrestricted exploratory RL actions are not executed on the physical manipulator. Exploratory actions can be mechanically costly and may transiently enter undesirable parts of the state-action space. The present workflow therefore confines computationally intensive policy optimization to the host and uses only the trained actor during physical evaluation.

2.6. Micro-ROS and ROS 2 integration

Micro-ROS connects the Teensy 4.1 embedded controller to the host-side ROS 2 environment while preserving the separation between supervisory functions and the time-critical control path. The workstation runs Ubuntu 24.04 and ROS 2 Jazzy and is responsible for visualization, telemetry recording, experiment supervision, and offline policy training. During deployment, the host does not compute the policy action; actor inference and Dynamixel command transmission remain local to the Teensy 4.1.

3. METHOD

3.1. Hardware setup and overclocking configuration

The physical platform consists of a Robotis OpenManipulator-X, a Teensy 4.1 controller based on the NXP i.MX RT1062 Arm Cortex-M7 processor, a ROS 2 host workstation, and a Micro-ROS communication layer. The OpenManipulator-X uses Dynamixel XM430-W350-T actuators. The Teensy 4.1 performs state processing, actor inference, safety-constrained joint-command generation, and direct actuator communication.

Two operating frequencies were used for the controlled physical comparison: the standard 600 MHz configuration and an overclocked 816 MHz configuration. Preliminary frequency screening also considered 720, 912, 960, and 1,008 MHz. The higher-frequency configurations were not retained because the tested system exhibited slower or inconsistent servo behavior despite higher processor benchmark scores. The 816 MHz setting was therefore selected as the highest tested configuration that provided the desired computational gain while maintaining stable manipulator operation and Dynamixel communication.

Clock frequency was configured through the Teensyduino development environment. The current working revision records that no manual CPU core-voltage override was applied in the experimental firmware and that core-voltage management associated with the selected clock configuration was left to the Teensy/Teensyduino clock-management implementation. This statement should be verified against the final firmware configuration before submission. Frequency and voltage operating conditions are closely related to processor power and thermal behavior [12].

The 816 MHz configuration used a stable regulated power supply, a compact heatsink, and a 12 V PWM-controlled brushless fan. Active cooling was treated as part of the experimental overclocked configuration rather than as an optional accessory. Preliminary acceptance of the 816 MHz setting required stable firmware execution, physical manipulation, and communication with the Dynamixel actuators under the tested laboratory conditions. Figure 2 shows the physical and communication layout.

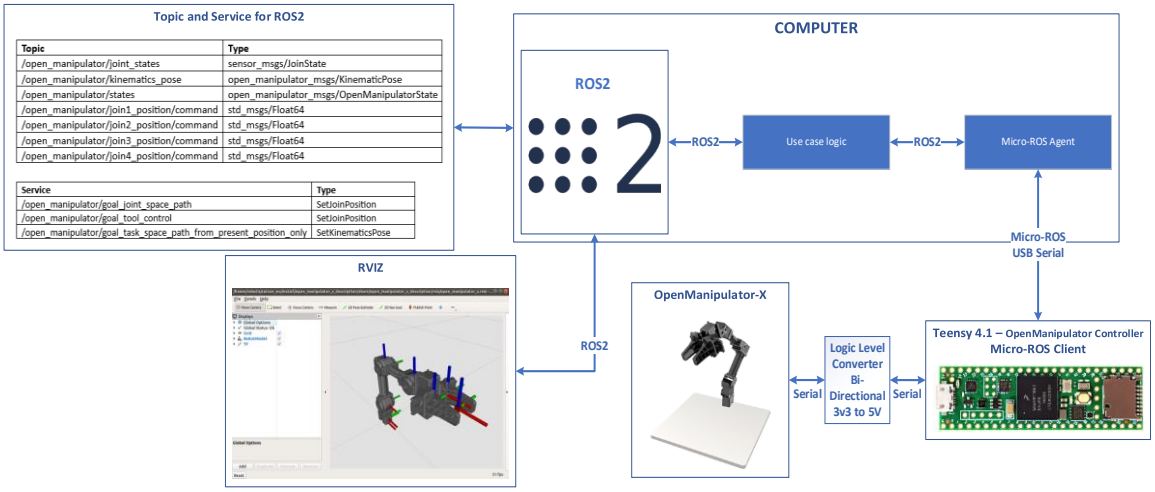

Figure 2. Physical OpenManipulator-X platform and ROS 2/Micro-ROS communication path used for local actor inference and experimental monitoring

3.2. Hardware safety, reliability, and sustainability considerations

Overclocking was applied only as a controlled research configuration. Increasing clock frequency provides additional processor throughput but can also increase power demand and thermal stress. Hardware risk was reduced through bounded joint commands, regulated power, and active cooling. The RL policy cannot directly command positions outside the firmware limits in (4), and unrestricted online RL exploration was not performed on the physical robot.

The 816 MHz configuration should be interpreted as a laboratory proof of concept rather than a production-qualified operating condition. Long-duration thermal cycling, continuous-duty stress testing, semiconductor-lifetime characterization, energy-per-inference measurements, actuator-bus error-rate characterization, electromagnetic compatibility testing, and industrial functional-safety certification were outside the experimental scope. Accordingly, this study makes no claim that overclocking improves energy efficiency or hardware lifespan.

The study involved no human or animal subjects. Ethical considerations therefore concern responsible hardware operation, avoidance of unnecessary mechanical stress, transparent reporting of the non-standard processor configuration, and avoidance of unsupported claims regarding long-term reliability or sustainability.

3.3. Actuator and communication implementation

The Teensy 4.1 communicates directly with the Dynamixel XM430-W350-T actuators through a UART interface operating at 1 Mbps. The servos use a shared Dynamixel daisy-chain bus [22], [23]. Because the Teensy 4.1 uses 3.3 V logic while the Dynamixel TTL interface uses 5 V signaling, a bidirectional logic-level converter is inserted between the microcontroller and the actuator network. The Dynamixel2Arduino library is used for packet generation, command transmission, and actuator-status acquisition [24]. During each control iteration, the embedded firmware obtains the required state variables, evaluates the frozen actor, applies the safety mapping in (3) and (4), and transmits the resulting joint targets to the actuator network [25].

3.4. Experimental protocol

The OpenManipulator-X has five degrees of freedom when the gripper is included. The RL controller operates Joints #1–#4 for end-effector positioning. Figure 3 shows the manipulator dimensions, whereas Figure 4 shows the five-DoF joint configuration. Each trial begins from the same predefined home configuration and evaluates reach-and-approach motion toward a target within a defined 0.4×0.4×0.4 m workspace.

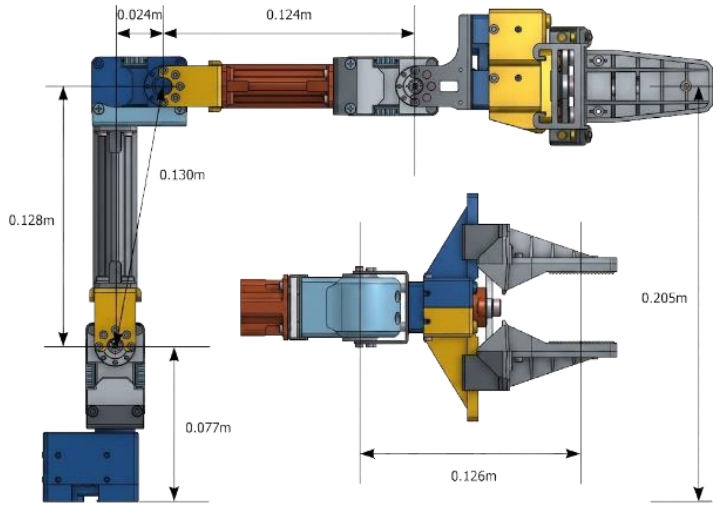

Figure 3. OpenManipulator-X dimensions

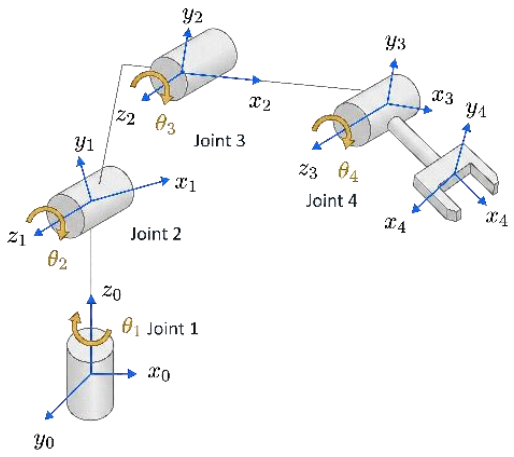

Figure 4. Five-DoF OpenManipulator-X joint configuration

Three target regions (Areas 1–3) and three joint-subset conditions are evaluated: Joints {#1, #2, #3}; Joints {#1, #2, #4}; and Joints {#1, #2, #3, #4}. The Cartesian target-coordinate ranges are summarized in Table 1 and their spatial locations relative to the manipulator are illustrated in Figure 5. Each combination of clock configuration, target region, and joint-subset condition is repeated for 10 independent trials, producing 180 physical trials in total. The same frozen actor, initial configuration, workspace, action mapping, joint limits, Dynamixel baud rate, and success criterion are used for both clock frequencies.

Table 1. Target-coordinate ranges for the three object areas used in the OpenManipulator-X manipulation experiments

Area	x	y	z
Area1	(-0.6, -0.5)	(-0.2, 0.2)	(0.1, 0.5)
Area2	(-0.5, -0.4)	(-0.2, 0.2)	(0.1, 0.5)
Area3	(-0.3, -0.2)	(-0.2, 0.2)	(0.1, 0.5)

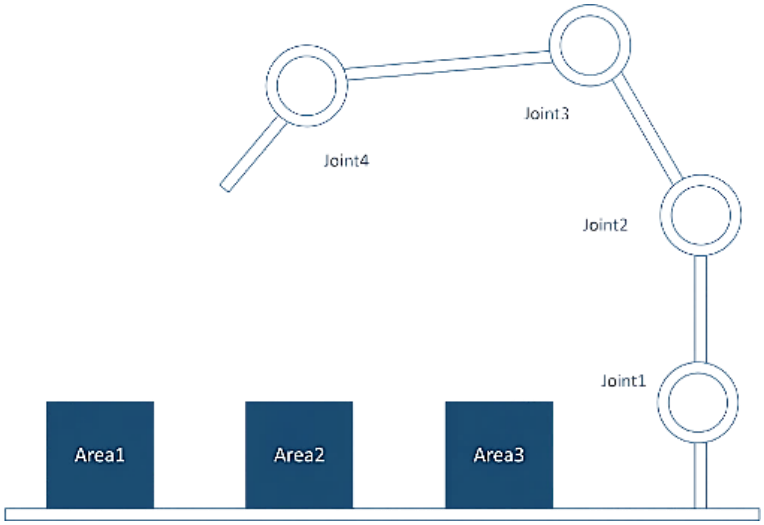

Figure 5. Spatial locations of Areas 1–3 relative to the OpenManipulator-X workspace used in the target-region experiments

After defining the workspace and target regions, a physical trial is classified as successful using the Cartesian condition in (15).

$$\|p_{ee} - p_{target}\|_2 < 10 \text{ mm} \tag{15}$$

Equation (15) must remain true for at least three consecutive control-loop cycles before the trial is counted as successful. The protocol records the number of policy-action steps, wall-clock episode response time with millisecond resolution (verified against the Teensy 4.1 cycle counter), and a binary success indicator. The comparison therefore isolates the practical effect of changing the same Teensy 4.1 controller from 600 MHz to 816 MHz under otherwise identical task conditions. Unless otherwise stated, all quantitative values reported using the ± notation are expressed as mean ± standard deviation (SD). The SD characterizes trial-to-trial variability under each experimental condition and should not be interpreted as a standard error or confidence interval [26], [27].

4. RESULTS AND DISCUSSION

The experiments evaluated whether additional microcontroller computational headroom improves execution of a frozen RL policy during physical OpenManipulator-X manipulation. The controlled comparison uses the same Teensy 4.1, trained actor, task definition, actuator interface, and success criterion at 600 MHz and 816 MHz. Accordingly, the observed differences concern deployment-time execution and should not be interpreted as changes in RL training, policy convergence, or learned policy quality.

4.1. Processor headroom and embedded execution timing

As shown in Figure 6, increasing the Teensy 4.1 clock from 600 to 816 MHz increased the measured CoreMark score from 2,400.77 to 3,265.04. The 36% increase in processor frequency therefore produced an approximately proportional increase in synthetic computational performance. For one embedded policy update, the execution time can be represented conceptually by (16).

$$T_{\text{embedded}} = T_{\text{state}} + T_{\text{actor}} + T_{\text{mapping}} + T_{\text{comm}} + T_{\text{other}} \quad (16)$$

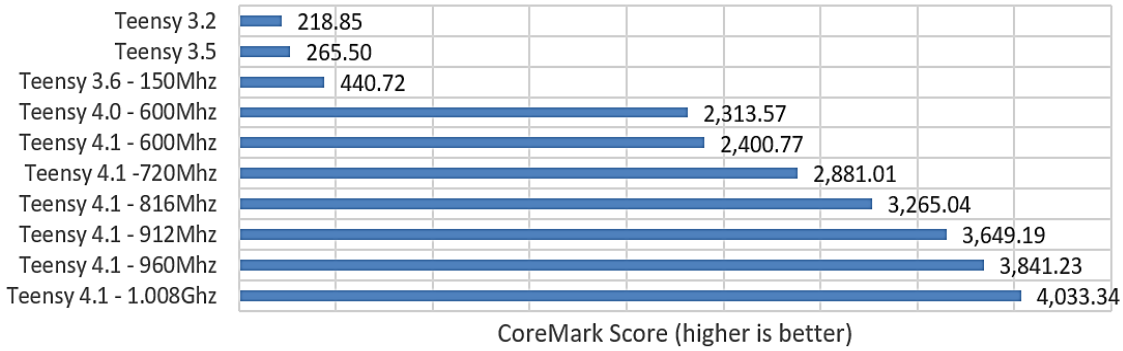

Figure 6. CoreMark benchmark comparison across Teensy platforms and Teensy 4.1 clock configurations

In (16), T_{state} is state-processing time, T_{actor} is actor forward-pass time, T_{mapping} is the normalized-action-to-safe-joint-command conversion time, T_{comm} is actuator-communication latency, and T_{other} contains remaining software overhead. Increasing CPU frequency primarily reduces the processor-bound components T_{state}, T_{actor}, and T_{mapping}. Serial communication and physical actuator dynamics do not necessarily scale with processor frequency.

The measurements reported in Tables 2 to 5 are task-level episode response times rather than separately instrumented actor-inference or communication latencies. Consequently, the results demonstrate an improvement in end-to-end manipulation response, while the contribution of each latency component in (16) cannot be quantified independently from the present experiment.

4.2. Effect on manipulation performance

Table 2 summarizes the controlled comparison across the three joint-subset configurations. Increasing the clock frequency from 600 to 816 MHz reduced the average number of policy-action steps by approximately 18–22% and shortened task-level response time by approximately 40–43%. Manipulation success also increased in each joint configuration.

Because the same frozen actor was used in both configurations, the improvement does not indicate a better learned policy. Instead, the higher clock frequency provides additional computational capacity for executing the existing policy and its associated state-processing and command-generation operations.

A plausible control-level interpretation is that shorter processor-side delay allows subsequent policy actions to be generated from more recent robot-state information. Near the target, this can reduce the interval during which the robot continues to move before the next corrective action is issued. The simultaneous reduction in episode duration and policy-action count is consistent with this interpretation. However, because actor inference, serial communication, and mechanical settling were not instrumented separately, their causal contributions cannot be isolated.

Table 2. Task-level manipulation performance at 600 and 816 MHz for the evaluated OpenManipulator-X joint configurations. Data are reported as mean ± SD

Experiment	Joints	Average Steps		Average Time (s)		Success Rate (%)	
		600 MHz	816 MHz	600 MHz	816 MHz	600 MHz	816 MHz
1	#1, #2, #3	48 ± 5	39 ± 6	4.2 ± 0.6	2.4 ± 0.6	88 ± 4	91 ± 3
2	#1, #2, #4	60 ± 5	49 ± 7	5.0 ± 0.3	3.0 ± 0.6	85 ± 5	88 ± 3
3	#1, #2, #3, #4	49 ± 5	38 ± 7	4.3 ± 0.3	2.6 ± 0.3	90 ± 3	93 ± 3

4.3. Target-region effects and kinematic limits

Table 3 reports the Area 1 results, Table 4 reports the Area 2 results, and Table 5 reports the Area 3 results. Taken together, these tables show that the effect of additional MCU performance depends on the mechanical characteristics of the task and cannot be interpreted independently of manipulator reachability.

Table 3. Manipulation performance in target Area 1, showing the effect of MCU operating frequency under an accessible workspace condition. Data are reported as mean ± SD.

Experiment	Joints	Average Steps		Average Time (s)		Success Rate (%)	
		600 MHz	816 MHz	600 MHz	816 MHz	600 MHz	816 MHz
1	#1, #2, #3	38 ± 4	28 ± 4	1.9 ± 1.5	1.7 ± 1.8	90 ± 4	93 ± 3
2	#1, #2, #4	123 ± 4	109 ± 4	6.2 ± 1.7	4.0 ± 2.0	87 ± 4	90 ± 4
3	#1, #2, #3, #4	35 ± 4	25 ± 4	2.0 ± 1.0	1.4 ± 1.0	92 ± 3	95 ± 3

As shown in Table 4, Area 2 produced the strongest overall performance. For the four-joint condition, increasing the clock frequency from 600 MHz to 816 MHz reduced the average action count from 40 ± 3 to 25 ± 2 and the task response time from 2.0 ± 1.0 s to 1.2 ± 1.2 s, while the reported success rate increased from 93 ± 3% to 97 ± 2%. This combination represents the best measured result in the present experiments.

Table 4. Manipulation performance in target Area 2, where the evaluated configuration produced the highest overall task success. Data are reported as mean ± SD

Experiment	Joints	Average Steps		Average Time (s)		Success Rate (%)	
		600 MHz	816 MHz	600 MHz	816 MHz	600 MHz	816 MHz
1	#1, #2, #3	25 ± 3	18 ± 4	3.0 ± 2.0	1.5 ± 0.5	91 ± 3	95 ± 3
2	#1, #2, #4	35 ± 3	17 ± 5	1.5 ± 1.5	0.8 ± 1.8	92 ± 3	96 ± 3
3	#1, #2, #3, #4	40 ± 3	25 ± 2	2.0 ± 1.0	1.2 ± 1.2	93 ± 3	97 ± 2

As shown in Table 5, Area 3 provides an important counterexample to the assumption that higher processor performance alone determines manipulation success. In the four-joint condition, response time decreased from 3.0 ± 2.0 s to 2.4 ± 1.5 s, whereas success increased only from 70 ± 5% to 74 ± 5%. Additional computational headroom therefore improves execution efficiency but cannot remove limitations imposed by workspace geometry, joint limits, or unfavorable manipulator configurations.

These results indicate two interacting regimes. When a target is mechanically favorable and processor-side computation remains relevant to the control loop, additional MCU performance can translate into faster manipulation and fewer corrective actions. When kinematic constraints dominate, the benefit of additional computation becomes smaller. Embedded computation and manipulator mechanics must therefore be considered jointly.

Table 5. Manipulation performance in target Area 3, illustrating the increasing influence of workspace geometry and kinematic constraints. Data are reported as mean ± SD

Experiment	Joints	Average Steps		Average Time (s)		Success Rate (%)	
		600 MHz	816 MHz	600 MHz	816 MHz	600 MHz	816 MHz
1	#1, #2, #3	69 ± 3	58 ± 3	3.5 ± 1.5	2.3 ± 1.0	67 ± 3	72 ± 3
2	#1, #2, #4	71 ± 3	63 ± 3	4.0 ± 2.0	3.1 ± 1.5	64 ± 3	70 ± 4
3	#1, #2, #3, #4	66 ± 4	54 ± 3	3.0 ± 2.0	2.4 ± 1.5	70 ± 5	74 ± 5

4.4. Thermal stability and practical operating limits

The benefit of increasing processor frequency is not monotonic. Preliminary screening included 720, 912, 960, and 1,008 MHz, but configurations above 816 MHz were not retained for the physical comparison because slower or inconsistent servo behavior was observed under the tested setup. These observations were not instrumented sufficiently to determine whether the limiting mechanism originated from processor operation, thermal conditions, peripheral timing, actuator communication, or their interaction.

The results therefore do not demonstrate that higher processor frequency inherently degrades Dynamixel communication. They establish only that, in this prototype, 816 MHz was the highest tested configuration that combined increased computational performance with stable manipulator operation.

Thermal management is part of this operating condition. The 816 MHz configuration used a compact heatsink and a 12 V PWM-controlled fan. Frequency scaling can alter computational performance, power demand, and thermal behavior [12], [28], while recent studies of AI workloads on MCU-class platforms also identify thermal behavior as an important design consideration. The reported performance should therefore be interpreted as the performance of an actively cooled 816 MHz configuration, not as evidence that the same frequency is suitable for passive or continuous-duty operation.

Long-term hardware reliability remains unresolved. The current experiments did not include junction-temperature logging, thermal cycling, semiconductor-aging characterization, or continuous-duration stress tests. Short-term operational stability therefore does not establish an unchanged hardware lifespan. For embedded robotic control, the appropriate operating point is not the highest attainable processor frequency, but the highest point that satisfies computational latency while preserving thermal margin, communication stability [29], actuator behavior, and acceptable hardware reliability.

4.5. Relevance to robotic automation

The practical significance of the architecture is that the trained continuous-control actor executes locally near the sensor–actuator layer rather than requiring host-side policy inference for every action. The workstation remains necessary for SAC-CQL-SI training, model preparation, logging, and supervision, while the MCU performs deployment-time inference and direct actuator control. This division is consistent with embedded robotic architectures in which microcontrollers perform time-sensitive sensing and actuation while more capable computers execute computationally intensive functions [14].

For cost-sensitive robotic cells, educational manipulators, laboratory automation, and compact handling systems, the results provide a proof of concept that MCU-class hardware can support local RL inference rather than acting only as a low-level command interface. Local execution may also reduce dependence on host communication within the time-critical state-to-action path.

The experimentally supported comparison in this study is restricted to the same Teensy 4.1 at 600 and 816 MHz. OpenCR, host-CPU inference, and GPU inference were not evaluated using identical tasks and measurement procedures and therefore should not be interpreted as experimentally validated performance baselines.

4.6. Limitations and Future Work

This study is limited to a single Teensy 4.1/OpenManipulator-X platform and a predefined reach-and-approach task. Although three workspace regions and several joint configurations were evaluated, the results do not establish generalization to different robot morphologies, payloads, end effectors, or industrial manipulation tasks.

First, the present study does not provide a task-identical performance comparison with OpenCR, host-CPU inference, or GPU-based RL inference. Future work should benchmark these platforms using the same frozen actor, state/action definitions, actuator conditions, and timing methodology. Such experiments should separately measure actor-inference latency, end-to-end control latency, jitter, power consumption, implementation cost, and sustainable control throughput, with the results summarized in a dedicated throughput table.

Second, this work evaluates deployment of an already trained SAC-CQL-SI actor rather than RL learning efficiency. Training-convergence histories were therefore outside the present experimental analysis. Future training campaigns should record episode return, actor and critic losses, task success, and convergence time to generate convergence-speed graphs and distinguish training efficiency from MCU inference efficiency.

Third, the current state representation does not incorporate heterogeneous sensing or controlled sensor degradation. Future experiments should integrate encoder, inertial, force/torque, and vision measurements through appropriate sensor-fusion methods. Robustness should then be evaluated using controlled sensor-noise stress tests, including increasing measurement noise, delayed or dropped data, calibration errors, actuator disturbances, and intermittent communication.

Fourth, the study does not establish the long-term reliability of the overclocked MCU. Future qualification should include continuous-duty operation, thermal cycling, junction-temperature monitoring, power and energy-per-inference measurements, communication-error logging, and long-duration hardware stress testing to quantify the trade-off between computational performance and hardware lifetime.

Finally, the control architecture should be extended to higher-DoF and multi-axis manipulators, followed by multi-agent or multi-robot configurations in which embedded controllers execute local policies under supervisory coordination. Validation in representative industrial environments will be required to determine reliability, safety, maintainability, and manufacturing-level performance.

5. CONCLUSION

This study provides a proof-of-concept validation of MCU-level performance optimization for on-device continuous-control RL on a physical OpenManipulator-X. SAC-CQL-SI training was performed offline on a host workstation, whereas the frozen actor was deployed to the Teensy 4.1 for local state processing, policy inference, safety-constrained joint-command generation, and direct Dynamixel communication.

Increasing the same controller from 600 MHz to an actively cooled 816 MHz improved deployment-time performance without modifying the learned policy. Across the evaluated joint-subset conditions, the 816 MHz configuration reduced policy-action steps by approximately 18–22% and task-level response time by approximately 40–43%. The best measured condition, obtained in Area 2 with four controlled joints, reached a reported success rate of 97 ± 2%. The smaller improvement in Area 3 confirms that additional MCU performance cannot compensate for unfavorable workspace geometry or kinematic constraints.

The contribution is therefore the integration of controlled MCU-level optimization with local RL inference inside the physical sensor–inference–actuator loop, rather than overclocking as an isolated processor benchmark. Under the tested laboratory conditions, 816 MHz provided useful additional computational headroom while maintaining stable manipulator operation with active cooling. The results support the feasibility of microcontroller-class hardware for cost-sensitive, embedded RL control, while broader platform benchmarking, robustness testing, and long-term hardware qualification remain necessary before industrial deployment.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the Department of Computer Science and Electronics, Universitas Gadjah Mada, for supporting the publication of this work through the 2026 Publication Funding Program.

FUNDING INFORMATION

This work was supported by the Department of Computer Science and Electronics, Universitas Gadjah Mada, under the 2026 Publication Funding Program.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Bakhtiar Alldino Ardi Sumbodo	✓	✓	✓	✓	✓	✓		✓	✓	✓			✓	
Jazi Eko Istiyanto		✓				✓			✓	✓	✓	✓		
Andi Dharmawan	✓		✓	✓			✓			✓	✓			

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

The authors state no conflict of interest.

DATA AVAILABILITY

Data is available upon request.

REFERENCES

[1] M. L. Nicora, R. Ambrosetti, G. J. Wiens, and I. Fassi, "Human–robot collaboration in smart manufacturing: Robot reactive behavior intelligence," *Journal of Manufacturing Science and Engineering*, vol. 143, no. 3, Mar. 2021, doi: 10.1115/1.4048950.

[2] U. Othman and E. Yang, "Human–robot collaborations in smart manufacturing environments: Review and outlook," *Sensors*, vol. 23, no. 12, p. 5663, Jun. 2023, doi: 10.3390/s23125663.
[3] C. J. Conti, A. S. Varde, and W. Wang, "Human-robot collaboration with commonsense reasoning in smart manufacturing contexts," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 3, pp. 1784–1797, Jul. 2022, doi: 10.1109/TASE.2022.3159595.
[4] D. Han, B. Mulyana, V. Stankovic, and S. Cheng, "A survey on deep reinforcement learning algorithms for robotic manipulation," *Sensors*, vol. 23, no. 7, p. 3762, Apr. 2023, doi: 10.3390/s23073762.
[5] N. Alhousani, M. Saveriano, I. Sevinc, T. Abdulkuddus, H. Kose, and F. J. Abu-Dakka, "Geometric reinforcement learning for robotic manipulation," *IEEE Access*, vol. 11, pp. 111492–111505, 2023, doi: 10.1109/ACCESS.2023.3322654.
[6] S. P. Yadav, R. Nagar, and S. V. Shah, "Learning vision-based robotic manipulation tasks sequentially in offline reinforcement learning settings," *Robotica*, vol. 42, no. 6, pp. 1715–1730, Jun. 2024, doi: 10.1017/S0263574724000389.
[7] Liu *et al.*, "Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review," *Robotics*, vol. 10, no. 1, p. 22, 2021, doi: 10.3390/robotics10010022.
[8] A. Shiri, A. N. Mazumder, B. Prakash, H. Homayoun, N. R. Waytowich, and T. Mohsenin, "A hardware accelerator for language-guided reinforcement learning," *IEEE Design & Test*, vol. 39, no. 3, pp. 37–44, Jun. 2022, doi: 10.1109/MDAT.2021.3063363.
[9] E. Manor and S. Greenberg, "Custom hardware inference accelerator for TensorFlow Lite for microcontrollers," *Ieee Access*, 2022, doi: 10.1109/access.2022.3189776.
[10] T. Szydlo, P. P. Jayaraman, Y. Li, G. Morgan, and R. Ranjan, "TinyRL: Towards reinforcement learning on tiny embedded devices," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, Oct. 2022, pp. 4985–4988. doi: 10.1145/3511808.3557206.
[11] B. P. Duisterhof *et al.*, "Tiny Robot Learning (tinyRL) for source seeking on a nano quadcopter," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, May 2021, pp. 7242–7248. doi: 10.1109/ICRA48506.2021.9561590.
[12] J. Zidar, T. Matić, I. Aleksi, and Ž. Hocenski, "Dynamic voltage and frequency scaling as a method for reducing energy consumption in ultra-low-power embedded systems," *Electronics*, vol. 13, no. 5, p. 826, Feb. 2024, doi: 10.3390/electronics13050826.
[13] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, May 2022, doi: 10.1126/scirobotics.abm6074.
[14] Z. Wang, S. Liu, D. Ji, and W. Yi, "Improving real-time performance of Micro-ROS with priority-driven chain-aware scheduling," *Electronics*, vol. 13, no. 9, p. 1658, Apr. 2024, doi: 10.3390/electronics13091658.
[15] F. Chen *et al.*, "Sensor fusion-based anthropomorphic control of a robotic arm," *Bioengineering*, vol. 10, no. 11, p. 1243, Oct. 2023, doi: 10.3390/bioengineering10111243.
[16] Á. Odry, I. Kecskes, P. Sarcevic, Z. Vizvari, A. Toth, and P. Odry, "A novel fuzzy-adaptive extended Kalman filter for real-time attitude estimation of mobile robots," *Sensors*, vol. 20, no. 3, p. 803, Feb. 2020, doi: 10.3390/s20030803.
[17] S. P. Yadav, "Roboverse," 2022. https://github.com/sudhirpratapyadav/roboverse (accessed Mar. 16, 2025).
[18] S. P. Yadav, "SAC-CQL-SI," 2022. https://github.com/sudhirpratapyadav/sac-cql-si (accessed Mar. 16, 2025).
[19] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, 2018, vol. 80, pp. 1861–1870.
[20] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-learning for offline reinforcement learning," in *Advances in Neural Information Processing Systems*, 2020, vol. 33, pp. 1179–1191.
[21] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proceedings of the 34th International Conference on Machine Learning*, 2017, vol. 70, pp. 3987–3995.
[22] R. Jánoš *et al.*, "Stability and dynamic walk control of humanoid robot for robot soccer player," *Machines*, vol. 10, no. 6, p. 463, Jun. 2022, doi: 10.3390/machines10060463.
[23] X. Sun, "Bidirectional multi-device daisy chain communication architecture and verification based on UART serial communication protocol," *Highlights in Science Engineering and Technology*, 2024, doi: 10.54097/tjz7j294.
[24] Robotis, "Dynamixel2Arduino: Arduino library for Dynamixel control," Dynamixel2Arduino GitHub repository, 2024.
[25] L. S. Lim and J. W. Cho, "Implementation of high-speed communication bridge for Dynamixel-based robotic systems," *International Journal of Advanced Robotic Systems*, vol. 15, no. 4, 2018.
[26] D. K. Lee, J. In, and S. Lee, "Standard deviation and standard error of the mean," *Korean Journal of Anesthesiology*, vol. 68, no. 3, p. 220, 2015, doi: 10.4097/kjae.2015.68.3.220.
[27] S. W. Choi and G. T. C. Wong, "It's just a standard deviation!," *Anaesthesia*, vol. 71, no. 8, pp. 969–971, Aug. 2016, doi: 10.1111/anae.13565.
[28] S. Jang, F. Kawsar, and C. Min, "Thermal characterization of AI applications on AI accelerators-equipped microcontrollers," in *Proceedings of the Workshop on Body-Centric Computing Systems*, Jun. 2024, pp. 17–22. doi: 10.1145/3662009.3662020.
[29] Dongwoo Hong, Chee-Kian Ong, and Kwang-Ting Cheng, "BER estimation for serial links based on jitter spectrum and clock recovery characteristics," in *2004 International Conference on Test*, 2004, pp. 1138–1147. doi: 10.1109/TEST.2004.1387388.

BIOGRAPHIES OF AUTHORS



Bakhtiar Alldino Ardi Sumbodo was born in Purwokerto, in 1987. He received his B.Sc. degree in electronics and instrumentation in 2009, and a master's degree in computer science in 2014 from Gadjah Mada University, Indonesia. From 2010 to 2014, he was a research assistant in the Department of Computer Science and Electronics at Gadjah Mada University, Indonesia. Since 2014, he has been a lecturer in the Department of Computer Science and Electronics at Universitas Gadjah Mada. He is a member of the Embedded Systems and Robotics Research Lab. His research topics are related to embedded electronics, robotics, PLC, and computer networks. He can be contacted at b.alldino.as@ugm.ac.id.



Jazi Eko Istiyanto has been a Full Professor of Electronics and Instrumentation at Universitas Gadjah Mada (UGM) since 2010. He previously served as Chairman of Indonesia's Nuclear Energy Regulatory Agency (BAPETEN) from 2014 to 2021, and held departmental leadership roles at UGM in computer science, electronics, and physics. He holds a B.Sc. in Nuclear Physics from UGM, alongside a Postgraduate Diploma, an M.Sc. in computer science, and a Ph.D. in electronic systems engineering from the University of Essex, UK. His research focuses on embedded and cyber-physical systems security. He can be contacted at jazi@ugm.ac.id.



Andi Dharmawan was born in Surakarta, Indonesia, in 1984. He received his B.Sc. in electronics and instrumentation in 2006, M.Sc. in computer science in 2009, and Ph.D. in computer science in 2017, all from Universitas Gadjah Mada, Indonesia. From 2007 to 2009, he worked as a Research Assistant at the Department of Computer Science and Electronics, Universitas Gadjah Mada. Since 2009, he has been a faculty member at the same department, where he is also a part of the Embedded Systems and Robotics Laboratory. His research interests include control systems for robotics, autonomous unmanned systems, and advanced control system development. He can be contacted at andi_dharmawan@ugm.ac.id.