

Digital Twin-Based Control and Monitoring of Industrial Robotic Arms Using Cloud CAD Platforms

Dragos-Alexandru Cazacu^{1,*}, Mihail Hanga², Carmen-Cristiana Cazacu², and Florina Chiscop²

¹ PTC Eastern Europe SRL, Romania and Robots and Production Systems Department, The Faculty of Industrial Engineering and Robotics, National University of Science and Technology Politehnica Bucharest, Splaiul Independenței 313, 060042 Bucharest, Romania

² Robots and Production Systems Department, The Faculty of Industrial Engineering and Robotics, National University of Science and Technology Politehnica Bucharest, National University of Science and Technology Politehnica Bucharest, Romania

Email: acazacu@ptc.com (D.A.C.); mihail.hanga@stud.fir.upb.ro (M.H.); carmen.cazacu@upb.ro (C.C.C.); florina.chiscop@upb.ro (F.C.)

*Corresponding author

Manuscript received November 5, 2025; accepted December 16, 2025; published January 14, 2026.

Abstract—This paper presents a low-cost, CAD-centric Digital Twin (DT) framework for joint-level control and monitoring of robotic arms using a cloud Computer Aided Design (CAD) platform. A Raspberry Pi 5 prototype (DRV8825 stepper driver and AS5600 magnetic encoder) is coupled to an Onshape assembly so that CAD mate values act as both command inputs and state variables. A lightweight Flask web interface provides local operation (slider/inputs, start-stop, emergency stop), while RESTful GET/POST requests synchronize the physical joint and the CAD model in near real time. To keep the motor loop responsive under cloud delays, encoder feedback is acquired on a dedicated 100 Hz thread and CAD updates are batched. Experiments report mean API round-trip latencies of ~195 ms (GET) and ~205 ms (POST), an average cloud synchronization delay of ~200 ms, and a mean angular error of $0.33^\circ \pm 0.05^\circ$ at 1/8 microstepping; local commands execute in ~0.8 s, while Onshape-mediated commands take 1.0–1.5 s depending on network load. Software interlocks, watchdog timeouts, and driver current limiting complement the mushroom-type E-stop. A simple scalability analysis shows that sequential per-joint calls would accumulate latency (e.g., ~1.2 s for 6 joints), motivating batched updates for supervisory DT operation and future multi-DOF validation.

Keywords—Internet of Things (IoT), Raspberry Pi, digital twin, Computer Aided Design (CAD)

I. INTRODUCTION

Industrial robotic arms increasingly need decision-safe, real-time execution that stays faithful to Computer Aided Design (CAD) design intent, yet shop-floor practice often decouples CAD from controller execution, relying on offline simulation with limited feedback leading to trajectory mismatches, commissioning delays, and safety risk. A Digital Twin (DT) can close this loop via live, bidirectional synchronization, but as summarized in Table 1, most DT-robotics platforms depend on external simulation stacks (e.g., Unity/Gazebo) that require exporting CAD to static meshes and losing parametric intelligence, while CAD-centric DT studies are often tied to specific industrial cells/middleware and rarely report end-to-end latency/accuracy/safety metrics; conversely, low-cost arm prototypes typically focus on mechanics/control without CAD-level DT coupling.

To address this gap, we present a compact, reproducible, low-cost CAD-centric DT that operates directly inside the native Onshape environment (CAD as the “Single Source of Truth”), eliminating intermediate conversions (e.g., Unified Robot Description Format (URDF) / Wavefront OBJ file format). Using the Onshape REST API and a lightweight

Flask interface, a physical joint prototype (Raspberry Pi 5 + DRV8825 + AS5600) performs closed-loop synchronization: CAD mate updates drive motion, and encoder feedback updates mates in CAD. We evaluate local vs. cloud-mediated operation and report ~200 ms synchronization latency and $\pm 0.3^\circ$ angular accuracy (1/8 microstepping), alongside practical safeguards (gearbox/encoder mapping, batched API updates, E-stop), providing a clear bridge from educational prototypes toward multi-DOF industrial DT scenarios.

II. LITERATURE REVIEW

The comparison in Table 1 summarizes representative DT-robotics frameworks, DT-CAD integrations, and low-cost robotic arm studies, highlighting their methods, key limitations, and how the proposed approach differs.

Table 1. Critical comparison of related work and positioning of the proposed approach

Ref.	Method / focus	Key limitation(s)	How this paper differs (proposal)
[1]	Cross-domain in DT survey/taxonomy	Generic; little guidance for cloud-CAD robotic control + sync metrics	Implements CAD↔hardware loop; reports latency/accuracy/safety
[2]	Onshape API DT concepts (parameters /mates) API-driven DT for assembly/manufacturing	Conceptual/educational; no robotics control + no quantitative eval.	Uses mates as control surface; encoder feedback; quantified results
[3]	IoT monitoring/control (wireless/R FID)	Cell-specific; industrial context; weak low-cost reproducibility	Low-cost, reproducible joint DT with calibration + bidirectional sync
[4]	5-DOF 3D-printed manipulator	Mainly mechanical; DT/cloud closed-loop not central	Adds DT closed loop between prototype and cloud CAD
[5]	DT safety-critical robotics overview	Conceptual; no practical low-cost safety recipe	Implements safeguards: limits, E-stop, batching, calibration mapping
[6]	2-DoF kinematic design/placement	No DT/cloud-CAD coupling or real-time metrics	Uses DT pipeline for validation/monitoring during execution
[7]	4-DOF arm prototype/validation	DT + CAD↔hardware bidirectional control not addressed	Provides integration layer linking such arms to cloud-CAD DT

[9]	ROS/Gazebo o DT trajectory control Unity3D	Heavy sim stack; model conversions; CAD intent not control surface	Uses cloud-CAD mates directly via lightweight web/REST
[10]	DT control (socket-based)	High complexity; often proprietary industrial targets	Low-cost, reproducible CAD-centric DT with measured API latency

Digital Twins have evolved into closed-loop systems that couple a virtual model, the physical asset, and a reliable bidirectional link for monitoring and control [1]. In robotics, many DT platforms rely on external simulation/VR stacks (e.g., ROS/Gazebo or Unity), which increase integration overhead and often decouple control from CAD parametric intent [9, 10].

Cloud-CAD APIs (e.g., Onshape) enable DT synchronization directly at the design level [2] and have been applied in manufacturing/assembly workflows [3], yet most CAD-centric studies depend on additional middleware and seldom report end-to-end latency/accuracy/safety metrics. In parallel, low-cost arm prototypes emphasize mechanism design and local control [5, 7, 8], with limited continuous CAD↔hardware coupling.

Based on this comparison, the key gap is a reproducible DT pipeline that (i) uses cloud CAD as an interactive control/monitoring surface, (ii) provides bidirectional CAD↔hardware synchronization with reported latency and accuracy, and (iii) integrates practical calibration and safety suitable for low-cost prototyping. This work addresses the gap by coupling a Raspberry Pi joint prototype (DRV8825 + AS5600) with an Onshape model via REST endpoints and evaluating synchronization delay, response time, and angular accuracy, including mitigation for API latency and the 3:1 gearbox.

III. MATERIALS AND METHODS

The Digital Twin framework was implemented on a low-cost 1-DOF prototype to synchronize a cloud CAD model with a physical joint and to evaluate latency, accuracy, and safety under representative command scenarios.

A. Hardware Components

The hardware platform was developed using cost-effective components to demonstrate the feasibility of the approach:

- Control Unit: Raspberry Pi 5, selected for its processing power and connectivity.
- The actuation is provided by a NEMA 17 stepper motor (Model 17HS8401S) with a 1.8° step angle and a holding torque of 0.59 Nm. It is driven at a supply voltage of 12 V with a current limit set to 1.7 A per phase via the DRV8825 driver.
- Feedback: AS5600 magnetic encoder, used to measure angular displacement and provide real-time feedback to the controller.
- Safety: A mushroom-type emergency stops interrupts actuator power. Additional fault-tolerance is implemented in software: joint limits, rate limiting, and input validation; watchdog timeouts for encoder acquisition and cloud/API stalls that force a safe stop and disable the driver via ENABLE; and an anomaly check comparing commanded vs measured encoder angles to

detect missed steps/backlash or obstruction. DRV8825 current limiting provides a practical torque cap and its built-in overcurrent/thermal protections add a hardware safety layer; predictive fault forecasting is left as future work once richer sensing (e.g., current/temperature logging) is integrated.

This 1-DOF setup represents a simplified industrial joint for validating DT synchronization and control.

B. Software Architecture

The software implementation consisted of two main components:

1) Local web interface

A streamlined interface was created utilizing Flask (Python) alongside an HTML and JavaScript front-end. The interface enabled users to enter angular commands via a numeric slider, issue start/stop commands, and observe the system's status visually.

2) Cloud CAD integration

The Onshape API enables bidirectional CAD–hardware alignment via HTTP GET/POST, sending virtual commands to the prototype and updating mate values from encoder feedback. To keep cloud latency from blocking the control loop, encoder acquisition runs in a separate `EncoderMonitor` thread that polls the AS5600 over I2C at 100 Hz, while the main thread handles API requests. Synchronization is also batched—updates are sent at controlled intervals or after motion completion—reducing average cloud-sync delay from > 1.0 s (continuous attempts) to ~200 ms under typical conditions.

C. Digital Twin Synchronization Loop

The DT loop follows a bidirectional CAD↔hardware cycle: mate targets are read from Onshape (CAD→Robot), executed locally, and the measured joint angle is written back to the corresponding mate (Robot→CAD) using a GET–modify–POST workflow (see Fig. 1). Multithreaded encoder monitoring and batched updates decouple real-time control from cloud communication.

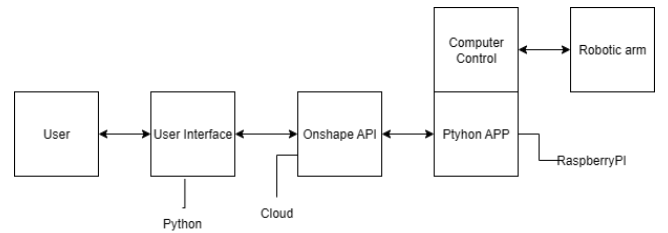


Fig. 1. System architecture diagram.

D. Methodology of Testing

To validate the system, the following experimental protocol was defined:

- Positioning Accuracy: was measured angular deviation between commanded and actual motor position.
- Response Time: latency was evaluated for local interface commands and cloud-initiated commands.
- Synchronization: measure time offset between CAD model updates and physical execution.
- Operational Safety: emergency stop function and system

recovery were tested.

The experiments were conducted multiple times involving single-point commands (e.g., 90°), multi-point trajectories (0° → 45° → 135°), and scenarios with disturbances that necessitated encoder corrections.

IV. RESULT AND DISCUSSION

A. System Overview

The prototype integrates a physical 1-DOF joint with its Onshape CAD counterpart to enable real-time monitoring, trajectory testing, and safe command execution through the DT loop.

B. Functional Architecture and Interfaces

A FAST diagram was used to decompose the primary goal (physical–virtual synchronization) into subfunctions for manipulation, interface, sensing, integration, and safety.

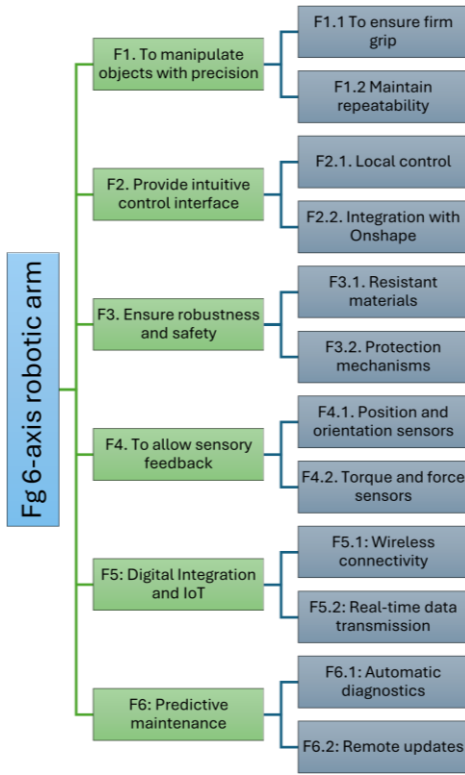


Fig. 2. FAST (function analysis system technique) decomposition of the proposed CAD-centric digital twin.

The FAST diagram (Fig. 2) decomposes the main objective bidirectional physical – virtual synchronization into six linked functions that justify the system design: microstepping actuation with encoder-based correction (precision), a browser UI with optional cloud control (interaction), interlocks and E-stop (safety/robustness), continuous AS5600 monitoring (feedback), REST mapping of joint motion to Onshape mate values (IoT/CAD integration), and deviation logging for early wear/backlash indicators (predictive maintenance). Together, this shows how UI, sensing, safety, and cloud CAD synchronization support the DT closed loop.

The control application is a streamlined web interface, available via any browser, and seamlessly integrated with the Onshape API. The interface features a slider position, numeric input, an emergency stop button, and a checkbox for Onshape live-sync.

C. Experimental Tests and API Integration

Raspberry Pi 5 acted as the controller, running a Python Flask application that included endpoints for joint rotation, angle acquisition, and Onshape synchronization. The Onshape API was evaluated for communication efficiency through both GET and POST requests, aiming for a latency of less than 200 ms.



Fig. 3. Screenshot of web interface.

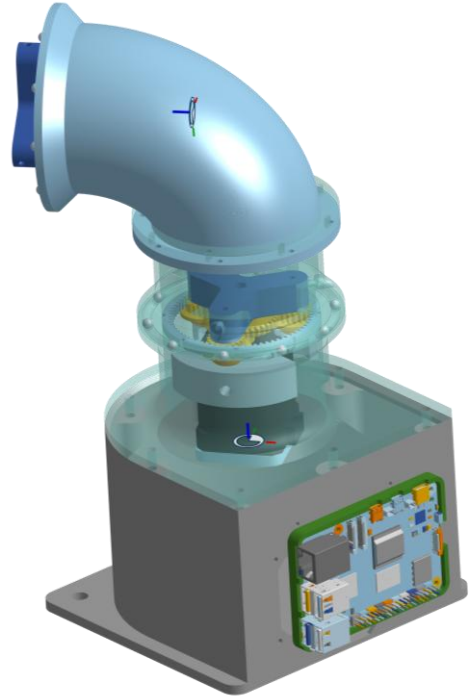


Fig. 4. Screenshot of onshape CAD model with mate values.

Scalability can be estimated from the measured Onshape round-trip time (~ 0.2 s). If multi-DOF synchronization uses sequential per-joint API calls, latency accumulates approximately as $T_{seq} \sim N \times T_{api}$; for a 6-DOF arm, this yields $T_{seq} \sim 6 \times 0.2 = 1.2$ s per full update. Using batched mate updates reduces request chaining, and network load scales as $B \sim N \times f_{sync} \times S$, where S is the average payload per joint

update. Using a conservative $S \sim 1$ kB, a 6-DOF arm synchronized at $f_{sync} = 2$ Hz requires $B \sim 12$ kB/s, while 10Hz yields $B \sim 60$ kB/s., motivating cloud CAD as a supervisory DT layer with controlled sync rates while the local loop maintains high-rate actuation

Table 2. API communication latency (GET vs POST, expected vs measured)

API Operation	Mean	Range	Notes
GET (angle from Onshape)	~ 195 ms	180–210 ms	Stable under local network conditions
POST (update CAD with joint angle)	~ 205 ms	190–220 ms	Slight variability depending on network load

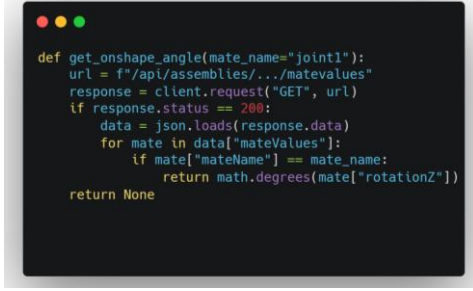


Fig. 5. Bidirectional onshape API workflow used for digital twin synchronization (CAD $\leftrightarrow$ robot).

Fig. 5 summarizes the bidirectional Onshape API workflow used for DT synchronization. In the CAD $\rightarrow$ Robot direction, the controller issues a GET request to the assembly matevalues endpoint, extracts the target mate's rotation (rotationZ) and converts it from radians to degrees, then maps the requested joint angle to motor motion (steps/microstepping and gear ratio) and executes the movement locally. In the Robot $\rightarrow$ CAD direction, the measured joint angle from the AS5600 is used to update the CAD state: the controller first retrieves the current mateValues structure via GET, modifies only the corresponding mate entry (rotationZ converted back to radians), and then transmits the updated payload via POST. This GET–modify–POST pattern ensures that the cloud CAD model remains synchronized with the physical joint while preserving the full mateValues structure returned by the API. The workflow is exposed in the Flask application through dedicated endpoints (angle acquisition, rotation command, and Onshape synchronization), enabling both manual control and automated fetch/sync cycles.

D. Performance Evaluation

Performance was evaluated using synchronization delay, response time, error rate, and angular accuracy (Table 3).

Table 3. Performance metrics used for system evaluation

Metric	Description	Evaluation Method	Target
Angular Accuracy	Difference between commanded and achieved joint angle	Encoder feedback vs commanded input	$\pm 0.5^\circ$
Response Time	Delay between command issuance and physical execution	Stopwatch-based measurement	< 2 s
Error Rate	Frequency of unexpected deviations	% of failed/incorrect commands	$< 1\%$
Synchronization Delay	Time difference between physical movement and Digital Twin update	Timestamp comparison	< 250 ms

E. Results

Table 4 summarizes angular accuracy across microstepping settings. The best trade-off was obtained at 1/8

microstepping (mean absolute error $0.33^\circ \pm 0.05^\circ$), while 1/16 showed slightly higher error ($0.45^\circ \pm 0.06^\circ$) despite finer nominal resolution because the lower incremental torque per microstep increases sensitivity to static friction, gearbox backlash, and load-induced vibration, so the rotor does not consistently settle on intermediate microstep positions. Integrated tests showed comparable accuracy in Local and Onshape modes ($\sim \pm 0.3^\circ$ – $\pm 0.4^\circ$), but longer response times in Onshape mode (1.0–1.5 s vs ~ 0.8 s locally) due to cloud/API overhead (Table 4).

Table 4. Angular accuracy under different microstepping settings (Full, 1/4, 1/8, 1/16)

Target Angle	Full Step Error	1/4 Step Error	1/8 Step Error	1/16 Step Error
0°	$\pm 1.2^\circ$	$\pm 0.8^\circ$	$\pm 0.3^\circ$	$\pm 0.4^\circ$
90°	$\pm 1.5^\circ$	$\pm 0.9^\circ$	$\pm 0.3^\circ$	$\pm 0.5^\circ$
180°	$\pm 1.4^\circ$	$\pm 0.7^\circ$	$\pm 0.3^\circ$	$\pm 0.4^\circ$
270°	$\pm 1.6^\circ$	$\pm 0.8^\circ$	$\pm 0.4^\circ$	$\pm 0.5^\circ$
Mean $\pm$ SD	$1.43^\circ \pm 0.17^\circ$	$0.80^\circ \pm 0.08^\circ$	$0.33^\circ \pm 0.05^\circ$	$0.45^\circ \pm 0.06^\circ$

Although 1/16 microstepping offers finer nominal resolution, it showed slightly higher error than 1/8 in our setup because the lower per-microstep torque increases sensitivity to static friction, gearbox backlash, and load-induced vibration, so the rotor does not reliably settle at intermediate microstep positions. Encoder noise is unlikely to dominate (AS5600 resolution is finer than the error and readings were stable), indicating the main cause is reduced torque margin and drivetrain non-idealities at higher microstepping (see Fig. 6 and Table 5).



Fig. 6. Assembled concept of the experimental robotic arm.

F. Challenges and Solutions

Several issues were identified during testing:

- Encoder calibration errors caused by the 3:1 gearbox ratio.
 - Latency in API communication.
 - Safety risks due to high torque during unexpected stops.
- Solutions were implemented through software compensation, batch synchronization, and integration of a mushroom-type emergency stop button.

Table 5. Challenges, digital twin contributions, and observed outcomes

Challenge	Digital Twin Contribution	Implemented Solution	Observed Outcome
Gearbox ratio (3:1) caused encoder mismatch	Enabled real-time feedback correction	Software calibration mapping motor to joint angle	Accurate angular mapping
API latency affecting Onshape mode	Digital Twin highlighted sync delays	Batch updates & controlled refresh rate	Reduced communication load, stable sync
Safety during unexpected stops	Digital Twin facilitated remote monitoring	Emergency stop button integrated	Immediate halt, safe manual repositioning
Need for dual operation modes	CAD ↔ Physical bidirectional mapping	Local web + Onshape API	Flexible control in both contexts

G. Case Applications and Practical Use

Real test sequences were used to validate the system's reliability in representative operating scenarios. In Local mode, a 90° rotation was completed in approximately 2 s with a measured positioning error of +0.3°. In Onshape mode, a 180° rotation was executed with an initial error of −0.2°, which was subsequently corrected to below 0.1° through closed-loop feedback.

A multi-point trajectory (0° → 45° → 135°) further confirmed stable tracking, yielding an average absolute error of ±0.5° while maintaining an average synchronization delay of ~200 ms between the physical joint and the cloud CAD model. This latency reduction is primarily attributable to batched API updates and the non-blocking encoder feedback thread, which decouple real-time sensing/control from cloud communication overhead.

Overall, these results indicate practical applicability of the proposed Digital Twin system for automated sorting tasks that require accurate, synchronized positioning, for education and training as a low-cost platform to learn robotics and CAD integration, and for rapid prototyping by enabling real-time validation of CAD-defined trajectories prior to physical execution.

V. CONCLUSION

In this study, a Digital Twin framework for robotic arm control and monitoring was designed, put into practice, and evaluated. The outcomes verified that real-time synchronization between virtual and physical systems was enabled by combining low-cost hardware with cloud-based CAD platforms.

This work demonstrated a low-cost, CAD-centric Digital Twin that synchronizes an Onshape assembly with a physical robotic joint (Raspberry Pi 5, DRV8825, AS5600) for command execution and state monitoring. Using a lightweight Flask interface, multithreaded encoder acquisition, and batched REST API updates, the system achieved ~200 ms cloud synchronization delay and joint accuracy of $0.33^\circ \pm 0.05^\circ$ at 1/8 microstepping, with local response times of ~0.8 s and Onshape-mediated response

times of 1.0–1.5 s depending on network load. Safety is supported through software interlocks, watchdog timeouts, driver current limiting, and a mushroom-type emergency stop.

A scalability discussion indicates that sequential per-joint cloud calls would accumulate latency (e.g., ~1.2 s for 6 joints), motivating batched mate updates and positioning cloud CAD as a supervisory DT layer while the local controller maintains high-rate actuation. Future work will extend validation to multi-DOF prototypes with longer repeated trials, time-series stability under load, and richer sensing (e.g., current/temperature logging) to support fault prediction and predictive maintenance.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

Dragos-Alexandru Cazacu and Carmen-Cristiana Cazacu conducted the research; Mihail Hanga carried out the applied part of the research and wrote the paper, Carmen-Cristiana Cazacu and Florina Chiscop analyzed the data and wrote the paper; all authors had approved the final version.

REFERENCES

- [1] M. E. Iliuță *et al.*, “Digital twin — A review of the evolution from concept to technology and its analytical perspectives on applications in various fields,” *Appl. Sci.*, vol. 14, 5454, 2024.
- [2] Onshape Digital Twins | Digital Transformation in Education. [Online]. Available: <https://ptc-education.github.io/docs/solutions/onshapedx>
- [3] C. C. Cazacu *et al.*, “Optimizing assembly in wiring boxes using API technology for digital twin,” *Appl. Sci.*, vol. 14, 9483, 2024.
- [4] N. Sudheer *et al.*, “Library Robo using IoT,” *International Journal of Advance Research and Innovative Ideas in Education*, vol. 6, no. 3, 2020.
- [5] A. N. Sharkawy and J. M. Nazzal, “Design and manufacturing using 3D printing technology of A 5-DOF manipulator for industrial tasks,” *International Journal of Robotics and Control Systems*, vol. 4, pp. 893–909, Jun. 2024.
- [6] S. Baidya *et al.*, “Digital twin in safety-critical robotics applications: Opportunities and challenges,” in *Proc. 2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2022, pp. 101–107.
- [7] K. Khalid, A. A. Zaidi, and Y. Ayaz, “Optimal placement and kinematic design of 2-DoF robotic arm,” in *Proc. 2021 International Bhurban Conference on Applied Sciences and Technologies (IBCAST)*, Islamabad, 2021, pp. 552–559.
- [8] A. Fahruzi *et al.*, “Design OF 4DOF 3D robotic arm to separate the objects using a camera,” *International Journal of Artificial Intelligence and Robotics*, vol. 3, 2021.
- [9] C. A. M. Montoya *et al.*, “Integration of a digital twin framework for trajectory control of a 2RRR planar parallel manipulator using ROS/Gazebo and matlab,” *Future Internet*, vol. 17, 2025.
- [10] W. Chang *et al.*, “Construction and validation of a digital twin-driven virtual-reality fusion control platform for industrial robots,” *Sensors*, vol. 25, 4153, 2025.

Copyright © 2026 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).