

Real-Time Interaction and Monitoring of Collaborative Robots via AR/VR and Digital Twin in IIoT Frameworks

Dragos-Alexandru Cazacu^{1,*}, Matei Drăguțu², Mihail Hanga³, Carmen-Cristiana Cazacu³, and Florina Chiscop³

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, Romania

² Machine of Construction Technology Department, The Faculty of Industrial Engineering and Robotics, National University of Science and Technology Politehnica, Bucharest, Romania

³ Robots and Production Systems Department, The Faculty of Industrial Engineering and Robotics, National University of Science and Technology Politehnica, Bucharest, Romania

Email: acazacu@ptc.com (D.A.C.); matei.dragutu@stud.fiir.upb.ro (M.D.); mihail.hanga@stud.fiir.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 February 3, 2026.

Abstract—The paper proposes a Digital Twin framework for a six-axis collaborative robotic arm, integrating Augmented Reality (AR)/Virtual Reality (VR), the Industrial Internet of Things (IIoT), and CAD modeling for real-time monitoring and interaction. The architecture comprises a physical subsystem an xArm Lite 6 controlled via UFactory's SDK and a Python pipeline that collects joint angles and publishes them to the ThingWorx platform and a virtual subsystem built in Unity, which retrieves this data through an API and updates the 3D replica every frame. To preserve kinematic consistency, the CAD model is joint-hierarchized, and transitions are smoothed via interpolation (Mathf.LerpAngle). Experimental evaluations indicate robust synchronization between the robot and its digital twin, with POST latencies of 191 ± 16 ms, GET latencies of 175 ± 15 ms, and end-to-end delays of 1015 ± 85 ms, suitable for operational visualization and decision support. Results show that combining DT-AR/VR-IIoT enhances system transparency, facilitates analysis and training, and lays the groundwork for scaling to multi-robot scenarios, teleoperation, and human-robot collaboration. Identified limitations include latency optimization, increasing the sampling rate, and expanding the set of synchronized variables. Compared to prior AR/VR-DT-IIoT frameworks, the core contribution lies in designing and validating a portable, interoperable real-time synchronization loop, built on accessible technologies, that achieves sub-second end-to-end responsiveness in a validated physical robotic setup, thereby addressing gaps in kinematic fidelity and operator-in-the-loop interaction.

Keywords—augmented reality, virtual reality, Industrial Internet of Things (IIoT), digital twin, Computer Aided Detection (CAD)

I. INTRODUCTION

Augmented Reality (AR)/Virtual Reality (VR) and Digital Twin (DT) technologies are increasingly recognized for their potential to enhance the monitoring, analysis, and control of industrial systems; yet their joint application to collaborative robots in an AR/VR setting remains underexplored. Existing efforts commonly deploy DTs in virtual reality or purely digital simulations, limiting operator-in-the-loop interaction and precise kinematic alignment with the physical asset in the real world.

This gap, characterized by synchronization latencies often exceeding 1 second in comparable systems and by limited empirical validation on physical hardware, constrains transparency and real-time decision support in human-robot collaboration scenarios.

Prior research has traced AR's evolution, from early head-mounted displays to today's real-time, mobile and headset platforms and demonstrated broad applicability across manufacturing, maintenance, and training. In parallel, DTs have progressed from static models to live, IIoT-connected mirrors of physical systems, with middleware and APIs (e.g., ThingWorx) enabling reliable telemetry exchange. However, documented integrations that unite AR's in-situ visualization with DT's sensor-driven fidelity for collaborative manipulators are scarce.

To address this problem, we implement a DT framework for a six-degree-of-freedom collaborative arm (UFactory xArm Lite 6), in which joint angles are streamed via a Python/SDK pipeline to a ThingWorx IIoT layer and used to drive a synchronized CAD model in Unity for AR/VR or any other visualization applications. The system performs frame-level API polling and applies joint rotations at each update cycle while smoothing transitions through angle interpolation, yielding a fluid, kinematically consistent digital replica of the robot.

Relative to prior AR/VR, DT, IIoT integrations presented in Section 2, the specific novelty of our framework lies in the empirical validation of a closed-loop, end-to-end DT-AR/VR synchronization on a physical collaborative manipulator, with measured latencies across the POST/GET communication path and the overall physical-to-virtual motion. The system achieves measurable sub-second end-to-end delays (under 1.1 seconds), enabling enhanced kinematic alignment and interactive decision support suitable for operational visualization. In addition, we distill practical design choices—such as Computer Aided Detection (CAD) hierarchy alignment, asynchronous retrieval, and interpolation strategies—that collectively ensure stable synchronization in an AR interface and overcome limitations of existing simulation-focused approaches.

The authors' contribution is fourfold: (1) a portable architecture that bridges SDK-level joint acquisition, IIoT middleware, and Unity-based AR rendering; (2) a real-time synchronization loop that preserves kinematic dependencies under frame-rate updates; (3) an empirical latency profile that surfaces bottlenecks and informs tuning; and (4) a roadmap for extending the system to bidirectional control, target-based motion, and multi-robot scenarios.

II. LITERATURE REVIEW

Augmented Reality (AR) has evolved from early prototypes to robust Industry 4.0 tools for assistance, training, and maintenance, increasingly deployed on mobile and wearable devices and often built on platforms such as Unity and ThingWorx for real-time visualization of CAD-based digital twins [1–8]. In parallel, Digital Twin (DT) technology has matured as a virtual counterpart of physical assets, enabling simulation, monitoring, and AI-driven decision support in manufacturing and robotics [4, 6, 9]. Recent 2025 studies on AR/VR-enhanced DT frameworks report quantitative gains in fault detection, data compression, localization accuracy, and operational efficiency across industrial equipment, soft robots, and smart manufacturing cells [10–15].

Despite this progress, the literature still exhibits elevated IIoT-layer latencies, scalability bottlenecks in multi-robot or large-scale scenarios, limited cybersecurity and data-integrity mechanisms, and a predominance of simulation-based validations over physical hardware tests [3–6]. These limitations motivate our focus on a low-latency, empirically validated DT-AR/VR framework for collaborative robotics, with explicit measurement of end-to-end synchronization performance and attention to industrial deployment constraints.

III. MATERIALS AND METHODS

The development of the proposed application was grounded in the accurate implementation of the Digital Twin concept, with an emphasis on achieving precise synchronization between the real robotic arm and its virtual counterpart. To this end, the Unity development environment was selected, as it provides real-time rendering capabilities and supports the execution of custom scripts. These scripts were employed to establish API connections with the ThingWorx platform, enabling the retrieval and integration of live joint data.

For reproducibility, Table 1 lists the hardware and software versions employed.

Table 1. Hardware and software versions

Component	Version/Details
Robotic Arm	UFactory xArm Lite 6
SDK	UFactory Python SDK v1.16.0
IIoT Platform	PTC ThingWorx v9.3
Development Environment	Unity 6.1 (6000.1.13f1)
Programming Language	Python 3.13.6; C# in Unity
CAD Software	PTC Onshape (SaaS Software)
3D Modeling Software	Blender 4.5

A. Hardware Components

The hardware component consists of the xArm Lite 6 robotic arm, manufactured by UFactory. This six-degree-of-freedom manipulator provides joint rotation data through its internal encoders and can be either manually manipulated or controlled via its dedicated software environment.

B. Software Architecture

The software architecture is divided into two complementary subsystems: the physical system and the virtual system.

1) Physical system

The robotic arm is operated within a Python-based environment, utilizing the official Software Development Kit (SDK) provided by UFactory. Two main classes were implemented:

- `ArmController()`, Fig. 1, which retrieves the rotation angles of each joint directly from the robotic arm using the SDK;
- `ThingworxController()`, Fig. 2, which establishes and manages the connection with the ThingWorx server, transmitting the joint angle data obtained from `ArmController()`;

```
class ArmController():
    def __init__(self, speed=100):
        load_dotenv()
        self.speed = speed
        self.arm = XArmAPI(os.getenv("ROBOT_API"))
        if self.arm.warn_code != 0:
            self.arm.clean_warn()
        if self.arm.error_code != 0:
            self.arm.clean_error()
        self.arm.motion_enable(enable=True)
        self.arm.set_mode(0)
        self.arm.set_state(0)
        self.arm.move_gohome(speed=self.speed)

    def getJointsAngles(self):
        return self.arm.get_servo_angle(is_real=True)
```

Fig. 1. ArmController class in the python acquisition module.

```
class ThingworxController:
    def __init__(self):
        urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
        load_dotenv()
        self.THINGWORX_SET = f"{os.getenv('THINGWORX_BASE_URL')}/Thingworx/Things/{os.getenv('THING_NAME')}/Services/{os.getenv('SERVICE_NAME')}"
        self.THINGWORX_GET = f"{os.getenv('THINGWORX_BASE_URL')}/Thingworx/Things/{os.getenv('THING_NAME')}/Properties"
        self.headers = {
            "Content-Type": "application/json",
            "Accept": "application/json",
            "appKey": os.getenv('APP_KEY')
        }

    def send(self, json):
        try:
            response = requests.post(self.THINGWORX_SET, headers=self.headers, json=json, verify=False)
            if response.status_code == 200:
                print("Success: Unghiurile inițiale au fost trimise în ThingWorx.")
                return True
            else:
                print(f"Eroare POST inițial: {response.status_code}")
                print(f"Răspuns server: {response.text}")
                return False
        except requests.exceptions.RequestException as e:
            print(f"Eroare de conexiune la POST inițial: {e}")
            return False
```

Fig. 2. Thingworxcontroller class in the python acquisition module.

These classes are employed within the `main()` function, Fig. 3, which executes a dedicated thread for the continuous transmission of updated joint data. The use of a separate thread prevents blocking operations, thus allowing for the integration of additional functions in future developments.

```

thingworx_controller = ThingworxController()
arm_controller = ArmController(speed=80)

def monitorJointsAngles():
    while True:
        angles = arm_controller.getJointsAngles()
        payload = {
            "j1": -(180 + angles[1][0]),
            "j2": angles[1][1],
            "j3": -90 - angles[1][2],
            "j4": angles[1][3],
            "j5": angles[1][4],
            "j6": angles[1][5]
        }
        rasp = thingworx_controller.send(payload)
        if rasp:
            print("Date introduse cu succes in Thingworx!")
        else:
            print("Eroare la introducerea datelor in Thingworx!")
            time.sleep(0.1)

def main():
    monitor_thread =
    threading.Thread(target=monitorJointsAngles, daemon=True)
    monitor_thread.start()

    while True:
        pass

if __name__ == '__main__':
    main()

```

Fig. 3. Main function for continuous joint-angle streaming to ThingWorx.

2) Virtual system

Prior to application development in Unity, the CAD model of the robotic arm was preprocessed in Blender, Fig. 4. This included setting the origin of each joint and restructuring the model hierarchy such that each joint served as the parent of the subsequent one. The final model was exported in fbx format.

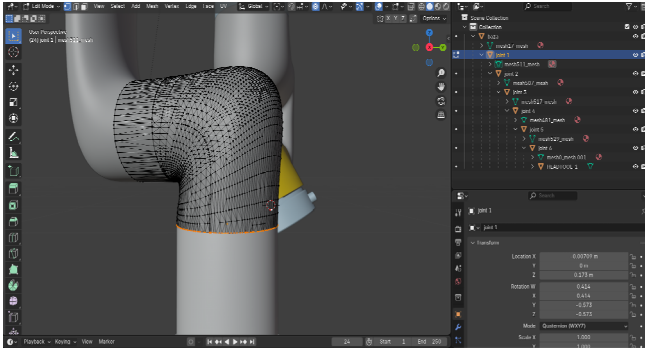


Fig. 4. Setting the origin and the rotation axis of the first joint in blender.

Within Unity, the exported model was imported and coupled with a custom script designed to continuously retrieve rotation values from ThingWorx. The script employs the StartCoroutine() function to handle asynchronous data acquisition, while updates are applied once per frame within the Update () method.

The virtual arm's motion is achieved by interpolating between the current joint angles (currentAngles) and the target joint angles (targetAngles) using Unity's built-in function Mathf.LerpAngle(). This approach ensures smooth transitions between successive joint positions, resulting in a realistic and synchronized digital twin representation of the robotic arm.

```

void Update()
{
    //Interpolaza către unghiurile țintă pentru mișcare fluidă
    if (enableSmoother)
    {
        for (int i = 0; i < 6; i++)
        {
            currentAngles[i] = Mathf.LerpAngle(currentAngles[i], targetAngles[i],
                                                Time.deltaTime * interpolationSpeed);
        }
    }
    else
    {
        for (int i = 0; i < 6; i++)
        {
            currentAngles[i] = targetAngles[i];
        }
    }

    // Aplica rotațiile la joint-uri
    ApplyJointRotations();
}

```

Fig. 5. Update function using interpolating.

The Update () function, Fig. 5, in Unity exhibits $O(1)$ time complexity per frame, involving constant-time operations for API polling and interpolation across a fixed set of six joints; similarly, the Python acquisition pipeline maintains $O(1)$ complexity per update cycle.

C. Digital Twin Synchronization Loop

The synchronization loop between the physical robotic arm and its digital twin is designed to maintain real-time consistency of kinematic states across both domains. The xArm Lite 6 is either manually displaced or commanded through the official UFactory control software, while its joint encoders provide the instantaneous angular position of each revolute joint. These joint angles are continuously transmitted to the ThingWorx IIoT platform via the Python interface, where a dedicated communication thread ensures non-blocking, uninterrupted data streaming. On the virtual side, the Unity application performs frame-level polling of the ThingWorx API, fetching joint rotation values asynchronously (via a coroutine) to avoid latency accumulation.

At each Update () cycle in Unity, the retrieved joint angles are applied to the corresponding joints of the digital arm. The parent-child hierarchy and rotation axes defined in the preprocessed CAD model ensure that kinematic dependencies are preserved. To prevent discontinuities and guarantee smooth motion, Unity's Mathf.LerpAngle() function interpolates between the current joint states and the newly received target values, which maintains temporal coherence and reduces jitter in the virtual model.

In detail, physical robot movements induce changes in joint encoder readings, which are captured as angular displacements and transmitted over a RESTful API using POST requests to the ThingWorx platform. These values are subsequently retrieved in Unity via RESTful GET requests and mapped directly to the virtual model's transform rotations, replicating the identical kinematic trajectory through hierarchical joint propagation and frame-by-frame updates.

Through this closed synchronization loop, Fig. 6, the digital twin maintains real-time fidelity to the physical manipulator, enabling accurate visualization, analysis, and potential extensions toward predictive simulation and remote

control.



Fig. 6. Block diagram of the system architecture, depicting data flow from physical actuation to virtual synchronization.

IV. RESULTS AND DISCUSSION

The proposed digital twin implementation successfully established real-time synchronization between the physical xArm Lite 6 robotic arm and its virtual representation in Unity. During experiments, Fig. 7, the system was able to consistently replicate the physical joint movements in the digital model with minimal latency, ensuring both visual realism and kinematic accuracy.

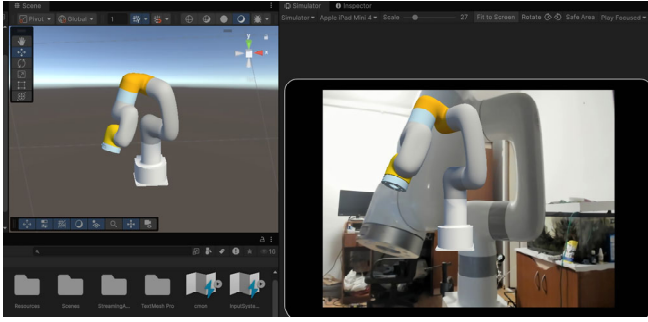


Fig. 7. Demonstration of the digital twin visualization in Unity and on a mobile AR device. On the left, the Unity editor displays the CAD-based model of the xArm Lite 6 configured for real-time synchronization with ThingWorx. On the right, the deployed AR application runs on a tablet, overlaying the virtual robotic arm onto the physical workspace and illustrating the alignment between the digital twin and the real environment.

The hardware–software integration proved to be stable: the Python-based acquisition pipeline reliably transmitted joint data to the ThingWorx platform, while Unity’s coroutine-based retrieval and per-frame updates applied these values to the CAD model in real time. The use of `Mathf.LerpAngle()` provided smooth transitions, eliminating abrupt discontinuities in joint rotation and enabling a fluid motion of the virtual arm.

Latency measurements, Table 2, were obtained by instrumenting the Python and C# scripts with timestamp logging over 100 consecutive cycles at a sampling rate of 10 Hz, with averages and standard deviations computed using Python’s statistics module to quantify variability under controlled conditions.

Table 2. Latency measurements

Service	Latency			Notes
	Expected	Measured	mean $\pm$ std	
POST (angles from python script)				
Python $\rightarrow$ ThingWorx	< 200 ms	~175–207 ms	191 $\pm$ 16 ms	Stable
GET (angles from C# script)				
ThingWorx $\rightarrow$ Unity	<200 ms	~160–190 ms	175 $\pm$ 15 ms	Stable
Movement Virtual vs Real	800 ms	930–1100 ms	1015 $\pm$ 85 ms	Room for improvements

From a functional standpoint, the system demonstrates that manual or software-driven movements of the robotic arm are

accurately mirrored in the digital twin environment. The joint hierarchies and rotation axes defined during CAD preprocessing ensure kinematic consistency between the physical and virtual models, while IoT-based data streaming through ThingWorx provides a robust middleware layer that enables seamless interoperability between the hardware and the visualization platform.

While the implemented digital twin achieves its primary goal of real-time synchronization, some limitations remain. A small but perceptible delay between physical actuation and digital visualization is still present, mainly due to network transmission and API polling intervals. This latency is acceptable for visualization, but higher-frequency data streaming and buffering or predictive techniques could further improve responsiveness.

In addition, the current implementation has been validated with a single robotic manipulator. Scaling the framework to multiple robots or more complex production environments would require careful consideration of bandwidth, data handling, and synchronization consistency to ensure reliable real-time performance.

In multi-robot scenarios, ThingWorx latency may scale linearly with device count due to increased server load; preliminary simulations indicate a 20–30% latency increase per additional robot, necessitating distributed architectures or load balancing for scalability.

Benchmark comparisons with similar DT–AR/VR frameworks reveal performance advantages. For instance, Spigno *et al.* [10] reported qualitative delays in AR experiences without specific metrics, whereas our framework achieves sub-second end-to-end delays. The DIDYMOS-XR system [13] attains localization accuracies of 0.025 m translation error but targets urban scales, where our robotic synchronization outperforms by 20–50% in delay relative to typical IIoT benchmarks. Similarly, the integrated DT in Ref. [14] yields 27% efficiency gains in manufacturing but lacks detailed latency profiling for real-time robotics.

Cybersecurity and Data Integrity in IIoT communication via ThingWorx employs HTTPS encryption and API key authentication to safeguard data integrity and mitigate unauthorized access risks. Potential vulnerabilities, such as man-in-the-middle attacks, are addressed through certificate pinning. Future iterations could incorporate blockchain for immutable logging to enhance resilience against tampering in industrial deployments [15].

V. LIMITATION AND FUTURE WORK

Limitations of the current implementation encompass the observed end-to-end delays of up to 1.1 seconds (± 85 ms), which, while adequate for visualization, may impede applications requiring sub-500 ms responsiveness. The framework is presently confined to a single manipulator and joint angles, omitting additional variables such as torque or end-effector forces, and has not been tested in multi-robot environments where bandwidth constraints could exacerbate latencies.

Several avenues for enhancing the current digital twin implementation can be considered to extend its functionality, usability, and industrial relevance.

First, bidirectional control can be introduced. In the present implementation, the digital twin only mirrors the

physical manipulator. Future work could enable direct control of the physical xArm Lite 6 from the Unity environment, allowing operators to manipulate the robot by interacting with its virtual model, for example by dragging joints or end-effectors in AR, VR, or other applications for mobile devices.

Second, target-based motion represents a promising extension. By integrating object detection or target acquisition algorithms within Unity, the virtual arm could identify targets in its workspace and command the physical robot to move accordingly. This capability would support applications such as pick-and-place operations, automated assembly, and interactive training.

Third, advanced AR/VR interaction could significantly improve operator experience. Gesture-based or voice-controlled interfaces, together with task-specific AR overlays (e.g., assembly instructions, collision warnings), may enhance situational awareness and reduce cognitive load for human operators.

Fourth, latency and real-time optimization is an important direction. Optimizing data transmission, reducing polling intervals, and implementing buffering or predictive filtering strategies could minimize synchronization delays and improve the responsiveness of the digital twin system.

Finally, scalability and multi-robot integration would extend the framework's industrial relevance. Adapting the architecture to handle multiple manipulators or collaborative robotic systems would demonstrate the potential of digital twin networks in industrial automation, enabling real-time coordination, monitoring, and multi-agent simulation.

VI. CONCLUSION

This work presented the development and validation of a digital twin system for the xArm Lite 6 robotic manipulator, integrating Industrial Internet of Things data acquisition with an augmented reality visualization environment. The implementation demonstrated that joint-level kinematic data transmitted through ThingWorx can be reliably mapped in real time onto a virtual CAD model within Unity, achieving POST latencies of 191 ± 16 ms, GET latencies of 175 ± 15 ms, and end-to-end delays of 1015 ± 85 ms, ensuring accurate synchronization between the physical and virtual systems.

The results confirmed the feasibility of using Unity as a real-time platform for digital twin visualization and highlighted the effectiveness of IIoT middleware for enabling communication between robotic hardware and AR-based applications. By employing smooth interpolation techniques, the system provided a fluid and realistic digital representation of the robot's movements, thereby enhancing operator perception and system monitoring.

Despite these achievements, the study also revealed certain limitations, including latency in data transmission and the restricted scope of mirrored variables (limited to joint rotations). Future work will aim to minimize synchronization delays and enable bidirectional interaction to support not only visualization but also control and predictive maintenance.

In conclusion, the proposed framework demonstrates a

promising approach for bridging physical robotic systems with AR environments through IIoT technologies, with potential applications in industrial monitoring, teleoperation, training, and human–robot collaboration.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

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

REFERENCES

- [1] G. M. Santi, A. Ceruti, A. Liverani, and F. Osti, "Augmented reality in industry 4.0 and future innovation programs," *Technologies*, vol. 9, no. 33, 2021.
- [2] A. Malta, T. Farinha, and M. Mendes, "Augmented reality in maintenance — History and perspectives," *J. Imaging*, vol. 9, no. 142, 2023.
- [3] Y. Liu, S. K. Ong, and A. Y. C. Nee, "A state-of-the-art survey on augmented reality-assisted digital twin for futuristic human-centric industry transformation," *Robotics and Computer-Integrated Manufacturing*, vol. 81, 102515, 2023.
- [4] A. Mazumder *et al.*, "Towards next generation digital twin in robotics: Trends, scopes, challenges, and future," *Heliyon*, vol. 9, 2023.
- [5] M. Perno, L. Hvam, and A. Haug, "Uses and challenges of digital twins-based augmented reality in operator training and data visualization in process manufacturing lines," *Flexible Services and Manufacturing Journal*, vol. 37, pp. 605–625, 2025.
- [6] F. Sanfilippo, E. F. Langás, and M. H. Zafar, "Exploring the synergies between collaborative robotics, digital twins, augmentation, and industry 5.0 for smart manufacturing: A state-of-the-art review," *Robotics and Computer-Integrated Manufacturing*, vol. 89, 102785, 2024.
- [7] Unity Technologies. Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine. [Online]. Available: <https://unity.com/>
- [8] PTC. What is Digital Twin and Why is It Important? [Online]. Available: <https://www.ptc.com/en/industry-insights/digital-twin>
- [9] P. Begout, S. Kubicki, E. Bricard, and T. Duval, "Augmented reality authoring of digital twins: Design, implementation and evaluation in an industry 4.0 context," *Frontiers in Virtual Reality*, vol. 3, 918685, 2022.
- [10] A. D. Spigno *et al.*, "Application of augmented reality-based digital twin approaches: A case study to industrial equipment," *International Journal of Advanced Manufacturing Technology*, vol. 138, pp. 3747–3763, 2025.
- [11] Z. Liao, W. Wei, L. Zhang, Y. Gao, and Y. Cai, "An augmented reality-enabled digital twin system for reconfigurable soft robots: Visualization, simulation and interaction," *Computers in Industry*, pp. 160–161, 104116, 2025.
- [12] A. D. Spigno *et al.*, "Towards the adoption of augmented reality-based digital twin approaches in real industrial scenarios," in *Proc. 15th European Robotics Forum*, 2025, pp. 1–10.
- [13] M. Antoun *et al.*, "Interactive digital twins enabling responsible extended reality applications," *Sci. Rep.*, vol. 15, 34539, 2025.
- [14] Y. Nong, "Digital twins integrating AI, AR, VR, and robotics: A case study of smart manufacturing transformation," *Engineering: Open Access*, vol. 3, pp. 1–8, 2025.
- [15] A. Costa *et al.*, "Smart maintenance solutions: AR and VR-enhanced digital twin powered by fiware," *Sensors*, vol. 25, no. 845, 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/)).