On the resource consumption of XR-enabled Digital Twin Applications in the edge-cloud continuum

Aias Sotiropoulos*, Dionysis Xenakis†, Giorgos Kalpaktsoglou*, Yannis Vittorakis†, Melina Dolapsaki†

***Abstract*—In this work, we present an experimental study on the resource consumption requirements and practical deployment considerations for a fully functional, ROS-compatible digital twin (DT) system of a robotic arm, featuring an extended reality (XR)-enabled graphical user interface (GUI). Our evaluation measures network, CPU, RAM and GPU usage across all principal system components that include both physical and virtualized robot controllers, deployed in Multi-access Edge Computing (MEC) infrastructures. The XR interface, developed for remote monitoring, optimization, and control in emerging Industry 4.0/5.0 scenarios, supports immersive supervision and interactive operation. The results provide practical insights for designing efficient, scalable, and XR-enabled digital twin applications towards the next-generation of manufacturing and robotic systems.**

***Index Terms*—Digital twin, robotic arm, ROS, Industry 4.0/5.0 assets, multi-access edge computing (MEC)**

I. Introduction

Digital twins are at the heart of the emerging Industry 4.0/5.0 paradigms, serving as fully functional, dynamic digital replicas of physical systems and processes [1], [2]. By leveraging continuous streams of sensor data, simulation, and advanced control algorithms, digital twins enable real-time monitoring, predictive analytics, remote or autonomous operation of industrial assets, from manufacturing lines to robotic arms. These ROS-compatible virtual systems are not merely static models operating offline but instead, they are continuously synchronized with their physical counterparts, supporting process optimization, proactive maintenance, safety validation, and remote handling—capabilities now considered indispensable for sectors such as the healthcare 4.0, automotive, manufacturing, pharmaceuticals, aerospace, and warehouse management [3]–[5].

The adoption of digital twins in robotics has accelerated the shift toward edge-enabled automation, where network latency and reliability are critical. Modern deployments utilize advanced network architectures, which include cloud-edge and 5G infrastructures, to offload time-sensitive computation closer to the data sources, ensuring fully-fledged, low-latency feedback loops between the digital and physical counterparts [6], [7]. Experimental studies have demonstrated that 5G, when combined with edge computing, can provide ultra-low latency, high-end reliability, and large bandwidth, which are all necessary for digital twin applications in manufacturing, making such architectures especially attractive for Industry 4.0/5.0 deployment setups [8]. This is particularly crucial for robotic arms in production, or remote assembly lines, where safe and precise tele-operation, adaptive control, and integration with wider cyber-physical systems (CPS) demand robust and scalable digital twin architectures [6]–[8].

Beyond manufacturing, digital twins have been applied to heavy industry [9], construction [4], pharmaceuticals, and logistics—anywhere robotics are essential for automating repetitive, hazardous, or precision-critical tasks. For example, digital twin-driven robotic arms have been used for real-time inspection and assembly in aircraft manufacturing [4], adaptive quality control in pharmaceutical lines [3], and as core elements in modular, human-robot collaborative work cells [5]. These systems, typically built on open frameworks such as ROS, are increasingly integrated with extended reality (XR) interfaces for immersive visualization, remote supervision, and operator training [2], [10]. XR-enhanced digital twins empower field engineers to interact with live systems, visualize process information in context, and optimize task performance from anywhere with network access.

Besides, fully-fledged digitization of physical assets in the production field through DTs (i.e., workers, mobile/fixed robots, stock, fleet) is key enabler for the adoption of innovative and emerging business models that distinct the ownership of assets from their daily usage and management. The so-called *servitization* empowers organizations that lack the know-how, technical staff, or resources for purchasing and operating high-end assets to immediately benefit from the utilization of specialized machinery, equipment and software, through pay-as-you-use services offered by Asset-as-a-Service (AaaS) providers; thus, transforming acquisition costs from fixed capital (CAPEX) to operating expenses (OPEX).

Several recent works have developed digital twin systems for robotics, each targeting specific industrial applications and integration scenarios. In [8], Mertes et al. present a 5G-enabled digital twin framework for machine tools, highlighting the role of advanced network architecture and edge computing in supporting responsive, closed-loop control over wireless infrastructure. Singh et al. [1] describe the implementation of a ROS-compatible digital twin for industrial robotic arms using Unity for visualization and control, with a focus on real-time synchronization between physical and virtual systems.

Feddoul et al. [2] provide a systematic review of digital twin use cases in human–robot collaboration, emphasizing the importance of XR interfaces and real-time process mon-

Aias Sotiropoulos and Giorgos Kalpaktsoglou are with Fogus Innovations and Services PC, Greece. Emails: {aias, gkalpak}@fogus.gr. Dionysis Xenakis, Yannis Vittorakis and Melina Dolapsaki are with the Department of Digital Industry Technologies at the National and Kapodistrian University of Athens, Greece. Emails: {nio, ivittora, melinadol}@uoa.gr. This work received funding from Horizon Europe under the Marie Sklodowska Curie actions: AIAS (GA: 101131292) and EXARCH (GA: 101236244).

itoring for collaborative work environments. He et al. [4] detail the deployment of a digital twin solution for robotic arm inspection in aircraft manufacturing, leveraging a cloud-edge platform to enable efficient and scalable quality control processes. Girletti et al. [6] explore digital twin-based orchestration of collaborative robots at the edge, demonstrating how distributed deployment strategies can support complex, real-time automation scenarios.

While some of these studies include measurements of latency, resource usage, or DT responsiveness, few have provided comprehensive, empirical evaluations of CPU, GPU, RAM, and network usage across all architectural components, including both virtualized robot controllers and streaming-based XR visualization. Additionally, there is limited discussion on the practical challenges of deploying XR-enabled digital twins as fully containerized multi-access edge computing (MEC) services, or applications, in the edge-cloud continuum.

To address this gap, the present work provides a detailed experimental study of a fully-fledged ROS-compatible digital twin system for robotic arms, featuring an XR-enabled graphical user interface to enable remote monitoring, optimization, and control of industry X.0 (4.0/5.0) assets. Through a comprehensive experimental setup, we systematically profile the resource usage and performance of each software component of the XR-enabled digital twin robotic arm under realistic industrial workloads and edge-cloud deployments, including both physical and emulated scenarios. Our results offer practical insights and measured requirements for deploying scalable, XR-enabled digital twin applications in next-generation manufacturing and robotics.

II. Robotic Arm Digital Twin

A. Infrastructure

A conceptual architecture of the proposed digital twin robotic arm, shown in Fig. 1, is structured around three principal modules:

- A ROS-based robotic arm (physical or emulated),
- An XR-enabled digital twin application,
- A video streaming service for remote observation.

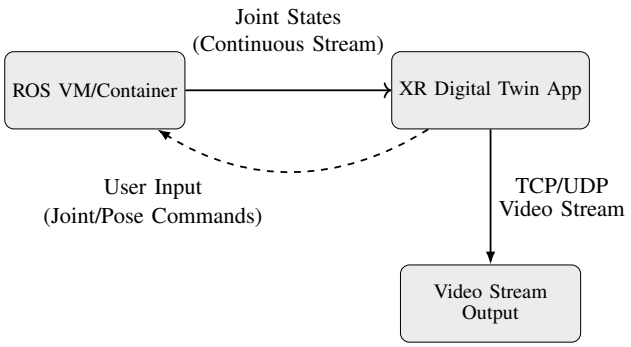

Fig. 1. High-level architecture: synchronous communication between ROS and the XR digital twin app, with user input and remote video stream output.

The ROS-enabled robotic arm provides a standardized robotics middleware environment for low-level control, state monitoring, and data exchange with external applications. The XR digital twin application acts as the system's central hub, integrating visualization, user interaction (including AR/VR input), and bi-directional communication with the robot. To support additional supervision or collaborative use cases, the system incorporates a dedicated video streaming component, allowing external users or observers to monitor the digital twin's state from any location or device. This modular structure supports flexible deployment in industrial, research, and training scenarios, with all core elements communicating via standard network protocols.

B. System Description

In our implementation, each module is realized with open-source, or commercial-off-the-shelf, tools to create a fully-fledged edge-enabled prototype for Industry X.0 setups:

ROS-Enabled Robotic Arm: The robotic arm, which can be either a physical Motoman Yaskawa, or a virtualized model, is managed using the ROS 2 Humble OS [11], deployed on Ubuntu 22.04 within a Docker container, or a standalone VM. Motion planning and inverse kinematics are provided by using the MoveIt 2 [12] and TRAC-IK libraries [13]. The robot receives both joint and pose commands (sent via the XR app) and continuously publishes joint state updates through standardized ROS topics. Communication with the digital twin robotic arm is handled via the ROS–TCP Connector [14] for robust and low-latency message exchange.

XR Digital Twin Application: The XR-enabled digital twin application is developed in Unity 6000.0.26f1 [15], leveraging the ROS–TCP Connector [14] to exchange high fidelity data with ROS (typically every 1-5 ms). The visualization uses a 3D Motoman model acquired from the Motoman GitHub [16] and imported through the URDF Importer plugin [17], adapted furthermore for Unity rendering and adjusted for the needs of our project. XR functionality is powered by the Mixed Reality Toolkit 3 (MRTK 3) [18], providing immersive input and a flexible GUI for teleoperation. The XR app features an interactive control menu that enables users to:

- Reset the robot to its initial configuration.
- Independently control each joint using buttons, or XR gestures (Joint Control).
- Manipulate the end effector's pose (position and orientation) with either GUI buttons or gestures (Tip Control).

The application runs either natively on AR headsets, or as a high-fidelity desktop application that is streamed via Unity Remoting, supporting both desktop and immersive access.

User: The user interacts with the system through either a desktop PC, or an AR device, such as Microsoft HoloLens 2. Input can be provided via mouse, keyboard, or immersive XR inputs (gesture, gaze, voice), while visual feedback is delivered in real-time through the XR digital twin application. For passive observation, the user may also connect as a viewer via standard video streaming clients (e.g., VLC), or web browsers.

Video Streaming Subsystem: For external monitoring, the XR app provides live camera feeds, which can be streamed from the HoloLens, or the Unity application, using the

HTTP/MP4, NDI, or RTSP protocols. Third-party clients (e.g., VLC) and web browsers can access these streams for passive observation. The system supports multiple virtual cameras, allowing for multi-angle scene viewing by additional users.

C. Information Flow Outline

The system's information flow is modular and bidirectional, enabling both real-time control and passive monitoring. Since our experimental study focuses on the performance of the Digital Twin Robotic Arm app residing at the MEC infrastructure, in the sequel we term the respective upstream/downstream information flows (connections) with respect to the digital components involved.

- **User → XR App:** Users place joint or pose commands via the GUI or XR input, e.g., using the Hololens 2 headset; these are transmitted to the XR app in real time. We term this stream of data as $Rx_{4.2}$ in the sequel.
- **XR App → ROS:** The XR app sends user commands to the ROS robot controller over TCP (using the ROS–TCP Connector). We term this stream of data as $Tx_{2.1}$.
- **ROS → XR App:** ROS publishes continuous joint state updates, which are received by the XR app and used to update the 3D visualization and telemetry. We term this stream of data as $Rx_{1.2}$. To avoid synchronization errors in the Digital Twin Robotic arm, we choose to synchronize the state of the Digital Twin app with its physical counterpart (emulated by a ROS VM).
- **XR App → User:** The XR app provides visual feedback and system state via its interactive GUI and immersive scene rendering, supporting both PC and AR displays. We term this stream of data as $Tx_{2.2}$.
- **XR App → Video Streaming Subsystem → Passive Users:** The XR app or HoloLens generates a live video stream (HTTP/MP4, NDI, RTSP) accessible by remote clients, such as VLC or browser-based viewers for remote monitoring. Multiple virtual camera angles can be configured. We term this stream of data as $Tx_{4.3}$.

This architecture enables interactive teleoperation, real-time system feedback, and scalable external observation, all implemented by using widely supported open technologies.

III. Experimental Study

A. Experimental setup

Fig. 2 illustrates the fully-softwarized and natively containerized architecture that we have developed for the purposes of our experimental study towards the resource consumption of the XR-enabled Digital Twin Robotic Arm application, similarly with [19]. All the components of our experimental setup are implemented in the form of Docker containers in Windows 11. We now detail their operation below.

- *ROS-enabled Robotic Arm Emulator (Container 1).* Responsible for emulating the state of a fully-fledged Yaskawa collaborative robotic (Cobot) arm HC10 using the ROS Humble OS in combination with the MoveIt2 library, which enables calculations on inverse kinematics. The Yaskawa Cobot robotic arm HC10 assumes a 6-axis *SLURBT* setup where the the S-axis rotates the body of the robotic arm horizontally, the L-axis moves the body forward-backward, the U-axis moves the arm up/down, the R-axis rotates the arm, the B-axis moves the tip of the arm up/down and the T-axis rotates the tip of the arm.
- *Digital Twin Robotic Arm Application (Container 2).* An XR- and MEC-enabled application built in Unity, which hosts the immersive 3D model of the Yaskawa HC10 Cobot. It receives a continuous stream of data through the $Rx_{1.2}$ interface from the ROS-enabled robot emulator, to synchronize the position of the robotic arm joints in real-time over a TCP connection. It processes the user input received from the User XR headset, i.e., using Hololens remoting, converts the user input into ROS-compatible control feedback and conveys the control feedback to the ROS-enabled Robot Emulator (Container 1). It visualizes the state of the 3D robotic arm model (digital twin) to the User XR headset.
- *User XR headset*. Deploys the Hololens remoting UDP service to feed the GUI interface of the end user in full compliance with the 3D model state at the Digital Twin Robotic Arm application, consuming feedback through a perfectly synchronized control loop with the physical robotic arm counterpart (at 50Hz). Enables the user to remotely handle the physical counterpart of the robotic arm, as emulated by Container 1, either by using *Tip Control*, i.e., the user handles a ball placed at the nose of the robot, or by using *Joint Control*, i.e., the user directly increases/decreases the position of a specific joint by a delta step and input speed (configurable parameters).
- *Video Client*. Receives a video stream from the User XR Headset to enable remote users passively look at what the User XR headset output. Implemented by Container 3.
- *Monitoring Toolset Service*. Implements our DT-tailored monitoring toolset. Hosted in Container 5, it utilizes multiple online available toolkits to remotely measure Rx/Tx throughput on a per socket basis as well as CPU, random access memory (RAM) and GPU usage on per PID basis at Container 2. Network usage statistics are derived with a sampling rate of once per 10ms for all the distinct communication ports (Tx and Rx) from/to the Digital Twin Robotic Arm, i.e., ports Tx2.1, Rx1.2, Rx4.2, Tx2.4, by utilizing the *Process Monitor v4.01* tool [20]. The *Performance Counter* class from the Windows System Diagnostics is also employed to measure instantaneous CPU utilization as well as real RAM consumption (i.e., the Memory Working set metric - see below) once per 0.5ms [21]. GPU utilization, memory and power consumption are measured by using the NVIDIA System Management Interface (Nvidia-smi), which is a command line utility for NVIDIA GPU devices [22].

All the aforementioned services (except from the User XR headset that is actual hardware device) have been built in an automated fashion using the Docker Compose framework.

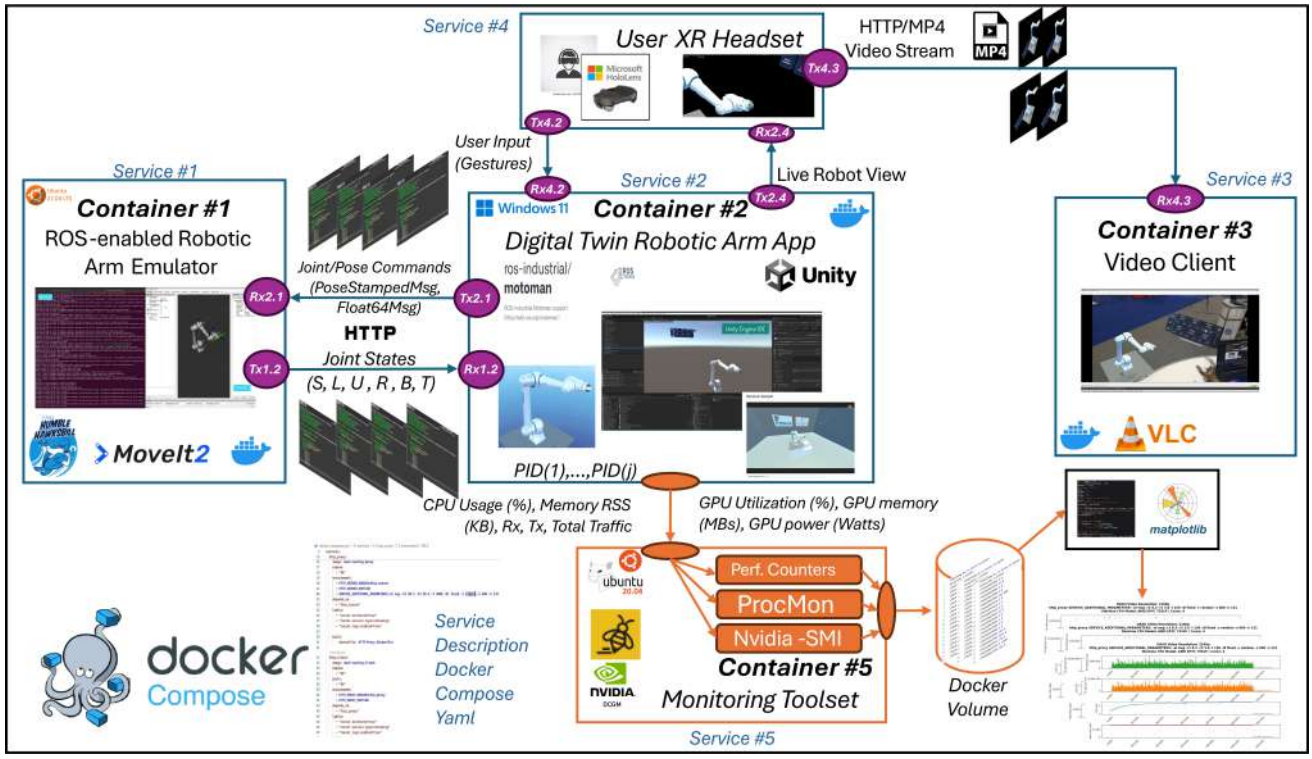

Fig. 2. Modular system architecture for XR-enabled digital twins: ROS-based robot emulation (Container #1), XR digital twin app with user input and live visualization (Container #2), real-time video streaming for external observation (Container #3), and a dedicated monitoring toolset (Container #4) for resource profiling. Main information and control flows are shown.

Its role is to start the containers in the right order, collect information about the environment (e.g. number of CPU cores, memory and GPU used), configure the experimental parameters and setup metadata (e.g., algorithm name, timestamp), dynamically name the resulting folder that will host the performance records on network, CPU, memory and GPU usage (recorded in .csv files) and collect the associated files from each container in the "Results" folder. The corresponding results where then processed and plotted in an automated fashion by using Python and the Matplotlib library.

The experimental campaigns presented in this work, assess the performance of the Digital Twin Robotic Arm application in view of the following important key metrics:

Rx1.2 [Mbps]: Also referred to as TCP Rx. Throughput measured at the reception port Rx1.2 (Fig. 2), i.e., ROS-enabled Robotic Arm Emulator to Digital Twin Robotic Arm app (Unity). It enables the estimation of the required transmission throughput from the physical to the digital counterpart of the robotic arm. Attaining the reported measurement values plays an instrumental role in the timely visualization of the actual Robotic Arm state to its Digital Twin counterpart.

Tx2.1 [Mbps]: Also referred to as TCP Tx. Throughput measured at the transmission port Tx2.1 (Fig. 2), i.e., Digital Twin Robotic Arm app (Unity) to ROS-enabled Robotic Arm Emulator. It enables the estimation of the required transmission throughput from the digital twin Robotic Arm app (residing at the MEC host) to the physical (or emulated) robotic arm. Attaining the reported measurement values plays a critical role in the timely transfer of the user control commands to the physical (or emulated) robotic arm counterpart.

Rx4.2 [Mbps]: Also referred to as UDP Rx. Throughput measured at the reception port Rx4.2 (Fig. 2), i.e., User XR Headset to Digital Twin Robotic Arm app (Unity). It provides an estimate of the required transmission throughput from the XR Headset of the user to the MEC-enabled Digital Twin Robotic Arm app (Unity). Attaining the reported measurement values plays a critical role in the timely transfer of the user control commands from the user to the digital twin app.

Tx2.1 [Mbps]: Also referred to as UDP Tx. Throughput measured at the transmission port Tx2.4 (Fig. 2), i.e., Digital Twin Robotic Arm app (Unity) to User XR Headset. It provides an estimate of the required transmission throughput from the digital twin Robotic Arm app (residing at the MEC host) to the User's GUI. Attaining the reported measurement values is of paramount importance to timely depict the joint state of both physical and digital counterparts to the user.

CPU utilization [%]: Percentage of local/instantaneous CPU usage by the Digital Twin App at the MEC service host, assuming single and isolated physical/digital instances.

Memory Working Set [MBs]: Actual RAM usage. The Working Set counter reports the current size, in bytes, of the set of memory pages in a process's virtual address space that are currently resident in physical RAM, i.e., the amount of the Digital Twin app's memory footprint that is actually loaded

in RAM at the moment the counter is sampled. If the OS has sufficient free RAM, pages may remain in the working set even if not immediately in use; when memory pressure rises, the OS may trim seldom-used pages out of the memory working set. If those pages are needed again, they'll be faulted back into RAM ("soft-faulted") before eviction.

GPU Utilization [%]: GPU usage in runtime for the Digital Twin app at the MEC host (Container 2), normalized over a single local processor capacity (i.e., can exceed 100%).

GPU Memory [GBs]: Percentage of allocated GPU memory used by the Digital Twin Robotic Arm app instance.

GPU Power [Watts]: Power consumption of the GPU used to support the Digital Twin Robotic Arm app instance.

B. Experimental results and discussion

Fig. 3 presents the instantaneous performance of the Digital Twin Robotic Arm app (Container 2) as measured using our monitoring toolset at Container 4. Regarding the *Rx4.2 (Hololens to Unity App traffic)*, the incoming UDP stream from the User XR headset to the Digital Twin Robotic Arm app results in an average of about 0.385 Mbps, with a minimum throughput of about 0 Mbps, maximum at about 2.00 Mbps and a standard deviation of about 0.34 Mbps). Overall, the derived performance results for the $Rx_{4.2}$ exhibit a steady baseline punctuated by short, high-amplitude bursts every few milliseconds, reflecting intermittent batches of spatial-tracking sensor packets arriving from the User XR headset. At the same time, in terms of transmission throughput from the Robotic Arm Digital Twin to the User XR headest, we observe an average throughput of about 19.8 Mbps, with a minimum of 0 Mbps and a maximum of 48.5 Mbps with a standard deviation at about 15 Mbps. After an initial buffer ramp-up, the transmitted throughput settles into a sustained high-bandwidth yet highly variant pattern, reflecting regular scene updates and state synchronization being pushed over the respective link.

Let us now move on the throughput requirements between the ROS-enabled Robotic Arm Emulator and its digital twin counterpart. The ROS control channel, measured in the $Rx_{1.2}$ port, exhibits small-bandwidth flows that average at 0.16 Mbps with a minimum and maximum values at 0 and 0.64 Mbps, respectively, characterized by a standard deviation of about 0.19 Mbps. This trace displays the continuous but low-amplitude transfer of the Robotic Arm's joint state that is characteristic of messages transmitted for telemetry and acknowledgements between the physical and digital robotic arm counterparts. On the other hand, the digital twin counterpart is shown to transfer control commands to the ROS-enabled Robotic Arm emulator with an average tranmission throughput of 0.008 Mbps, having minimum and maximum values at 0 and 0.09 Mbps, respectively, characterized by a standard deviation of about 0.025 Mbps. The respective performance trend aligns with the dense series of small spikes that reflect command packets sent back into the ROS system.

Moving the CPU utilization, the digital twin MEC application maintains an 81.4% average CPU load, oscillating between 0% and 100% with a standard deviation of about 17.7%. The utilization curve alternates between higher-compute intervals—likely during physics or collision updates, and lower plateaus, which correspond to network or I/O waits at the Unity app. In terms of Memory Working Set (RAM), the resident memory footprint of the digital twin application at the MEC host is essentially flat, averaging close to 660.2 MBs and varying only between 659 MB and 661 MB with a standard deviation of about 0.4 MB, indicating stable RAM usage by the Robotic Arm Digital Twin app at the MEC host.

In terms of GPU Utilization and Memory, we observe that the GPU load at the MEC host averages to 42.3% with minimum and maximum values ranging between 40 and 45%, respectively, having a standard deviation of about 1.2%. The GPU memory usage averages at about 3.2 GBs with a roughly static pattern (standard deviation close to 0). The respective results indicate that the support of MEC-enabled rendering and computing kernels consume a moderate but relatively consistent share of the available GPU's resources. Finally, in terms of power consumption at the GPU card, the Digital Twin Robotic Arm application consumes about 52.2 Watts of power by ranging between 37 and 60 Watts with a standard deviation of about 2.43 Watts. Thus, apart from a single dive to 37 W around 40 secs, which aligns with a slight CPU utilization drop, the power curve holds a steady profile, reflecting that the Digital Twin Robotic Arm application offloads to the GPU a rather regular compute and rendering workload at the host.

IV. Conclusion

In this work, we have presented a fully-fledged ROS-compatible Digital Twin MEC application of a 6 degrees of freedom (DOF) Collaborative Robotic (Cobot) arm and assessed its resource consumption through thorough experimental campaigns. We have detailed all the components necessary for the support of the Digital Twin Robotic Arm, in the context of remote monitoring, optimization and control of Industry X.0 assets, and implemented a fully automated natively containerized experimental setup capable to assess key performance metrics like Rx/Tx throughput, CPU and Memory RSS utilization as well as GPU utilization, memory usage and power consumption at the MEC Service host. Our results provide benchmark values for the deployment of MEC-enabled Digital Twin applications in emerging Industry 4.0/5.0 setups, opening the road ahead for resource-efficient digitalization and remote operation of Industry X.0 assets.

References

[1] M. Singh et al., "Unity and ROS as a Digital and Communication Layer for Digital Twin Application: Case Study of Robotic Arm in a Smart Manufacturing Cell," *Sensors*, vol. 24, no. 17, Article 5680, 2024.

[2] Y. Feddoul et al., "Exploring human-machine collaboration in industry: a systematic literature review of digital twin and robotics interfaced with extended reality technologies," *Intern. J. of Advanced Manufact. Tech.*, vol. 129, pp. 1749–1769, Oct. 2023.

[3] X. Tu, J. Autiosalo, R. Ala-Laurinaho, C. Yang, P. Salminen, and K. Tammi, "TwinXR: Method for using digital twin descriptions in industrial eXtended reality applications," *Frontiers in Virtual Reality*, vol. 4, Article 1019080, 2023.

[4] Y. He, Z. Xue, X. Li, Y. Jin, and J. Yan, "Application of Cloud-Edge Based Digital Twin in Robotic Arm Scanning of Aircraft Fuselage Panels," *IEEE Access*, vol. 10, pp. 101436–101449, 2022.

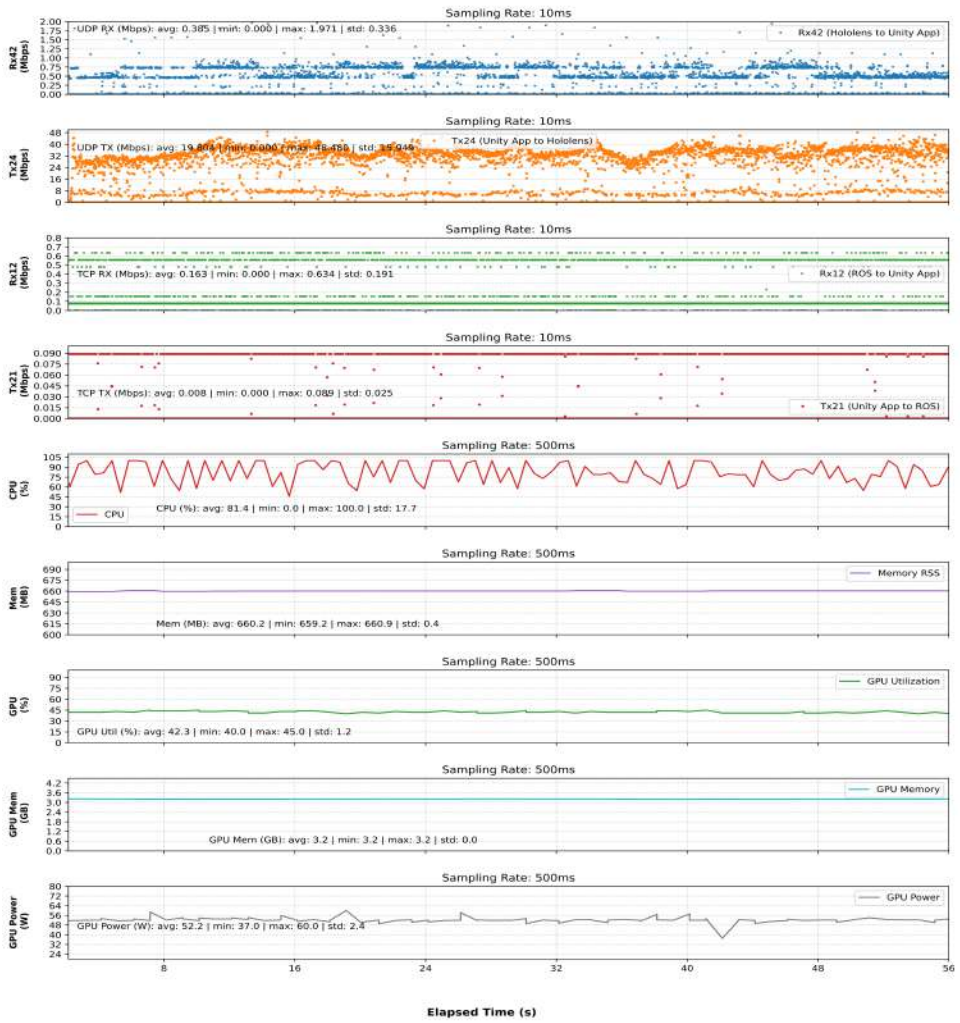

Fig. 3. Instantaneous performance of the Digital Twin Robotic Arm app (as measured at Container 2).

[5] Y. Xu, X. He, R. Zhang, and H. Zhang, "A generic and modularized Digital twin enabled human-robot collaboration," *Procedia CIRP*, vol. 108, pp. 882–887, 2022.

[6] M. Girletti, G. Delnevo, S. Giordano, G. Ferrari, and P. Ferrari, "An Intelligent Edge-based Digital Twin for Robotics," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 8, pp. 9198–9208, Aug. 2023.

[7] Q. Zhang, W. Zhuang, Z. Xu, et al., "DEEP: A Vertical-Oriented Intelligent and Automated Platform for the Edge and Fog," *IEEE Network*, vol. 35, no. 2, pp. 173–179, Mar.–Apr. 2021.

[8] J. Mertes, C. Schellenberger, L. Yi, M. Schmitz, M. Glatt, M. Klar, B. Ravani, H. D. Schotten, and J. C. Aurich, "Experimental evaluation of 5G performance based on a digital twin of a machine tool," *CIRP Journal of Manufact. Science and Techn.*, vol. 55, pp. 141–152, 2024.

[9] S. Wei, H. Zhang, and L. Ma, "Experimental evaluation of 5G performance based on a digital twin of a machine tool," *Journal of Manufacturing Systems*, vol. 69, pp. 221–233, 2024.

[10] S. Ghosh et al., "Digital Twin for Remote Control of Robotic Arm via Wearable Glove in Smart Agriculture," *IEEE Access*, 2023.

[11] Open Robotics, "ROS 2 Humble Hawksbill," [Online]. https://docs.ros.org/en/humble/index.html. Accessed: July 2025.

[12] PickNik Robotics, "MoveIt 2," [Online]. https://moveit.picknik.ai/humble/index.html. Accessed: July 2025.

[13] TRACLabs, "TRAC-IK Kinematics Plugin," [Online]. https://bitbucket.org/traclabs/trac_ik/src/HEAD/trac_ik_kinematics_plugin/. Accessed: July 2025.

[14] Unity Technologies, "ROS–TCP Connector," [Online]. https://github.com/Unity-Technologies/ROS-TCP-Connector. Accessed: July 2025.

[15] Unity Technologies, "Unity Real-Time Development Platform," [Online]. https://unity.com/releases/unity-6. Accessed: July 2025.

[16] ROS-Industrial, "motoman – ROS-Industrial support for Motoman robots," [Online]. https://github.com/ros-industrial/motoman. July 2025.

[17] Unity Technologies, "URDF Importer," [Online]. https://github.com/Unity-Technologies/URDF-Importer. Accessed: July 2025.

[18] Microsoft, "Mixed Reality Toolkit 3 (MRTK3)", [Online]. https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-overview/. Accessed: July 2025.

[19] I. Vittorakis, D. Xenakis, G. Kalpaktsoglou, "On the Resource Consumption of 2D Human Pose Estimation Services in MEC-enabled Networks", 2025 IEEE NFV-SDN, 10–12 November 2025, Athens, Greece.

[20] Microsoft, "Process Monitor v4.01". [Online]. https://learn.microsoft.com/en-us/sysinternals/downloads/procmon. Accessed July 2025.

[21] Microsoft, "Performance Counters class: System Diagnostics". [Online]. https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.performancecounter. July, 2025.

[22] NVIDIA, "NVIDIA System Management Interface (SMI)". [Online]. https://developer.nvidia.com/system-management-interface. Accessed July, 2025.