

A Comparative Implementation Benchmark of Gazebo and Unity for xArm Lite6 Robotic Grasping Simulation

Haotian Yang^{*}, Yijia Zhang^{}

Xi'an Jiaotong-Liverpool University, Suzhou, China

Email: *13812753155@163.com, zhangyijia041010@outlook.com

How to cite this paper: Yang, H.T. and Zhang, Y.J. (2026) A Comparative Implementation Benchmark of Gazebo and Unity for xArm Lite6 Robotic Grasping Simulation. *Journal of Computer and Communications*, 14, 155-166.

<https://doi.org/10.4236/jcc.2026.149009>

Received: August 20, 2026

Accepted: September 27, 2026

Published: September 30, 2026

Abstract

Robot simulation platforms are widely used for motion planning, control validation, grasping-task prototyping, and digital-twin demonstration, but platform comparisons are often based on qualitative descriptors. This paper presents an implementation benchmark of Gazebo and Unity using the same xArm Lite6 six-degree-of-freedom robot and a six-stage pick-and-place task. Gazebo uses ROS Noetic, MoveIt!/OMPL, joint controllers, gazebo_ros_control, and gazebo_grasp_fix; Unity uses the URDF Importer, Articulation Body joints, staged C# targets, and trigger-based parent-child attachment. The revised evaluation reports auditable indicators rather than unsupported impressions, and it explicitly separates completed implementation evidence from prospective repeated-trial measurements. Both implementations completed all six task stages (6/6). In the implemented prototypes, Gazebo scored 1 for online planning and 1 for controller-chain execution, whereas Unity scored 0 for both because its trajectory was scripted. Both scored 1 for deterministic attachment dependence and 0 for physics-preserving grasp transport. The results therefore support Gazebo for planning/control-pipeline validation and Unity for scene-level interactive reproduction, while neither implementation supports direct claims about physical grasp robustness. The paper further analyzes Sim-to-Real failure mechanisms, specifies a controlled Gazebo ablation, and documents implementation failure conditions.

Keywords

Robot Simulation, Gazebo, Unity, xArm Lite6, Robotic Grasping, Digital Twin, MoveIt, ROS

*Corresponding author.

1. Introduction

Robotic simulation has become an important stage in the development of manipulation systems. Compared with repeated physical testing, simulation can reduce hardware cost and operational risk while supporting robot modeling, controller tuning, motion planning, collision checking, and task-level validation [1]-[3]. With the increasing use of digital twins in manufacturing and automation, simulation platforms are also expected to provide not only functional robot execution, but also visual clarity, interaction capability, and extensibility for virtual-physical integration [4] [5].

Gazebo and Unity represent two commonly used but technically different routes for robot simulation. Gazebo is closely connected with the Robot Operating System (ROS), URDF-based modeling, MoveIt!, OMPL, and controller execution, making it suitable for algorithm verification and planning-oriented robotics research [6] [7]. Unity, in contrast, is widely used for real-time visualization and interactive scene construction. With the Unity Robotics URDF Importer and Articulation Body physics component, URDF robot models can be reconstructed in Unity and controlled through scripts or external communication interfaces [8] [9].

The difference between these two platforms becomes especially visible in robotic grasping tasks. A complete pick-and-place task involves not only joint motion, but also contact, friction, collision response, object stability, and task-state management. In practice, simulation systems often use simplified grasping mechanisms to ensure stable task reproduction. Gazebo projects may use plugins such as `gazebo_grasp_fix` to attach an object to the gripper after contact conditions are satisfied [10]. Unity implementations may use detection volumes and parent-child binding to make a target object follow the gripper during transportation. These solutions are useful for reproducible simulation, but they are not equivalent to full physical grasping.

This paper compares Gazebo and Unity by implementing the same xArm Lite6 pick-and-place task in both environments. The purpose is not to declare one platform universally better, but to identify how each platform supports a different simulation objective. The main contributions are as follows:

- A dual-platform implementation of an xArm Lite6 grasping task using Gazebo and Unity.
- An operationalized comparison using six-stage completion and binary indicators for planning, controller execution, deterministic attachment, physics-preserving transport, and scene-level interaction.
- A mechanism-level analysis of Sim-to-Real risks, implementation failure modes, and a reproducible Gazebo physics-only versus plugin-assisted ablation protocol.

2. Related Work

Robot simulation involves multiple technical layers, including model description, physical simulation, motion planning, controller execution, and result visualization. ROS provides a common communication and software framework for robot applications [1], while Gazebo offers a three-dimensional dynamic simulation en-

vironment that has been widely used in robotics research [2]. Collins *et al.* reviewed physics simulators for robotic applications and emphasized that different simulators vary in physical modeling capability, computational performance, supported features, and ease of use [3].

For robotic manipulation, MoveIt! is a commonly used motion-planning framework that supports robot model configuration, collision checking, planning groups, and trajectory execution [6]. It can call OMPL to generate sampling-based motion plans in high-dimensional configuration spaces [7]. These tools make Gazebo suitable for validating ROS-based robot control workflows.

Unity follows a different technical direction. The Unity Robotics URDF Importer allows a URDF robot model to be imported into a Unity scene [8]. The Articulation Body component uses reduced coordinates and is designed for articulated systems such as robot arms [9]. This makes Unity useful for visually clear and interactive robot demonstrations, especially when digital-twin presentation is a major goal.

Digital twin research further increases the need to understand platform suitability. Kritzinger *et al.* distinguish digital models, digital shadows, and digital twins by the strength of data integration between physical and virtual entities [4]. Tao *et al.* argue that digital twins require the integration of models, data, and services in intelligent manufacturing [5]. For robotic arms, this means that a simulation platform should be evaluated not only by whether a task can be displayed, but also by how well it supports control integration, state synchronization, visual interaction, and future physical validation.

3. Methodology

3.1. Robot Model and Task Definition

The experimental object is the xArm Lite6, a desktop-level six-degree-of-freedom robotic arm. According to UFACTORY documentation, the Lite6 has a compact mechanical structure, a maximum payload of 600 g, a maximum end-effector speed of 500 mm/s, and a repeatability of ± 0.5 mm [11]. These characteristics make it suitable for small-scale manipulation and teaching-oriented simulation tasks.

To reduce model inconsistency, both platforms use the same source URDF/Xacro description of the xArm Lite6. The benchmark task is a predefined pick-and-place sequence rather than an open-ended grasping algorithm. This design keeps the focus on platform implementation and simulation workflow. **Table 1** summarizes the task definition.

3.2. Benchmark Procedure and Evaluation Criteria

The benchmark is an implementation comparison rather than a claim that the two physics engines are equivalent. To replace qualitative labels with auditable measures, the common task is decomposed into six ordered stages: approach, grasp, lift, transfer, lower, and release. Task-stage completion is $C = n_{\text{completed}}/6$. Five binary implementation indicators are also recorded: $P = 1$ when an

Table 1. Definition of the xArm Lite6 grasping benchmark.

Item	Description
Robot model	xArm Lite6 six-degree-of-freedom robotic arm
Task type	Pick-and-place grasping simulation
Initial state	Home pose with target object placed in reachable workspace
Task sequence	Approach, grasp, lift, transfer, lower, and release
Compared platforms	Gazebo and Unity
Comparison aspects	Control workflow, motion behavior, grasping mechanism, visual presentation, development workflow

online motion planner generates the executed trajectory; $R = 1$ when that trajectory is executed through a robot-controller chain; $A = 1$ when successful transport depends on an explicit attachment or parenting operation; $F = 1$ only when the transported object remains governed by contact, friction, mass, and inertia without kinematic attachment; and $I = 1$ when the implemented prototype provides scene-level scripted interaction for reproduction and display. These indicators describe the submitted implementations, not every capability of Gazebo or Unity; they also make clear which claims are supported by current evidence and which require additional repeated-trial experiments.

The procedure contains four steps. First, the same xArm Lite6 URDF/Xacro description and the same six-stage task are prepared for both platforms. Second, the platform-specific joint and scene components are configured. Third, each implementation is checked against the complete stage sequence. Fourth, C, P, R, A, F, and I are assigned from inspectable implementation evidence, as defined in **Table 2**. The available submission package contains the implemented workflows and task sequence but no repeated-trial log, timing trace, force trace, or joint-error

Table 2. Evaluation criteria used in the benchmark.

Criterion	Operational Measure
Task-stage completion	$C = n_completed/6$ across approach, grasp, lift, transfer, lower, and release.
Online planning	$P = 1$ only if an online planner generates the executed trajectory; otherwise $P = 0$.
Controller-chain execution	$R = 1$ only if the trajectory is sent through a robot-controller chain; otherwise $R = 0$.
Attachment dependence	$A = 1$ if transport depends on explicit attach/parent logic; otherwise $A = 0$.
Physics-preserving transport	$F = 1$ only when contact, friction, mass, and inertia remain active without kinematic attachment.
Scene-level interaction	$I = 1$ when the implemented prototype exposes scripted scene-level interaction for reproduction/display.

record. Accordingly, the revision reports no fabricated success-rate, duration, oscillation, or frame-rate statistics; Section 4.4 instead specifies the controlled experiment and reporting fields required to obtain them.

3.3. Gazebo Implementation

The Gazebo implementation is built around the ROS Noetic ecosystem. The xArm Lite6 URDF/Xacro model is loaded into a ROS workspace and instantiated in the Gazebo simulation environment. The robot description contains the link, joint, visual mesh, collision mesh, inertial, and joint-limit information required for simulation. To drive the virtual joints through ROS controllers, the active arm joints and gripper joint are configured with transmissions.

Motion planning is implemented using MoveIt!. The arm planning group and gripper control group are configured so that the manipulator can plan motions between predefined poses. OMPL is used through MoveIt! to generate feasible joint trajectories. The generated trajectories are then sent to Gazebo through joint controllers and gazebo_ros_control. PID controller parameters are adjusted to reduce slow response and overshoot during target tracking.

The original contact-only behavior in Gazebo was not sufficiently stable for transporting the small object in this setup: small pose errors, limited contact area, friction/inertial mismatch, or controller oscillation can produce slip or loss of contact. gazebo_grasp_fix is therefore used after its contact condition is satisfied. The plugin creates a deterministic attachment to a gripper-related link, which prevents drops during the remaining stages but removes slip, force-closure, payload-coupling, and regrasp behavior from the transport phase. Thus, the Gazebo implementation is suitable for testing planning and controller sequencing, not for learning or validating a low-level physical grasp policy.

3.4. Unity Implementation

The Unity implementation uses the same xArm Lite6 URDF model. The model is imported through the Unity Robotics URDF Importer, which reconstructs the link hierarchy, mesh references, and joint structure. Each robot joint is represented using Articulation Body, which is more suitable than ordinary Rigidbody components for serial-chain robot structures.

The task sequence is controlled by C# scripts. The script defines staged joint targets corresponding to the approach, grasp, lift, transfer, lower, and release stages. During execution, joint target positions are interpolated to reduce abrupt changes in motion. This makes the Unity implementation a scripted reproduction of the task rather than a planner-driven trajectory-generation workflow.

The Unity grasping mechanism is also deterministic. A spherical trigger is placed near the gripper; when the object enters the region and the grasp stage is active, the object becomes a child of a GrabPoint or gripper transform. Parenting guarantees pose following until release but bypasses contact-force, friction, slip, payload inertia, and force-closure constraints. Trigger radius and timing therefore be-

come hidden task parameters: an oversized trigger can capture an object before valid contact, a small trigger can miss a valid grasp, and a parenting or Rigidbody-state error can cause penetration, discontinuous velocity, or an incorrect release pose.

As shown in **Figure 1**, both implementations start from the same URDF/Xacro model but diverge in their execution pipelines. Gazebo uses ROS Noetic, MoveIt!/OMPL, joint controllers, and gazebo_grasp_fix, whereas Unity uses the URDF Importer, Articulation Body joints, and staged C# targets.

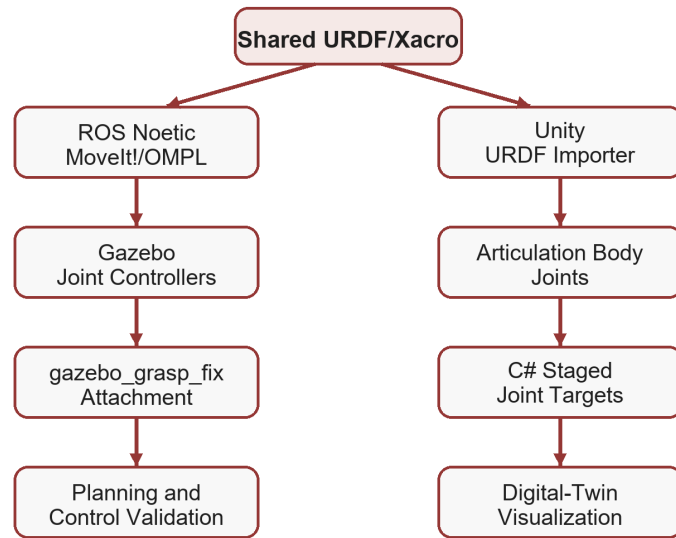


Figure 1. Comparison of the Gazebo and Unity implementation workflows for the same xArm Lite6 grasping task.

4. Results and Discussion

4.1. Task Execution Results

Both implementations completed the predefined six-stage sequence, giving $C = 6/6$ for Gazebo and $C = 6/6$ for Unity. In Gazebo, MoveIt!/OMPL generated the robot trajectory and the ROS controller chain executed it, so $P = 1$ and $R = 1$. These values demonstrate the presence of an integrated planning-and-execution path; they do not measure path optimality, controller accuracy, or real-robot success.

Gazebo completed object transport only with plugin-assisted stabilization in the submitted implementation, giving $A = 1$ and $F = 0$. Contact-only behavior could lose the object, while gazebo_grasp_fix removed this failure by attaching it after the contact condition. **Figure 2** documents the initial, approach/grasp, lift/hold, and transfer/release states. Because no paired repeat-run logs were supplied, the revision does not assign a numerical improvement to the plugin; the required paired comparison is defined in Section 4.4.

Unity also completed all six stages ($C = 6/6$). The current prototype executed predefined interpolated C# targets rather than online planning or ROS-controller execution, giving $P = 0$ and $R = 0$. Object transport depended on trigger-based parenting, giving $A = 1$ and $F = 0$. Scene-level scripted interaction was present (I

= 1). These indicators document implementation structure and repeatable task reproduction; they do not establish lower tracking error, higher physical grasp success, or a smaller Sim-to-Real gap.

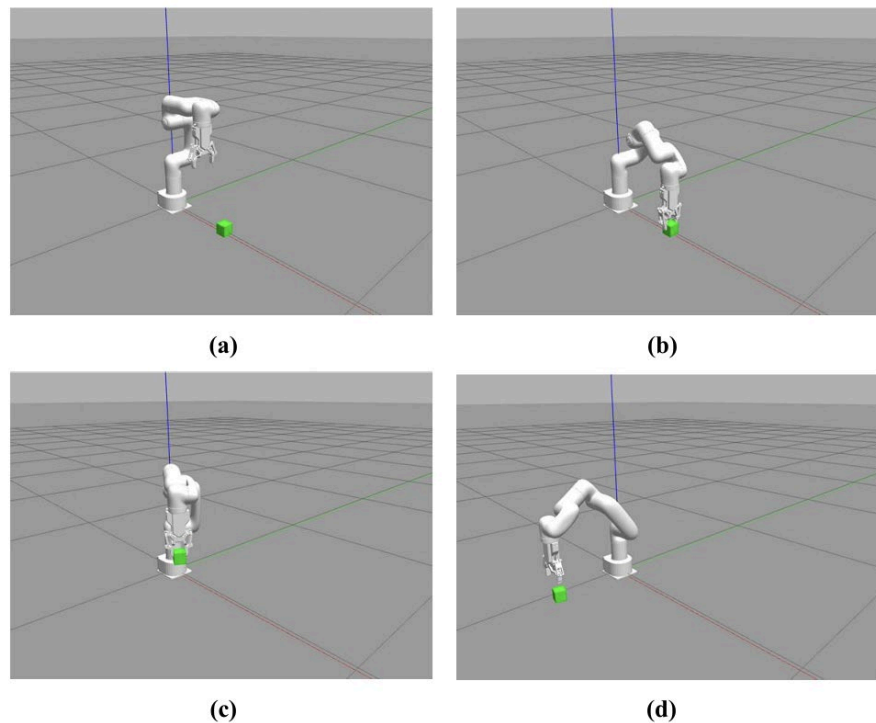


Figure 2. Gazebo grasping task execution sequence: (a) initial position; (b) target approach and grasping; (c) object lifting and holding; (d) object transfer and release.

4.2. Comparative Analysis

Table 3 lists the implementation components that determine the audit indicators. The comparison is limited to the submitted prototypes: Gazebo combines an online planner and ROS controller chain with plugin-assisted transport, whereas Unity combines staged script targets with trigger-based parenting and scene-level interaction.

Table 3. Main components used in the two implementations.

Aspect	Gazebo	Unity
Model import	URDF/Xacro in ROS workspace	URDF Importer in Unity
Joint execution	ROS controllers and PID parameters	Articulation Body drive targets
Motion generation	Online MoveIt!/OMPL planning	Predefined interpolated C# targets
Object transport	<code>gazebo_grasp_fix</code> attachment	Trigger detection and parenting
Online planner in prototype	Present	Not present
Scene-level scripted interaction	Not implemented	Implemented

Table 4 reports the operationalized results. The equal completion value (6/6) shows that both prototypes reproduced the scripted task, but the remaining indi-

cators show that they did so through different mechanisms. In particular, $A = 1$ and $F = 0$ for both platforms, so completion cannot be interpreted as evidence of physically robust grasping.

Table 4. Quantified audit of the implemented prototypes.

Metric	Gazebo	Unity
Task-stage completion, C	6/6	6/6
Online planning, P	1	0
Controller-chain execution, R	1	0
Deterministic attachment, A	1	1
Physics-preserving transport, F	0	0
Scene-level interaction, I	0	1

4.3. Sim-to-Real Consequences of Simplified Grasping

For `gazebo_grasp_fix`, a policy or task controller can appear successful after the plugin converts a contact event into a rigid attachment even when the real gripper would not achieve sufficient normal force, friction margin, or force closure. Direct deployment to an xArm Lite6 can therefore fail under jaw-pose error, uncertain friction, object compliance, mass-distribution error, controller delay, or acceleration-induced slip. The plugin is appropriate when the research question begins after grasp confirmation, such as testing collision-free transfer or task sequencing; it is not appropriate when grasp acquisition, holding force, or slip recovery is the research target.

Unity parenting creates a stronger abstraction boundary: the object pose is kinematically inherited from the `GrabPoint`, so payload-dependent torque, contact impulse, slip, rebound, and release velocity are not learned or validated. A controller tuned in this state can fail on hardware because the real arm must accelerate the payload, maintain contact during motion, and release with nonzero residual velocity. Parenting is therefore suitable for visualization, operator training, and state-machine validation, but a Sim-to-Real control study should retain Rigidbody dynamics, model gripper contact, randomize mass/friction/compliance, and validate the resulting command sequence on hardware [12]–[15].

For both platforms, attachment success must be logged separately from physical grasp success. A reproducible transfer study should define a grasp-success event from finger closure and object retention before attachment is enabled, then evaluate whether the same command survives randomized object pose, friction, mass, latency, and sensor noise. Without that separation, deterministic binding systematically underestimates the Sim-to-Real gap.

4.4. Controlled Grasping Ablation Protocol

The reviewer-proposed Gazebo control experiment should use paired trials of (a) contact/friction-only grasping and (b) the identical task with `gazebo_grasp_fix` enabled. The robot model, controller gains, trajectory, object model, initial pose,

physics step, random seed, and stopping criteria must be held constant. A minimum of 30 paired trials per condition is proposed to estimate uncertainty; each pair should use the same randomized object pose and friction/mass sample.

Table 5. Proposed paired gazebo grasping ablation.

Protocol element	Physics-only arm	Plugin-assisted arm
Contact handling	Contact and friction only	Same contact model plus gazebo_grasp_fix
Paired trials	≥ 30 ; identical seeds/poses	≥ 30 ; matched seeds/poses
Primary outcomes	Success rate; drop count	Success rate; drop count
Motion outcomes	Task time; joint RMS error; peak oscillation	Same measures
Contact outcomes	Slip distance; contact impulse	Attach latency; false attach/detach
Reporting	Mean/SD or median/IQR; 95% CI	Paired difference and effect size

The current submission package does not contain the repeat-run logs required to populate **Table 5** with measured outcomes. The table is therefore a prospective ablation protocol, not a report of completed trials. This evidence boundary is stated explicitly so that the 6/6 stage-completion audit is not misrepresented as a statistical performance comparison.

4.5. Implications for Platform Selection

Platform selection follows from the audited mechanisms rather than from visual adjectives. The implemented Gazebo workflow has $P = 1$ and $R = 1$, so it directly exercises online planning and controller execution. It is therefore the relevant prototype when the research question concerns MoveIt!/OMPL integration, collision-aware planning, ROS command flow, or controller behavior. Its $A = 1$ and $F = 0$ values also limit any conclusion about physical grasp robustness.

The implemented Unity workflow has $I = 1$ but $P = 0$ and $R = 0$. It is therefore useful for interactive scene reproduction, operator-facing visualization, and task-state demonstration, while the current scripted prototype cannot validate online motion planning or ROS-controller behavior. Unity could address this boundary by receiving the same ROS/MoveIt command source through ROS-TCP-Connector and by replacing parenting with contact-governed transport when physical grasp validity is required.

The equal completion ratio does not make the two platforms interchangeable. Both have $A = 1$ and $F = 0$, meaning that deterministic binding—not verified contact mechanics—ensures object retention. The benchmark therefore supports claims about workflow integration and task reproduction only; force closure, slip margin, grasp success under perturbation, and hardware transfer require the controlled and physical experiments described above.

4.6. Reproducibility Considerations

For a conference submission, reproducibility is strengthened by clearly separating platform-dependent implementation details from platform-independent task settings. The task settings include the robot model, initial pose, target object, grasping pose, placing pose, and stage order. These settings define the benchmark goal and should remain consistent across platforms. The implementation details, however, are allowed to differ because they reflect the normal development workflow of each platform. In Gazebo, reproducibility depends on the ROS package versions, MoveIt! configuration, controller parameters, Gazebo world file, and grasp-fix plugin settings. In Unity, reproducibility depends on the URDF Importer version, Articulation Body drive parameters, collider settings, C# script values, object detection radius, and parenting logic.

This separation also makes the comparison useful for later extension. If future experiments introduce physical hardware, the Gazebo workflow can be used to transfer planned joint trajectories to the real robot, while the Unity workflow can be used to build an interactive monitoring or demonstration interface. If the two platforms are connected through ROS-TCP-Connector or a similar bridge, the same motion command source could drive both the algorithmic simulator and the visual digital-twin scene. Such an integrated setup would provide a stronger basis for evaluating the Sim-to-Real gap.

4.7. Limitations Encountered during Implementation

Gazebo can fail before grasping when MoveIt! cannot find an inverse-kinematics or collision-free solution, when joint limits or URDF collision geometry are inconsistent, or when the planned trajectory is not accepted by the configured controller. During execution, poorly tuned PID gains or a mismatch between controller rate and physics step can cause lag, overshoot, or oscillation. During grasping, low friction, inaccurate inertial parameters, sparse collision geometry, or an unfavorable contact normal can cause slip. `gazebo_grasp_fix` can itself produce false attachment when the contact threshold is too permissive or missed attachment when contacts are intermittent; detachment thresholds can also release the object too early or keep it attached after the intended release.

Unity can fail during import when joint axes, hierarchy, mesh scale, collider geometry, or center-of-mass settings do not match the URDF. Articulation drives can oscillate or lag when stiffness, damping, force limits, or fixed timestep are inconsistent. Because the motion sequence is scripted, an object-position change or unexpected collision is not replanned and can make the gripper miss the target. The spherical trigger can bind an object without valid finger contact or fail to bind because the radius/timing is too small. Parenting while Rigidbody state is inconsistent can create interpenetration, velocity discontinuity, tunneling, or an incorrect release trajectory. These limitations are specific engineering conditions that must be checked in addition to whether the animation reaches the final state.

5. Limitations and Future Work

This work has four main limitations. First, the benchmark uses one predefined six-stage sequence rather than randomized object positions, shapes, and target poses. Second, the source package contains no repeated-trial logs, force traces, timing records, joint-error records, or frame-rate traces; therefore the reported quantitative evidence is limited to stage completion and binary implementation indicators. Third, the proposed physics-only versus plugin-assisted Gazebo ablation has not yet been executed. Fourth, neither implementation has been validated on a physical xArm Lite6. Future work should execute **Table 5**, report confidence intervals and effect sizes, connect Unity to the same ROS command source, replace deterministic binding with contact-governed transport, and measure the Sim-to-Real gap on hardware.

A complete follow-up should record task success, drops, task duration, joint RMS tracking error, peak oscillation, slip distance, contact impulse, attachment latency, false attachment/detachment, and rendering rate under matched seeds. The same randomized object pose, mass, friction, compliance, controller delay, and sensor-noise distributions should be applied across conditions. This would convert the present implementation audit into a repeatable performance benchmark and identify which simplified assumptions dominate transfer failure.

6. Conclusion

This paper presented an implementation benchmark of Gazebo and Unity for an xArm Lite6 pick-and-place task and revised the comparison around auditable evidence. Both prototypes completed all six stages ($C = 6/6$). Gazebo implemented online planning and controller-chain execution ($P = 1, R = 1$), whereas the current Unity prototype used predefined C# targets ($P = 0, R = 0$) and provided scene-level scripted interaction ($I = 1$). Both depended on deterministic object binding ($A = 1$) and neither preserved contact-governed grasp transport ($F = 0$). These results support Gazebo for ROS planning/control workflow validation and Unity for interactive task reproduction, but they do not establish physical grasp reliability. The added Sim-to-Real analysis shows how attachment can hide failures caused by pose error, friction, compliance, payload dynamics, latency, and slip; the proposed paired Gazebo ablation and documented failure conditions define the measurements required for a stronger follow-up study.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

- [1] Quigley, M., *et al.* (2009) ROS: An Open-Source Robot Operating System. *ICRA Workshop on Open Source Software*, Kobe.
- [2] Koenig, N. and Howard, A. (2004) Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator. 2004 *IEEE/RSJ International Conference on Intelli-*

- gent Robots and Systems (IROS)* (IEEE Cat. No.04CH37566), Sendai, 28 September-2 October 2004, 2149-2154. <https://doi.org/10.1109/iros.2004.1389727>
- [3] Collins, J., Chand, S., Vanderkop, A. and Howard, D. (2021) A Review of Physics Simulators for Robotic Applications. *IEEE Access*, **9**, 51416-51431. <https://doi.org/10.1109/access.2021.3068769>
 - [4] Kritzinger, W., Karner, M., Traar, G., Henjes, J. and Sihm, W. (2018) Digital Twin in Manufacturing: A Categorical Literature Review and Classification. *IFAC-PapersOnLine*, **51**, 1016-1022. <https://doi.org/10.1016/j.ifacol.2018.08.474>
 - [5] Tao, F., Zhang, H., Liu, A. and Nee, A.Y.C. (2019) Digital Twin in Industry: State-of-the-Art. *IEEE Transactions on Industrial Informatics*, **15**, 2405-2415. <https://doi.org/10.1109/tii.2018.2873186>
 - [6] Coleman, D., Sucan, I.A., Chitta, S. and Correll, N. (2014) Reducing the Barrier to Entry of Complex Robotic Software: A Move It! Case Study. *Journal of Software Engineering for Robotics*, **5**, 3-16.
 - [7] Sucan, I.A., Moll, M. and Kavraki, L.E. (2012) The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, **19**, 72-82. <https://doi.org/10.1109/mra.2012.2205651>
 - [8] Unity Technologies (2026) URDF Importer. Unity Robotics Hub. <https://github.com/Unity-Technologies/URDF-Importer>
 - [9] Unity Technologies (2026) ArticulationBody. Unity Scripting API Documentation. <https://docs.unity3d.com/ScriptReference/ArticulationBody.html>
 - [10] ROS Documentation (2026) Gazebo: GazeboGraspFix Class Reference. <https://docs.ros.org/>
 - [11] UFACTORY (2026) Lite6 Technical Specifications. UFACTORY Documentation. <https://www.ufactory.cc/>
 - [12] Jakobi, N., Husbands, P. and Harvey, I. (1995) Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics. In: Morán, F., Moreno, A., Merelo, J.J. and Chacón, P., Eds., *Advances in Artificial Life*, Springer, 704-720. https://doi.org/10.1007/3-540-59496-5_337
 - [13] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W. and Abbeel, P. (2017) Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. 2017 *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, 24-28 September 2017, 23-30. <https://doi.org/10.1109/iros.2017.8202133>
 - [14] Peng, X.B., Andrychowicz, M., Zaremba, W. and Abbeel, P. (2018) Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. 2018 *IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, 21-25 May 2018, 3803-3810. <https://doi.org/10.1109/icra.2018.8460528>
 - [15] Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N., *et al.* (2019) Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience. 2019 *International Conference on Robotics and Automation (ICRA)*, Montreal, 20-24 May 2019, 8973-8979. <https://doi.org/10.1109/icra.2019.8793789>.