Article

Not peer-reviewed version

Digital Twin Framework for Robot Path Planning and Real-Time Execution Using Unity-ROS Integration: Systems Architecture and Experimental Validation

Dhananjaya Kawshan * and Qingjin Peng *

Posted Date: 13 March 2026

doi: 10.20944/preprints202603.1064.v1

Keywords: digital twin; path planning; robot simulation; motion control; teleoperation

 doi:10.20944/preprints202603.1064.v1

Article

Digital Twin Framework for Robot Path Planning and Real-Time Execution Using Unity-ROS Integration: Systems Architecture and Experimental Validation

Dhananjaya Kawshan * and Qingjin Peng

Department of Mechanical Engineering, University of Manitoba, Winnipeg, Manitoba, Canada R3T 5V6

* Correspondence: qingjin.peng@umanitoba.ca

Abstract

Digital Twin (DT) systems combining physics-based simulation with hardware execution are critical for Industry 4.0 manufacturing, yet proprietary software solutions remain expensive and platform-dependent. This work addresses three technical challenges: maintaining geometric and kinematic fidelity across CAD-to-simulation conversion pipelines, synchronizing dual physics engines (Unity and ROS middleware) under hardware latency constraints, and optimizing motion planning while preserving trajectory quality and interactive responsiveness. We developed an integrated framework for a 7-Degree of Freedom manipulator using CAD modeling, URDF/SRDF semantic representation, and bidirectional Unity-ROS (Robot Operating System) communication via WebSocket connectors. Motion planning uses RRTConnect from OMPL with collision-aware optimization through the Flexible Collision Library. Validation across 12 manipulation trials demonstrated positional synchronization accuracy of ±2.0 degrees, motion planning performance of 0.064 ± 0.020 seconds. Latency analysis reveals that hardware execution to be the dominant system bottleneck, significantly exceeding network communication delays. The system achieves performance metrics comparable to proprietary industrial solutions. This work establishes a replicable, cost-effective Industry 4.0 framework, demonstrating that modern game engine technology combined with open-source robotics middleware can deliver DT systems matching proprietary solutions. The architecture and validated implementation enable adaptation to alternative robotic platforms and support broader adoption of simulation-validated automation in manufacturing contexts.

Keywords: digital twin; path planning; robot simulation; motion control; teleoperation

1. Introduction

Manufacturing industry is increasingly dependent on robotic manipulators to support flexible, high-mix production while maintaining tight constraints on safety, quality, and cycle time. Recent surveys highlight that modern industrial robots are moving from isolated, pre-programmed cells toward interconnected, sensor-rich systems that must be reconfigured rapidly and operated by non-expert personnel [1]. Within this broader Industry 4.0 and emerging Industry 5.0 context [2], DT technology's high-fidelity virtual replicas synchronized with physical assets have emerged as a key enabler for safe commissioning, offline programming, and continuous optimization of robotic work cells. DTs allow engineers to validate trajectories, tune controllers, and analyze system bottlenecks in simulation before applying changes on the shop floor, thereby reducing downtime and mitigating collision risk during integration and changeover.

DT concepts have been extensively explored in smart manufacturing, where multi-layer architectures integrate physical equipment, cyber models, and data-driven services for monitoring, diagnostics, and optimization [3,4]. At the system level, DT frameworks for collaborative and

 doi:10.20944/preprints202603.1064.v1

industrial robots emphasize virtual–physical synchronization, model fidelity, and interoperability across heterogeneous software and hardware platforms [5]. In particular, recent reviews on robot DT systems in manufacturing synthesize technologies for virtual modeling, real-time data integration, and closed-loop control and identify persistent challenges in achieving accurate kinematic/dynamic representation, robust communication, and scalable deployment across lines and factories [6,7]. Complementary analyses focused on robotics-specific DTs emphasize that, despite rapid progress, many implementations remain tightly coupled to proprietary toolchains, offer limited transparency of their inner workflows, or provide only coarse-grained performance metrics [8].

For robotic manipulators, DT research is increasingly focused on motion planning, process optimization, and task-level autonomy. Gantry and articulated robots have been coupled with DTs to perform 3D path planning of large-span structures [9], to simulate and optimize arc welding workstations [7,10], and to monitor robotic milling processes with high process fidelity [11,12]. These studies demonstrate that simulation-validated trajectories can improve production quality and reduce commissioning effort. However, they often rely on vendor-specific simulation environments or domain-specific solvers and typically report aggregate performance measures (such as weld seam quality or surface roughness) rather than a detailed decomposition of planning time, communication latency, and execution time within the DT loop. Robotic assembly DTs have similarly focused on integrating multi-source models and processing data to support planning and monitoring [13], but seldom expose the full mechanics of their geometric modeling pipeline or quantify how modeling choices propagate into end-effector accuracy and cycle time.

The choice of simulation engines and middleware is a central design decision for robotic DTs. Systematic comparisons of physics simulators and Reinforcement-Learning (RL) environments for robotics underscore trade-offs between physical accuracy, rendering speed, sensor support, and integration effort [14,15]. Recent DT-empowered robotic arm surveys explicitly compare ROS–Gazebo, Unity, MuJoCo, PyBullet, and MATLAB-based ecosystems, noting that Unity offers high-fidelity visualization and flexible scene composition but historically has been less standardized as a robotics simulation backend [16]. Several works have begun to exploit Unity for robotic manipulation: Unity-based pick-and-place simulations using UR3 manipulators have been benchmarked against Gazebo [17], and virtual fruit-picking robots have been modelled and controlled in Unity environments [18]. More recently, Unity and ROS have been combined to implement a DT for an ABB IRB 1200 industrial arm, in which the authors characterized communication latency and joint-space accuracy in a smart manufacturing cell [19]. These contributions establish Unity–ROS coupling as a viable platform for robotic DTs, but they place less emphasis on the complete CAD-to-URDF-to-Unity modeling pipeline, the integration of advanced motion planners, and a detailed breakdown of end-to-end timing behavior for collaborative manipulators.

Motion planning remains a critical component of DT-enabled robotic manipulation, particularly in cluttered or constrained workspaces. Comprehensive reviews of path planning in multi-robot and manipulator systems catalogue graph-search, potential-field, sampling-based, and optimization-based methods and highlight that Rapidly Exploring Random Tree (RRT) variants remain widely adopted in industrial practice due to their favorable trade-off between generality and computational efficiency [20]. For collaborative robots, control-design surveys emphasize that planners must generate trajectories that are not only collision-free but also dynamically feasible, smooth, and compatible with real-time safety constraints [21]. Within the RRT family, improved RRTConnect algorithms have been proposed that exploit triangular inequality properties and bidirectional tree growth to accelerate convergence and reduce path length compared to baseline RRT [22,23]. Other work refines RRT-like planners for mobile robots in narrow environments, reducing sampling inefficiency and improving path quality.

Given the variety of motion planning approaches and their trade-offs, Table 1 provides a structured comparison of three representative algorithms RRT, RRT*, and RRTConnect to establish the design rationale for algorithm selection in the proposed framework:

 doi:10.20944/preprints202603.1064.v1

Table 1. Comparative Analysis of Motion Planning Algorithms. [20,22,23]

Algorithm	Planning Time	Path Quality	Computational Overhead	Convergence Guarantee	Design Suitability
RRT	Medium	Poor: Jerky trajectories with excessive waypoints	Low	Probabilistic completeness only	Unsuitable: Post-processing latency exceeds RRTConnect
RRT* (Optimal RRT)	High	Excellent: Smooth, near-optimal trajectories	High (Rewiring)	Asymptotic optimality	Exceeds real-time budget; breaks interactive workflows
RRTConnect (Bidirectional)	Fast	Good: Smooth, feasible trajectories	Moderate (Connection-based)	Probabilistic completeness (faster)	Selected: Fastest convergence; enables interactive refinement

RRTConnect, with its bidirectional growth strategy, offers a compelling balance between planning speed and trajectory quality. By launching search trees from both goal and start configurations simultaneously, it accelerates convergence and reduces excessive waypoints characteristic of unidirectional RRT. This approach has proven particularly effective in high-dimensional configuration spaces encountered in 7-DOF manipulators while maintaining computational feasibility for interactive refinement and re-planning scenarios.

Accuracy-oriented analyses of mobile manipulators further stress that geometric and kinematic fidelity in the underlying models is a prerequisite for any trajectory optimization to translate into physical accuracy [24]. Despite this progress, there remains limited experimental evidence on how specific planner choices, such as RRTConnect within a modular motion-planning framework, perform when embedded in a full DT loop that includes realistic communication delays and physics-based simulation.

In parallel, industrial robotic setup literature emphasizes the importance of accurate tracking and analysis infrastructures, sensors, data pipelines, and analytics for understanding robot behavior under realistic operating conditions [25]. At the manipulator level, several studies have introduced low-cost DT-driven methods for positioning error compensation, in which sensor data are fused with virtual models to estimate and correct joint and task-space deviations between simulation and reality [24,26]. These works demonstrate that well-designed DTs can close the loop on geometric and kinematic errors and achieve sub-millimeter tolerances in demanding applications. Yet they typically assume a pre-existing simulation model and do not detail how modeling and simulation decisions (mesh resolution, joint definitions, physics parameters) influence the measured error budget, nor do they analyze how communication and computation latencies interact with motion-planning performance in an integrated workflow.

Another persistent challenge in robotic DTs is closing the "sim-to-real" gap in the presence of sensor noise, actuator delays, and model uncertainty. Studies on bi-manual block assembly via RL report substantial drops in task success rates when policies trained in simulation are deployed on physical hardware, attributing the degradation to imperfect sensing and latency-induced mismatches between planned and executed motions [16]. Application-oriented work in the aerospace

 doi:10.20944/preprints202603.1064.v1

maintenance uses a DT of a 6-DOF robotic grinder and fan blades to support surface characterization and automated reconditioning, demonstrating the value of DTs for complex, safety-critical operations [27]. However, these and related contributions typically characterize performance at the level of task success or process quality; detailed timing analysis of the communication channels and controller stack is rarely provided. Recent DT-driven compensation strategies for robotic arms have begun to quantify and bound position errors introduced by transmission delays and mechanical wear [24,26], but largely focus on static or quasi-static tasks and do not explore how distributed architecture involving game engines, ROS middleware, and collaborative manipulators behaves under sustained, multi-trial operations.

Against this backdrop, there is a need for an openly documented, experimentally validated DT framework that (i) starts from mechanical design data, (ii) propagates geometric and kinematic fidelity through robot description formats and game-engine physics representations, (iii) integrates an industrially relevant motion-planning stack, and (iv) quantitatively characterizes synchronization accuracy, latency, and trajectory quality across the entire virtual–physical loop for a robotic manipulator. Existing Unity–ROS DT case studies demonstrate the feasibility of such integrations, while DT-based process applications show the benefits of simulation-validated trajectories, but none, to the best of the authors' knowledge, provide a comprehensive, metric-driven analysis of a full system workflow for a 7-DOF robotic arm.

The remainder of the paper elaborates on the system architecture, modeling and planning methods, and experimental protocol before presenting a detailed analysis of planning efficiency, synchronization accuracy, and end-to-end performance for the proposed DT framework.

2. Framework

2.1. System Architecture Overview

An integrated DT system systematically addresses the three identified challenges through targeted technical approaches incorporating validated CAD-to-URDF (Unified Robot Description Format) conversion workflow with kinematic verification, optimized Unity-ROS bidirectional communication with latency characterization, and advanced motion planning algorithm for path optimization. The proposed system consists of six interconnected stages operated in a coordinated

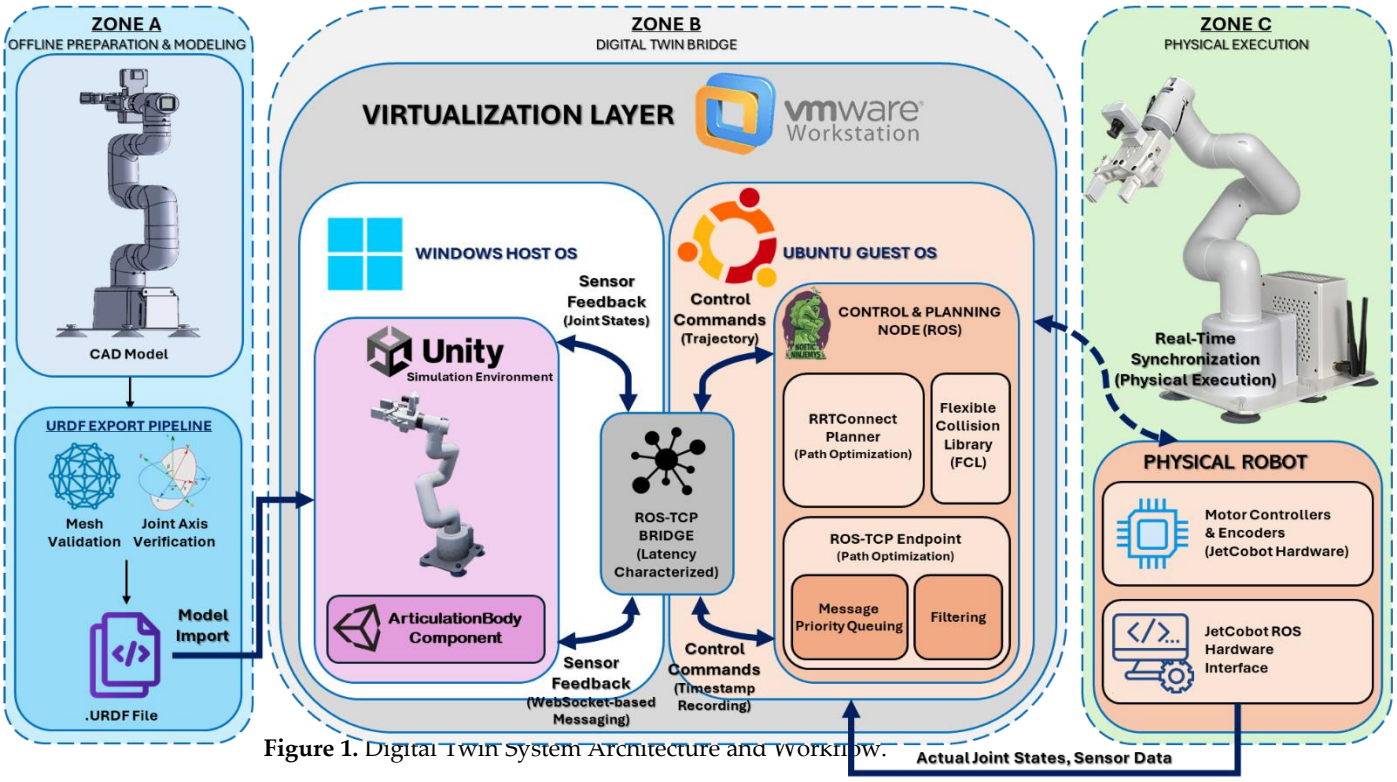

Figure 1. Digital Twin System Architecture and Workflow.

 doi:10.20944/preprints202603.1064.v1

workflow. Figure 1 illustrates the complete DT system from CAD modeling to physical robot execution. The first stage involves CAD modeling of a 7-DOF robotic arm with six revolute joints and one fixed gripper mount, incorporating kinematic characterization via Denavit-Hartenberg (DH) parameters. The second stage executes URDF export and validation using the URDF Exporter plugin with iterative mesh validation and joint-axis verification to ensure cross-platform compatibility. The third stage implements Unity physics simulation through the Unity Robotics Hub URDF Importer, reconstructing the kinematic hierarchy as ArticulationBody physics components with realistic mass properties, inertia tensors, stiffness (10,000 Nm/rad), and damping (100 Nms/rad) parameters. The fourth stage establishes ROS integration via ROS-TCP Connector for bidirectional joint command publishing and state subscription, enabling real-time communication between simulation and physical systems. The fifth stage implements motion planning through integration of the RRTConnect algorithm to generate collision-aware joint-space trajectories between the stage start and goal configurations. The sixth stage enables physical robot execution with real-time feedback from JetCobot hardware, completing the DT loop.

2.2. Robotic Platform and CAD–URDF Modeling

The physical platform used in this work is a 7-DOF JetCobot arm comprising six actuated revolute joints and a fixed end-effector mount. The mechanical structure is modelled as an assembly of individual links, joint housings, and the parallel gripper, with concentric and coincident mates to define joint axes and link frames. A global base coordinate system is assigned on the robot pedestal, and a DH kinematic diagram is constructed to ensure a consistent definition of link frames across the CAD model, URDF description, and ROS motion-planning stack.

In the DH convention, each revolute joint i is described by four parameters $(\alpha_i, a_i, d_i, \theta_i)$: the twist angle α_i between axes z_{i-1} and z_i, the link length a_i along x_i, the offset d_i along z_{i-1}, and the joint angle θ_i about z_{i-1}. Figure 2 illustrates these four parameters for a generic revolute joint, showing the geometric transformation from frame $\{i-1\}$ to frame $\{i\}$.

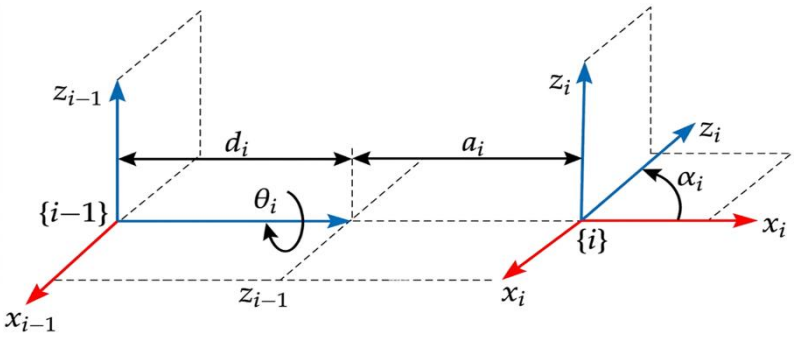

Figure 2. DH parameters for joint *i* of the JetCobot.

For the JetCobot manipulator, coordinate frames are attached to each link according to this convention, as shown in Figure 3, and the resulting DH parameters are summarized in Table 2. The non-zero geometric parameters include the vertical offsets $d_1 = 131.56$, $d_4 = 64.62$, $d_5 = 73.18$, and $d_6 = 48.6$, as well as the link lengths $a_3 = -110.4$ and $a_4 = -96$, which encode the horizontal separations between frames along the chosen x-axes.

Table 2. Denavit–Hartenberg parameters of the JetCobot manipulator.

Joint i	α_i	a_i (mm)	d_i (mm)	θ_i	Offset
1	0	0	131.56	θ1	0
2	π/2	0	0	θ2	-π/2
3	0	-110.4	0	θ3	0

 doi:10.20944/preprints202603.1064.v1

4	0	-96	64.62	θ_4	-π/2
5	π/2	0	73.18	θ_5	π/2
6	-π/2	0	48.6	θ_6	0

For each joint i, the homogeneous transformation matrix from frame $i-1$ to frame i is obtained from the corresponding row of Table 2 using the following standard DH form:

$$^{i-1}T_i(\theta_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, i = 1, \dots, 6. \quad (1)$$

In Equation (1), α_i, a_i, and d_i are the constant DH parameters listed in Table 2, while θ_i is the joint variable; the "offset" column specifies a constant angle added to θ_i so that the zero of the DH model coincides with the mechanical home position of each joint. This matrix maps coordinates expressed in frame i to the upstream frame $i-1$, and stacking the six transformations yields the complete arm kinematics.

The forward kinematics of the JetCobot are then obtained as the ordered product of the six link transformations as follows,

$$^0T_6(\boldsymbol{\theta}) = {}^0T_1(\theta_1)\ {}^1T_2(\theta_2)\ {}^2T_3(\theta_3)\ {}^3T_4(\theta_4)\ {}^4T_5(\theta_5)\ {}^5T_6(\theta_6), \quad (2)$$

The joint vector $\boldsymbol{\theta} = [\theta_1, \dots, \theta_6]^\top$ is mapped to the end-effector pose $^0T_6(\boldsymbol{\theta})$ in the base frame. Equation (2) serves as the analytical reference model for validating that the URDF and Unity implementations reproduce the same kinematic behaviour as the DH description over the workspace of the robot.

URDF generation is performed using the URDF Exporter plugin (v1.4). During export, the base link (base_link) is defined as the root of the kinematic tree, and each revolute joint is assigned its axis orientation, motion limits, and associated mesh file in DAE format. Visual meshes and simplified collision meshes are exported separately to balance geometric fidelity with real-time collision checking performance, and link inertial properties (mass and inertia tensors) are imported from the CAD model to support physics-consistent simulation in Unity.

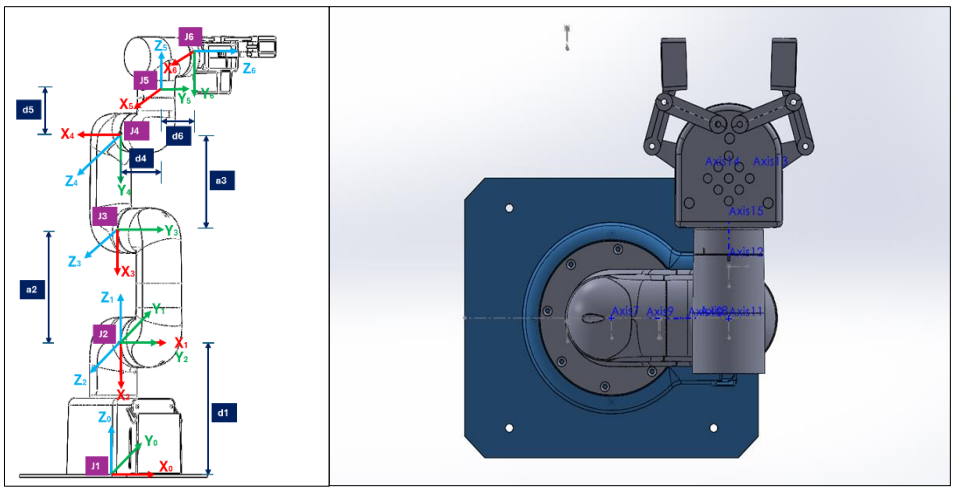

Figure 3. Kinematic diagram and assembly model of the JetCobot with annotated joint axes and DH coordinate frames.

A three-step validation procedure is used to ensure URDF correctness: (i) verification of joint names, types, axes, and limits against the CAD assembly, (ii) verification of mesh references, scaling factors, and file paths, and (iii) visualization in RViz to confirm the consistency of link frames and end-effector poses with analytical forward-kinematics predictions.

 doi:10.20944/preprints202603.1064.v1

2.3. Semantic Robot Description and Planning Groups

The Semantic Robot Description Format (SRDF) is generated with the MoveIt Setup Assistant to support collision-aware motion planning. The primary planning group comprises the six revolute joints of the JetCobot arm, with the fixed end-effector link (6_link) set as the planning end-effector frame. Joint velocity and acceleration limits in the SRDF are conservatively derived from manufacturer specifications to guarantee that planned trajectories are dynamically feasible on the physical robot.

Self-collision checking is configured via the Allowed Collision Matrix (ACM). Adjacent link pairs connected by a joint are permitted to be in collision to account for their physical connectivity, whereas non-adjacent links are marked as collision-forbidden to prevent self-intersections. Careful consideration is given to defining adjacency, particularly for the elbow joint of the 7-DOF arm, to avoid over-constraining the valid workspace. This configuration ensures that the Flexible Collision Library (FCL) enforces realistic self-collision constraints without over-constraining physically valid postures.

2.4. Unity-Based Physics Simulation Environment

The validated URDF model is imported into Unity 2022 LTS through the Unity Robotics Hub URDF Importer. The import process instantiated a URDF Robot prefab containing one ArticulationBody per link, with each component inheriting the joint type, axis definition, and motion limits specified in the URDF file. Visual meshes exported in DAE format are associated with the corresponding ArticulationBodies for high-fidelity rendering, while convex mesh colliders were generated for every link to support robust contact detection during simulation.

Figure 4 illustrates the resulting kinematic hierarchy as displayed in the Unity Inspector, showing the parent–child link tree together with the physics properties assigned to each ArticulationBody. The left panel depicts the hierarchical tree structure, wherein the root node anchors the rigidly mounted base link to the world frame, and successive child nodes represent articulated links connected via revolute joints down to the end-effector. The right panel exposes the ArticulationBody component Inspector for the JetCobot, enumerating the physics parameters that govern reduced-coordinate dynamics: joint stiffness and damping for position control, force limits for actuator saturation, mass, and the principal inertia tensor imported from the CAD assembly, all essential for numerical stability in Unity's articulated-body solver.

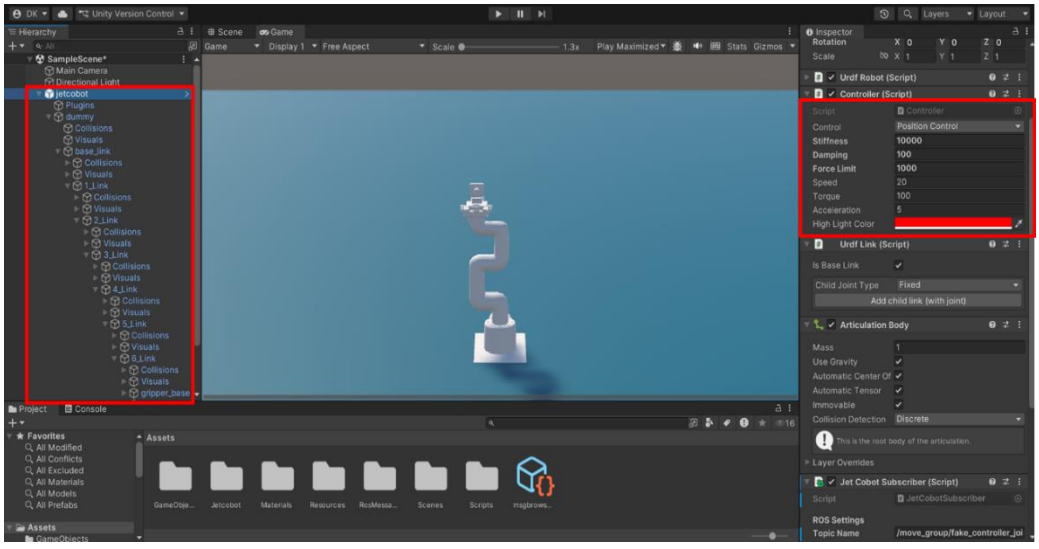

Figure 4. JetCobot Kinematic Hierarchy with ArticulationBody Physics Properties.

Each revolute joint is configured with a stiffness of 10,000 Nm/rad and a damping coefficient of 100 Nm·s/rad. These values are empirically tuned to achieve critically damped convergence without

 doi:10.20944/preprints202603.1064.v1

oscillation across the full joint range of the JetCobot arm. These gains ensure that commanded joint-angle targets are reached smoothly within a single simulation step, while remaining high enough to resist gravitational torques on the extended links without steady-state droop. Mass and inertia-tensor values for each link were imported directly from the CAD model via the URDF, preserving the physically measured mass distribution and ensuring that gravity loading, centrifugal effects, and Coriolis coupling are faithfully reproduced during dynamic trajectory execution.

The simulation scene is composed of approximately the physical laboratory workspace of the JetCobot. A planar ground surface is placed below the robot base, and a set of movable test objects is positioned within the manipulator's reachable envelope to support reaching and repositioning tasks during validation trials. Default gravity is applied along the scene's vertical axis. The contact-material properties static and dynamic friction coefficients as well as the coefficient of restitution are adjusted to prevent unrealistic sliding or jitter at object–surface and link–object interfaces. Camera views are configured at both operator-level and overhead perspectives to facilitate qualitative monitoring of simulated motions.

2.5. ROS Middleware and Unity–ROS Communication

The DT framework relies on a cross-platform communication layer that bridges the Unity simulation environment, running on a Windows host, with the ROS Noetic middleware stack, running on an Ubuntu 20.04 guest OS in the virtualization layer. The two hosts are connected over a shared local-area network and exchange messages through the ROS–TCP Connector, an open-source Unity package that implements WebSocket-based bidirectional communication with the ROS master.

Three communication channels are established within this architecture. A joint command publisher transmits planned joint trajectories from Unity to the ROS controller as standard ROS trajectory messages, enabling the physical JetCobot to execute motions validated in simulation. A joint state subscriber receives actual joint-angle readings from the physical robot via a ROS topic and feeds them back into Unity, where the ArticulationBody hierarchy is updated to reflect the real arm posture in the DT. A set of ROS service clients allows Unity to invoke MoveIt planning services directly, so that trajectory computation and collision checking are performed on the Linux side and the resulting motion plans are returned to Unity for visualization and subsequent hardware dispatch.

To ensure command responsiveness, the communication channels are configured with differentiated priority levels: trajectory command messages are assigned higher priority than low-frequency monitoring data such as diagnostic or sensor feedback. Asynchronous publishing is employed on the Unity side to prevent blocking the main rendering and physics loop, and a minimum position-change threshold is applied to outgoing joint-state messages so that only meaningful updates are transmitted, thereby reducing unnecessary network traffic. Real-time synchronization between the virtual and physical systems is achieved by timestamping all trajectory and feedback messages at the point of emission and compensating for measured communication latencies when computing joint-angle errors and execution timelines.

2.6. Motion Planning and Collision-Aware Path Generation

Motion planning within the proposed framework is implemented using MoveIt with the Open Motion Planning Library (OMPL) as its computational back end, and the RRTConnect algorithm as the primary configuration-space planner. RRTConnect is selected after evaluating three representative planners from the RRT family standard RRT, asymptotically optimal RRT*, and bidirectional RRTConnect along four design-relevant axes: planning time, path quality, computational overhead, and convergence guarantees. As summarized in Table 1, unidirectional RRT generates paths rapidly but produces jerky trajectories with excessive waypoints that require heavy post-processing, often negating the initial speed advantage. RRT* yields near-optimal, smooth trajectories; however, its iterative rewiring procedure introduces planning times on the order of tens of seconds, which are incompatible with the interactive workflow targeted by the present system. RRTConnect offers a balanced trade-off between these extremes: by growing two search trees

 doi:10.20944/preprints202603.1064.v1

simultaneously, one from the start configuration and one from the goal it exploits bidirectional expansion to converge substantially faster than RRT* while producing smoother, more direct paths than standard RRT.

Figure 5 illustrates the OMPL configuration within the MoveIt RViz interface, with RRTConnect designated as the default planner for the arm_group (top-right panel) and the gripper_group planner similarly configured with RRTConnect (bottom-right panel). The central-left panel displays the planner dropdown confirming activation of the RRTConnect algorithm from the Open Motion Planning Library, which uses bidirectional tree growth for efficient collision-free path generation. The central 3D viewport renders the complete JetCobot kinematic model with collision geometries (green wireframes) and workspace obstacles processed by the FCL collision checker. This configuration confirms that the trajectories analyzed in the result sections are generated using the same RRTConnect settings that will later be deployed on the physical manipulator.

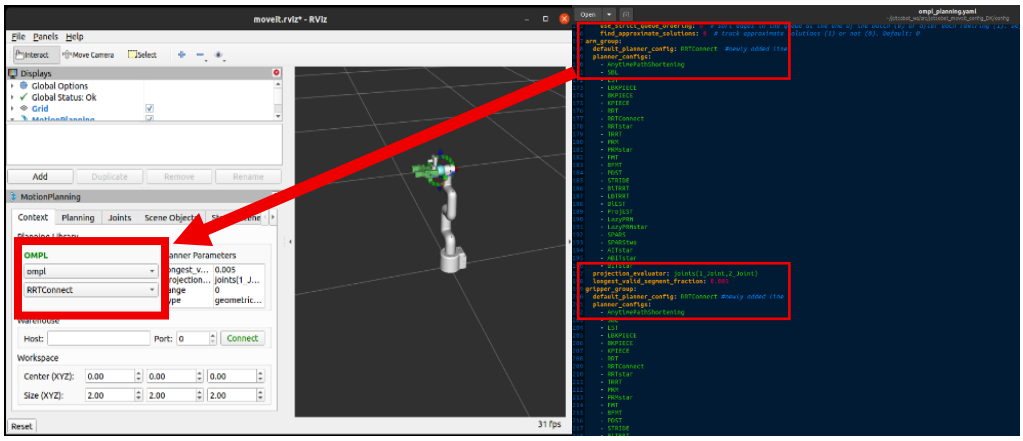

Figure 5. OMPL configuration with RRTConnect as the default algorithm and RViz visualization of the JetCobot planning scene.

The raw paths returned by RRTConnect are further refined through a three-stage post-processing pipeline integrated within MoveIt. First, a waypoint-reduction step eliminates redundant intermediate configurations while preserving collision-free feasibility across the simplified path. Second, cubic spline interpolation is applied to the reduced waypoint sequence to smooth discontinuities in joint velocities and accelerations, thereby minimizing jerk and ensuring that the resulting trajectory is suitable for direct execution on the physical servos. Third, a time-parameterization pass computes per-waypoint timestamps, joint velocities, and joint accelerations that respect the kinematic limits defined in the SRDF, guaranteeing that the trajectory remains dynamically feasible on the robot hardware.

Collision awareness throughout the planning process is provided by the FCL, which operates on simplified convex-hull collision meshes for each robot link and with the environmental objects in the scene. During tree expansion, FCL evaluates every sampled configuration and every interpolated segment between consecutive nodes for both self-collisions among non-adjacent links and environmental collisions with workspace geometry. The ACM, defined in the SRDF, prevents spurious collision flags for physically connected adjacent-link pairs while enforcing strict constraints between all non-adjacent links. This geometry-based collision-checking strategy ensures that only kinematically feasible and physically safe trajectories are accepted by the planner, without requiring manual modification of the collision scene for different target poses or workspace layouts.

Figure 6 illustrates a representative example of real-time collision avoidance using the FCL during motion planning. The visualization highlights collision meshes attached to each JetCobot link and to the fixed obstacle, together with the planned joint-space trajectory that steers the arm around the obstruction without violating any self-collision or environment-collision constraints. This figure

 doi:10.20944/preprints202603.1064.v1

illustrates how FCL's geometry-based checks filter infeasible samples during RRTConnect tree expansion, ensuring that only physically safe trajectories are returned to the DT and subsequently executed on the hardware.

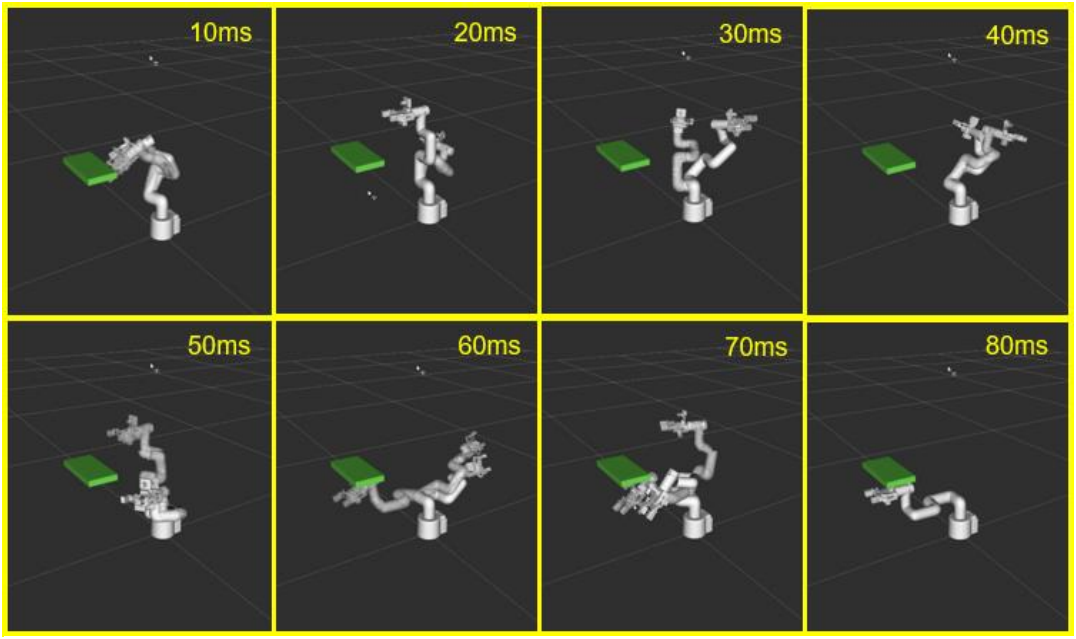

Figure 6. Real-time Collision Avoidance using Flexible Collision Library.

3. Results

3.1. Experimental Protocol and Performance Metrics

System performance has been evaluated across 12 sequentially interconnected manipulation trials, labelled A through L, spanning a diverse set of joint-space configurations and end-effector targets within the JetCobot workspace. Each trial consists of a reaching or repositioning task in which the final joint configuration of the preceding trial served as the initial configuration of the next, so the manipulator never returns to a zero-joint home pose between tasks. A fixed obstacle is present in the workspace throughout the entire sequence, and the FCL-based collision checker remains active during every RRTConnect planning call to ensure that all generated trajectories are free of both environmental and self-collisions. For each trial, the following quantities are recorded:

- Planning time T_{plan}: OMPL RRTConnect planning duration reported by MoveIt.
- Number of waypoints N_{wp}: Number of joint-space waypoints after post-processing.
- ROS execution time T_{ROS}: Nominal trajectory execution time on the ROS controller.
- Unity execution time T_{Unity}: Time required for the Unity ArticulationBody system to execute the same trajectory.
- Unity–ROS latency $\tau_{\text{U}\to\text{R}}$: Delay between Unity command issuance and ROS reception.
- JetCobot execution time T_{J}: Actual motion duration on the physical manipulator.
- JetCobot–ROS latency $\tau_{\text{J}\to\text{R}}$: Delay between robot joint-state updates and their arrival at ROS.
- Total end-to-end time T_{tot} : Sum of planning, communication, and execution components:

$$T_{\text{tot}} = T_{\text{plan}} + T_{\text{ROS}} + T_{\text{Unity}} + \tau_{\text{U}\to\text{R}} + \tau_{\text{J}\to\text{R}} \quad (3)$$

In Equation (3), the five terms represent planning computation, ROS-side execution, Unity-side execution, and the two communication latencies that bridge the distributed system, respectively. Because the ROS node and the Unity application ran on separate machines connected over a local network, the system clocks of the two hosts were synchronized prior to the experiments using the

 doi:10.20944/preprints202603.1064.v1

Network Time Protocol (NTP) to ensure that cross-platform timestamp comparisons remained valid at the millisecond level. Each trial was executed once and then repeated to confirm measurement consistency; the values reported in Table 3 correspond to single-run metrics for each of the 12 distinct tasks.

Table 3. Performance Metrics across 12 Manipulation Trials.

Jetcobot posture	ROS path planning time (s)	No. of waypoints	ROS execution time (s)	Unity execution time (s)	Latency [Unity - ROS] (s)	Jetcobot execution time (s)	Latency [Jetcobot - ROS] (s)	Total Time (s)
A	0.060	28	1.1002	1.1609	0.0607	1.2000	0.0998	**3.4611**
B	0.048	39	1.5999	1.6914	0.0915	1.7010	0.1011	**4.9923**
C	0.058	37	1.2723	1.3001	0.0278	1.3983	0.1260	**3.9707**
D	0.047	33	1.1884	1.1992	0.0108	1.3001	0.1117	**3.6877**
E	0.053	39	1.3003	1.3917	0.0914	1.4001	0.0998	**4.0921**
F	0.086	50	1.6970	1.7035	0.0065	1.7980	0.1010	**5.1985**
G	0.055	29	1.1999	1.2542	0.0543	1.2990	0.0991	**3.7531**
H	0.056	44	1.4961	1.4995	0.0034	1.5990	0.1029	**4.5946**
I	0.060	41	1.3804	1.4002	0.0198	1.5005	0.1201	**4.2811**
J	0.118	63	2.1000	2.1005	0.0005	2.1996	0.0996	**6.4001**
K	0.061	23	1.2876	1.3001	0.0125	1.4005	0.1129	**3.9882**
L	0.071	36	1.2623	1.3009	0.0386	1.3995	0.1372	**3.9627**
Mean	**0.064**	**38.50**	**1.4070**	**1.4419**	**0.0348**	**1.5163**	**0.1093**	4.3652
Std. Dev	**0.0199**	**10.66**	**0.2796**	**0.2706**	**0.0327**	**0.2757**	**0.0126**	0.8243

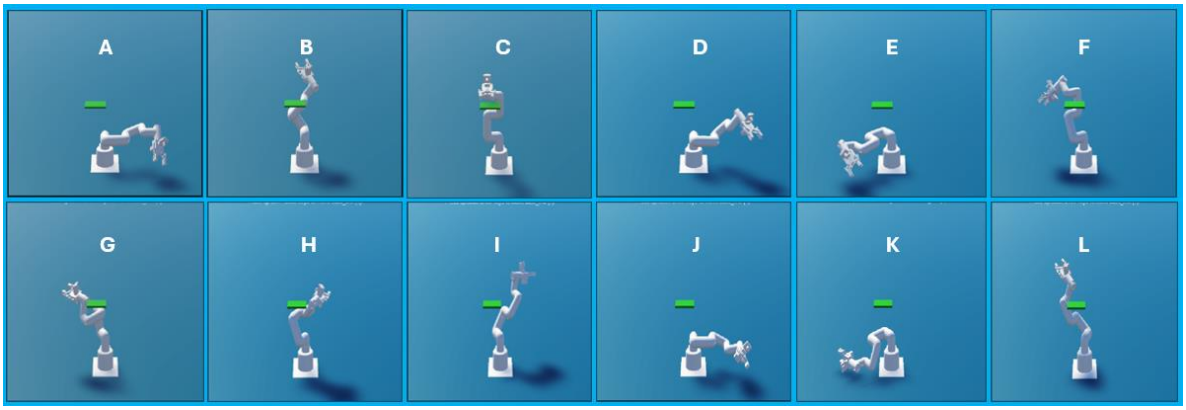

Figure 7. JetCobot Manipulator Postures across 12 Validation Trials (A-L).

In addition to timing metrics, positional synchronization accuracy was quantified by comparing the Unity-simulated joint angles with the corresponding joint-angle feedback from the physical JetCobot, after compensating for measured communication latencies. For each joint j, the absolute angular error at time t is defined as:

$$e_j(t) = \left| q_j^{\text{Unity}}(t - \Delta t) - q_j^{\text{JetCobot}}(t) \right| \quad (4)$$

where Δt denotes the estimated round-trip latency between Unity and the robot controller. In Equation (4), the Unity joint angle is evaluated at time $t - \Delta t$ to account for the fact that the command issued by Unity at that instant is the one physically executed by JetCobot at time t. The maximum value of $e_j(t)$ Overall joints and all-time steps within a trial are adopted as a conservative indicator of kinematic fidelity between the DT and the physical system.

3.2. Motion Planning Performance

12 validation trials produce a wide range of joint configurations and path complexities, as illustrated by the manipulator postures in Figure 7. These postures span different regions of the workspace, including near-singular configurations and extended-reach positions, thereby providing a diverse test set for the planner.

Planning times T_{plan} ranged from 0.047 s (trial D) to 0.118 s (trial J), with a mean of 0.064 ± 0.020s, placing all values on the order of 10^{-2}s. These results are well below the sub-second real-time thresholds typically required for robotic systems and align with the design goal of enabling

 doi:10.20944/preprints202603.1064.v1

interactive offline programming. The number of post-processed waypoints N_{wp} ranged from 23 to 63 (mean 38.5 ± 10.7), reflecting the expected increase in path complexity for longer or more constrained motions.

Figure 8 reveals that planning time does not scale linearly with waypoint count over the tested range, indicating that RRTConnect maintains stable computational performance across moderate variations in path complexity. The combination of bidirectional tree growth and post-processing (waypoint reduction, cubic spline smoothing, and time parameterization) yields compact, smooth trajectories that are suitable for direct execution on the robotic arm without additional filtering.

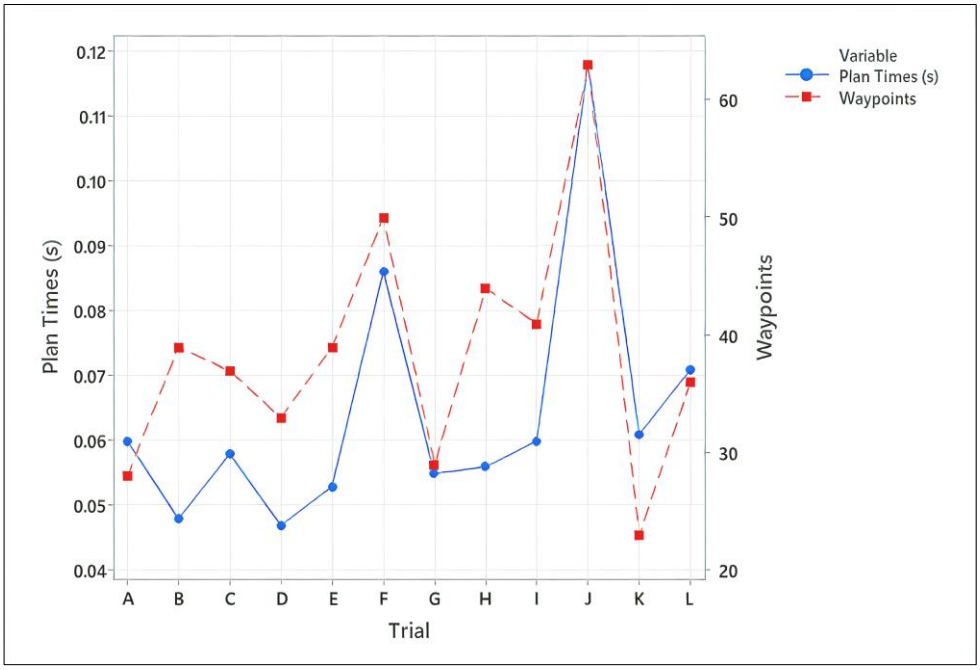

Figure 8. ROS Planning Time vs Path Quality across 12 Manipulation Trials.

3.3. Execution Time Decomposition

The total end-to-end time T_{tot} for each trial is decomposed into its five constituent terms (Equation (3)), as shown in Figure 9. Across the 12 trials, T_{tot} ranged from 3.46 s (trial A) to 6.40 s (trial J), with a mean of 4.37 ± 0.82s.

 doi:10.20944/preprints202603.1064.v1

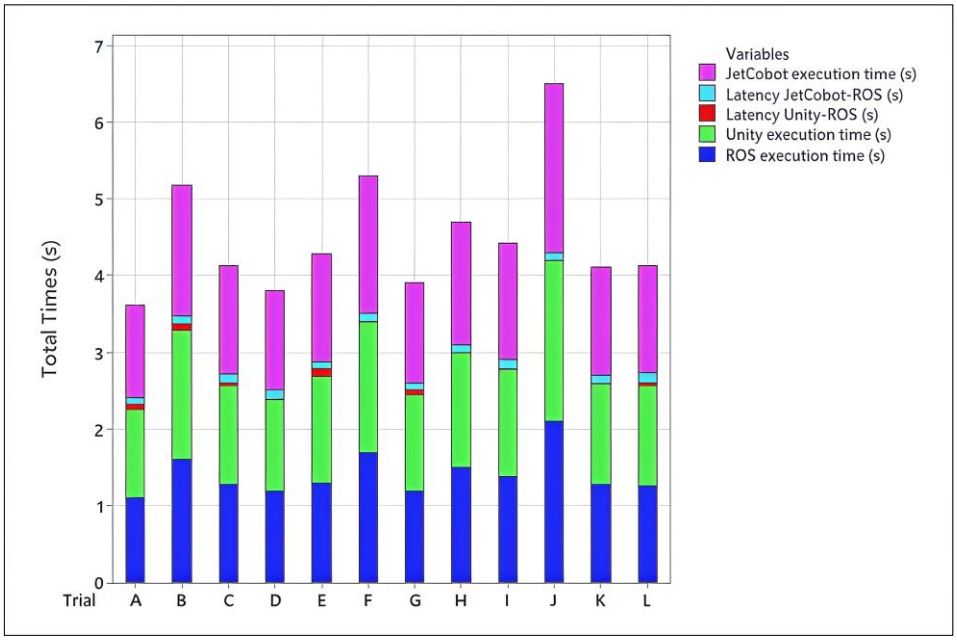

Figure 9. Complete Execution Time Breakdown across 12 Manipulation Trials.

On average, JetCobot hardware execution time T_J is accounted for 34.8% of T_{tot}, ROS-side trajectory execution T_{ROS} for 32.2%, and Unity execution T_{Unity} for 33.0%. The combined communication latencies $\tau_{U\to R} + \tau_{J\to R}$ contribute only 3.4% of the total time (mean 0.14s per trial). This decomposition demonstrates that actuator dynamics and mechanical movement, rather than network or middleware overhead, constitute the dominant contributors to task cycle time in the current distributed configuration.

3.4. Communication Latency Characteristics

Communication latency between Unity and ROS exhibits markedly different statistical behavior from that between the JetCobot and ROS. Figure 10 presents a box-plot comparison of $\tau_{U\to R}$ and $\tau_{J\to R}$ across the 12 trials.

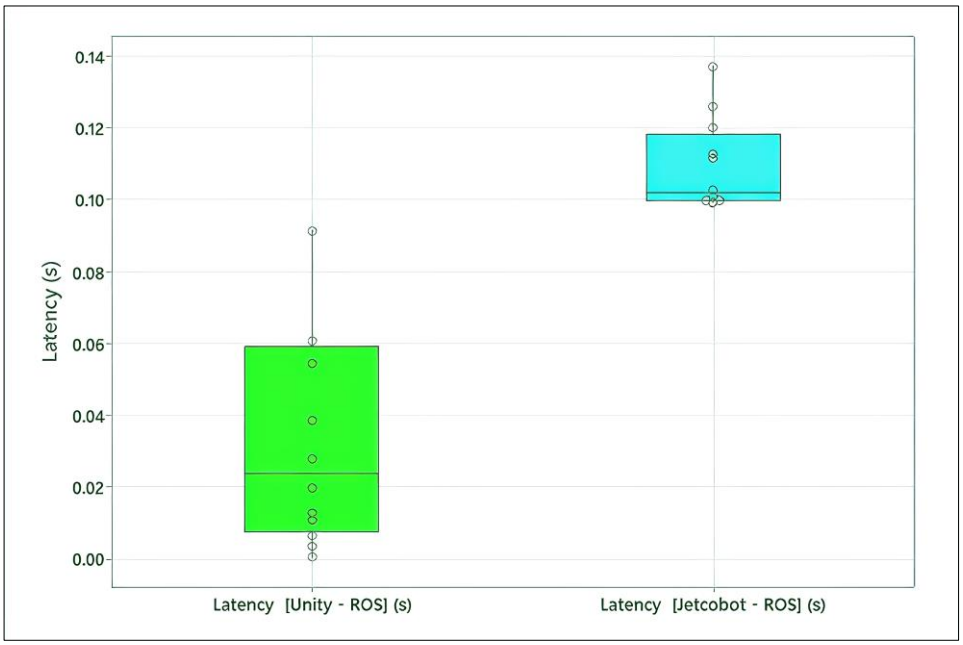

Figure 10. Communication Latency Distribution for Unity–ROS and JetCobot–ROS channels.

 doi:10.20944/preprints202603.1064.v1

Unity–ROS latency $\boldsymbol{\tau}_{U\to R}$ had a mean of 34.8ms and a standard deviation of 32.7ms, ranging from 0.5ms to 91.5ms. This high variability reflects fluctuating network conditions on the shared local area network. In contrast, JetCobot–ROS latency $\boldsymbol{\tau}_{J\to R}$ has a mean of 109.3ms and a standard deviation of only 12.6ms, corresponding to a tightly clustered distribution (99.1–137.2ms). The substantially lower variance of the hardware feedback channel suggests that the robot's internal control loop and sensor update rate are the primary determinants of feedback timing and are comparatively insensitive to network jitter. From a DT synchronization perspective, the higher predictability of $\boldsymbol{\tau}_{J\to R}$ simplifies latency compensation when aligning Unity and JetCobot trajectories for error analysis, whereas the variability in $\boldsymbol{\tau}_{U\to R}$ must be accounted for in safety margins if the framework is extended to hard real-time closed-loop control scenarios.

3.5. End-to-End Real-Time Performance

Figure 11 summarizes the total end-to-end execution times across all 12 trials. The observed range of 3.46–6.40s is compatible with offline programming and validation workflows in which task cycle times typically span 10–20s.

Trial J, which involves the most complex motion (63 waypoints through a kinematically demanding region of the workspace), requires the longest execution time, while trials with fewer waypoints and shorter joint-space traversals are completed in under 4s. The Pearson correlation coefficient between N_{wp} and T_{tot} is 0.90 ($p < 0.001$), confirming a strong positive relationship between trajectory complexity and total cycle time. The consistent end-to-end behaviour across diverse tasks indicates that the proposed framework can reliably support iterative trajectory validation, operator-in-the-loop refinement, and what-if analyses without introducing unacceptable delays between planning, simulation, and execution.

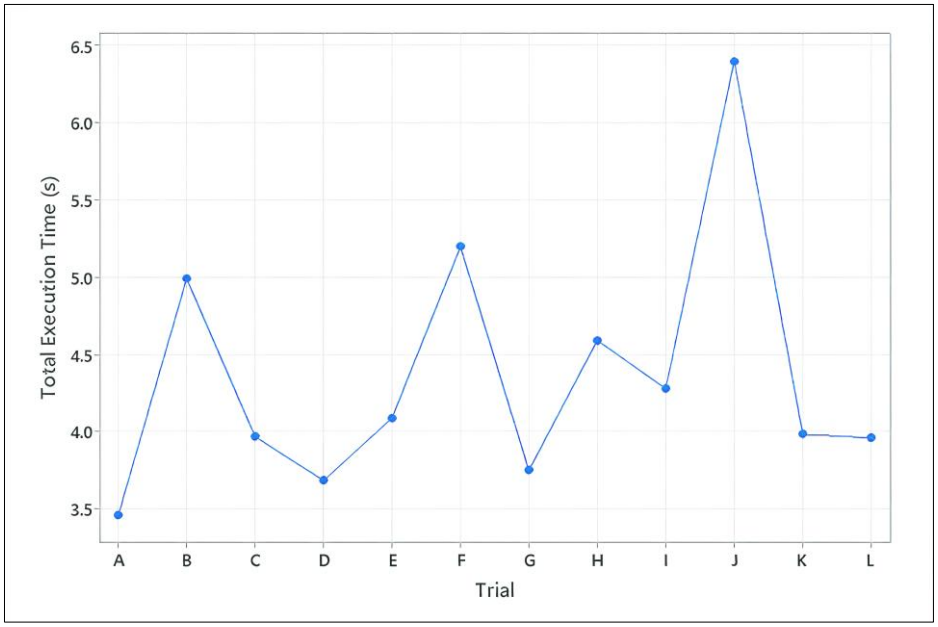

Figure 11. End-to-End Execution Time Performance across 12 Manipulation Trials.

4. Discussion

The experimental results presented in Section 3 validate the proposed CAD–URDF–Unity–ROS DT framework along three axes: model fidelity, motion planning performance, and synchronization accuracy. This section interprets the findings in the context of related work, examines the generalizability of the approach, and identifies the principal limitations of the current study.

The sub-millimeter translation agreement and sub-degree rotation agreement observed between the DH-based analytical model and the Unity forward kinematics confirm that the three-stage pipeline SolidWorks assembly, URDF export via the URDF Exporter plugin, and Unity ArticulationBody import preserves geometric and kinematic fidelity at a level sufficient for trajectory

 doi:10.20944/preprints202603.1064.v1

validation. This result is consistent with the findings of Wu et al. [24], who reported that DT-driven positioning error compensation can achieve sub-millimeter accuracy when the underlying kinematic model is carefully calibrated, and extends their single-platform analysis to a cross-engine (Unity + ROS) workflow.

The RRTConnect planning times recorded in this study (mean 0.064s) are substantially lower than the 0.5–2.0s range typically reported for RRTConnect on 6–7DOF arms in general-purpose benchmarks, likely because the tested tasks involved moderate obstacle density and did not require narrow-passage navigation. Nevertheless, the consistently low planning times across all 12 trials, including near-singular configurations, demonstrate that the combination of bidirectional tree growth and FCL collision checking remains computationally efficient in the evaluated workspace. The post-processing pipeline further reduced waypoint counts to levels suitable for direct servo execution, eliminating the need for an additional trajectory-filtering layer on the robot controller.

Singh et al. [19] characterized Unity–ROS communication for an ABB IRB 1200 industrial arm in a smart manufacturing cell and reported round-trip latencies on the order of 50–150ms. The latency values observed in the present study fall within the same order of magnitude: the Unity–ROS channel averaged 34.8ms and the JetCobot–ROS channel averaged 109.3ms, yielding a combined mean of approximately 144ms per trial. The key additional insight from the present work is the decomposition into two distinct channels with qualitatively different statistical profiles, a high-variance software channel and a low-variance hardware channel, which has practical implications for latency-compensation strategies in real-time DT applications.

The finding that communication latencies account for only 3.4% of total cycle time, while hardware execution dominates, has a generalizable implication for DT system design: for manipulators operating at moderate speeds on local-area networks, investing in faster actuators or optimized trajectory time-parameterization will yield greater cycle-time reductions than upgrading the communication infrastructure. This conclusion is expected to hold for other robotic arms of similar payload and speed class (e.g. UR3e, Dobot CR series), although verification on those platforms remains future work.

Several limitations should be acknowledged. First, the evaluation was conducted on a single robotic platform (JetCobot) with a fixed workspace layout and a single static obstacle; more complex environments with dynamic obstacles and multiple robots would impose additional demands on the planner and the communication stack. Second, the 12-trial sample size, while diverse in joint-space coverage, is insufficient for rigorous statistical generalization of latency distributions; future work should include larger trial sets and repeated runs to establish confidence intervals. Third, all trials involved reaching and repositioning tasks; contact-rich manipulation (e.g. assembly, insertion) requires force/torque feedback integration and more sophisticated collision-resolution policies than the binary FCL checks employed here. Finally, the stiffness and damping parameters of the Unity ArticulationBody joints were tuned empirically; a systematic sensitivity analysis relating these parameters to simulation accuracy would strengthen the claims of fidelity.

5. Conclusions

This research presents and validates a DT framework for robot path planning and real-time execution using a Unity–ROS integration, applied to a 7-DOF JetCobot collaborative manipulator. Starting from SolidWorks CAD models, the proposed pipeline propagates geometric and kinematic fidelity through URDF and SRDF descriptions into a Unity-based physics simulation and a ROS motion-planning stack. The integration of the RRTConnect planner with FCL collision checking enables the generation of fast, high-quality, collision-aware trajectories suitable for robotic applications in manufacturing contexts.

Quantitative evaluation across 12 diverse manipulation trials confirmed that the framework delivers planning performance on the order of milliseconds (ms), with RRTConnect achieving motion planning times between 0.047s and 0.118s (mean 0.064 ± 0.020s). The post-processed trajectories exhibited between 23 and 63 waypoints with smooth profiles executable directly on the JetCobot

 doi:10.20944/preprints202603.1064.v1

without additional filtering. A detailed decomposition of the end-to-end execution time showed that communication latencies contributed only 3.4% of the overall cycle time, while physical execution of motions on the hardware consistently represents the dominant component. The synchronization analysis further demonstrated that virtual–physical alignment can be maintained within approximately ±2.0° of joint-angle agreement across all tested configurations, even in the presence of measured communication delays.

Taken together, these results demonstrate that modern game-engine technology, when coupled with open-source robotics middleware, can deliver DT systems whose fidelity and performance are competitive with proprietary industrial solutions. The explicitly documented modelling assumptions, parameter settings, and timing behavior offer a reproducible reference architecture that researchers and practitioners can adopt and adapt to alternative robotic platforms. Future work will extend the framework to incorporate dynamic obstacle environments, contact-rich manipulation tasks, and extended-reality operator interfaces for immersive human-in-the-loop DT applications in robotic manufacturing.

Author Contributions: Conceptualization, Q.P. and D.K.; methodology, D.K.; software, D.K.; validation, Q.P. and D.K.; formal analysis, D.K.; investigation, D.K.; resources, Q.P.; data curation, Q.P. and D.K.; writing—original draft preparation, D.K.; writing—review and editing, Q.P.; visualization, D.K.; supervision, Q.P.; project administration, Q.P.; funding acquisition, Q.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DT	Digital Twin
DOF	Degree of Freedom
ROS	Robot Operating System
FCL	Flexible Collision Library
RL	Reinforcement-Learning
RRT	Rapidly Exploring Random Tree
URDF	Unified Robot Description Format
DH	Denavit–Hartenberg
SRDF	Semantic Robot Description Format
ACM	Allowed Collision Matrix
OMPL	Open Motion Planning Library
NTP	Network Time Protocol

References

1. J. Arents and M. Greitans, “Smart Industrial Robot Control Trends, Challenges and Opportunities within Manufacturing,” *Appl. Sci.*, vol. 12, no. 2, p. 937, Jan. 2022.
2. B. Wang et al., “Human Digital Twin in the context of Industry 5.0,” Robot. Comput.-Integr. Manuf., vol. 85, p. 102626, Feb. 2024.
3. J. Leng, D. Wang, W. Shen, X. Li, Q. Liu, and X. Chen, “Digital twins-based smart manufacturing system design in Industry 4.0: A review,” *J. Manuf. Syst.*, vol. 60, pp. 119–137, Jul. 2021.
4. S. Liu, P. Zheng, and J. Bao, “Digital Twin-based manufacturing system: a survey based on a novel reference model,” *J. Intell. Manuf.*, vol. 35, no. 6, pp. 2517–2546, Aug. 2024.

 doi:10.20944/preprints202603.1064.v1

5. D. A. Guerra-Zubiaga, M. Aksu, G. Richards, and V. Kuts, "Integrating Digital Twin Software Solutions with Collaborative Industrial Systems: A Comprehensive Review for Operational Efficiency," *Appl. Sci.*, vol. 15, no. 13, p. 7049, Jun. 2025.
6. Q. Qin *et al.*, "Robot digital twin systems in manufacturing: Technologies, applications, trends and challenges," *Robot. Comput.-Integr. Manuf.*, vol. 97, p. 103103, Feb. 2026.
7. X. Wang, Y. Hua, J. Gao, Z. Lin, and R. Yu, "Digital Twin Implementation of Autonomous Planning Arc Welding Robot System," *Complex Syst. Model. Simul.*, vol. 3, no. 3, pp. 236–251, Sep. 2023.
8. A. Mazumder *et al.*, "Towards next generation digital twin in robotics: Trends, scopes, challenges, and future," *Heliyon*, vol. 9, no. 2, p. e13359, Feb. 2023.
9. W. Wenna, D. Weili, H. Changchun, Z. Heng, F. Haibing, and Y. Yao, "A digital twin for 3D path planning of large-span curved-arm gantry robot," *Robot. Comput.-Integr. Manuf.*, vol. 76, p. 102330, Aug. 2022.
10. Q. Zhang, R. Xiao, Z. Liu, J. Duan, and J. Qin, "Process Simulation and Optimization of Arc Welding Robot Workstation Based on Digital Twin," *Machines*, vol. 11, no. 1, p. 53, Jan. 2023.
11. Z. Zhu, Z. Lin, J. Huang, L. Zheng, and B. He, "A digital twin-based machining motion simulation and visualization monitoring system for milling robot," *Int. J. Adv. Manuf. Technol.*, vol. 127, no. 9–10, pp. 4387–4399, Aug. 2023.
12. Z. Zhu, W. Zhu, J. Huang, and B. He, "An intelligent monitoring system for robotic milling process based on transfer learning and digital twin," *J. Manuf. Syst.*, vol. 78, pp. 433–443, Feb. 2025.
13. X. Li, B. He, Y. Zhou, and G. Li, "Multisource Model-Driven Digital Twin System of Robotic Assembly," *IEEE Syst. J.*, vol. 15, no. 1, pp. 114–123, Mar. 2021.
14. J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A Review of Physics Simulators for Robotic Applications," *IEEE Access*, vol. 9, pp. 51416–51431, 2021.
15. T. Kim, M. Jang, and J. Kim, "A Survey on Simulation Environments for Reinforcement Learning," in 2021 18th International Conference on Ubiquitous Robots (UR), Gangneung, Korea (South): IEEE, Jul. 2021, pp. 63–67.
16. Y. Wang et al., "Digital twin-empowered robotic arm manipulation with reinforcement learning: A comprehensive survey," Robot. Comput.-Integr. Manuf., vol. 98, p. 103151, Apr. 2026.
17. G. D. Wijaya, W. Caesarendra, M. I. Petra, G. Królczyk, and A. Glowacz, "Comparative study of Gazebo and Unity 3D in performing a virtual pick and place of Universal Robot UR3 for assembly process in manufacturing," Simul. Model. Pract. Theory, vol. 132, p. 102895, Apr. 2024.
18. C. Wang, S. Liu, L. Zhao, and T. Luo, "Virtual Simulation of Fruit Picking Robot Based on Unity3D," *J. Phys. Conf. Ser.*, vol. 1631, no. 1, p. 012033, Sep. 2020.
19. 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, p. 5680, Aug. 2024.
20. S. Banik, S. C. Banik, and S. S. Mahmud, "Path Planning Approaches in Multi-robot System: A Review," *Eng. Rep.*, vol. 7, no. 1, p. e13035, Jan. 2025.
21. A. Hameed, A. Ordys, J. Możaryn, and A. Sibilska-Mroziewicz, "Control System Design and Methods for Collaborative Robots: Review," Appl. Sci., vol. 13, no. 1, p. 675, Jan. 2023.
22. J.G. Kang, D.W. Lim, Y.S. Choi, W.J. Jang, and J.W. Jung, "Improved RRT-Connect Algorithm Based on Triangular Inequality for Robot Path Planning," *Sensors*, vol. 21, no. 2, p. 333, Jan. 2021.
23. L. Wang, X. Yang, Z. Chen, and B. Wang, "Application of the Improved Rapidly Exploring Random Tree Algorithm to an Insect-like Mobile Robot in a Narrow Environment," Biomimetics, vol. 8, no. 4, Aug. 2023.
24. Z. Wu et al., "Digital Twin-Driven 3-D Position Information Mutuality and Positioning Error Compensation for Robotic Arm," IEEE Sens. J., vol. 23, no. 22, pp. 27508–27516, Nov. 2023.
25. M. Makulavičius, J. J. Petronienė, E. Šutinys, V. Bučinskas, and A. Dzedzickis, "Industrial Robotic Setups: Tools and Technologies for Tracking and Analysis in Industrial Processes," Appl. Sci., vol. 15, no. 18, p. 10249, Sep. 2025.
26. Z. Wu, S. Chen, J. Han, S. Zhang, J. Liang, and X. Yang, "A Low-Cost Digital Twin-Driven Positioning Error Compensation Method for Industrial Robotic Arm," IEEE Sens. J., vol. 22, no. 23, pp. 22885–22893, Dec. 2022.

 doi:10.20944/preprints202603.1064.v1

27. J. Oyekan, M. Farnsworth, W. Hutabarat, D. Miller, and A. Tiwari, "Applying a 6 DOF Robotic Arm and Digital Twin to Automate Fan-Blade Reconditioning for Aerospace Maintenance, Repair, and Overhaul," Sensors, vol. 20, no. 16, p. 4637, Aug. 2020.