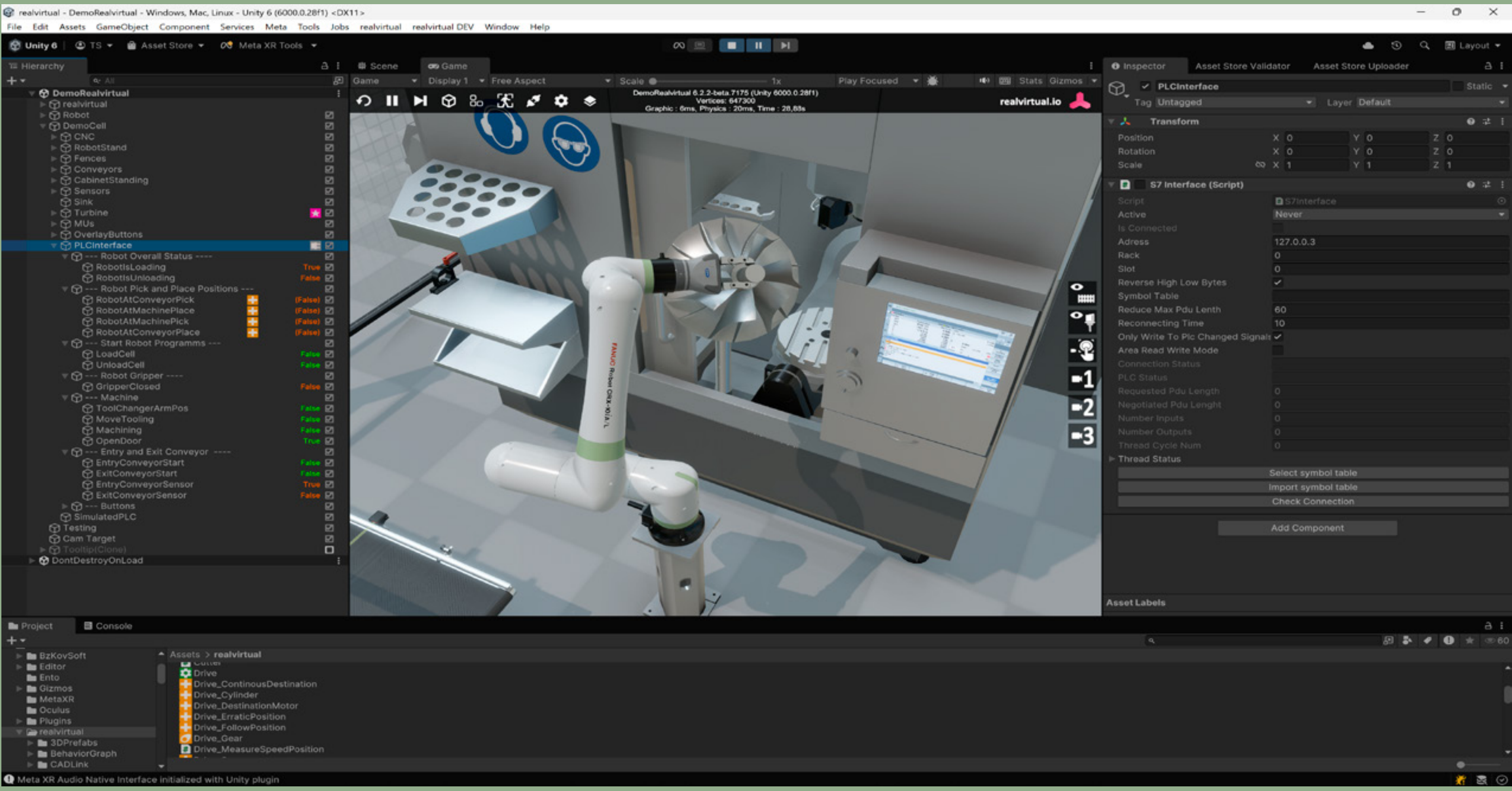
→ E-BOOK

Design, simulate, deploy: Building scalable industrial digital twins with Unity

Author: Thomas Strigl, CEO, Realvirtual.io

Co-author: Thomas Pospiech, Professor, Heilbronn University of Applied Sciences, and CTO, OPTIMA pharma.


realvirtual - DemoRealvirtual - Windows, Mac, Linux - Unity 6 (6000.0.28f1) <DX11>
File Edit Assets GameObject Component Services Meta Tools Jobs realvirtual realvirtual DEV Window Help
Hierarchy
Scene
Game
Inspector
Asset Store Validator
Asset Store Uploader
DemoRealvirtual 6.2.2-beta.7175 (Unity 6000.0.28f1)
realvirtual.io
Project
Console
Meta XR Audio Native Interface initialized with Unity plugin

PLCInterface
Tag Untagged Layer Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

S7 Interface (Script)
Script S7Interface
Active Never
Adress 127.0.0.3
Rack 0
Slot 0
Reverse High Low Bytes ✓
Reduce Max Pdu Lenth 60
Reconnecting Time 10
Only Write To Plc Changed Signals ✓
Requested Pdu Length 0
Negotiated Pdu Lenght 0
Number Inputs 0
Number Outputs 0
Thread Cycle Num 0

Virtual commissioning model connected to SEW PLC and motion controller via SEW MQTT SimInterface using the Unity Editor.

Unity®

Contents

Design, simulate, deploy: Building scalable industrial digital twins with Unity

A practical guide for robotics, automation, and manufacturing teams

As industrial systems evolve toward smarter, more connected, and increasingly autonomous operations, the role of digital twins has expanded dramatically. No longer just static 3D models or isolated simulations, digital twins are now real-time, interactive tools that support the entire lifecycle of machines and production systems—from concept and commissioning to operation, optimization, and training.

This e-book provides guidance and best practices for robotics engineers, automation experts, and industrial innovators who want to leverage Unity as a platform for building next-generation digital twins. Drawing from real-world use cases and lessons learned across industries, it explains not only what digital twins are, but how to implement them effectively—without falling into the trap of buzzwords or siloed thinking.

The guide walks through core architectural concepts, best practices for computer-aided design (CAD) import, kinematics, behavior simulation, and programmable logic controller (PLC) integration. It explores how Unity's real-time engine, open C# scripting, and multi-platform support make it uniquely suited for industrial applications, and highlights how tools like realvirtual.io accelerate development by providing ready-to-use components and interfaces to Siemens, Beckhoff, Fanuc, and more.

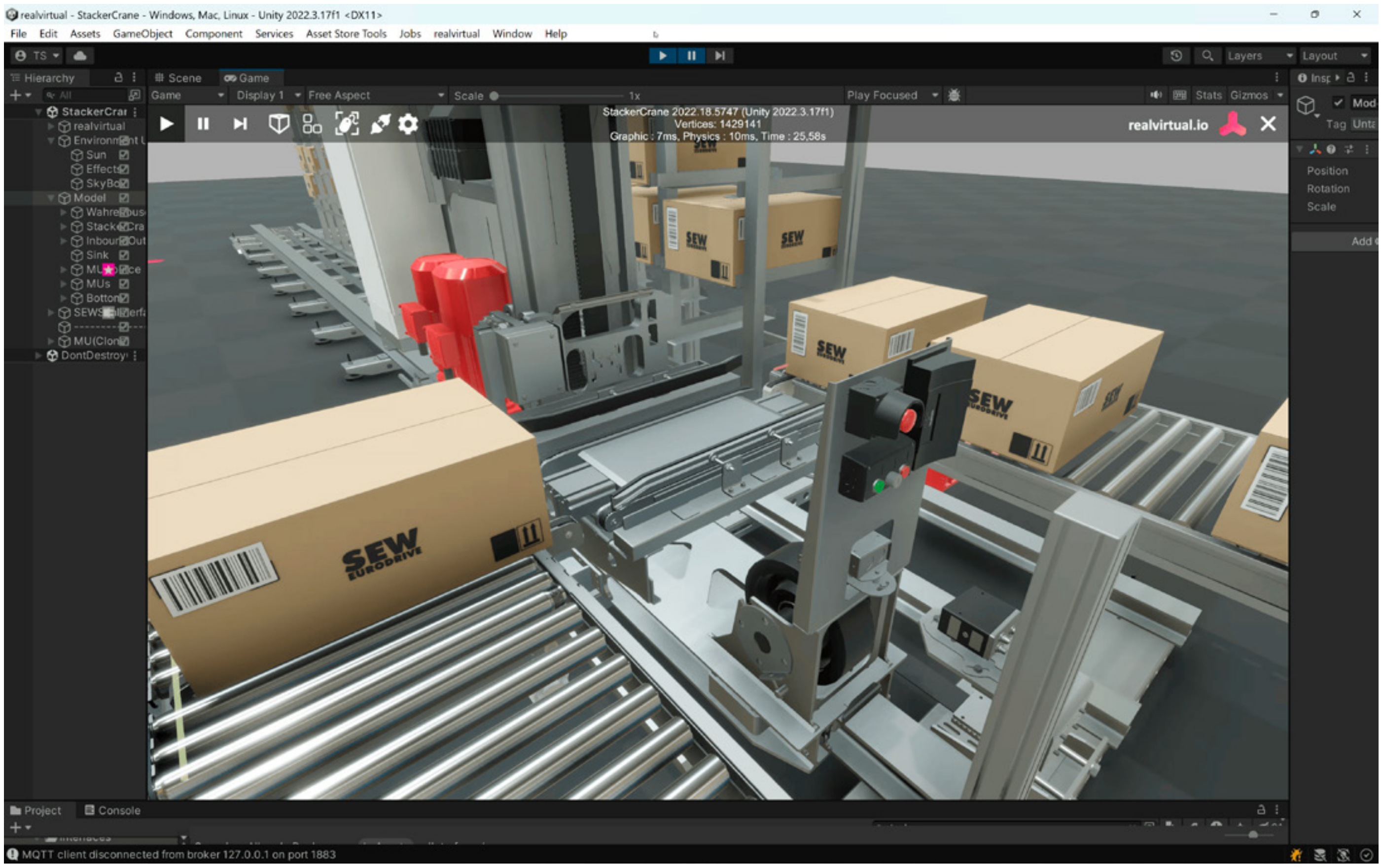
The shift to predictive and autonomous industry

Industrial production is undergoing a transformation. As manufacturers strive to increase efficiency, reduce labor costs, and respond to dynamic market demands, automation systems are becoming more complex. Robotics, autonomous systems, and machine-driven decisions are taking over tasks that once required human intervention.

At the same time, emerging technologies like artificial intelligence are unlocking new capabilities. Machine learning, computer vision, and even large language models are being increasingly used to optimize operations, detect anomalies, and interpret data in real-time. These systems are no longer isolated — they're interconnected, dynamic, and intelligent.

This shift comes with challenges. Skilled workers are becoming harder to find and retain, while the diversity of suppliers and technologies means that systems must be interoperable and modular. Production lines must be flexible and robust enough to support globally distributed teams—line builder in Germany, robot supplier in China, production in Mexico, stakeholder and production planning in the USA. Industrial operations are no longer local or static.

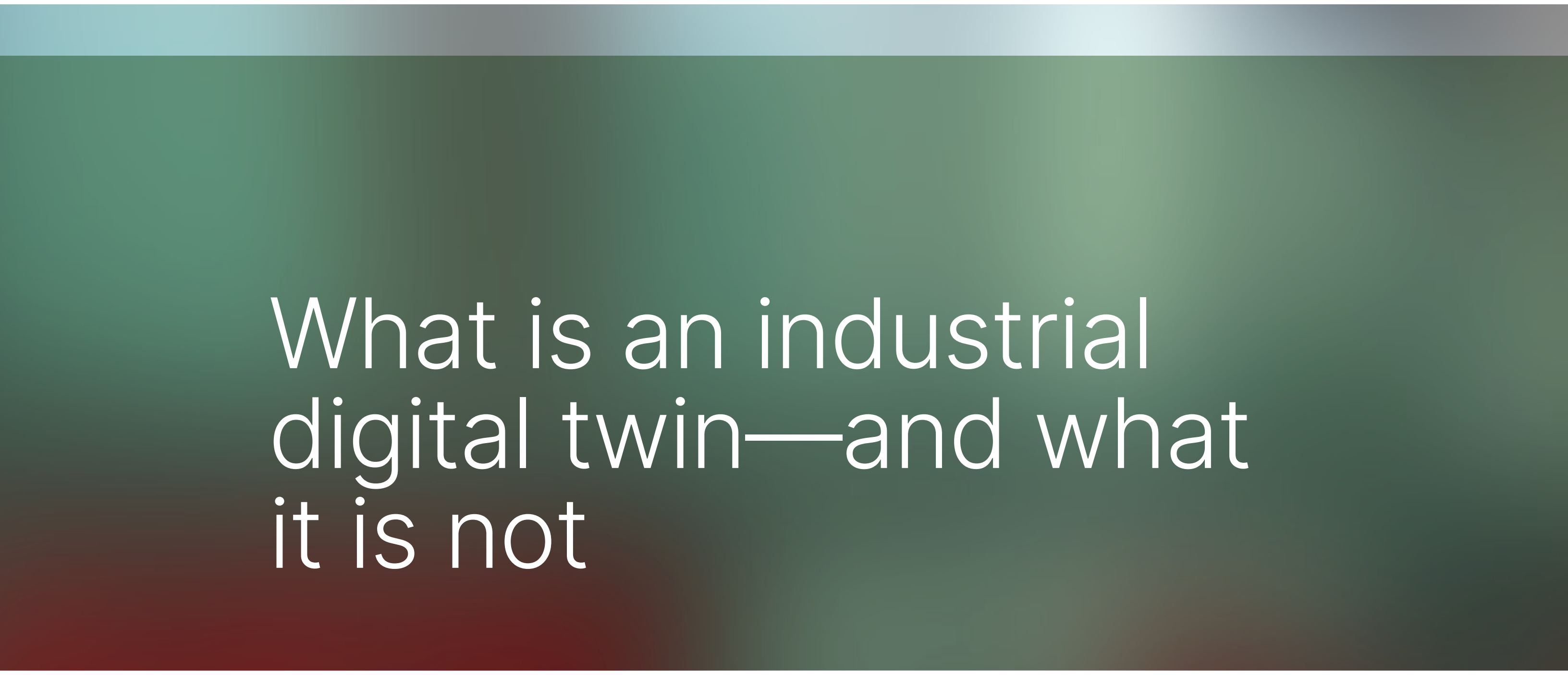
In this new reality, real-time, interactive 3D environments are not just a nice-to-have. They are essential. Digital twins powered by Unity can bridge the physical and digital worlds, enabling visualization, simulation, testing, and decision-making —all before reaching the shop floor.


realvirtual - StackerCrane - Windows, Mac, Linux - Unity 2022.3.17f1 <DX11>
File Edit Assets GameObject Component Services Asset Store Tools Jobs realvirtual Window Help
Hierarchy
Scene
Game
Display 1
Free Aspect
Scale
1x
Play Focused
Stats
Gizmos
Layers
Layout
realvirtual.io
Position
Rotation
Scale
Project
Console

```
StackerCrane 2022.18.5747 (Unity 2022.3.17f1)
Vertices: 1429141
Graphic : 7ms, Physics : 10ms, Time : 25,58s
```

```
MQTT client disconnected from broker 127.0.0.1 on port 1883
```

Image courtesy SEW-EURODRIVE and Realvirtual.io.



What is an industrial digital twin—and what it is not

Before the term “digital twin” became popular, the industry was already using digital tools to plan and optimize production. The concept of the ‘digital factory’ emerged in the late 1990s to early 2000s, driven by advancements in computer technologies and graphic cards, CAD, PLM (product lifecycle management), and 3D modeling. These tools enabled virtual validation of production environments long before physical commissioning.

The digital factory encompassed solutions like 3D layout planning, material flow simulation, robotic simulation, and ergonomic studies. These helped organizations test production setups and improve designs with minimal physical risk or cost.

Today, however, ‘digital twin’ is often used too loosely. Even a basic 3D animation gets labeled as a digital twin. This misuse risks reducing the term to a buzzword—detaching it from the real value it can bring.

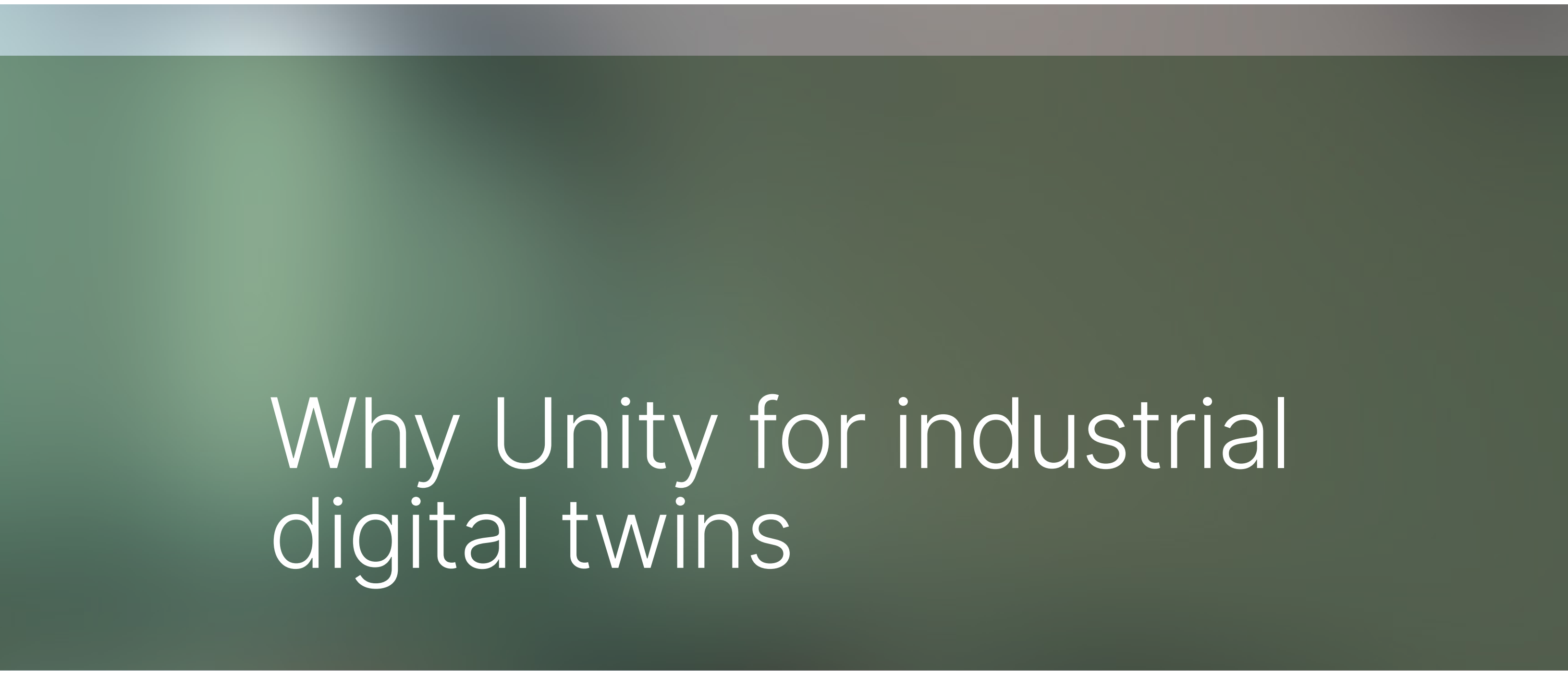
A true industrial digital twin is not created for its own sake. It serves a concrete purpose: better production planning, higher throughput, improved system reliability, faster training cycles, and increased operational insight. It is a functional, living replica of a system—connected to real data used for a real-world use case.

Real-world **use cases** of digital twins:

- **Simulation**: Accurately modeling system behavior based on technical data (not just animations) to detect problems, bottlenecks, and opportunities for improvement before the system is built or ordered—useful during sales or early system planning.
- **Virtual commissioning**: Testing automation logic, robotics, and PLC systems before connecting to real hardware. This enables faster ramp-up, test-driven automation development, and validation of safety concepts.
- **Training**: Providing immersive training environments for machine and line operators, as well as PLC and robotics programmers, to learn operation, handling, and programming tasks in advance.
- **AI development**: Using machine learning or AI vision systems trained on synthetic data. Digital twins enable large-scale simulations and data generation for optimal AI model performance.
- **Operations**: Operating machines with a 3D human-machine interface (HMI) that connects to the real system (digital shadow), delivering real-time feedback on status, maintenance needs, and possible problem resolutions—all integrated into a unified 3D context.

Digital twins go far beyond pure 3D keyframe animation. Also, what gets called a “digital twin” can vary widely—from predictive maintenance dashboards to data management standard or cloud machine data management, many of which do not directly involve 3D environments at all.

This guide focuses on *industrial digital twins* specifically that are based on *3D data and real-time simulation*. It is not our goal to provide a universal or scientific definition of the digital twin, but rather offer clarity and practical insights for those building systems with a focus on visualization, interaction, and operational integration.

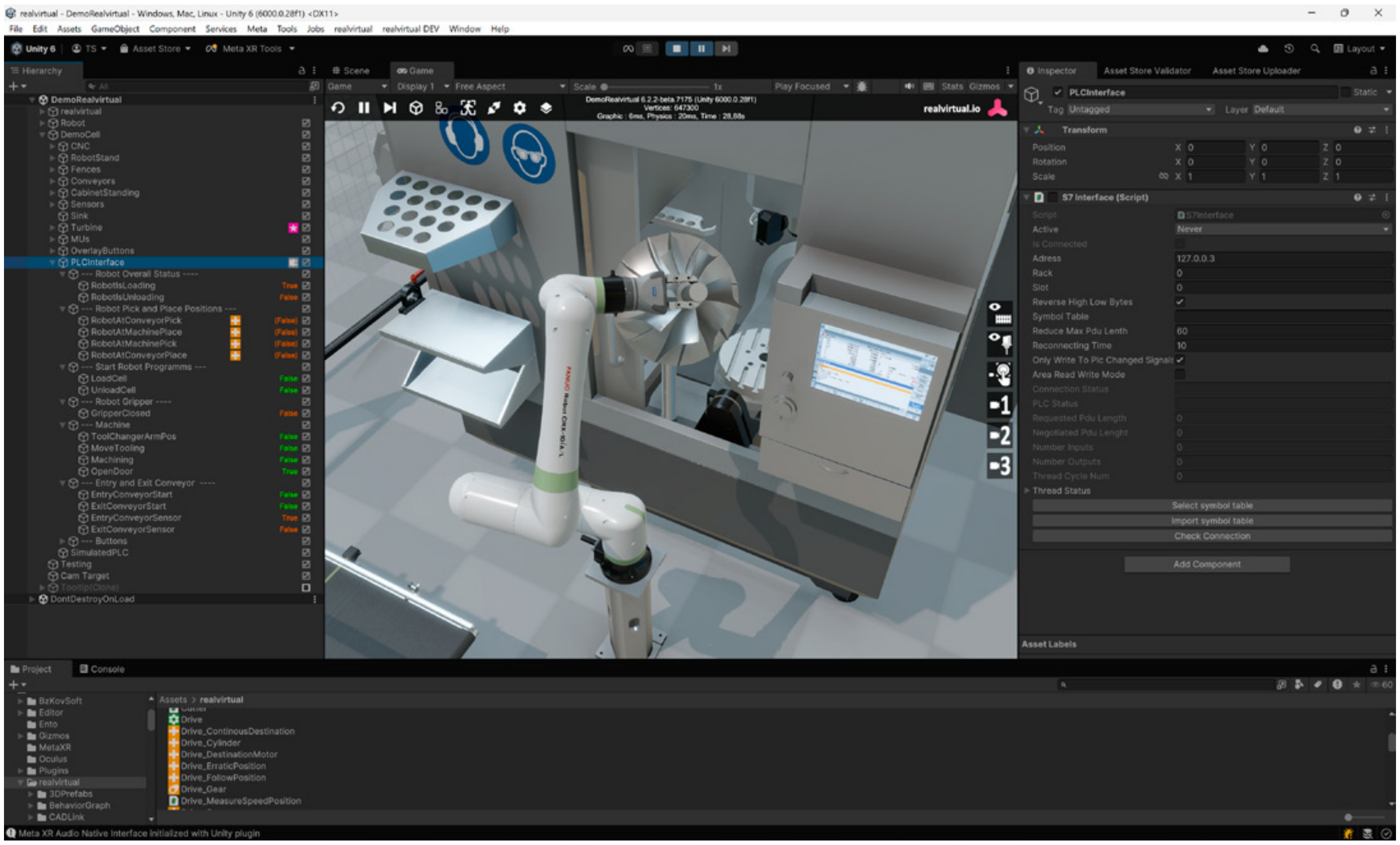


Why Unity for industrial digital twins

Digital twins span the entire industrial lifecycle—from early concept and sales visualization, through engineering and automation, to operations, training, and service. At every stage, having a performant, interactive 3D replica of a machine or production system is invaluable. Whether used in a sales presentation, an operator training scenario, or a live HMI system, a high-quality 3D model is the common foundation.

This is where Unity excels. Originally built for the gaming industry, Unity is designed for real-time performance across a broad range of hardware and platforms — including Windows, Linux, macOS, Android, iOS, WebGL, and XR devices like HoloLens, Quest, and industrial AR headsets. The same 3D model and simulation logic can be deployed across all of them with minimal adaptation.

This multi-platform strength enables a powerful workflow: **build once, reuse everywhere**. A well-made Unity-based 3D digital twin can serve different departments and use cases across an organization — from sales demos to engineering tests, training modules, and live machine monitoring.


PLCInterface
Tag Untagged   Layer Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

S7 Interface (Script)
Script S7Interface
Active Never
Is Connected
Adress 127.0.0.3
Rack 0
Slot 0
Reverse High Low Bytes ✓
Symbol Table
Reduce Max Pdu Lenth 60
Reconnecting Time 10
Only Write To Plc Changed Signals ✓
Area Read Write Mode
Connection Status
PLC Status
Requested Pdu Length 0
Negotiated Pdu Lenght 0
Number Inputs 0
Number Outputs 0
Thread Cycle Num 0
Thread Status

Virtual commissioning model connected to SEW PLC and motion controller via SEW MQTT SimInterface using the Unity Editor.

By contrast, many legacy tools from the digital factory era are highly specialized and siloed. They perform well within a narrow scope—like ergonomic simulation or robot reachability studies—but their outputs can't be easily reused or integrated beyond their original context. Often, the only way to share results is through static screenshots or exported videos, rather than interactive, updatable 3D environments.

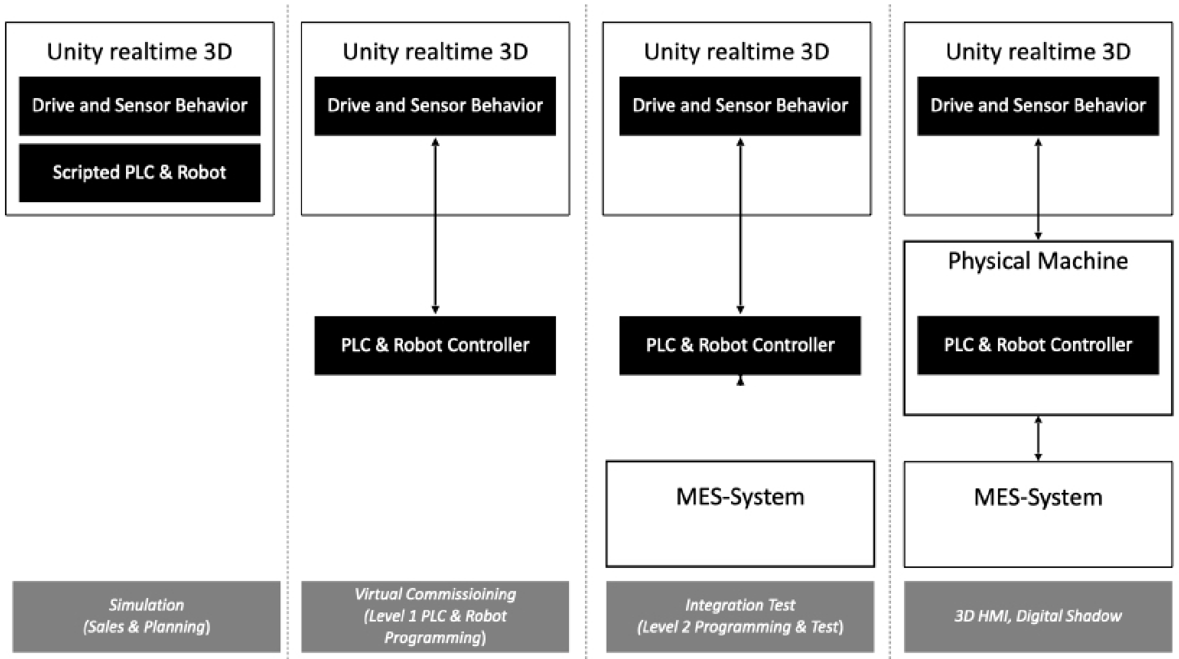
Another major strength of Unity is its open and extendable architecture. With full access to a powerful C# scripting API, developers can integrate virtually any custom logic, interface, or third-party system. Whether you're building a custom PLC protocol handler, an AI model runtime, or a proprietary UI layer—Unity gives you the flexibility to implement your exact requirements. Yes, this approach can be more complex than using a pre-configured, closed system, but, it also means there are no limits. Unity provides the foundation—and teams can build exactly what is needed, without compromise.

Equally important is the Unity development ecosystem. Unity benefits from one of the largest and most active developer communities in the world. This brings a massive advantage: tutorials, plugins, asset libraries, templates, and documentation are readily available — not just from Unity itself but from a vibrant ecosystem of independent developers, industrial partners, and open-source contributors. Whether you're starting from scratch or scaling to an enterprise-grade solution, you're never building in isolation.

And in contrast to many SaaS-based platforms, Unity offers freedom of deployment. Especially in sensitive industries like machinery and automation, not every company wants to rely on cloud-only infrastructure. With Unity, digital twins can be run locally with on-premise hardware or even offline.

Core architecture of a Unity-based digital twin

Not all digital twins are created equal. Depending on the application phase—from sales and planning to commissioning and operation—digital twins require different levels of integration and technical depth. Unity allows companies to scale their architecture along this continuum, from simple behavior-driven 3D simulations to full integration with live production systems.


Unity realtime 3D
Drive and Sensor Behavior
Scripted PLC & Robot
Simulation (Sales & Planning)
Unity realtime 3D
Drive and Sensor Behavior
PLC & Robot Controller
Virtual Commissioining (Level 1 PLC & Robot Programming)
Unity realtime 3D
Drive and Sensor Behavior
PLC & Robot Controller
MES-System
Integration Test (Level 2 Programming & Test)
Unity realtime 3D
Drive and Sensor Behavior
Physical Machine
PLC & Robot Controller
MES-System
3D HMI, Digital Shadow

From simulation to digital shadows: Four levels of an industrial digital twin. Image courtesy of Thomas Strigl.

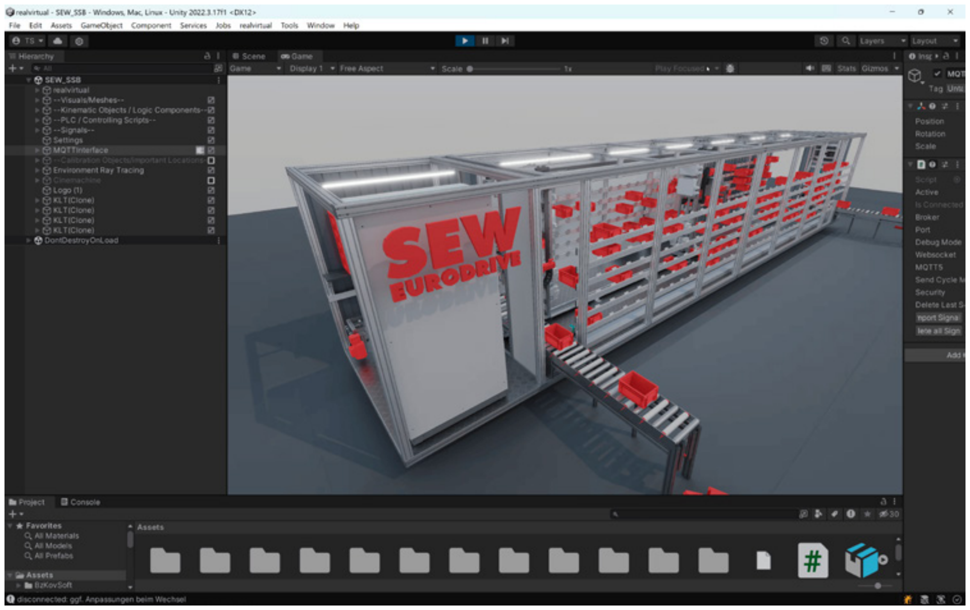
— At the most basic level, digital twins serve as **planning, sales and simulation tools**. Here, Unity can be used to create interactive 3D environments that simulate behavior through scripted logic and event-driven components. It is important to mention here the difference to a simple keyframe animation. The underlying logic needs to mirror what would later be implemented at the PLC or MES level. Drives, actuators, and axes are modeled with realistic motion parameters—acceleration, deceleration, limits, sensor states and timing — to reflect how the system will actually behave. This allows early validation of motion sequences, timing coordination, and space requirements before engineering is finalized.

— The next level brings in real automation systems. **Virtual commissioning** enables teams to connect Unity to actual PLCs and robot controllers via interfaces like OPC UA, Beckhoff ADS, or Siemens S7. Instead of simulated control, real control software is tested within a 3D simulation, offering a safe and efficient method for debugging, validating logic, and reducing on-site commissioning time.

— As systems mature, **integration testing** becomes essential. At this stage, the digital twin in Unity is connected not just to controllers but also to manufacturing execution systems (MES). This makes it possible to simulate and validate entire processes— including production order execution, data exchange, and inter-machine communication. Unity becomes a real-time orchestrator for visualizing and verifying how machines, data systems, and human operators will interact.

— Finally, the highest level of maturity brings Unity into live production environments as a **digital shadow or 3D HMI**. Here, Unity doesn't simulate—it reflects. The real system is mirrored in a live, data-driven 3D environment that shows actual machine states, sensor readings, alarms, and performance indicators. Operators can interact with this visualization locally or remotely to understand system status, diagnose issues, and access contextual information in a visual way.

What unites all these stages is Unity's ability to integrate real-world data, simulate realistic physics and machine behavior, and structure content in a modular, scalable way. Whether used offline for planning or online for real-time operation, a Unity-based digital twin grows with the systems—bridging teams, phases, and technologies in one platform.



Digital twin used for virtual commissioning based on realvirtual.io and Unity connected to SEW MOVI-C® Controller (Automation and design developed by INperfektion GmbH). Image courtesy SEW-Eurodrive and Realvirtual.io.

How to integrate industrial systems

Integrating industrial systems means establishing communication with PLCs, robots, manufacturing execution systems (MES), and other connected components in an industrial setup. There is no single right way to do this—the correct approach depends on what data needs to be exchanged, how often, and with what latency requirements.

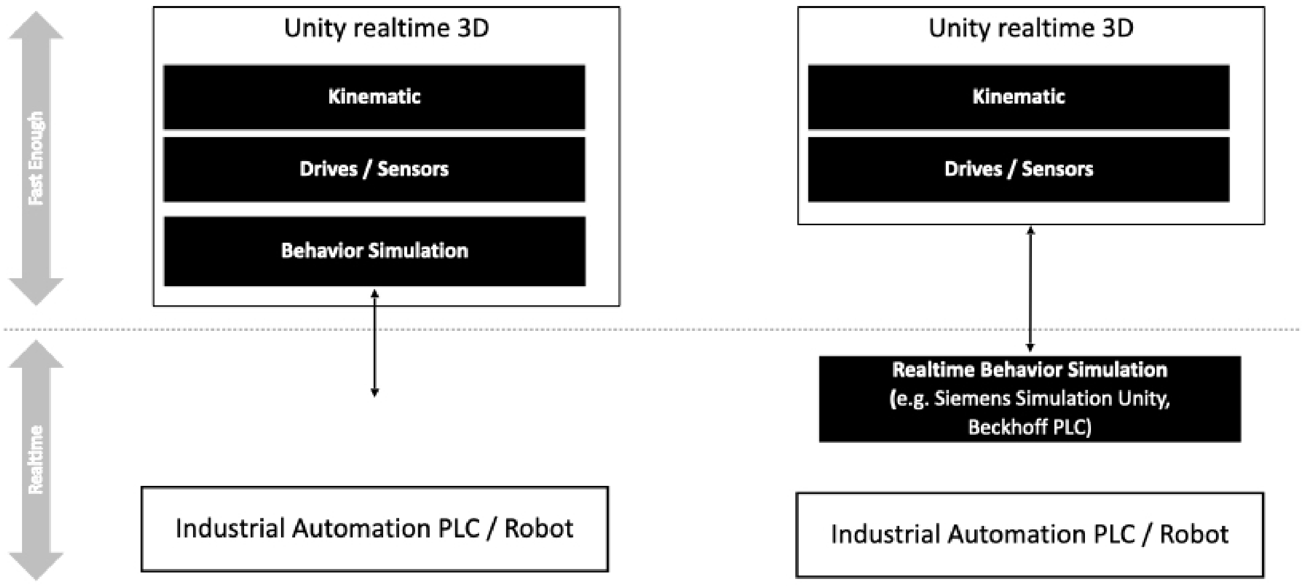
Standardized communication protocols like open platform communications unified architecture (OPC UA) are highly effective for MES-level data exchange, where robust structure and semantic interoperability are more important than real-time speed. However, when fast, cyclic communication is needed—for example, when simulating real robot or axis motion—vendor-specific protocols such as Siemens S7 TCP-IP or Beckhoff ADS often offer significantly better performance.

For digital twins that need to mirror fast real-world processes with high temporal accuracy, the choice of protocol becomes critical. Smooth, realistic motion and real-time feedback demand low-latency, high-frequency communication—something that general-purpose industrial protocols may not deliver efficiently.

It's important to clarify what "real-time" actually means in this context. In industrial automation, real-time communication refers to deterministic timing—the guarantee that data is delivered within a specific time window, reliably and predictably. Most general-purpose operating systems like Windows are **not** real-time capable by default. They may perform well in many scenarios but are not designed to guarantee microsecond or millisecond-level consistency required in certain closed-loop applications.

That said, full real-time simulation is **not required** in most digital twin use cases. Many industrial motion controllers already simulate their internal drive behavior with high fidelity. In these cases, it is usually sufficient to exchange drive positions and state data at a fast and stable update rate. The accuracy of the simulation depends more on the update frequency and the control structure than on achieving hard real-time execution within Unity.

Industrial solutions like **Siemens SIMIT and Simulation Units** or **Beckhoff's EtherCAT Simulation** provide highly accurate representations of real-time behavior—either via dedicated simulation hardware or via mirrored PLC architectures. Unity can act as a visual and interactive front-end to these simulation layers, leveraging their timing precision while providing a flexible, high-performance 3D environment for validation, visualization, and training. This is why protocol performance and the runtime environment both matter when designing integrated digital twin systems.


Fast Enough
Realtime
Unity realtime 3D
Kinematic
Drives / Sensors
Behavior Simulation
Industrial Automation PLC / Robot
Unity realtime 3D
Kinematic
Drives / Sensors
Realtime Behavior Simulation (e.g. Siemens Simulation Unity, Beckhoff PLC)
Industrial Automation PLC / Robot

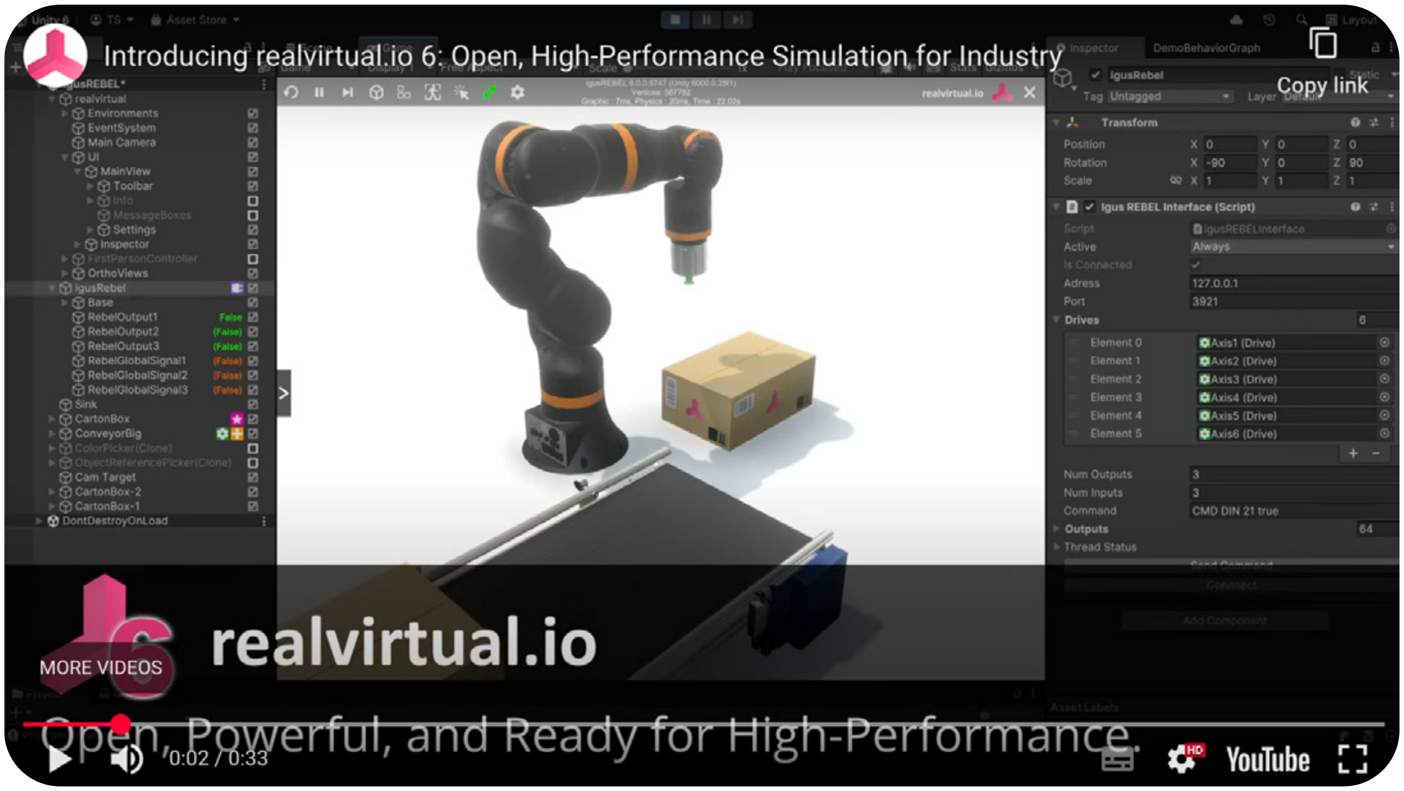
Realtime or "fast enough" behavior simulation. Image courtesy Thomas Strigl

Unity does not include built-in support for industrial communication protocols. However, thanks to its open and extensible C#/.NET environment, developers can integrate virtually any custom or standardized communication layer.

This open approach gives developers the full flexibility to implement the communication logic that fits their specific application, including connecting to proprietary hardware or implementing edge-device interfaces. Unlike many closed or siloed systems, Unity imposes no artificial restrictions—only the limits of what your development team can build.

Tools like **realvirtual.io** bridge this gap by offering Unity-native access to a comprehensive range of industrial protocols—from low-level PLC-specific channels like S7, Beckhoff ADS, and Ethernet/IP to higher-level IoT and cloud-friendly protocols such as Message Queuing Telemetry Transport (MQTT).

This flexibility means that Unity can be tailored to suit both fast real-time applications and slower, event-driven system layers. Whether you're building a live commissioning environment or a distributed MES interface, Unity can adapt to your specific integration needs—without forcing your architecture into a predefined communication model.


Introducing realvirtual.io 6: Open, High-Performance Simulation for Industry
Copy link
realvirtual.io
MORE VIDEOS
Open, Powerful, and Ready for High-Performance
0:02 / 0:33
YouTube

Introducing realvirtual.io 6: Open, High-Performance Simulation for Industry

To summarize, choosing the right communication protocol is always a compromise between several factors—ease of implementation, performance characteristics, and the availability of technical solutions both on the Unity side and the automation system side. There is no one-size-fits-all answer. The ideal protocol depends on the use case, the technical environment, and the intended fidelity of the digital twin.

***Pro tip**: If you are an Original Equipment Manufacturer (OEM) and planning to integrate automation protocols into your Unity application, consider starting with a WebSocket-based communication layer. This approach offers high platform flexibility—especially across mobile, desktop, and WebGL deployments—while still delivering solid performance and real-time responsiveness. Solutions like the realvirtual.io Beckhoff HMI interface or MQTT exemplify this flexibility.*

Update cycles for digital twins

A common question when designing a digital twin is: *How fast should the communication be?* While it depends on the application, a useful rule of thumb is that for most industrial use cases, update cycles between **10–50 milliseconds** are sufficient to ensure smooth motion and accurate state feedback. This typically supports applications such as motion visualization, drive synchronization, or sensor emulation. For slower control layers or monitoring-only systems, even update intervals of **100–200 milliseconds** may be acceptable.

Only in highly time-critical or closed-loop control scenarios — such as safety simulation or hardware-in-the-loop—would update rates faster than **5 milliseconds** be necessary. In these cases, simulation architecture and platform selection must be carefully engineered.

Using digital twins as 3D HMI – The digital shadow in operation

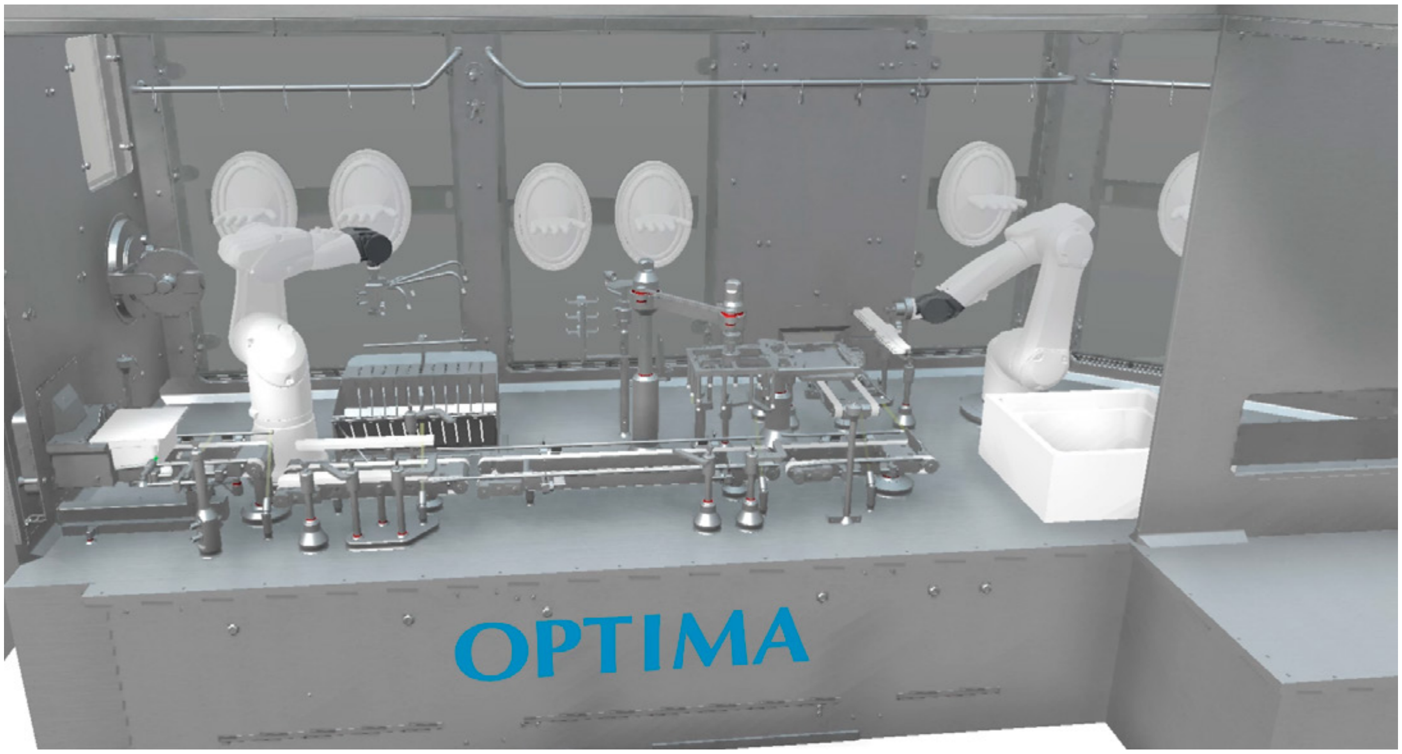
Digital twins are not just for simulation or commissioning. When connected to real automation systems, they can evolve into powerful operational tools—acting as fully interactive, real-time visual interfaces for machines and processes. This role is often described as a digital shadow: a live 3D representation of a machine or line that reflects its actual state, logic, and performance.

Unlike traditional HMIs that display symbolic data on flat panels, a Unity-based 3D HMI offers an immersive, spatial view of a system. It shows what's happening—where, when, and why—in the context of the machine's structure and layout. The operator or technician doesn't just read numbers on a screen. They see the status directly in the digital twin: machines changing state, sensors triggering, drives moving, and errors occurring—exactly where they are in the real world.

This 3D interface can display IDs of sensors and components, sensor states, part positions, and error messages located on the machine. It can show maintenance instructions, access PLC variables, or overlay live and historical data. In extended scenarios, it may even include AI-generated insights—predicting problems, recommending actions, or explaining complex system behaviors.

Another powerful advantage is the ability to link live 3D context to support content. For example, when a fault occurs, the user can click on the affected machine component in the 3D view and immediately access the corresponding service manual, maintenance instruction, or troubleshooting guide. This minimizes search time and helps less experienced operators resolve issues more efficiently—all without leaving the HMI.

Because Unity supports multiple platforms, the 3D HMI can be experienced in different ways. On a desktop or shopfloor panel, it may be part of a traditional interface. On a tablet or mobile device, it becomes a portable diagnostic tool. In a VR headset or through AR glasses, it becomes a collaborative, immersive environment—useful for training, support, and high-complexity troubleshooting.



Digital twin for virtual commissioning in pharmaceutical production. Image courtesy Optima Pharma GmbH.

For embedded and industrial use cases, Unity applications can also be deployed to Linux-based industrial PCs and—in some configurations—even run directly on modern PLC hardware that supports containerized or custom runtime environments.

Alternatively, Unity's WebGL build option allows you to integrate the 3D HMI directly into existing web-based HMI platforms, offering real-time 3D visualization as part of your browser-accessible dashboard—without requiring a separate application.

The 3D HMI doesn't need to be complicated to create value. Visualizing just a few signals or showing status indicators in a 3D layout already brings clarity that conventional HMIs cannot match. With tools like realvirtual.io Professional providing a wide range of industrial interfaces and dedicated HMI features, you can connect to a live PLC, display real sensor data in context, and start building a digital shadow—right away.

And while today's 3D HMI might focus on real-time sensor status and operator feedback, it opens the door to what's next: AI-powered decision support, remote collaboration, and context-aware interfaces that understand not just what's happening —but what should happen next.

Behavior simulation in digital twins

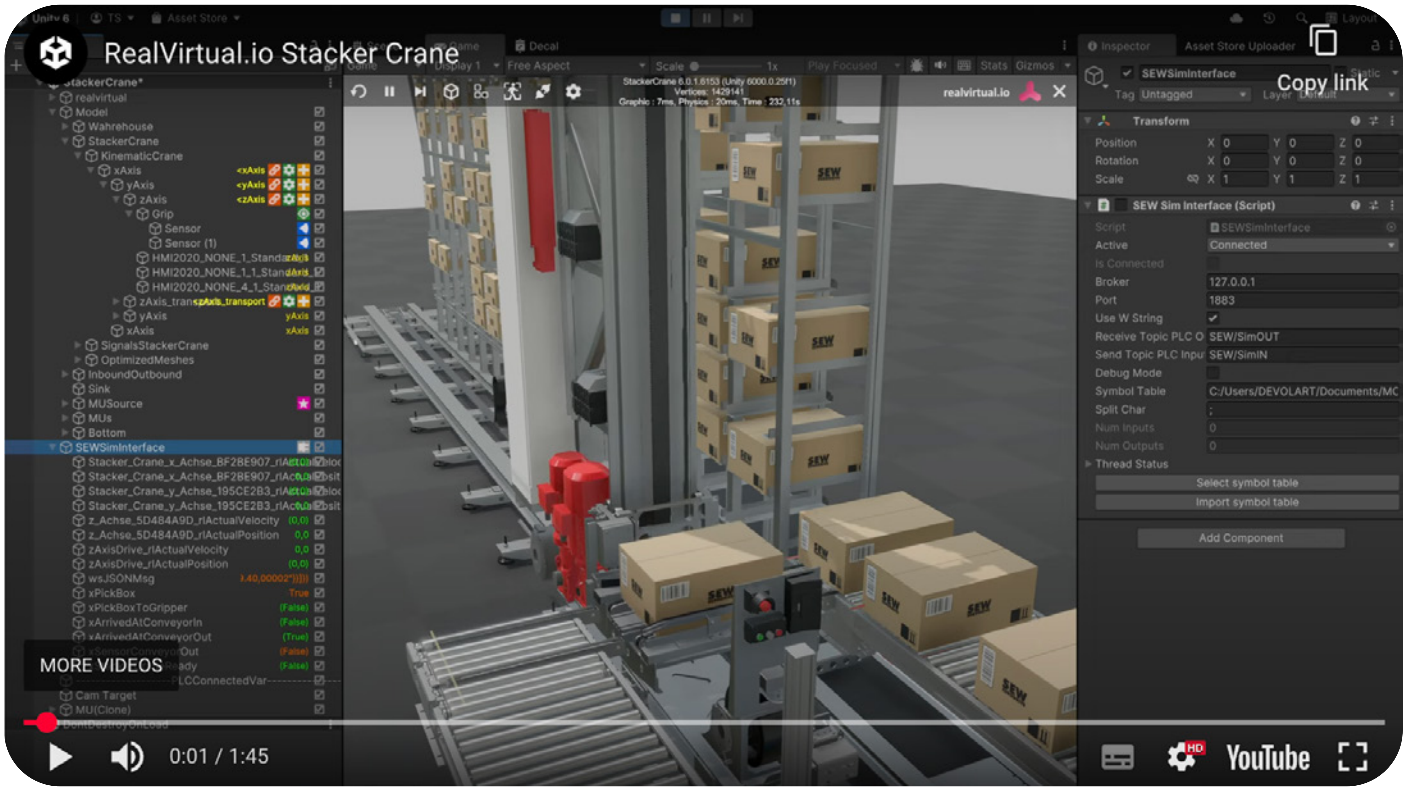
A digital twin typically begins as a 3D CAD model, which is imported into Unity and transformed into a kinematic structure. Each axis, actuator, or mechanical linkage is defined with motion directions, physical limits, and hierarchical relationships. However, for the twin to move beyond pure visualization, it requires a **behavior simulation layer**—the logic that defines how control signals (typically coming from a PLC or robot controller) are translated into physical movement.

Behavior simulation represents the digital twin's dynamic response: how fast a drive accelerates or decelerates, how it reaches a target position, how sensor states are triggered, and how mechanical constraints are enforced. This enables the simulation to realistically reflect system behavior under automation control—allowing early-stage validation, functional testing, and performance analysis before deployment.

Behavior simulation is especially important for virtual commissioning and detailed system simulation, where precise behavior is needed to test sequences, interlocks, and timing. In contrast, training applications or 3D HMI systems usually don't require detailed internal simulation of drives. In those cases, it's often sufficient to receive high-level position or status data from an external controller or simulation system.

Crucially, behavior simulation can be done within Unity—using C# scripts and physics-based logic—or externally, using standard simulation tools. Solutions like MATLAB/Simulink, FMI/FMU-based models, or systems like Siemens SIMIT and Beckhoff Simulation Units can all provide detailed behavior simulations that are connected to Unity as the 3D visualization and interaction layer.

The right choice depends on the complexity of the system and the availability of validated simulation models. For example, if a detailed and tested drive or process simulation already exists in MATLAB Simulink as part of a system engineering workflow, it can be highly effective to reuse that model and link it to Unity's 3D environment.



Broker 127.0.0.1
Port 1883
Receive Topic PLC O SEW/SimOUT
Send Topic PLC Inpu SEW/SimIN

https://www.youtube.com/watch?v=EaAk8iWKFUg

Toolkits like realvirtual.io provide this exact flexibility—supporting both internal behavior simulation within Unity and integration with external tools like Simulink or SIMIT. This gives developers and OEMs a modular architecture where Unity serves as a real-time 3D platform that adapts to existing engineering tools and scales across simulation depth.

Best practices for modeling robotics and machinery in Unity

Creating accurate and maintainable digital twins of industrial machines in Unity requires more than importing 3D models—it demands a disciplined modeling approach that reflects real-world function, structure, and control logic. The goal is not just to visualize machines, but to simulate and interact with them in a meaningful way throughout their lifecycle: planning, engineering, training, and operation.

1. Bridging engineering and simulation with smart CAD import

Start by importing your computer aided-design (CAD) data using tools like Unity Asset Transformer (previously known as Pixyz included with Unity Industry or realvirtual.io, both of which support a wide range of standard CAD formats such as standard for the exchange of product data (STEP), SolidWorks, CATIA, and more. These tools not only allow direct access to native CAD files but also offer filtering, tessellation control, and material assignment for optimized real-time use. Proper CAD import is the foundation of a reliable and efficient digital twin.

During import, the parametric CAD geometry is converted into tessellated meshes, which are essential for real-time rendering and visualization. This transformation enables Unity to display and interact with complex industrial models in a performant way.

Choose a mesh quality that balances visual clarity and simulation performance. The goal is not to replicate every screw thread or fine chamfer, but to preserve the essential visual identity and spatial accuracy of the machine or component. Overly detailed meshes increase rendering load and can significantly affect performance — especially on constrained platforms like mobile devices or WebGL deployments. Clean, simplified geometry works best in most real-time use cases.

With the right import tools and pipeline, no manual data preparation is needed on the CAD side. The CAD system should always serve as the source of truth, maintaining ownership over the geometry. That's why a fast, smooth, and updateable import pipeline is essential.

2. Define kinematic hierarchies

To enable realistic motion and automation behavior, it's essential to define a kinematic hierarchy that mirrors the mechanical structure of the real machine. Ideally, the CAD model is already organized in a way that reflects moving components — for example, with each rotating or linear axis placed in its own transform node. However, in practice, this is rarely the case. Most CAD assemblies are structured for manufacturing, not for real-time simulation, which makes them unsuitable for direct use without additional preparation.

One option is to manually restructure the GameObject hierarchy in Unity to establish a proper kinematic chain. But this approach comes with a major drawback: it breaks the connection to the original CAD data. Once the structure is changed inside Unity, automatic updates become impossible—a critical limitation in projects where CAD models evolve frequently.

This is where tools like realvirtual.io offer a major advantage. realvirtual.io provides features that allow you to define a kinematic hierarchy in parallel to the imported CAD hierarchy, without modifying the original structure. This means you can keep your CAD data intact while overlaying motion logic, behaviors, and axis relationships on top of it.

Even more importantly, realvirtual.io supports non-destructive CAD updates. When new CAD files are imported, existing simulation logic, axis configurations, and component references remain intact. This capability makes it possible to maintain a continuous and agile development process—where the digital twin evolves in step with the mechanical design, without requiring manual rework or reconfiguration.

When implementing kinematics in Unity, there are two main architectural approaches:

— **GameObject transform hierarchies**: This is the most practical and stable approach for most industrial applications. For systems with defined motion paths — such as robots, gantries, and pick-and-place mechanisms — using clean, nested GameObject transforms allows for precise control via cyclic position updates from motion controllers or robot programs. It's easy to manage, highly performant, and works reliably across platforms.

— **PhysX joint-based kinematics**: Unity's physics engine can simulate mechanical constraints using rigid bodies and joints (e.g., hinge joint, configurable joint). This is useful for cases where real-time physical interaction, collisions, or compliance behavior are needed. However, these setups are more complex and less deterministic when driven by external control systems.

In most cases, the best approach is to use the GameObject hierarchy for predictable, high-performance kinematic control, and to use PhysX-based kinematics only where physically reactive behavior is truly needed—such as simulating collisions or mechanical compliance.

By selecting the right kinematic architecture and using tools that support flexible, non-destructive updates, you can ensure your digital twin remains scalable, maintainable, and in sync with both your engineering data and automation systems.

3. Design for behavior simulation

As discussed in the first part of the e-book [insert link], behavior simulation adds motion logic to your digital twin. To make this manageable, isolate drive logic from visualization. Use modular scripts or state machines to define how each component responds to control inputs. Reuse behavior templates across similar machines, and make sure to separate simulation logic from live controller integration so the twin can switch between simulated and real inputs easily. Realvirtual.io provides a solid foundation for this or you can develop this approach on your own based on C# scripting.

SEW-EURODRIVE and realvirtual.io: Accelerating virtual commissioning with Unity
Copy link
SEW EURODRIVE
realvirtual.io
SEW-EURODRIVE AND REALVIRTUAL.IO
MORE VIDEOS
Standardizing Virtual Commissioning with Unity

https://youtu.be/z7Y7FRSfQVc

4. Structure your scenes modularity

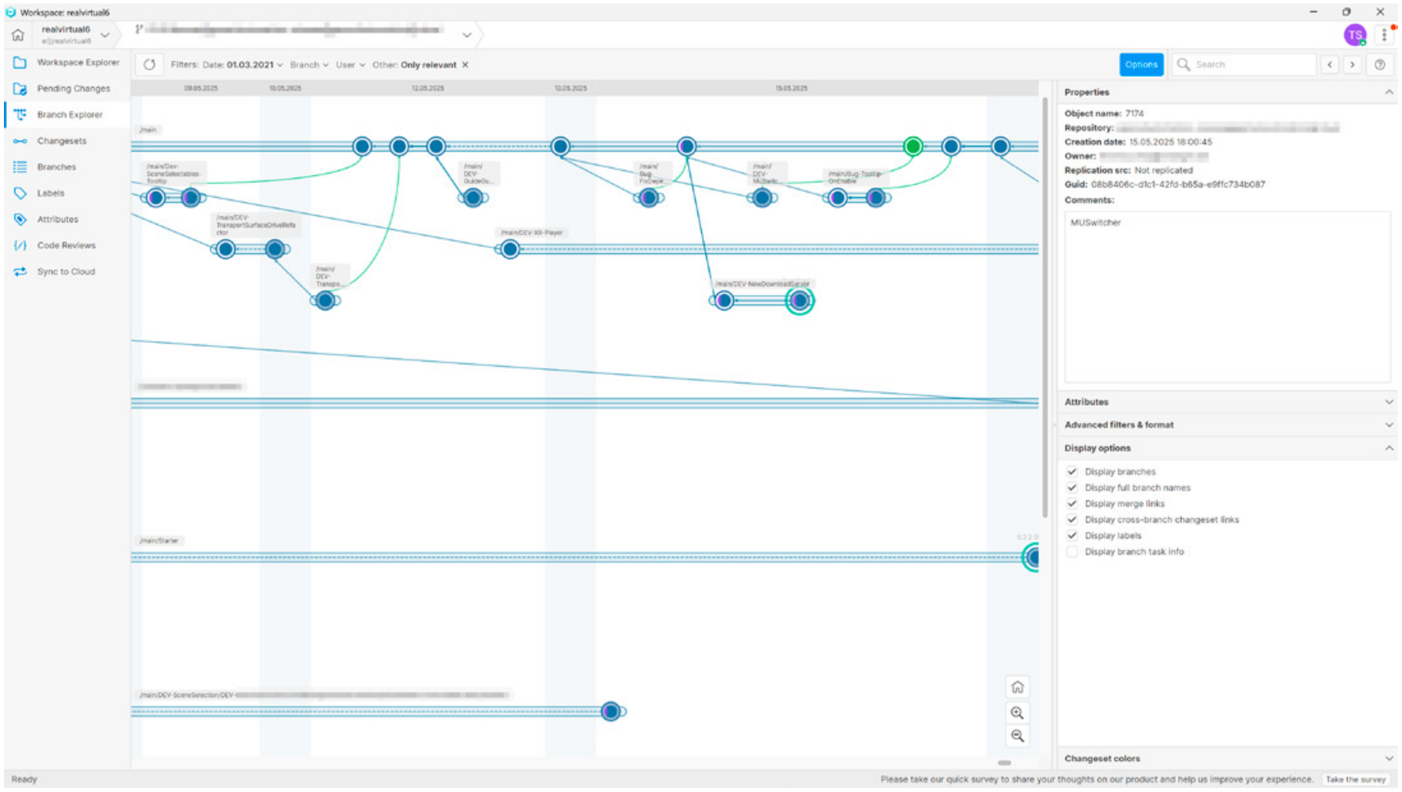
Use Unity prefabs and reusable components to define machine modules, sensor setups, product carriers, or robotic tools. This allows you to build libraries of tested assets that can be assembled into new systems quickly—essential for projects which are based on reusable standard components.

5. Integrate sensors and feedback loops

Model sensors in Unity not just as visual indicators but as functional elements. Use raycasts, collider triggers, or custom logic to simulate proximity sensors, light barriers, encoders, and safety switches. Output their states via signals that can be consumed by behavior logic or external control systems. This enables realistic interaction and test-driven commissioning.

6. Document and version your models

Industrial projects often evolve over years and are a team effort. Document your digital twin components, logic, and interfaces. Use version control, for example Unity's Version Control (previously Plastic SCM), to track changes, and consider scriptable object-based settings to centralize configuration. A well-maintained twin will remain usable and expandable as the real system changes.



Properties

Object name: 7174
Repository: [redacted]
Creation date: 15.05.2025 18:00:45
Owner: [redacted]
Replication src: Not replicated
Guid: 08b8406c-d1c1-42fd-b65a-e9ffc734b087
Comments:

MUSwitcher

Attributes

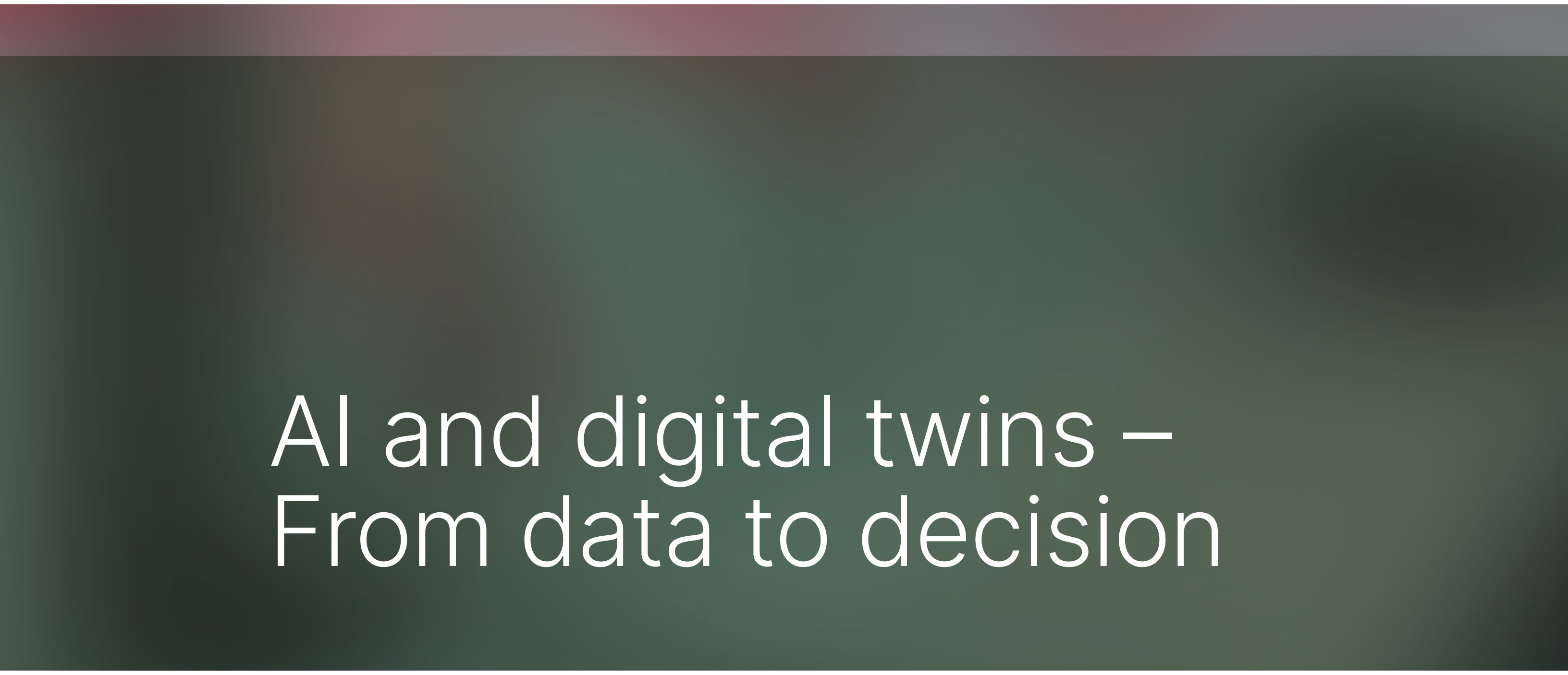
Advanced filters & format

Display options

- [✓] Display branches
- [✓] Display full branch names
- [✓] Display merge links
- [✓] Display cross-branch changeset links
- [✓] Display labels
- [ ] Display branch task info

Changeset colors

Version and revision management with Plastic. Image courtesy Thomas Strigl



AI and digital twins – From data to decision

Artificial intelligence (AI) is rapidly changing how we design, operate, and optimize industrial systems — and digital twins play a central role in enabling this transformation. By providing a controlled, data-rich, and interactive 3D environment, Unity-based digital twins serve as ideal platforms for developing, testing, and deploying AI solutions in manufacturing and automation.

It's important to note that AI itself is a broad and rapidly evolving field—one that could easily fill an entire e-book. In this section, we focus specifically on the connection between AI and Unity-based digital twins: how simulation, synthetic data, and real-time 3D environments enable smarter systems and faster implementation.

1. From visuals to vision: Unity + ONNX for AI inference

One of the most impactful use cases for AI in digital twins is computer vision—particularly for inspection, object recognition, sorting, and robotic guidance. With Unity's Sentis support for ONNX (Open neural network exchange), it is possible to integrate pre-trained machine learning models directly into a simulation. This allows Unity to perform real-time inference using AI models for tasks such as:

— Detecting objects on a conveyor belt

— Classifying product types or quality levels

— Guiding pick-and-place robots based on AI vision output

— Responding to anomalies in live environments

AI models can be embedded locally in Unity scenes or accessed through external Python runtimes and inference servers, depending on the deployment target and performance needs.

2. Synthetic data for training AI models

When deploying AI-based systems in industry, one of the biggest challenges is the lack of available training data. In most cases, the production line doesn't yet exist—and there are no real images or labeled datasets to train vision models. This is where digital twins become essential.

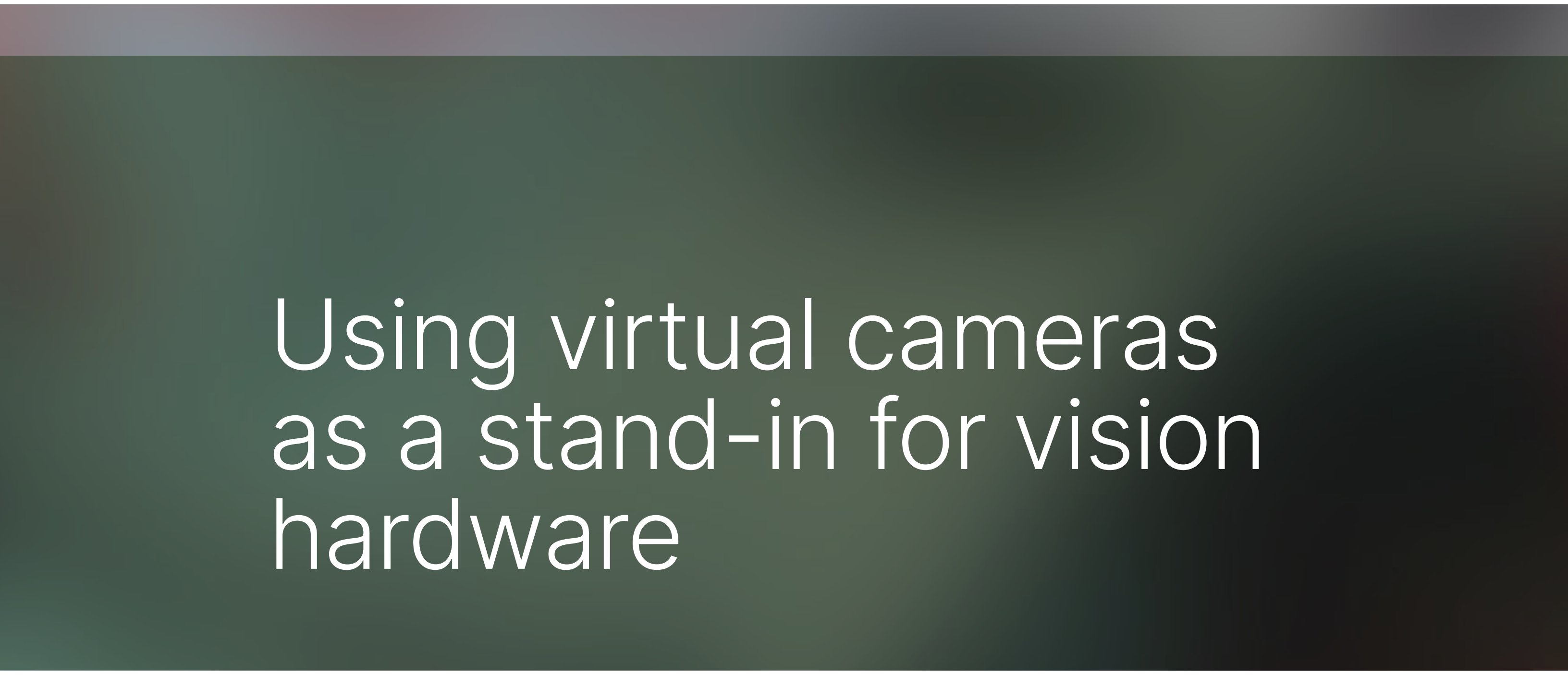
Unity's digital twin environment can simulate lighting, textures, object variation, and camera perspectives—enabling the generation of synthetic training datasets at scale. These images, along with perfectly aligned ground-truth labels, can be exported to train AI models for:

- Object detection
- Classification
- Segmentation
- Pose estimation

This method dramatically reduces the time, cost, and logistical hurdles of building AI solutions — especially in early design phases.

SEW-EURODRIVE and realvirtual.io: Accelerating virtual commissioning with Unity
Copy link
SEW EURODRIVE
realvirtual.io
SEW-EURODRIVE AND REALVIRTUAL.IO
MORE VIDEOS
Standardizing Virtual Commissioning with Unity

Demystifying IoT for operational digital twins | Unite 2024



Using virtual cameras as a stand-in for vision hardware

Unity’s virtual cameras can be positioned and configured just like real industrial cameras—including field of view, resolution, perspective, and even lens distortion. This allows you to replicate what an AI vision system will “see” before any physical vision hardware is installed on the production line.

These virtual cameras can feed simulated image streams to real AI pipelines, or be used to develop and configure vision systems in advance—giving engineers a major head start on setup and verification.

1. Real-time feedback loops

Once trained, AI models can be deployed back into the Unity digital twin to form real-time feedback loops. For example, an AI-powered sorting system can classify items on a conveyor and send control signals to diverter actuators—all simulated inside Unity before deploying to hardware. This makes it possible to test logic, verify decision quality, and measure system responsiveness in a safe, repeatable virtual environment.

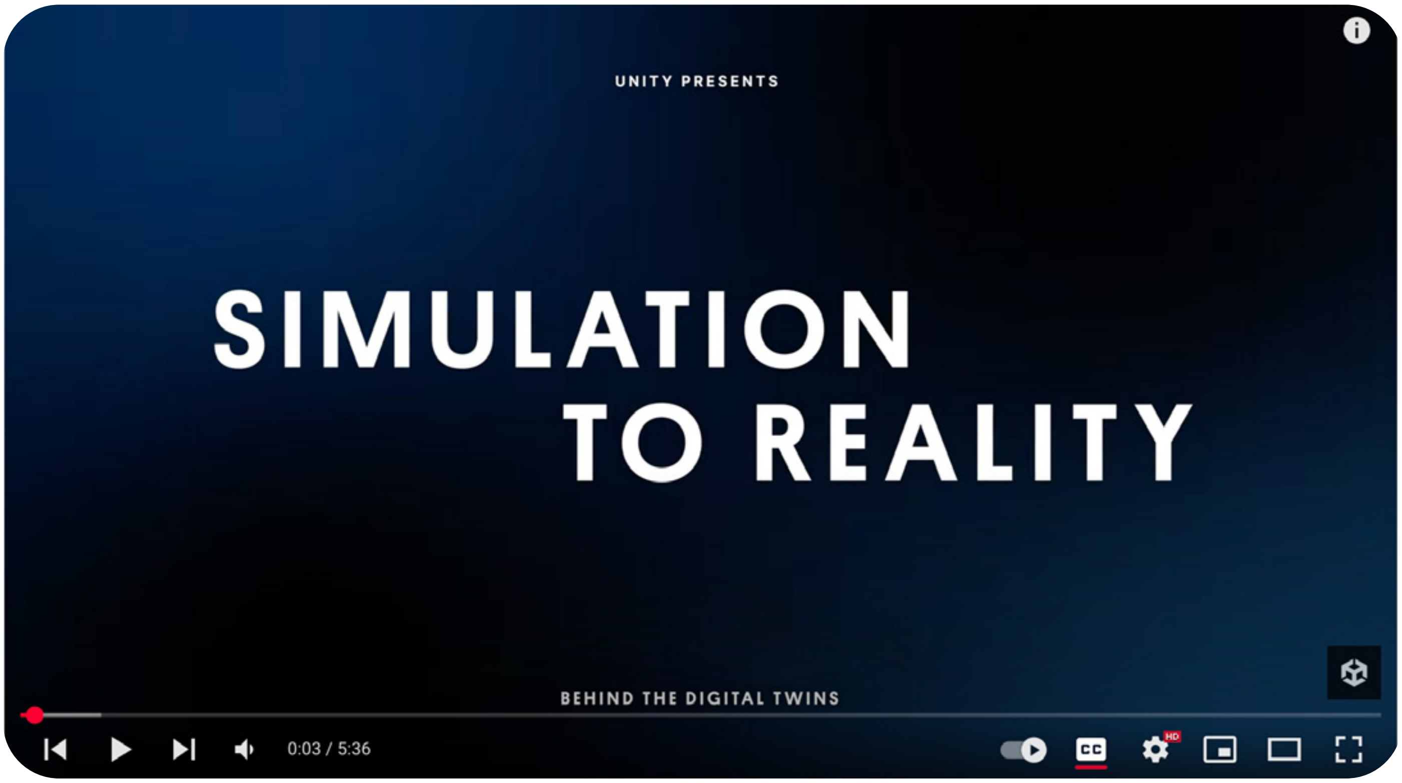
2. Beyond vision: Predictive models and optimization

AI in digital twins is not limited to image processing. Predictive maintenance, energy optimization, and production planning all benefit from AI models trained on historical or simulated data. These models can forecast failures, detect anomalies, or suggest more efficient machine settings.

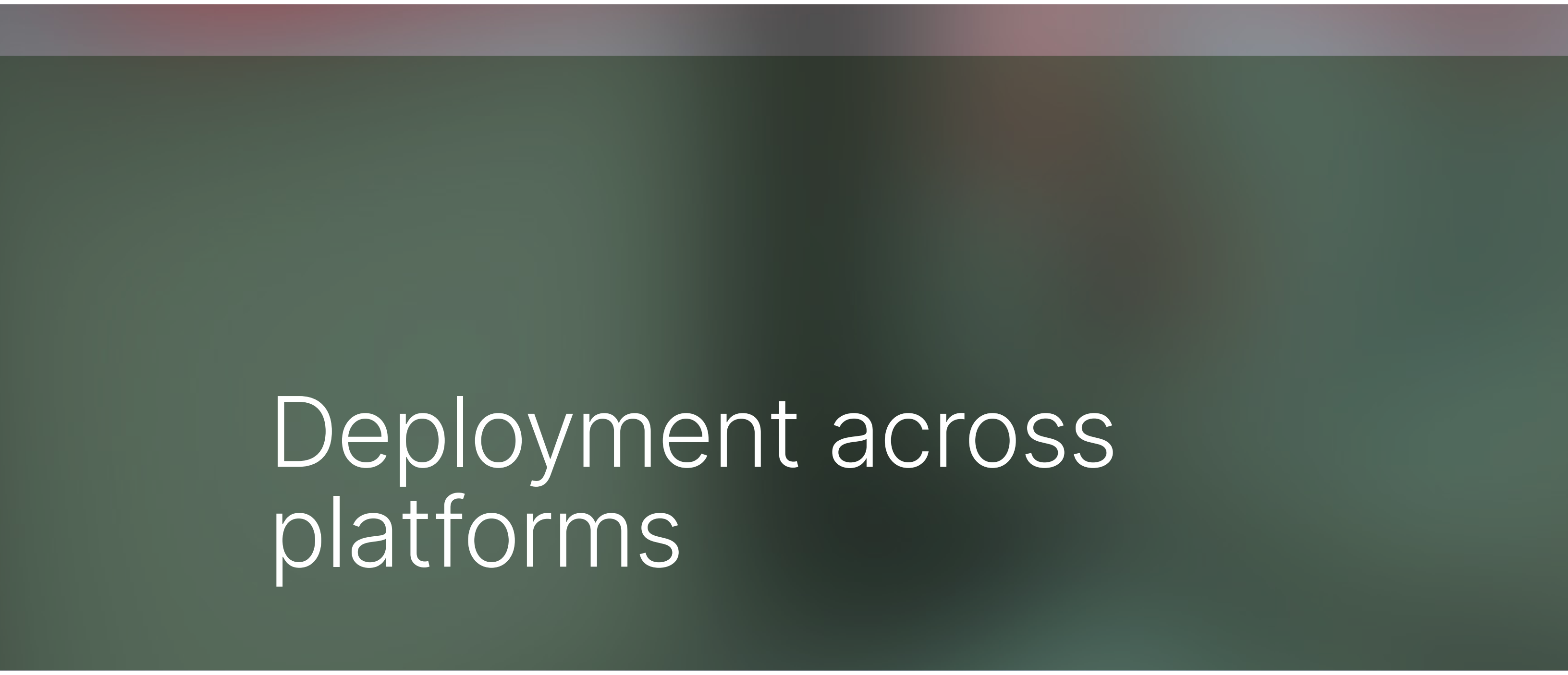
Unity can be used to visualize and interact with these predictions in real-time. A predictive model trained in Python, for example, can run in the background and send alerts or optimization suggestions into the Unity twin, where users can understand and act on them in context.

3. Unity as an AI-ready platform

What makes Unity especially powerful is its ability to host AI logic directly or serve as a bridge to external AI frameworks. Thanks to Unity's C# API and support for ONNX via Unity Sentis, Unity can run AI inference embedded in the simulation or act as a flexible frontend to cloud-based or edge-deployed inference systems.



From simulation to reality: improving safety and operations with digital twins



Deployment across platforms

One of Unity's most powerful advantages in industrial use is its ability to deploy digital twins across a wide range of platforms. A Unity-based simulation can run just as easily on a Windows desktop as it can on a Linux-based industrial PC, an Android tablet on the shop floor, a WebGL browser interface, or a standalone AR/VR headset.

This multi-platform support empowers a key strategic principle: **build it once, use it multiple times**. A digital twin developed for engineering review can later serve as a training tool, a sales visualization, a virtual commissioning environment, or even a live 3D HMI—all without needing to rebuild the core logic and structure. Reusability across devices and departments drastically reduces development cost and increases ROI.

Industrial digital twin – Connected to the automation controller in a multiuser AR session. Image courtesy SEW-Eurodrive and Realvirtual.io

This is a critical insight for management: when investing in digital twins, it's essential to avoid siloed, one-off solutions that cannot scale or adapt. Encouraging a unified platform strategy—one that supports reuse and interoperability—creates long-term value across the entire organization.

Unity's build system allows the same project to be exported as a standalone application for:

— Windows, Linux, macOS

— Android and iOS

— WebGL (browser-based deployment)

— XR platforms (Meta Quest, HoloLens, HTC Vive, Varjo, Apple Vision)

— Edge or embedded systems (via IL2CPP builds)

OEMs and solution providers can turn their digital twins into commercial products—embedding them into customer offerings, integrating licensing models, or distributing them as run-time environments tailored to specific machines or workflows. With full control over the codebase, branding, data handling, and feature scope, companies can align the digital twin exactly with their value proposition.

In short, Unity doesn't lock you into a vendor-driven ecosystem. Instead, it gives you the tools to create, scale, and commercialize your digital twin strategy—on your terms.

Lessons learned and common pitfalls

Building digital twins in Unity opens up tremendous opportunities—but it also comes with real-world challenges. Over time, teams working on industrial projects consistently encounter similar roadblocks. This section distills those insights into practical takeaways for decision-makers, developers, and system integrators.

1. Don’t overmodel — Focus on what matters

One of the most common mistakes is overmodeling: spending time creating highly detailed 3D geometry, simulating every mechanical nuance, or building full physical fidelity when it’s not needed. The goal of a digital twin is to create value—not to mirror reality in every bolt.

Instead, focus on the functional fidelity needed for the use case. A sales demo may only need approximate motion. Virtual commissioning may require precise drive behavior but no surface textures. A training system might require realistic UI and operator interactions, not perfect geometry.

Another common trap is adding colliders or rigid bodies everywhere by default—even when they’re not needed. Physics components should only be used when interaction, collision, or realistic physical response is required. Unnecessary physics objects increase CPU overhead, reduce performance, and often add instability without contributing value to the simulation.

***Pro tip**: Ask yourself: what problem does this digital twin solve? Let the answer guide the level of detail.*

2. Real-time ≠ frame rate

In Unity, it’s easy to think of real-time as a visual metric — 60 FPS or smooth camera motion. But in industrial automation, real-time means deterministic system behavior: fast I/O cycles, consistent update intervals, and reliable synchronization with external controllers.

Frame rate and logic rate are not the same**.** A simulation might run visually smooth at 60 frames per second, yet still fail to maintain the precision or responsiveness needed to test or operate automation logic accurately.

To ensure timing correctness, always use Unity’s fixed update loop for control logic and communication layers. Industrial communication (for example, PLC signal exchange) and behavior simulation (for example, drive responses) should be fully decoupled from the visual update loop. This allows the simulation to run with consistent, predictable time steps that match automation cycle times—typically in the 10–20 millisecond range.

This separation between rendering and logic is essential when designing Unity applications that are meant to simulate or interface with real industrial control systems.

3. Avoid siloed thinking

Digital twins live across organizational boundaries—engineering, automation, IT, training, service. Too often, Unity-based systems are built for one team without considering future use cases or system expansion. This leads to duplicated effort, conflicting data models, and wasted opportunity.

Start with a modular design, even if you’re only building a single-use system today. Use naming conventions, prefab structures, and configuration files that make reuse easier. And most importantly: collaborate across departments early.

4. Start small—but think ahead

A common and often underestimated pitfall is aiming too big, too early. It’s tempting to start with the vision of a fully integrated, lifecycle-spanning 3D digital twin. But trying to realize that all at once can stall progress, overcomplicate decisions, and delay tangible outcomes.

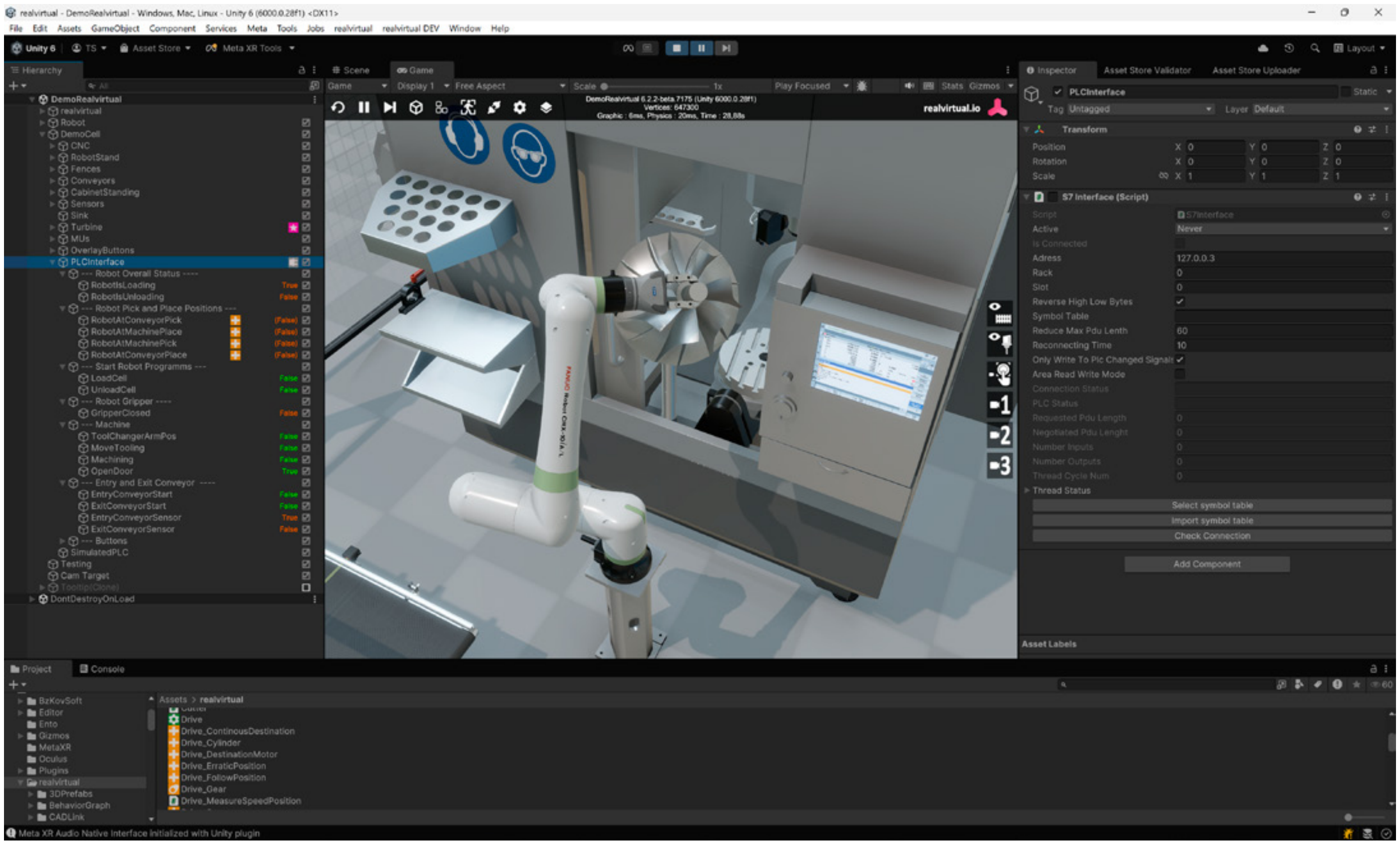
Instead, always begin with a first concrete use case—something specific, achievable, and valuable to your team. Whether it’s simulating a single robot cell, visualizing a process flow, or generating training data for an AI model, small wins build momentum and buy-in.

The key is to create fast, visible success—while laying the technical groundwork that can evolve toward a broader strategy later. Unity makes this possible: it enables fast prototyping, scalable architecture, and reuse across the digital twin lifecycle. Just don’t let the dream of a big, integrated future prevent you from delivering value today.

5. Management commitment is essential

Some of the most powerful applications of digital twins—such as virtual commissioning — require more than technology. They demand a shift in processes and mindset. Simulating a machine when it's already on the customer site is too late. The real value comes from starting earlier: testing logic, detecting errors, and validating integration long before hardware is shipped.

This kind of change requires management attention and commitment. Organizations must prioritize early simulation, allocate time for test-driven automation development, and most importantly, make their best people available to push this transition forward. Digital twins are not just a technical project—they are an organizational investment in doing things smarter and earlier.


realvirtual - DemoRealvirtual - Windows, Mac, Linux - Unity 6 (6000.0.28f1) <DX11>
File Edit Assets GameObject Component Services Meta Tools Jobs realvirtual realvirtual DEV Window Help
Unity 6
TS
Asset Store
Meta XR Tools
Layout
Hierarchy
Scene
Game
Display 1
Free Aspect
Scale
1x
Play Focused
Stats
Gizmos
Inspector
Asset Store Validator
Asset Store Uploader
realvirtual.io
Project
Console
Assets > realvirtual
Meta XR Audio Native Interface initialized with Unity plugin

DemoRealvirtual 6.2.2-beta.7175 (Unity 6000.0.28f1)
Vertices: 647300
Graphic : 6ms, Physics : 20ms, Time : 28,88s

```
DemoRealvirtual
  realvirtual
  Robot
  DemoCell
    CNC
    RobotStand
    Fences
    Conveyors
    CabinetStanding
    Sensors
    Sink
    Turbine
    MUs
    OverlayButtons
    PLCInterface
      --- Robot Overall Status ----
        RobotIsLoading                True
        RobotIsUnloading              False
      --- Robot Pick and Place Positions ---
        RobotAtConveyorPick           (False)
        RobotAtMachinePlace           (False)
        RobotAtMachinePick            (False)
        RobotAtConveyorPlace          (False)
      --- Start Robot Programms ---
        LoadCell                      False
        UnloadCell                    False
      --- Robot Gripper ----
        GripperClosed                 False
      --- Machine
        ToolChangerArmPos             False
        MoveTooling                   False
        Machining                     False
        OpenDoor                      True
      --- Entry and Exit Conveyor ----
        EntryConveyorStart            False
        ExitConveyorStart             False
        EntryConveyorSensor           True
        ExitConveyorSensor            False
      --- Buttons
    SimulatedPLC
  Testing
  Cam Target
  Tooltip(Clone)
DontDestroyOnLoad
```

```
PLCInterface                          Static
Tag Untagged                          Layer Default
Transform
Position        X 0    Y 0    Z 0
Rotation        X 0    Y 0    Z 0
Scale           X 1    Y 1    Z 1
S7 Interface (Script)
Script                                S7Interface
Active                                Never
Is Connected
Adress                                127.0.0.3
Rack                                  0
Slot                                  0
Reverse High Low Bytes                ✓
Symbol Table
Reduce Max Pdu Lenth                  60
Reconnecting Time                     10
Only Write To Plc Changed Signals     ✓
Area Read Write Mode
Connection Status
PLC Status
Requested Pdu Length                  0
Negotiated Pdu Lenght                 0
Number Inputs                         0
Number Outputs                        0
Thread Cycle Num                      0
Thread Status
[Select symbol table]
[Import symbol table]
[Check Connection]
[Add Component]
Asset Labels
```

```
BzKovSoft          Cutter
Editor             Drive
Ento               Drive_ContinousDestination
Gizmos             Drive_Cylinder
MetaXR             Drive_DestinationMotor
Oculus             Drive_ErraticPosition
Plugins            Drive_FollowPosition
realvirtual        Drive_Gear
  3DPrefabs        Drive_MeasureSpeedPosition
  BehaviorGraph
  CADLink
```

Demo scene in realvirtual.io Starter – available for free on the Unity Asset Store

The Unity ecosystem for industry

Digital twins are more than just technology—they rely on tools, people, and shared knowledge to succeed. One of Unity's biggest advantages is the size and strength of its ecosystem. From the Asset Store to third-party plugins, from open-source libraries to enterprise support plans, Unity provides a flexible and well-supported environment for industrial applications.

1. **Access to development resources**
   Unity has one of the world's largest developer communities. That means it's easy to find tutorials, documentation, sample projects, and answers to technical questions. Whether you're integrating a PLC, creating a robot simulation, or optimizing shaders for WebGL — chances are, someone has done it before and shared it.

2. **A growing set of industrial tools**
   Thanks to Unity Industry and packages like Unity Asset Transformer, industrial developers now have access to high-quality CAD importers, mesh optimizers, and data management tools directly inside the Unity editor. These tools simplify workflows and reduce the barrier between mechanical design and interactive simulation.

3. **Plugins and frameworks**
   Third-party frameworks like realvirtual.io extend Unity's industrial capabilities even further. With realvirtual.io, you can get started for free using the starter version, and upgrade to the professional edition when needed. The toolkit provides ready-to-use components for defining kinematics, modeling drives and sensors, and connecting to all major PLC brands including Siemens, Beckhoff, Rockwell, and others. This allows users to go from CAD to fully integrated simulation — fast, flexible, and scalable.

4. **Asset management and reuse**
   Unity includes powerful prefab and asset management systems. Combined with packages like Addressables or external solutions, teams can manage large libraries of reusable machines, devices, and control logic. For enterprise-scale workflows, Unity now offers Asset Manager—a cloud-based system designed to securely store, organize, version, and distribute 3D content at scale across global teams. This is especially useful for managing product variants, localized assets, or shared simulation components across multiple divisions and sites.

5. **Custom workflows and automation**
   Because Unity is fully scriptable and extendable, companies can build their own workflows, UI systems, editors, and automation logic on top of the engine. Unity doesn't force you into a black box—it gives you the foundation to create your own tools and pipelines that match your exact project needs.

Getting started – Build your first digital twin today

TUTORIAL 13
Getting Started with realvirtual 6 & Unity 6
Industrial Simulation & Virtual Commissioning Basics
0:01 / 23:43 · Introduction

PLCInterface
Tag Untagged Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
S7 Interface (Script)
Script S7Interface
Active Never
Adress 127.0.0.3
Rack 0
Slot 0
Reverse High Low By ✓
Reduce Max Pdu Len 60
Reconnecting Time 10
Only Write To Plc Chi ✓

Tutorial 13-Getting started with realvirtual & Unity 6: Installation & industrial simulation basics

Getting started with Unity for industrial digital twins doesn't need to be complicated. With the right tools, you can explore real-time simulation, behavior logic, and PLC integration— without the need to build everything from scratch.

A simple and effective entry point is the realvirtual.io starter package, available for free on the Unity Asset Store. It includes a complete demonstration cell with everything preconfigured: drive components, behavior simulation models, sensor logic, and a working Siemens S7 PLC interface. You can run the scene, examine how the system behaves, and simulate real industrial logic within Unity—all in one place.

This gives you an immediate way to understand how a digital twin works from end to end. It's ideal for experimenting, testing PLC code virtually, or presenting the concept of digital twins to internal teams. You can interact with the model, observe signal flows, and see the effects of behavior changes—without needing to import CAD or set up complex integrations yourself.

Once you're familiar with the starter setup, there are two natural directions to go next. You can upgrade to realvirtual.io professional, which unlocks advanced features such as Beckhoff, OPC UA, Fanuc and many more interfaces. Or, you can use what you've learned to build your own digital twin system from the ground up—tailored to your specific use case, using Unity's full flexibility and scripting environment.

What matters most is to get started. And with the tools already available, you can begin today — and grow from there.

Authors

Thomas Strigl has 18+ years of experience in creating and distributing simulation and commissioning software solutions. He recognizes that the modern robotics and machinery industry requires innovative solutions. This is why Thomas founded realvirtual.io in 2018, with the aim to pursue opportunities to embrace gaming technology to solve the challenges of complex automation systems.

Thomas Pospiech has over 20 years of industrial experience in automation technology. Over the past 15 years, he has specialized in the integration of industrial robots in mechanical engineering and finding optimized algorithms for mechatronic systems. After working as Engineering Director at OPTIMA pharma GmbH until 2016, he was appointed Professor of Automation and Control Technology at Heilbronn University of Applied Sciences. He believes digital twins, especially in mechanical engineering, can deliver added value for systems and machines.

Unity®

unity.com