POLITECNICO
MILANO 1863

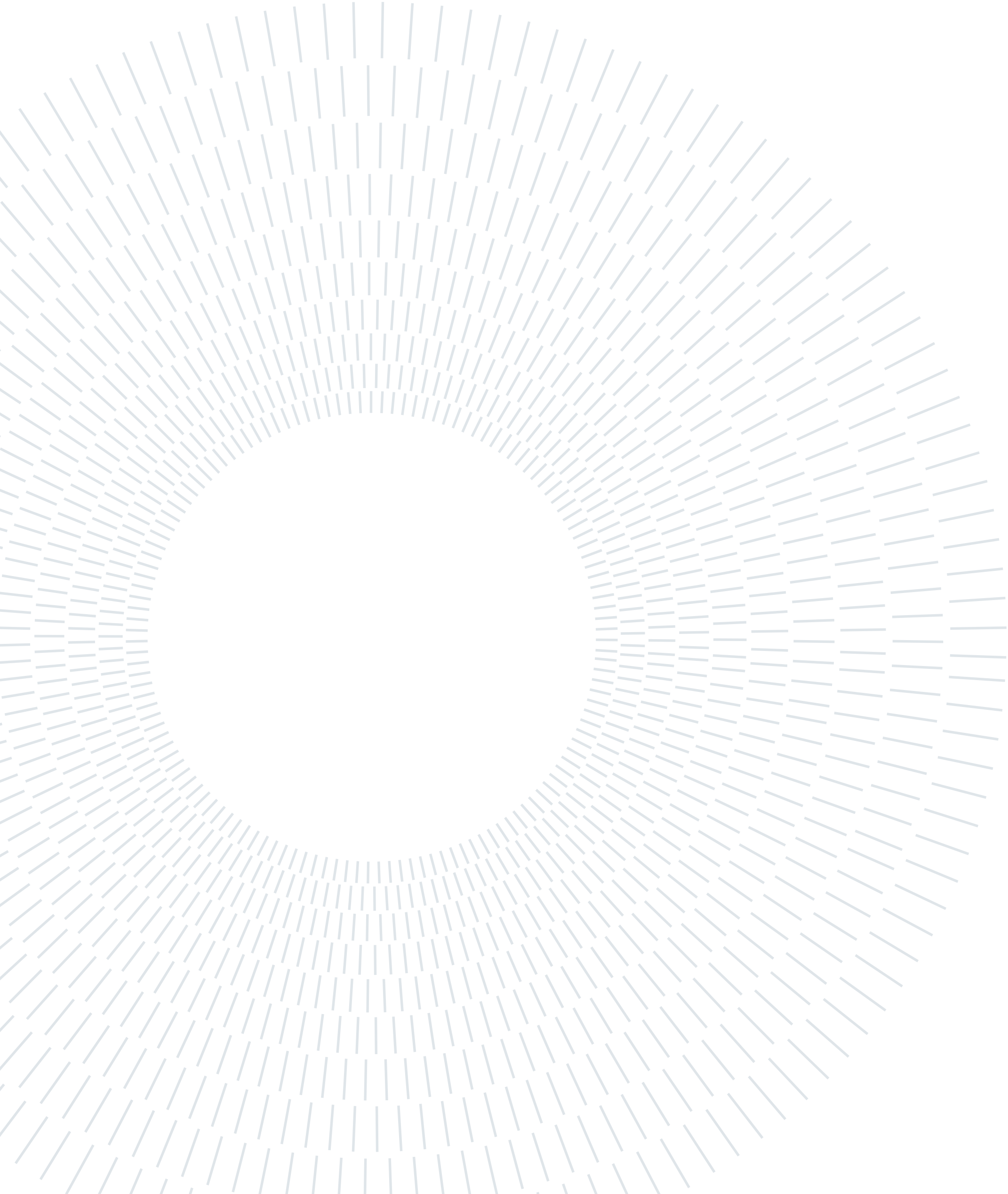
SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Designing User Interactions in Virtual Reality Environments for Industrial Digital Twins

TESI DI LAUREA MAGISTRALE IN
MANAGEMENT ENGINEERING - INGEGNERIA GESTIONALE

Author: **Sergio Benedet**

Student ID: 245781
Advisor: Prof. Marcello Urgo
Co-advisors: Dr. Walter Terkaj
Academic Year: 2024-25



Abstract

The ongoing digital transformation of manufacturing systems, within the Industry 4.0 paradigm, calls for innovative educational approaches that enable industrial engineers to interact with cyber-physical environments governed by real-time logic, automation, and human-machine interfaces. Conventional, theory-driven teaching methods struggle to convey the operational complexity and contextual dynamics of such systems, particularly when physical access to industrial machinery is not available.

This thesis presents the design, implementation, and validation of a modular framework focused on immersive industrial interactions in Virtual Reality (VR). The objective is to support engineering education through realistic and operationally coherent *Digital Twin* environments, where users can interact with virtual machines and control systems as active participants. Central to the proposed approach is a component-based architecture in which each interactive asset is driven by a *Finite State Machine* (FSM), defined externally via JSON configuration files. This separation between behavioural logic and visual representation allows for high modularity and flexible reuse across scenarios.

The framework was validated within a simulated Pick & Place manufacturing cell, designed to replicate realistic industrial workflows and immersive interaction modalities. This environment served as the main testbed for validating the framework, enabling structured assessment of usability, responsiveness, and educational effectiveness through immersive sessions conducted with standalone VR headsets.

Experimental validation demonstrated that the proposed framework effectively supports meaningful interaction, operational realism, and educational engagement. Users highlighted the clarity of system feedback, the intuitiveness of control mappings, and the perceived sense of agency within the virtual environment. The adoption of a finite state machine (FSM) architecture enabled modular behaviour definition and seamless integration of new components, ensuring the scalability of the system across diverse industrial training scenarios.

Keywords: Virtual Reality, Immersive Interactions, Finite State Machines, Industrial Engineering Education, Digital Twin, Immersive Training

Abstract in lingua italiana

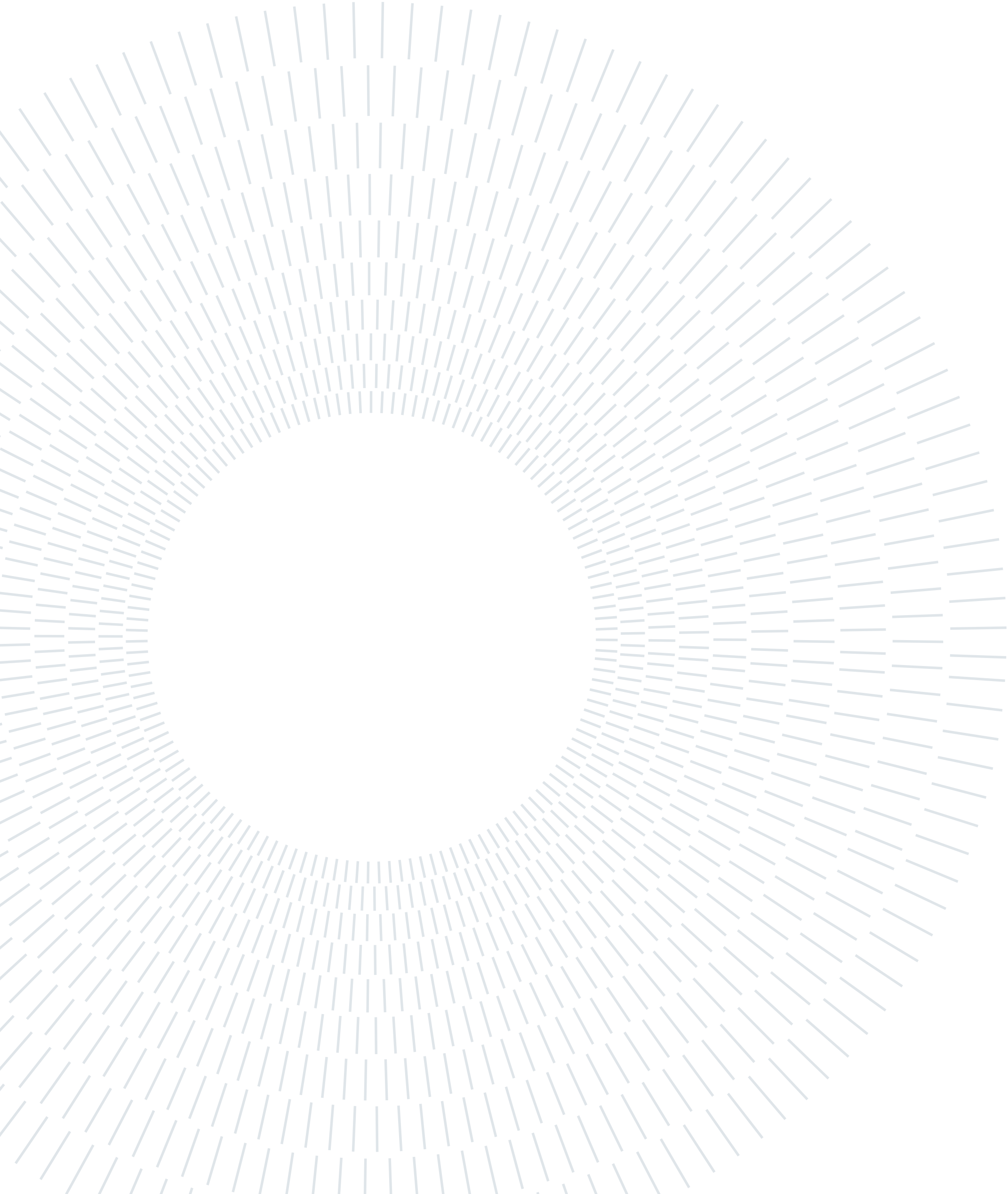
La trasformazione digitale in atto nei sistemi manifatturieri, nell'ambito dell'Industria 4.0, richiede approcci educativi innovativi che permettano agli ingegneri industriali di interagire con ambienti cyber-fisici governati da logiche in tempo reale, automazione e interfacce uomo-macchina. I metodi didattici convenzionali, basati prevalentemente sulla teoria, faticano a trasmettere la complessità operativa e le dinamiche contestuali di tali sistemi, specialmente in assenza di accesso diretto ai macchinari industriali.

Questa tesi presenta la progettazione, l'implementazione e la validazione di un framework modulare per interazioni industriali immersive in Virtual Reality (VR). L'obiettivo è supportare la formazione ingegneristica attraverso ambienti *Digital Twin* realistici, in cui gli utenti possano interagire con macchinari virtuali e sistemi di controllo, assumendo il ruolo di operatori attivi. L'approccio adottato si basa su un'architettura modulare in cui il comportamento di ciascun asset interattivo è regolato da una *Macchina a Stati Finiti* (FSM), definita esternamente tramite file JSON. Questa separazione tra logica comportamentale e rappresentazione visiva consente elevata modularità e riutilizzo in diversi scenari.

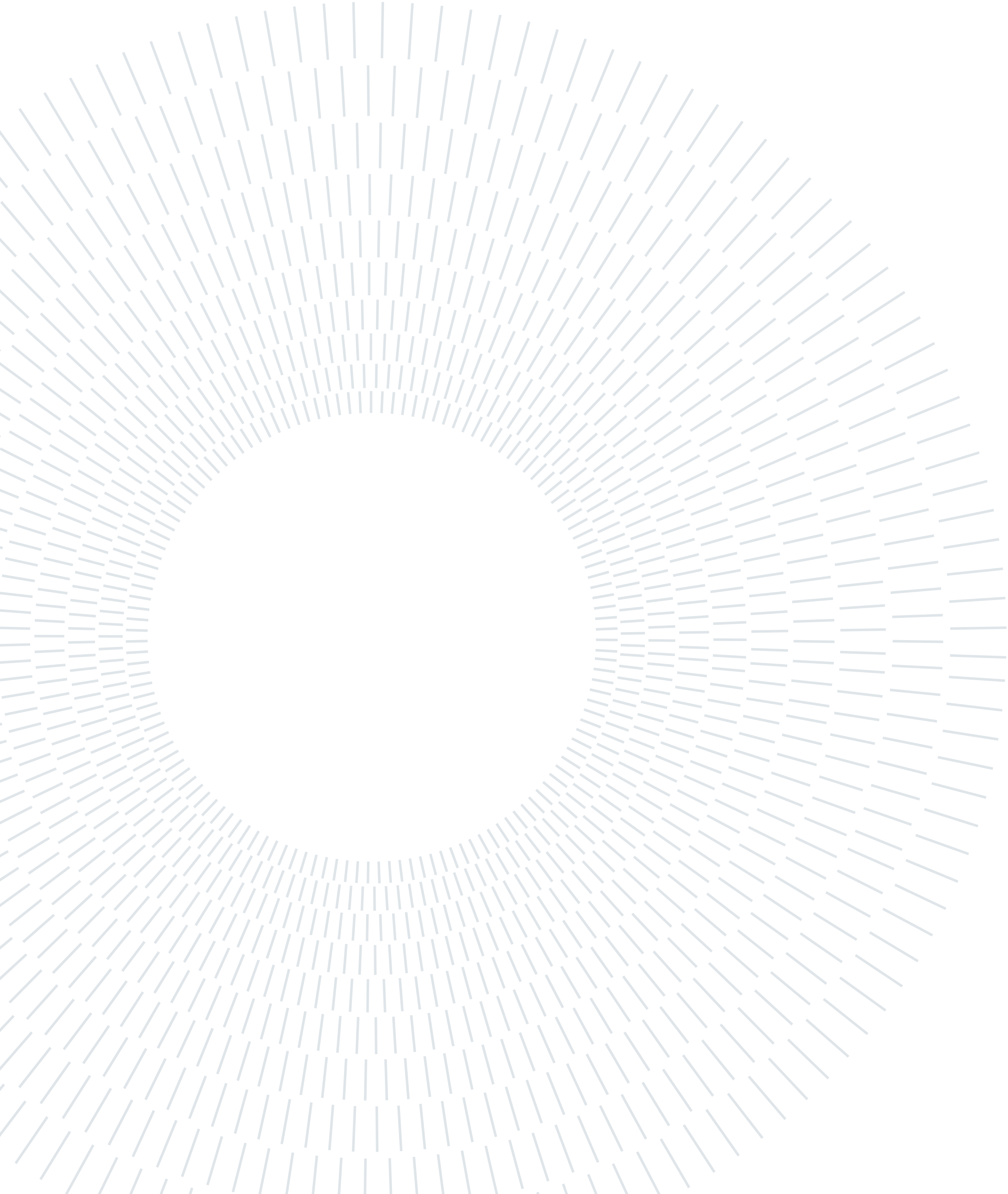
Il framework è stato validato all'interno di una cella di produzione *Pick & Place* simulata, progettata per replicare flussi di lavoro industriali e modalità di interazione immersiva. Questo ambiente ha rappresentato il banco di prova per la verifica del sistema, permettendo una valutazione strutturata di usabilità, reattività ed efficacia educativa, tramite sessioni immersive condotte con visori VR.

La validazione ha confermato l'efficacia del framework nel promuovere un'interazione significativa, un realismo operativo coerente e un solido coinvolgimento formativo. Gli utenti hanno apprezzato la chiarezza dei feedback di sistema, la naturalezza dei comandi e la sensazione di controllo. L'uso di FSM ha facilitato la definizione di comportamenti modulari e l'integrazione di nuovi componenti, garantendo la scalabilità del sistema in contesti formativi industriali.

Parole chiave: Realtà virtuale, Interazione immersiva, Macchine a stati finiti, Formazione per Ingegneria industriale, Gemello digitale, Apprendimento immersivo



Contents



1 | Introduction and Problem Statement

Training industrial engineers requires the development of both technical and soft skills, preparing future professionals to deal with complex manufacturing environments, critical decision-making processes, and the integration of advanced technologies. As manufacturing systems evolve under the paradigm of Industry 4.0, an industrial transformation driven by cyber-physical systems, interconnected assets, and real-time data management, engineers must be able to understand, configure, and operate highly dynamic production systems. This paradigm shift has introduced new concepts such as the *Digital Twin*, a digital replica of physical assets that enables simulation, monitoring, and optimization of industrial processes in real time. Within this context, education must not only convey theoretical foundations, but also prepare students to interact with smart systems, interpret feedback, and make informed decisions based on digital representations of industrial environments.

Traditional educational methods, however, often struggle to keep up with the increasing complexity of operational contexts and the tools used to manage them. Access to physical machinery may be limited, and the ability to safely explore process dynamics or system logic is constrained by cost, logistics, and safety concerns. In this scenario, immersive learning technologies, especially *Virtual Reality* (VR), are gaining attention as a powerful means to complement conventional teaching methods. By providing safe, cost-effective, and highly controllable environments, VR enables students to explore simulated scenarios that closely replicate real-world industrial systems. These virtual environments allow hands-on experimentation, system exploration, and the testing of control logic without the constraints imposed by physical laboratories, such as safety regulations, equipment availability, or spatial limitations.

The integration of immersive simulations into engineering education is not merely a technological enhancement, but a shift toward more interactive and engaging learning paradigms. Immersive tools can improve student engagement, support the development of problem-solving skills, and foster a deeper understanding of abstract concepts by making them tangible and actionable. Furthermore, they are particularly useful for training human–machine interaction, a domain that is central to modern digital factories, where operators must interface effectively with machines, sensors, and control systems. In this sense, effective learning does not emerge from passive observation, but from the ability to act, receive feedback, and refine decisions through iterative and purposeful interaction.

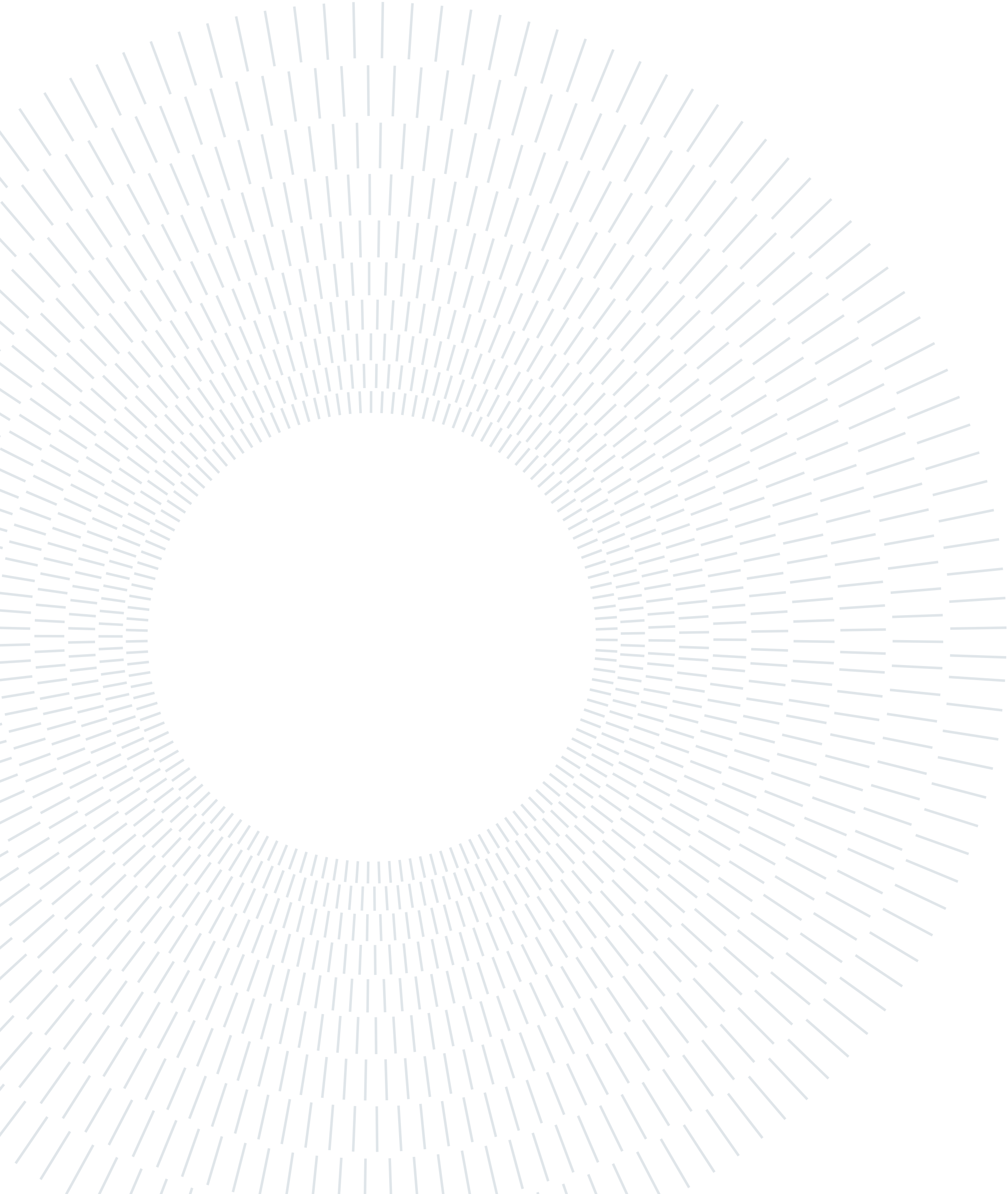
However, the effective adoption of immersive tools in education still faces several challenges: many VR training solutions prioritize graphical realism or basic navigation over realistic and flexible interactivity, overlooking the critical role of operational logic and user control in reproducing real manufacturing workflows. This limits their potential for simulating authentic industrial behavior. In real production environments, operators continuously interact with machinery through control systems that demand timely feedback, clear logic, and reactive behavior. These interactions are governed by operational constraints such as exclusivity between commands, conditional dependencies, and real-time responsiveness, all of which must be realistically reproduced to ensure effective simulation and learning. Capturing and replicating this interactive complexity in a modular and scalable way is essential to creating educational simulations that reflect authentic industrial practices and prepare engineers for real-world decision-making.

Among the many technological frameworks that enable immersive development, web-based solutions offer a promising alternative to traditional device-dependent software. These tools provide a greater level of accessibility and compatibility, allowing applications to run across different platforms and hardware configurations without installation overhead. Nevertheless, to be effective in industrial education, immersive solutions must combine ease of access with high levels of realism and adaptability. Designing meaningful interactive experiences that can be easily configured and reused remains a key challenge, especially when the goal is to simulate the dynamic logic of complex systems and allow students to interact with them directly.

This thesis contributes to this field by proposing and validating a configurable system of interactions designed for immersive training and learning in virtual factory environments. The work is framed within the broader objective of supporting engineering education through the development of modular and reusable Digital Twin components that respond to realistic operational logic. The focus is on defining a generalizable interaction framework in which users can manipulate virtual controls, activate state-driven behaviors, and explore the consequences of their actions in a safe but realistic industrial context. The proposed solution is structured around reconfigurable components, modeled through declarative logic and finite state machines, which enable consistent behavior across different scenarios while supporting flexible adaptation. By decoupling control logic from implementation, the system ensures both pedagogical relevance and technical scalability, bridging the gap between immersive experience and engineering rigor. Ultimately, this work aims to support a new generation of immersive learning environments in which users are not passive observers but active agents, engaging with systems, controlling operations, and exploring industrial logic through direct, meaningful interactions.

The following chapters are organized to guide the reader through the conceptual, technical, and applicative dimensions of the work:

- **Chapter 2** reviews the state-of-the-art on VR applications in education and industry, with a focus on interaction methods, available frameworks, and relevant hardware.
- **Chapter 3** outlines the problem definition, the specific objectives and the requirements that shaped the project.
- **Chapter 4** describes the architectural design and conceptual modeling of the interactive components developed during the thesis work.
- **Chapter 5** illustrates the implementation of the interactions and their flexible instantiation, with behavior and logic configurable via spreadsheet-to-JSON workflows.
- **Chapter 6** presents the application of the developed actuators in a practical industrial Pick & Place virtual scenario and the results of the functional validation.
- **Chapter 7** summarizes the findings, discusses technical limitations, and outlines potential future developments.



2 | State of the Art and Technical Background

This chapter provides an overview of the state-of-the-art in the field of Virtual Reality applied to industrial and educational training, with particular attention to interaction methods, development frameworks, and devices. The aim is to highlight the background and technological rationale behind the solution proposed in this thesis.

2.1. Current Applications of VR in Industrial and Educational Contexts

Virtual Reality has emerged as a powerful tool in both industrial and educational training contexts, offering immersive and interactive experiences that enhance learning effectiveness, engagement, and skill acquisition. Its adoption is steadily increasing, driven by the need to provide hands-on training opportunities that are safe, repeatable, and scalable, particularly in domains where real-world practice is constrained by cost, risk, or logistical limitations.

In higher education, VR-based platforms are used to replicate laboratory settings, simulate complex engineering processes, and provide contextualized learning experiences. For example,[35] demonstrated that after undergoing a VR-based training focused on the production process of DME (Dimethyl Ether), chemical engineering students significantly improved their operational knowledge and practical skills. The average test score increased from 46 to 66.8 out of 100, especially in topics related to chemical equipment and procedures.

Similarly, [23] noted that 65% of participants in a virtual power plant tour reported an improved understanding of operational procedures.

Practical skill development is another key area where VR has shown great promise, particularly for novice users. [27] found that students using immersive VR outperformed peers in both accuracy and execution speed on mechanical tasks. In a different study focused on welding, [26] reported a 41.6% increase in certification rates for students trained via VR-integrated systems compared to traditional methods.

In industrial contexts, VR is employed for assembly, maintenance, and safety training. [11] demonstrated the use of a VR/AR platform to train actuator assembly, observing improved task performance and reduced error rates. Multi-user VR platforms, such as those described by [13], enable remote, collaborative training for hydraulic maintenance.

Despite its potential, the widespread adoption of VR faces several challenges. These include the cost of hardware and infrastructure, lack of academic preparedness, uncertainty about pedagogical effectiveness, and the limited availability of suitable content and support structures within higher education institutions [8]. Additionally, developing high-quality, purpose-aligned VR content remains resource-intensive, as it requires not only careful design of immersive educational scenarios but also the integration of configurable assets, real-time interactions, and dynamic system behaviors within a unified framework [31]. Early research highlighted the importance of reconfigurability and data-driven customization, emphasizing the need for modular, ontology-based tools, which support layout design, animation, and seamless integration with other manufacturing applications through standardized semantic structures [30].

[24] demonstrated the effectiveness of Virtual Reality in Lean Manufacturing education through a quasi-experimental study. Students using a VR-based simulation showed a significant 7.5 point improvement in post-test scores and reported high engagement, ease of use, and emotional immersion, highlighting the value of immersive environments in enhancing learning outcomes.

Frameworks such as the Design, Play, and Experience (DPE) model [9] and its industrial extension DPE-IE [31] provide structured methodologies for aligning immersive technology design with educational objectives. The DPE model emphasizes the integration of pedagogical intent, interactive mechanics, and experiential depth, offering a balanced foundation for developing serious games and VR training environments. Building on this, DPE-IE adapts these principles to industrial education, incorporating domain-specific requirements and supporting modularity and reconfigurability. This approach ensures that immersive experiences are not only engaging but also tailored to promote effective skill acquisition and knowledge transfer in complex technical domains.

In conclusion, VR supports cognitive and procedural learning in engineering by enhancing engagement, retention, and skill transfer, while enabling safe experimentation in high-risk or inaccessible scenarios.

2.2. Current Immersive Interaction Techniques in VR

In immersive Virtual Reality (VR) systems, interaction techniques are fundamental to enabling meaningful engagement with virtual environments and objects. Particularly in industrial and engineering training contexts, the ability to manipulate components, activate controls, and interact with machinery via intuitive methods is critical to the realism and effectiveness of simulation-based learning. This section surveys the principal interaction techniques employed in VR and discusses their configurability in relation to object manipulation and control activation, essential capabilities for designing interactive industrial components.

Among the most prevalent methods for object selection is ray-casting, which projects a virtual ray, typically from a controller or hand, to point and select elements in the environment. This technique is widely adopted due to its speed and intuitive aiming capabilities, especially for large or nearby objects. However, it exhibits decreased accuracy when applied to distant or small targets, and benefits greatly from visual feedback mechanisms to enhance usability [6, 36].

To address ray-casting's limitations in fine manipulation, virtual hand techniques enable direct one-to-one mapping between a user's real and virtual hands, allowing natural interaction for nearby tasks [21]. While realistic, these approaches are inherently limited by the user's physical reach. The Go-Go technique overcomes this by dynamically extending the virtual arm, allowing interaction with distant objects through nonlinear scaling [6], although this may come at the cost of reduced precision.

Advanced strategies like PRISM (Performance-based Rate-Independent Scaled Manipulation) and Erg-O offer further refinement. PRISM dynamically adjusts movement scaling to improve accuracy in tasks requiring precision [10], while Erg-O focuses on ergonomic optimization by retargeting object positions to reduce user strain [19]. These approaches have demonstrated performance gains, but sometimes at the expense of spatial consistency or immersion.

Gesture-based interaction through hand tracking, enabled by devices such as Leap Motion, represents another intuitive modality, supporting grasping and rotation without physical controllers. Although realistic interfaces enhance user immersion, they can also present challenges in terms of usability. In particular, instability in object manipulation and inaccuracies in tracking under certain conditions can lead to user discomfort and increased task completion time [15]

In industrial contexts, controller-based inputs remain the most reliable interface for interaction with virtual buttons, switches, and sliders. Devices such as VR controllers provide robust support for both discrete actions and continuous input types, making them suitable for a wide range of immersive interactions [17]. Their consistent behavior and tactile feedback make them ideal for constructing complex virtual control panels [34].

Importantly, no single technique offers universal superiority. Rather, each method's effectiveness depends on task-specific variables, including object distance, interaction complexity, and hardware limitations. Ray-casting and virtual hands excel in local selection tasks; Go-Go aids in reaching distant targets; PRISM enhances precision; and Erg-O improves comfort in prolonged sessions. Additionally, alternative navigation strategies such as gaze-based steering have shown promising results, offering more intuitive and efficient movement in immersive environments compared to traditional techniques like map-dragging, particularly in scenarios requiring continuous spatial awareness [6].

Configurability plays a crucial role in enhancing usability, allowing interaction techniques to be dynamically tailored to specific scenarios. Parameters such as scaling factors, activation thresholds, and feedback cues can be fine-tuned to optimize performance and user experience. This adaptability is especially valuable in industrial VR simulations, where interactive logic must reflect real-world constraints, such as snapping objects to zones or toggling machine states through selectors.

Ultimately, immersive interaction in VR is most effective when approached with a hybrid and user-centered design mindset. Developers should tailor methods to match ergonomic comfort, task demands, and device capabilities. Structured taxonomies of interaction techniques, such as those proposed by [34], provide useful guidelines for selecting and integrating interaction models that balance usability, immersion, and efficiency, core goals in educational and industrial training environments.

2.3. Tools and Frameworks for Web-Based Immersive Development

Developing immersive and interactive digital environments for industrial or educational use cases requires a set of tools that are not only graphically performant and hardware-compatible, but also modular, extensible, and easily accessible across platforms. Web-based solutions have gained increasing attention for these purposes due to their ability to run directly in modern browsers without additional installations, offering cross-platform availability and fast deployment. While multiple tools are available for immersive development, WebXR, Babylon.js, and VEB.js stand out for their complementary features and support in building engaging experiences.

WebXR is a web standard developed and maintained by the W3C (World Wide Web Consortium), the main international organization responsible for defining web technologies and protocols [32]. It provides APIs (Application Programming Interfaces), which are standardized sets of functions that allow developers to interact with complex system components, such as VR headsets or motion controllers, without needing to manage low-level hardware instructions.

Through these APIs, WebXR enables developers to create immersive experiences directly in the browser, supporting both augmented reality (AR) and virtual reality (VR). It is compatible with a wide variety of devices, including VR headsets (e.g., Meta Quest), AR glasses, and smartphones. One of its key advantages is hardware abstraction: a single codebase can be used across different platforms, whether immersive (e.g., Head Mounted Displays) or non-immersive (desktop screens), with built-in support for spatial tracking, controller input, and environmental sensing.

Despite these benefits, WebXR adoption still faces limitations due to inconsistent browser support, varying performance across devices, and the need for specific runtime environments. To mitigate the inconvenience of testing applications exclusively through physical headsets, developers can use tools such as the WebXR Emulator Extension for Chrome. This extension simulates the behavior of headsets and controllers, allowing for faster and more convenient debugging and interaction testing directly within the browser, without the need to wear a device during development iterations[33].

Babylon.js is a powerful and widely adopted open-source 3D engine built on top of WebGL, the low-level browser standard for GPU-accelerated rendering [2]. It offers a robust set of tools to build, render, and animate 3D scenes in real time, including features such as:

- Advanced material and lighting systems,
- Physics simulation through modular engines (e.g., Cannon, Havok),
- Animation blending and skeletal rigging,
- Input and camera control systems,
- Full integration with WebXR for immersive deployment.

Its main strengths lie in performance optimization, scene expressiveness, and ease of use for developers familiar with JavaScript. Babylon.js also includes an extensive ecosystem of extensions and documentation, making it an attractive choice for developing browser-based XR applications. Nevertheless, being a low-level engine, it requires significant development effort to manage scene logic, interaction behaviors, and application structure, especially in complex simulations. For this project, version 8.10.0 of Babylon.js was used.

To address the complexity of configuring immersive environments, while enhancing scalability, modularity, and reuse, higher-level frameworks such as VEB.js (Virtual Environment based on Babylon.js) have been adopted. VEB.js is a web-based, model-driven application developed by Dr. Walter Terkaj (STIIMA-CNR) with contributions from Dr. Marcello Urgo (Politecnico di Milano), specifically designed to support configurable virtual simulations for industrial and educational use cases [29].

Built atop Babylon.js, VEB.js inherits all core rendering and interaction capabilities (e.g., lighting, physics, animation, WebXR integration), while introducing a higher abstraction layer that cleanly separates interaction logic from geometric representation. Its architecture is data-driven, meaning that:

- 3D assets (e.g., robots, conveyors, sensors) are loaded from `GLTF` models generated via CAD and mesh editing pipelines;
- Behaviors and logic are externally defined via `.json` configuration files structured around semantic ontologies;
- Interactive dynamics are implemented using finite state machines (based on statecharts), which define transitions, animations, and control actions over time.

The typical workflow involves CAD modeling (exported in formats like STEP or IGES), followed by mesh processing and rendering optimization (e.g., with OBJ or GLTF + PBR materials), and finally enriched with behavioral semantics through JSON descriptors. The result is a fully reconfigurable virtual environment, dynamically built at runtime from the fusion of geometric and logical definitions [4].

VEB.js provides several key functionalities:

- Dynamic instantiation and hierarchical placement of scene elements based on spatial coordinates and parent-child relations;
- Declarative behavior specification, through JSON-based configuration without requiring hardcoded logic;
- Centralized lifecycle management for user interface elements and 3D objects;
- Support for scene re-export, enabling updates and version control via modified JSON outputs.

A defining strength of this system lies in its semantic decoupling of form and function:

any 3D mesh, regardless of its visual origin, can be dynamically assigned a functional role through its type and associated statechart configuration. This approach allows developers to reuse visual assets across multiple functional roles, promoting consistency and minimizing development effort.

Furthermore, thanks to its Babylon.js foundation, VEB.js enables advanced rendering techniques (e.g., physically-based materials), immersive navigation through WebXR, and interactive manipulation of objects. As a web application, it supports seamless cross-platform execution: it runs directly in any modern WebGL-compatible browser, whether on a PC, tablet, smartphone, or VR headset.

VEB.js bridges the gap between conceptual modeling and technical implementation, offering a powerful infrastructure for building scalable, maintainable, and interactive digital twins and virtual learning environments.

In summary, the combination of WebXR, Babylon.js, and VEB.js constitutes a comprehensive stack for web-based immersive applications, striking a balance between graphical fidelity, interaction complexity, and deployment simplicity. Their integration supports the development of scalable, maintainable, and highly interactive training environments for the manufacturing domain.

2.4. Immersive Interaction Devices: VR Headsets and Controllers

Immersive interaction devices represent the primary interface between the user and the virtual environment. In virtual reality (VR) applications, the input/output system typically consists of two key components: the head-mounted display (HMD) and the hand-held controllers.

The headset is responsible for generating the immersive experience. By projecting high-resolution stereoscopic images to each eye and tracking the user's head movements in real time (head tracking), the headset enables users to perceive the 3D scene in a realistic and dynamic manner. Some models also include passthrough functionality and environmental tracking, which enhance safety and improve interaction with the physical surroundings.

The VR controllers serve as a physical extension of the user's hands. Equipped with buttons, triggers, and motion sensors, they allow users to manipulate virtual objects, activate commands, and navigate through immersive space. Modern technologies also incorporate haptic feedback and gesture recognition, enhancing the sense of presence and the intuitiveness of interaction.

The Meta Quest 3S was used in this project, a standalone virtual reality headset released in 2024. It features high-resolution displays and a fast processor, which ensure a smooth and responsive VR experience with refresh rates up to 120 Hz [18].

To track the user's movements and hands, the headset includes several built-in cameras, four infrared and two RGB, which allow it to understand where the user is and how they move, without the need for external sensors. Inside-out tracking is especially useful in interactive simulations where the user needs to walk around and grab or press virtual objects [18].

The system is complemented by the Meta Touch Plus controllers, featuring an ergonomic design without external tracking rings. Sensors are integrated directly into the body of the controller and, in combination with the headset's cameras, provide accurate tracking. The controllers include analog triggers, interaction buttons, joysticks, and gesture support, making them well-suited for complex interactions such as triggering, grabbing, and object rotation or translation [18].

The use of the Meta Quest 3S enabled the validation of the project's immersive functionalities under realistic operating conditions, ensuring a coherent, responsive, and pedagogically effective user experience.

Figure 2.1 and Figure 2.2 illustrate the headset used and the button mapping of the Meta Touch Plus controllers, respectively.

Figure 2.2 provides an overview of the physical layout of the Meta Touch Plus controllers, highlighting the main input elements available to the user. Both controllers are equipped with analog sticks for navigation, a pair of front-facing buttons (A and B on the right controller, X and Y on the left), a trigger button on the rear, and a lateral grip button.

The analog sticks typically support two main locomotion methods in VR environments: free movement, where the user moves continuously in the direction the stick is pushed (relative to the headset orientation), and teleportation, where the user points at a spot on the ground and instantly moves there, minimizing motion sickness and offering better spatial awareness in limited physical spaces.

The A, B, X, and Y buttons function as digital inputs and only support simple press/release actions. In contrast, the trigger and grip buttons are analogical, meaning they detect different levels of pressure. This allows for more precise and context-sensitive interactions, such as simulating variable resistance or activating functions progressively.

Additionally, each controller features a system-level button: the Menu button on the left controller and the Meta button on the right, used to access application settings or system interfaces. This button configuration enables a rich variety of immersive interactions and ensures compatibility with the diverse set of user actions required by the simulation.

In the upcoming chapters, it will be shown how some of these buttons have been associated with specific code-driven functionalities to support immersive and reactive interactions. In particular, selection and activation of objects within the scene are managed through a virtual ray cast from the front of each controller as shown in Figure 2.3. This ray acts as a pointing device, conceptually similar to a traditional mouse cursor, allowing users to aim at objects, highlight them, and trigger actions through button presses. This approach ensures precision in object targeting and offers an intuitive way to interact with virtual assets, especially in scenarios that involve remote selection or activation without direct grabbing.

Figure 2.1: Meta Quest 3S VR Headset and controllers used in the project[1].

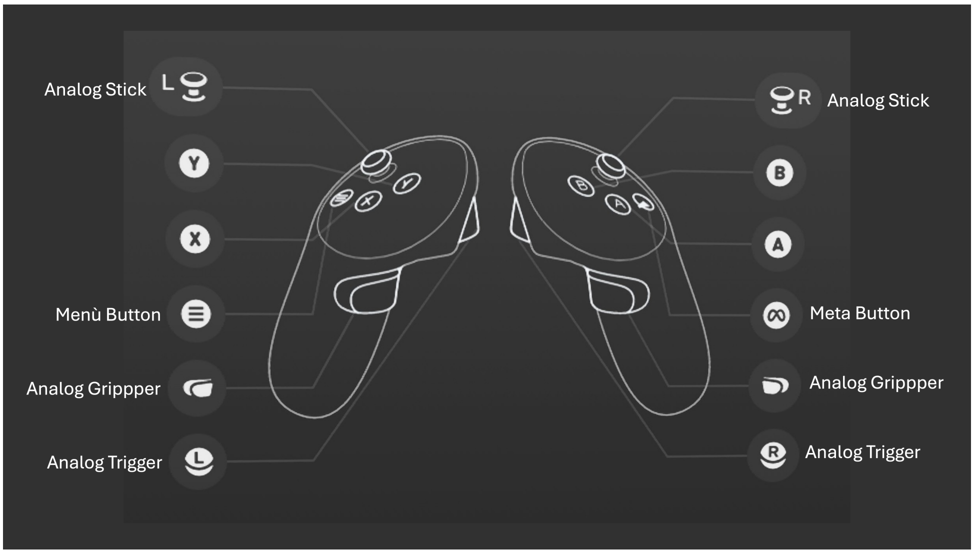


Figure 2.2: Mapping of the main input elements on the Meta Touch Plus controllers.

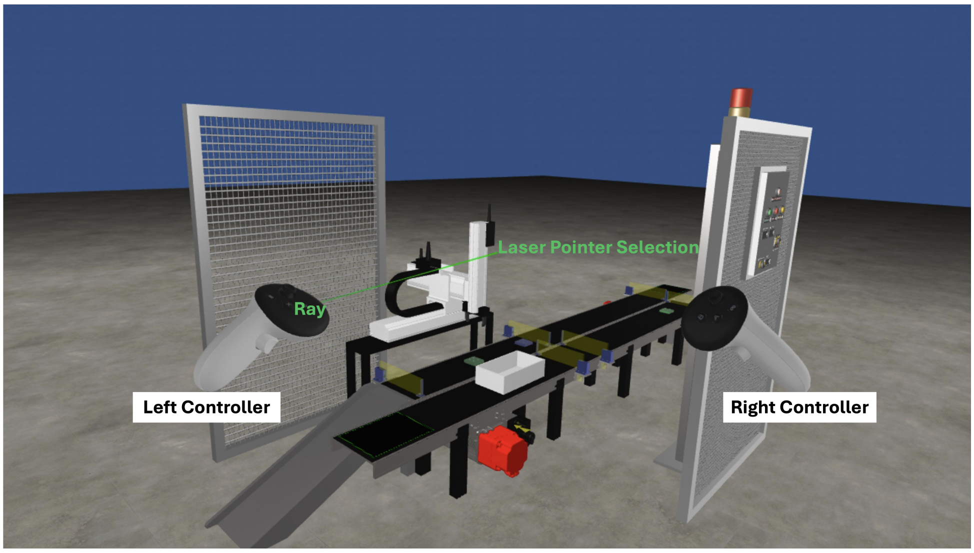
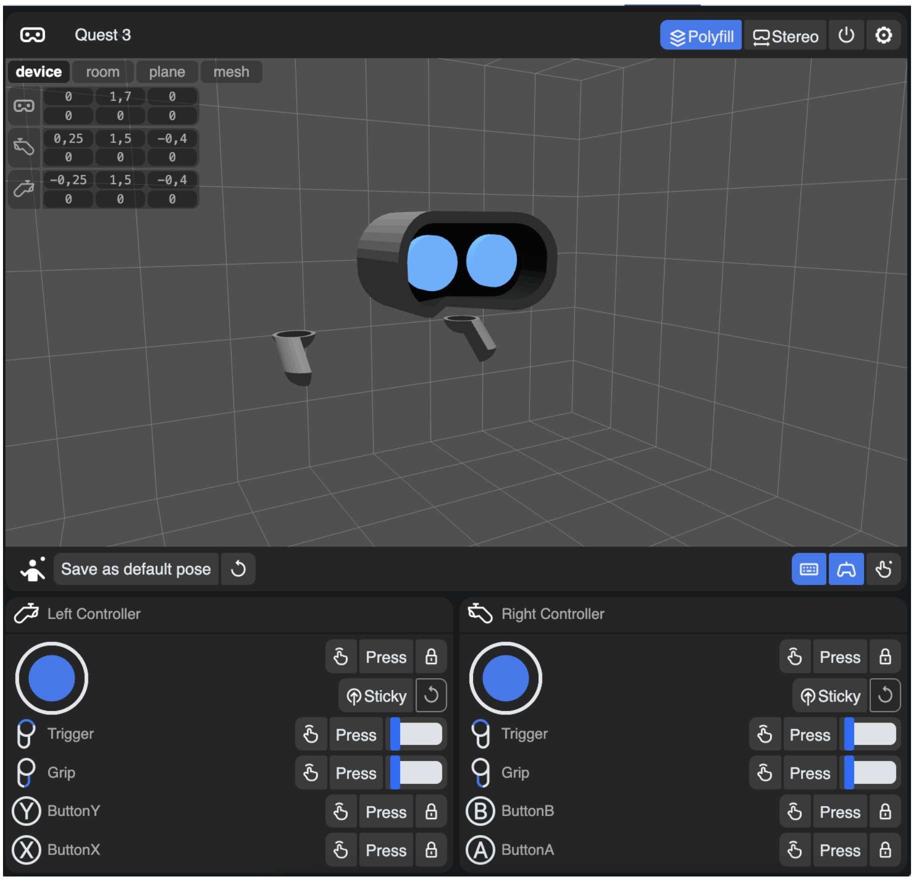


Figure 2.3: Ray-based object selection using the Meta Touch Plus controllers in the Pick & Place virtual scene.



0	1,7	0
0	0	0
0,25	1,5	−0,4
0	0	0
−0,25	1,5	−0,4
0	0	0

Figure 2.4: WebXR Headset Emulator Chrome Extension interface[33].

3 | Problem Definition, Objectives and Design Requirements

This chapter defines the industrial and educational motivations that underpin the development of the proposed VR interaction framework. It begins by identifying a critical gap in existing immersive training systems, which often prioritize visualization over realistic and flexible interaction. The chapter proceeds by formalizing the core problem addressed by this thesis, articulating the specific objectives pursued during the development process, and detailing the functional and methodological requirements that guided the system's design and implementation.

3.1. Problem Definition and Contextual Gap

Despite the growing diffusion of immersive technologies in educational and industrial contexts, as it has emerged from the literature review, many VR-based training applications still lack effective and versatile interaction methods. The focus has often remained on visual fidelity or navigation, while less attention has been given to the functional manipulation of virtual components, which is essential in simulating realistic workflows and operative tasks in manufacturing environments. This involves interacting with the virtual environment through Human-Machine Interfaces (HMIs) and realistic control elements, which are essential to accurately simulate the behavior of an industrial system.

In real production systems, workers interact with a wide range of physical elements, such as buttons, rotary selectors, lights, and mechanical devices, that require precise actions, contextual feedback, and dynamic responsiveness. However, this interactive depth is rarely mirrored in current VR training solutions, which often rely on simplified interfaces or static sequences that fail to reproduce the complex interplay between user actions and system behaviour. Even when interactive elements are present, they are frequently embedded in rigid, scenario-specific implementations, making it difficult to adapt or repurpose them for other contexts without full reprogramming. In many cases, users are exposed to pre-scripted scenes or automatic animations that simulate activity without offering direct control over objects, machines, or dynamic system elements, thus limiting true operational engagement.

This lack of modularity and configurability represents a major barrier to scalability and reusability. As a consequence, instructors and learners cannot easily modify the virtual environment to reflect different configurations or training objectives, limiting the educational potential of the platform. In a context like industrial training, where machine states, user decisions, and safety logic evolve continuously, the inability to flexibly reconfigure the interaction model poses a concrete limitation.

This challenge is particularly evident within the broader *Digital Factory* course at Politecnico di Milano.[1], led by Dr. Marcello Urgo and Dr. Walter Terkaj, which aims to develop a modular and reconfigurable platform for simulating industrial digital twins in both immersive and desktop environments. Within this vision, interaction is not a secondary or accessory feature, it represents the operative layer through which users control, test, and understand the functioning of the simulated system. Without a robust and adaptable interaction layer, the simulation risks becoming a static visualization rather than a dynamic and exploratory tool for learning and experimentation.

The present thesis arises within this framework, identifying a contextual gap in the availability of immersive interaction methods that are flexible, realistic, and easily configurable. While this work does not seek to replace industrial simulation platforms, it contributes to the development of a didactic and prototypical environment, where interaction becomes a vehicle for learning, experimentation, and system validation.

3.2. Thesis Objectives and Proposed Contribution

This thesis contributes to the broader development of immersive Digital Twin platforms by addressing a specific and practical challenge: how to enable configurable and responsive low-level control within a virtual manufacturing cell. In this context, low-level control refers to the logic that governs the behaviour of fundamental industrial components, such as actuators, signal devices, and manipulable elements, ensuring their reaction is coherent, dynamic, and adaptable to real-world operational logic.

The project focuses on developing an interaction framework that is both modular and reconfigurable, enabling the flexible simulation of industrial scenarios. Each component

---

[1] See course page [20]:

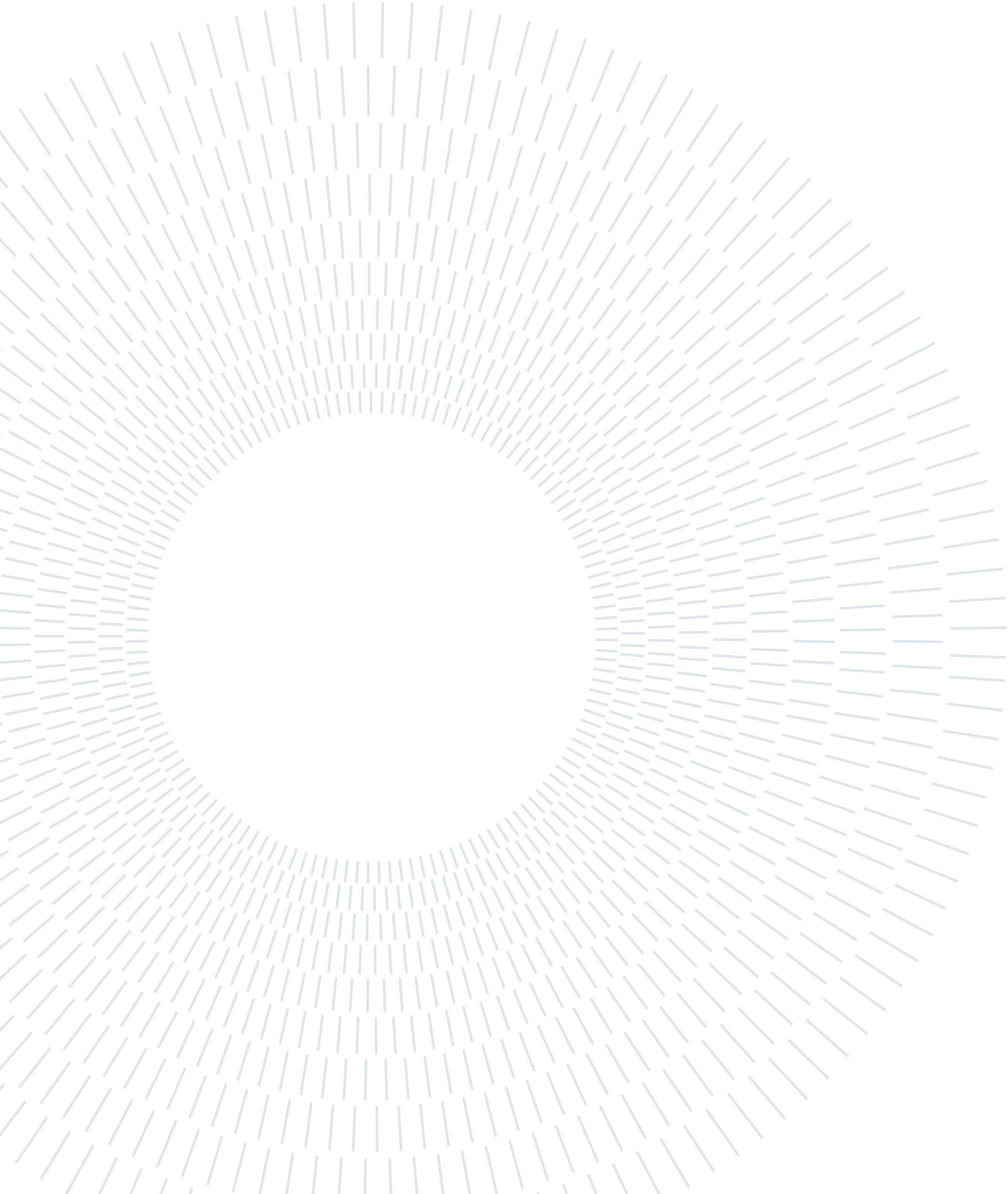
within the virtual environment is governed by external JSON configuration files, which define its behaviour through declarative state-based logic. This architecture separates behaviour definition from the executable code, promoting reuse, extensibility, and rapid adaptation across different use cases, without the need for direct code modifications. At the same time, the framework supports both desktop and VR deployment, allowing users to explore, operate, and understand the system through direct interaction rather than relying on pre-scripted sequences.

This paradigm is valuable in the context of industrial engineering education, where future professionals benefit from realistic, hands-on engagement with complex systems. By interacting with configurable control logic in a safe and immersive environment, engineering students can deepen their understanding of operational principles and gain practical experience that closely mirrors real industrial conditions. In this setting, the user is positioned not as a passive observer, but as an active operator who engages with the digital environment in a purposeful and meaningful way.

To support this general objective, the system was developed with the following specific goals:

1. **Design and integration of configurable interactive components** that emulate real-world operational interfaces, enabling users to engage with virtual representations of industrial control elements in a realistic and meaningful manner.
2. **Adoption of a decoupled behavioural modelling approach**, using externally defined logic structures, such as finite state machines and declarative configuration files, to govern system behaviour in a flexible and scenario-independent way.
3. **Implementation of immersive interaction strategies** aimed at facilitating natural and intuitive user engagement, within VR contexts, through actions like selection and manipulation of assets ensuring alignment with real operational practices.
4. **Assessment the system's performance and usability** within a realistic industrial simulation scenario, to verify the consistency of interactions, the accuracy of the modeled behaviors, and the adaptability of the framework to different operational configurations.

These goals collectively define the foundation for immersive industrial interaction design.



4 | Conceptual Design of Immersive Interactions

This chapter introduces the conceptual modeling process behind the immersive actuators developed in this thesis. It outlines the methodological approach used for their design, prototyping, and refinement, and describes the structure and behavior of each component through general control paradigms. By formalizing shared interaction principles and illustrating actuator types from buttons to snapping zones, it provides a scalable and reusable logic architecture.

4.1. Methodology for State-Driven Interactions

The design and development of the interactive control system presented in this thesis are grounded in a conceptual and methodological framework that emphasizes modularity, configurability, and behavioural realism. Rather than developing interaction logic through rigid programming constructs, this approach adopts a generalizable and scalable model in which each interactive element behaves as a state-driven actor, capable of responding autonomously and meaningfully to both user inputs and environmental events.

This design philosophy aligns with the needs of industrial training, where users must engage with digital systems not as passive observers, but as active operators who influence and monitor the behaviour of machines. In this context, interactivity is not limited to triggering predefined animations or visual feedback, but becomes a structured and reactive process governed by well-defined operational logic.

At the heart of this methodology is the use of finite state machines (FSMs), which serve as the foundational model for defining the logic behind each interactive component. An FSM is a formal structure used to represent the behaviour of a system through a finite number of discrete states, each corresponding to a specific condition or mode of operation. At any given time, a component can exist in one of these defined states, such as `idle`, `active`, or `error`.

The system transitions from one state to another based on events, discrete occurrences that signal a change in condition, such as a button being pressed. These events act as triggers for transitions, which are the rules that define how and when a component moves from one state to another. Each transition may be associated with specific actions, meaning that when a transition is triggered, the system not only changes state but may also execute a behaviour, such as turning on a light, sending a signal, or animating a mesh.

To support modularity and maximize reusability, the behavioural logic of each interactive component is defined externally through JSON-based declarative configuration files. These files act as the blueprint for each FSM, specifying the list of possible states, the events that trigger transitions, and the actions to execute during those transitions. Crucially, this logic is described in a way that is completely decoupled from the visual representation of the asset, that is, the 3D mesh, material, or spatial properties of the object are not embedded in the behavioural definition.

This separation between logic and representation is a key enabler of flexibility. It means that the same FSM, such as one describing the behaviour of a binary toggle or a multistate selector, can be reused across different scenarios, simply by associating it with a new mesh or component in the scene. The FSM provides the rules of behaviour, while the mesh provides the physical embodiment; the connection between the two is established at runtime through configuration, not hardcoded logic.

For instance, a statechart designed to model a n-position selector can be mapped to various visual implementations: a rotary knob, a switch, or even a pressure gauge. All share the same behavioural logic, but differ in how they are rendered and operated by the user. This abstracted and declarative approach allows developers and educators to rapidly prototype new training experiences, modify interaction patterns, and adapt simulations to evolving pedagogical needs without rewriting core functionality.

By making behaviour modular and interchangeable, this architecture supports the creation of scalable, consistent, and easily maintainable immersive environments, an essential quality for educational applications where variability and experimentation are central to learning.

Moreover, FSMs are not designed to function in isolation: their strength lies in their ability to operate in a coordinated, event-driven ecosystem. Each interactive component is not only capable of managing its own internal state, but can also emit events or react to external ones generated by other FSMs in the environment. This mechanism enables a form of behavioural communication between elements, where changes in one component can meaningfully influence the state and actions of others.

In this model, events act as messages exchanged between state machines, representing user actions or system conditions that trigger state transitions. For example, when a virtual button is pressed, its FSM transitions to a new state and emits an event (e.g., `buttonPressed`), which is picked up by the FSM of another component, such as a conveyor belt, that interprets it and responds accordingly (e.g., transitioning to an `active` state and initiating movement). These cause-effect chains can be extended across multiple layers, allowing even complex logic to emerge from the composition of simple, reusable building blocks.

This behavioral orchestration mirrors the structure of real-world industrial systems, where subsystems are coupled and coordinated through discrete signals, PLC commands, or sensor feedback. By modelling this paradigm digitally, the framework supports the design of coherent, modular simulations in which each component behaves autonomously but contributes to a shared operational logic.

Crucially, this architecture remains fully transparent and configurable: every interaction rule is defined declaratively within the statechart files, making it easy to trace, modify, or extend the logic of the system. Students can not only use the system but also inspect and experiment with its inner workings, gaining insight into how complex behaviours emerge from well-structured interaction models.

To support this model, the system adopts a component-based architectural structure, organized around a hierarchy of reusable classes. In this context, a class represents a software template that defines the structure and behaviour of a group of similar elements, in this case, interactive objects within the virtual environment. Each class defines the properties (e.g., references, states) and methods (e.g., behaviors, interactions) that its instances should implement, ensuring consistency throughout the system.

At the base of this hierarchy lies the abstract class `ControlledAsset`, which encapsulates the essential functionalities common to all interactive components. These include the connection to the global scene controller, the reference to the associated 3D node or mesh, and, when applicable, the binding to an external statechart. This base class provides general mechanisms to subscribe to state transitions, interpret external events, and trigger actions as defined in the FSM logic. It is intentionally detached from the specific role or appearance of the object it governs, focusing instead on enabling it to behave as a state-aware and reactive entity.

Built upon this foundation is the `Actuator` class, which extends `ControlledAsset` to model any interactive element that can produce visible or physical effects in the environment, such as moving, rotating, changing visibility, or modifying physics-based properties. This class retains the reactive logic of its parent but introduces additional functionalities tailored to dynamic control, making it suitable for simulating realistic operational elements in an industrial context.

These two foundational classes form the core of the system's modular design and will be examined in greater detail in Section 5.1, where their internal structure and role in specific actuator implementations will be explored.

Each specific interactive actuator developed in the project (e.g., buttons, switches, snapping zones) inherits from this class, defining its particular response patterns while leveraging the core state machine management features. Thanks to this design, new components can be rapidly developed by extending the base logic, binding a new mesh, and configuring a statechart, without modifying any core code. This not only ensures consistency and maintainability but also promotes the development of a scalable and pedagogically reusable interaction library.

The interaction system was developed with support for both immersive and non-immersive deployment. Rapid prototyping and intermediate validation were conducted in desktop mode using keyboard and mouse with the WebXR Emulator, while full validation was performed using Meta Quest 3S headsets. This dual-path testing ensured that interaction behaviour remained coherent and responsive across modalities, allowing the same components to be used flexibly in both teaching demonstrations and hands-on VR training sessions.

Throughout development, usability and user agency were prioritized. Components were designed not only for technical correctness but also to ensure intuitive control mappings, visible effects, and smooth feedback cycles. Gestures in VR were carefully mapped to expected physical interactions, reinforcing a sense of agency and reducing cognitive load.

Finally, all components were integrated and validated within a realistic Pick & Place simulation cell, which will be detailed in Chapter 6. This application served as both a testbed and a pedagogical demonstrator, enabling the deployment of all actuators under coordinated and operationally consistent conditions. It exemplifies how the developed framework can replicate industrial procedures and support immersive training without compromising on reusability or abstraction. In this context, the user becomes an active decision-maker, manipulating controls, observing outcomes, and engaging with system dynamics in a structured, realistic, and educationally meaningful way.

4.2. Core Interactive Actuators

This section presents the primary actuators designed to support configurable and immersive interaction in the virtual environment. The focus is placed on their logical structure and behavioural patterns, described independently of the technologies used (e.g., VEB.js or Babylon.js and of any specific use case), in order to highlight their reusability across different platforms and applications.

The described actuators include discrete input elements such as push buttons and rotary selectors, as well as visual feedback mechanisms like status lights. Each actuator is conceived as a modular and declarative component, designed to be easily instantiated, configured, and integrated into diverse virtual scenes. Collectively, they represent the operational backbone of the interaction system, enabling direct user control over dynamic virtual assets. Their implementation directly supports the central goal of this thesis: improving the realism, responsiveness, and configurability of human–digital twin interactions in immersive industrial environments.

4.2.1. Button Actuator: Buttons and Discrete Behaviour

The `ButtonActuator` represents the most elementary actuator type in the designed system, conceived to replicate the function of physical push buttons that activate discrete control logic. Its conceptual model is based on a binary interaction pattern: upon user input, the actuator transitions from an initial `idle` state to a `pressed` state, executes one or more associated actions, and then returns to its original condition. This logic is governed externally through a statechart that declaratively defines transitions and associated behaviours.

The component supports structured instructions, integrating a clear mapping between physical gestures and digital state changes. Actions such as `press` and `release` are not simple visual effects, but are designed to complement the logical `trigger` event by providing a coherent physical simulation of a button press. This interaction is further enhanced by an auditory feedback, a brief synthetic click sound, which reinforces the perception of tactility. The combination of visual and auditory cues strengthens the user's sense of having performed a meaningful action, offering a realistic experience that closely mirrors real-world interaction dynamics in virtual environments.

In addition to simple triggering, the `ButtonActuator` supports more expressive behaviours. It can:

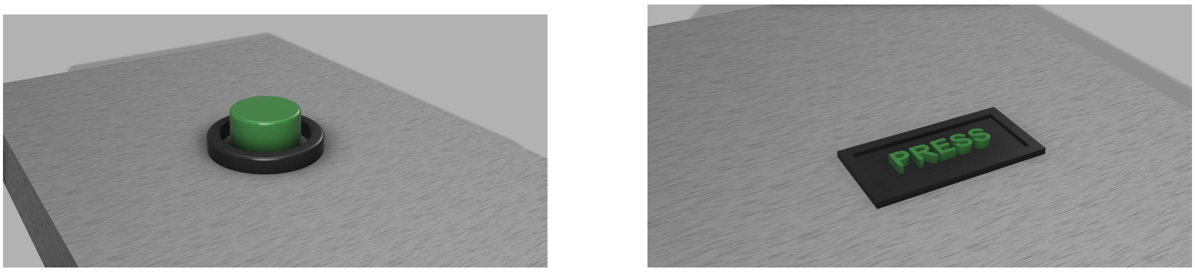
- execute sequences of actions with configurable delays,

- verify the state of other actuators before acting,
- emit periodic events to simulate recurring triggers.

These capabilities are dynamically defined through external configuration, making the component adaptable and highly reusable.

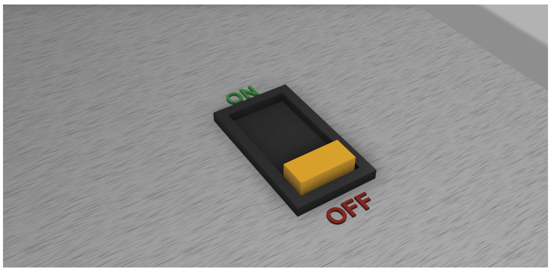
Thanks to its intuitive semantics and clear affordances, the `ButtonActuator` can be instantiated in a wide variety of contexts that require fidelity of interaction, temporal control, or visual and auditory feedback realism.

To demonstrate the flexibility and reusability of the `ButtonActuator` logic, the following figure 4.1 shows a variety of virtual buttons that can be instantiated using the same underlying class. These examples highlight how the same conceptual and programming structure can be adapted to support different designs and use cases within an XR environment. The mesh geometry, press direction, and visual styling can all be customized, while the core behaviour remains fully consistent.



(a) Simple push button.

(b) Flat button where the “PRESS” label itself is the interactive element.



(c) ON/OFF toggle switch with horizontal movement.

Figure 4.1: Examples of different button designs using the same `ButtonActuator` logic.

4.2.2. Rotary-Switch Actuator: State Selectors and Cyclic Logic

The `RotarySwitchActuator` models a rotary selector used to switch between multiple discrete operating states through cyclic interaction. Conceptually, it represents a physical knob or dial that users rotate to navigate through predefined modes in a sequential loop.

This component is characterized by a cyclic transition pattern: each interaction triggers a `next` event, causing the actuator to advance to the subsequent state in its defined statechart. The statechart determines both the sequence of logical states and the corresponding rotation values, enabling each state to be associated with a specific orientation.

The configuration supports complete flexibility:

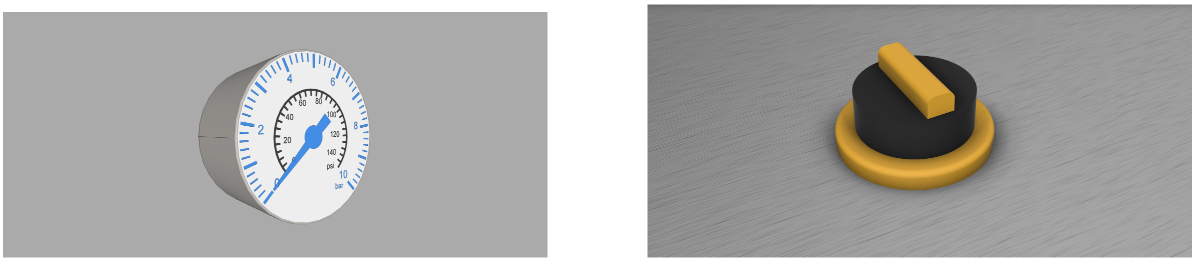
- The number of states can be arbitrary and does not require equal angular spacing.
- Rotation parameters such as axis (e.g., `x`, `y`, `z`) and direction (clockwise/counterclockwise) are customizable.
- Target angles for each state are explicitly declared, allowing for realistic or stylized selector behavior.

This design makes the selector adaptable to different geometries (e.g., circular knobs, polygonal dials) and usage scenarios. For instance, a rotary switch could rotate across five states with angles $[0°, 60°, 130°, 210°, 270°]$, independently of 360° symmetry.

Each interaction is mapped to a physical feedback loop: actions such as `rotateTo` are used to animate the switch's movement, while optional commands like `playSound` reinforce realism via sound effects. Logical branching is also supported through conditional actions and event dispatching, enabling the rotary switch to influence other actuators based on the selected mode.

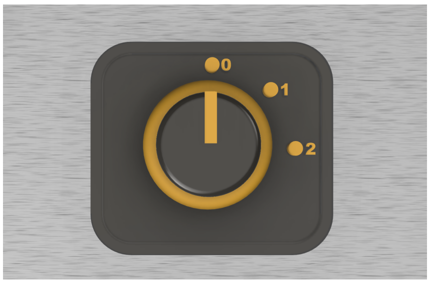
The modular design, separation of logic and representation, and reliance on external configuration make the `RotarySwitchActuator` ideal for immersive control systems that require stateful selection, such as machine mode setting, system toggles, or contextual changes in a virtual environment.

A dedicated set of example images is provided in Figure 4.2 to illustrate how different types of rotary selectors can be modeled and animated. These examples include an analog pressure gauge, a key switch, and a multi-position state selector demonstrating the actuator's versatility.



(a) Analog pressure gauge. Adapted from [12].

(b) Key switch used to activate or disable specific system functions.



(c) State selector with multiple rotary positions.

Figure 4.2: Examples of physical rotary-based components that can be modeled as `RotarySwitchActuator`.

4.2.3. Visual Light Actuator: Status Lights and Feedback

The `VisualLightActuator` models a general class of actuators used to visually reflect the state of a system through dynamic light behaviour. It is designed to represent status indicators, such as signal lamps, warning LEDs, or interface lights, that respond to state changes by altering their appearance. These elements provide immediate visual feedback, reinforcing the system's current condition in a non-verbal and intuitive manner.

The actuator operates on a set of logical light units, each associated with a named visual element. These lights are not tied to a specific geometry or arrangement; instead, their configuration is abstract and fully externalized. The actuator responds to high-level actions defined in an external statechart, which dictates how each light should behave in relation to system transitions.

The fundamental actions supported by this component are declarative and include:

- `lightOn` / `lightOff`: activate or deactivate a named light, simulating a switch in operational status;

- `startBlinking` / `stopBlinking`: initiate or interrupt a periodic visual pulse, useful for signaling warnings or transitions;
- `turnAllOff`: deactivate all defined lights simultaneously, resetting the visual state of the actuator.

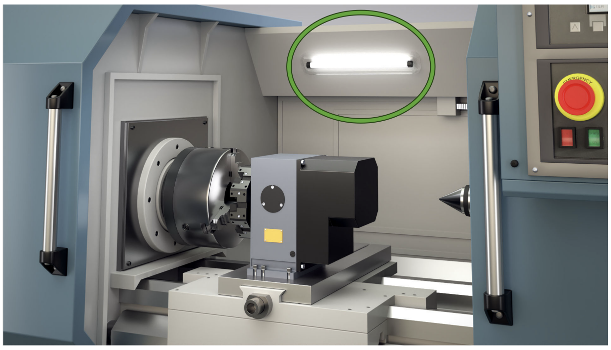
Each action is interpreted dynamically and applied to the corresponding light unit, which is assumed to possess a controllable visual parameter (e.g., brightness, colour intensity, emissive response). This abstraction allows lights to behave consistently regardless of their underlying geometry or rendering implementation.

The actuator reacts to events by changing the state of its lights, offering both simple on/off feedback and more complex multi-state indicators. Its logic is fully modular and works with any number or type of lights, making it suitable for many contexts where clear visual feedback is important.

A dedicated set of example images is provided in Figure 4.3 to illustrate how different visual signaling devices can be implemented using the `VisualLightActuator` logic. These examples include a vertical signal pole with multiple indicators, an LED light embedded into an HMI panel, and an application of integrated CNC machine lighting, showcasing the actuator's adaptability across various industrial use cases.

(a) Three-light signal pole used for machine status indication.

(b) LED indicator integrated into a machine HMI panel, positioned alongside its corresponding control button.



(c) Industrial CNC machine with mounted light. Adapted from [7].

Figure 4.3: Examples of visual lights that can be modeled using `VisualLightActuator`.

4.3. Complementary Interactions

This section presents interaction modalities that, while not functioning as actuators in a strict sense, play a fundamental role in enhancing the user's ability to manipulate and interact with virtual elements. These modalities include direct physical interaction through grabbing and the use of snapping mechanisms for guided placement. Unlike core actuators, these components do not typically initiate system-level state changes, but they contribute to the intuitiveness and precision of the immersive experience. Their implementation is designed to be modular and flexible, supporting diverse interaction scenarios and adapting to the needs of different virtual environments.

4.3.1. Physics-Based Grabbing of Virtual Objects

This section introduces a generalized approach for enabling the physical grabbing of virtual objects in immersive environments. The interaction is based on the natural gesture of squeezing the analog grip button on the VR controllers, which allows the user to pick up, move, and release 3D objects in real time.

The core logic is governed by a dynamic check on the target mesh: the system verifies whether the object, selected by the user's interaction, is explicitly marked as grippable through an external configuration attribute (e.g., a `gripping:true` flag). This property indicates that the mesh is intended to be interactable via grabbing.

The check is performed recursively along the object's hierarchy, allowing developers to tag only the top-level parent of a complex structure rather than each individual subcomponent. This hierarchical validation ensures that grabbing is only permitted when the user targets an authorized and complete entity.

By enforcing this rule, the system avoids unintended manipulations, such as grabbing isolated fragments of composite models, preserving the object's integrity and intended behavior. The tagging mechanism ensures robust interactions by allowing only explicitly designated elements to respond to user input, leaving static or decorative parts unaffected.

When a valid object is identified, selected via the laser emitted from the user's VR controller, it is programmatically attached to the controller's grip point, effectively synchronizing its position and orientation to the user's hand movements.

The interaction system also supports the activation of complementary components, such as snapping zones, which can be triggered contextually during the grabbing process. This allows for seamless transitions from free manipulation to constrained placement, enriching the realism and coherence of the user experience.

All behaviors are governed externally via configurable properties, making the implementation highly modular, robust, and reusable across different applications. This grabbing mechanism is particularly suitable for immersive training and digital twin environments where users must interact with objects in a realistic and physically consistent manner.

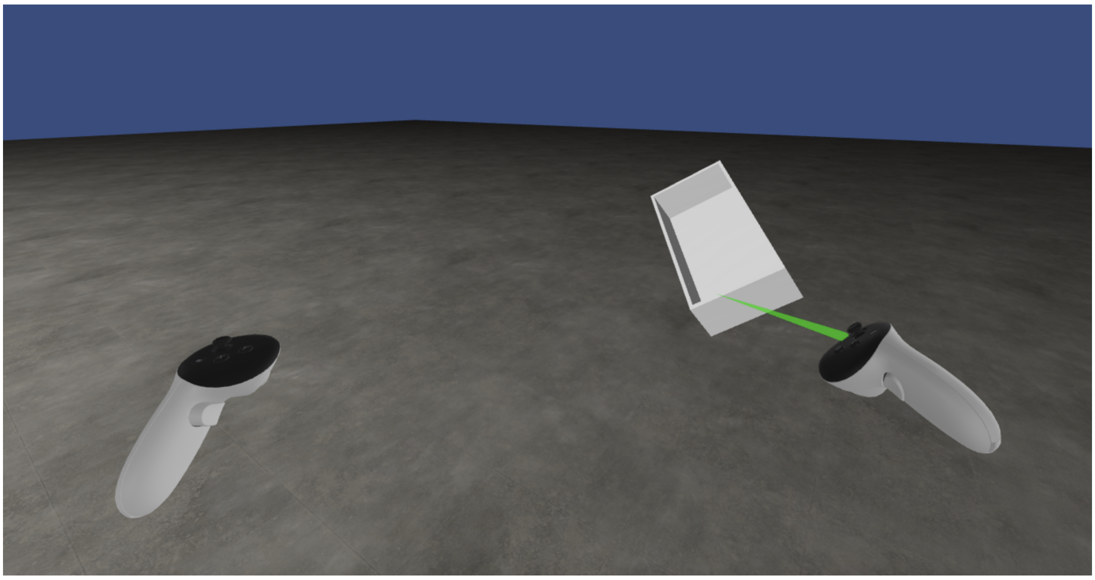


Figure 4.4: Example of object interaction: the right VR controller grabs a white box using the laser pointer.

4.3.2. Snapping Zones and Constrained Placement

The `SnapZoneActuator` enables structured and constrained object placement within immersive environments, simulating "docking zones" that support intuitive, accurate positioning of grabbable elements. It abstracts the concept of predefined placement regions, typically invisible during idle phases, that become active and interactive during manipulation.

Each snapping zone is modeled as a 3D mesh and can assume various geometries depending on the scenario (e.g., rectangular areas, disks, or custom bases). Optional configuration parameters, `snapPosition` and `snapRotation`, allow precise definition of where and how an object should be aligned upon snapping. If these are not specified, the object will automatically align to the center and orientation of the Snap-Zone itself, enabling flexible usage across both simple and highly controlled placement scenarios.

Internally, each zone is governed by a dedicated statechart, typically transitioning between an `inactive` and an `active` state. Upon activation, usually triggered by a grabbing event, the zone becomes visible and is associated with a static physics body, allowing it to detect intersections with other meshes in the scene. This detection is managed through an overlap-check routine executed at defined intervals (e.g., every 500 ms), ensuring responsiveness without excessive computational load.

When a valid overlapping object is detected, the actuator invokes a structured snapping procedure. This process detaches the mesh from the controller, repositions and reorients it according to the configured target.

This interaction model ensures a smooth and immersive snapping experience, balancing constraint and user control. The snapping logic is fully decoupled from specific scene elements, making it reusable and easy to adapt. The actuator itself operates on generic properties, preserving modularity and enabling full reconfiguration through JSON files.

This snapping mechanism is particularly suited for applications such as guided assembly and structured sorting, scenarios where correct part placement is critical to achieving functional accuracy and pedagogical effectiveness. In such contexts, Snap-Zones help ensure that users follow predefined steps, reducing the likelihood of incorrect configurations. In educational contexts, they can be employed to reinforce procedural understanding, allowing learners to receive immediate visual confirmation upon successful placement and progression control.

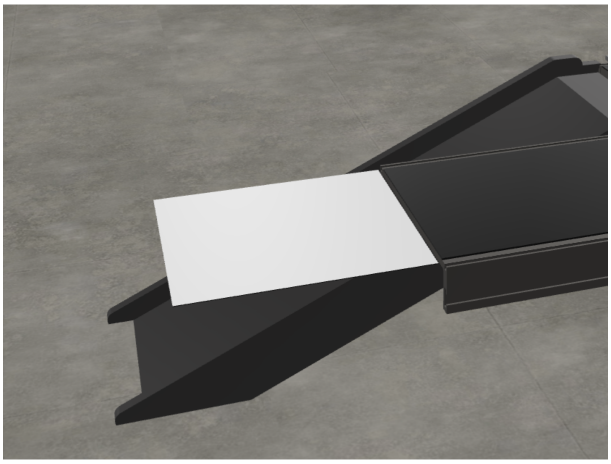


Figure 4.5: SnapZone (in white) positioned at the entrance of a conveyor belt, used to guide the correct placement of virtual objects.

4.4. Contextual Information Displays in VR

This section describes the set of visualization tools integrated into the immersive environment to support user awareness and system feedback. These tools, such as head-anchored or controller-linked panels, are used to adjust the main navigation parameters within the virtual scene and to display the current configuration on demand, offering users greater control and awareness of the immersive environment. Designed to be non-intrusive yet always accessible, these elements serve as complementary interactive interfaces that enrich the user experience.

4.4.1. Head Up Display and Information Panels

To provide continuous system feedback and interaction control in immersive contexts, a set of custom information panels has been specifically designed and implemented within the environment. These components are tailored to adapt to user preferences and ensure accessible monitoring of key parameters during exploration and task execution.

The **Main Control Panel**, the primary interface, is a head-anchored configuration panel attached to the XR camera. It maintains a fixed relative position slightly below eye level and is toggled via the `Y` button on the left controller. This ensures that the panel is visible only when needed, preserving immersion.

This panel includes interactive buttons, each linked to runtime variables and controller logic. The user can dynamically modify several parameters:

- **Speed + / −**: Adjusts movement speed (active only in free movement mode);
- **Gravity ON/OFF**: Enables full 3D navigation or locks vertical motion to simulate grounded walking;
- **Teleportation ON/OFF**: Switches between free locomotion and point-based teleportation;
- **Collisions ON/OFF**: Toggles collision detection with the camera;
- **HUD / iTablet**: Controls the visibility of the two auxiliary panels;
- **HUD position**: Shifts the HUD along X (sides), Y (height) and Z (depth) axes to improve ergonomics and personalisation.

All buttons provide press animations, audio feedback (click sound), and are color-coded to communicate function. Interactions occur via laser pointer selection from the XR controllers.

To avoid invalid command combinations, the interface enforces runtime constraints. For instance, when teleportation is active, speed controls are temporarily disabled. In such cases, the system displays contextual alerts directly in front of the user's view, ensuring clarity and preventing confusion.

This utility generates temporary, one-line informational messages anchored to the XR camera. These are automatically dismissed after a few seconds, maintaining visual cleanliness while reinforcing system feedback. The same mechanism is extensible to other alerts or instructions.

The **HUD** (Head Up Display) is a semi-transparent horizontal panel aligned to the headset's direction and rendered near the top of the field of view. It displays live status for key parameters (speed, gravity, teleportation, collisions), using color-coded labels and emoji symbols for quick readability.

Its position can be adjusted through dedicated buttons, allowing the user to fine-tune its placement based on comfort and visibility preferences. If the user attempts to reposition the HUD while it is hidden, a warning is issued guiding consistent usage.

The **iTablet** is a controller-linked panel, parented to the left-hand XR controller and designed to behave like a wrist-mounted device. It mirrors the HUD content but presents it in a vertical layout, optimized for near-field inspection. By simply raising the arm, the user can consult system information in a natural and unobtrusive manner.

This format supports private access to runtime variables without crowding the central view and is particularly suited to scenarios requiring momentary verification of system status or immersive configuration changes.

Both the HUD and iTablet panels are updated in real time using the same backend logic and dynamic texture updates. Their visual footprint is minimal, and they are fully configurable at runtime. Combined with the main control interface, these components provide a scalable and ergonomic feedback model, essential for immersive workflows where responsiveness, awareness, and configurability are key.

An overview of the three-panel system (Main Panel, HUD, and iTablet) is shown in Figure 4.6.

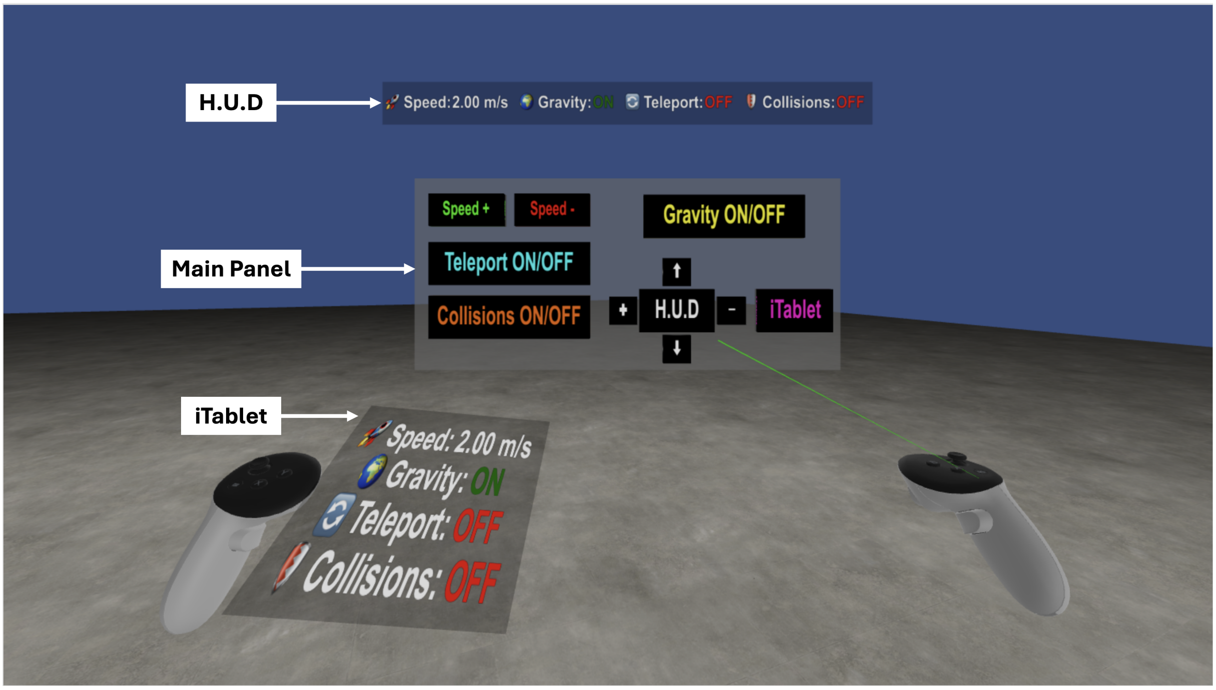


Figure 4.6: Three-layer information system in XR: Main Control Panel (center), HUD (top), and iTablet (bottom), providing real-time interaction and feedback.

5 | Implementation of Immersive Interactions

This chapter illustrates how the interaction models conceptually introduced in Chapter 4 have been implemented in code and instantiated independently of the specific use case, validating their reusability and configurability. The focus is placed on the translation from abstract interaction logic into a modular JavaScript-based architecture, supported by declarative configuration via JSON files and spreadsheet interfaces.

5.1. System Architecture and Modular Structure

5.1.1. Basic Architecture of the Control System

The control system, intended here as the logical architecture that governs how virtual components behave and respond to user input, underlies the VEB.js framework and has been structured around a hierarchy of classes, reusable code templates that define the properties, states, and interaction rules of each element in the virtual environment. This hierarchy allows the definition of common functionalities shared across all interactive assets, while also enabling the creation of specialized behaviours through a mechanism known as *inheritance*, in which new classes extend existing ones. This design supports the reuse of code and the flexible configuration of actuators, whose behaviour is governed by external statecharts modelled in JSON format.

Base Class: `ControlledAsset`

At the root of the hierarchy lies the abstract class `ControlledAsset`, which represents an asset controlled by state-based logic. Each object derived from this class is associated with:

- a reference to the global scene controller,
- a 3D node (`node`) corresponding to the visual mesh,
- a statechart (if provided), defined as an external JSON object.

The constructor initializes these references and, if a state machine is present, automatically starts its execution using the `interpret()` function from the `XState` library, a JavaScript library for declarative and observable finite state machine management.

Once activated, the state machine is monitored via `actor.subscribe()`, which enables real-time observation of state changes. Each time a transition occurs:

- an MQTT message with the updated state is published,
- the dynamic interface is updated,
- any action specified is triggered through the `interpretAction(action)` function.

This mechanism allows each asset to be reactive to its internal state and capable of dynamically performing visual or logical operations defined externally from the main code.

Derived Class: `Actuator`

The `Actuator` class extends `ControlledAsset` and represents a generic actuator, enriched with additional functionalities for managing physical and visual actions in the 3D scene. All interactive actuators developed during the project derive from this class, such as push buttons (`ButtonActuator`), rotary selectors (`RotarySwitch`), visual light actuators (`VisualLightActuator`) and snap zones (`SnapZoneActuator`). For this reason, the `Actuator` class serves as the common foundation and introduction to the following sections, which will describe specific extensions in detail.

The `interpretAction()` method is overridden in the `Actuator` class to handle basic and differentiated actions, including:

- conditional asset visibility (`show`),
- kinematic animations for asset translation and rotation (`moveTo`),
- real-time physics-based movements (`moveToPhysics`), using the Havok.js engine, a modern and high-performance physics library for simulating realistic dynamics in 3D environments.

For the `moveTo` action, two separate animations are generated: one for position and one for rotation of the target object. The animation is computed by considering the hierarchy of 3D objects and the relative scale between elements.

For the `moveToPhysics` action, a velocity vector consistent with the desired transformation is calculated and applied to the node's physics body. This approach allows realistic movement simulation based on force and inertia.

Additional features implemented in the `Actuator` class include:

- `send(eventName)`, for simplified event dispatching between state machines;
- `getCurrentState()`, for retrieving the current state of an actuator, which can be used to synchronize it with other actuators or logic.

This structure allows for complete separation of logic from visual behaviour, making each actuator autonomous and extensible through inheritance and modular design. Moreover, it facilitates the definition and management of new types of interactions, supporting the development of immersive systems in complex scenarios. Any new actuator can be created by extending `Actuator`, inheriting base behaviours and specializing them as needed.

5.1.2. Relation Between Actuators, Configurable Assets, and StateCharts

In the developed framework, the behaviour of each actuator is not defined directly within the source code but is instead described externally via JSON files containing a statechart structured according to the principles of finite state automation. This approach enables the modelling of asset operational logic in a clear, reusable, and decoupled manner from the implementation. As a result, actuators become generic components capable of dynamically adapting to different logic or assets, depending on the provided configuration.

Each configurable asset is initialized by assigning it a unique identifier (`id`), a reference 3D mesh (`node`), and, if available, a statechart object. The associated mesh can either be created natively within the scene using Babylon.js primitives, useful for basic geometric shapes, or, as is often the case in industrial simulations, imported from an external `GLTF` file. These `GLTF` files are typically the result of 3D modelling performed in dedicated CAD software and exported via rendering optimization softwares (e.g., Blender [5]) in a format optimized for real-time rendering and interaction.

The statechart defines the possible states of the associated actuator (e.g., `active`, `inactive`) and the transitions between them, triggered by specific events (e.g., `press`, `release`). Transitions can be enriched with actions, which represent visual or logical behaviours to be executed during state changes, such as triggering an animation or sending an event to another actuator.

The link between the asset and its state machine is made through the `interpretAction()` function, which interprets each action defined in the statechart and executes its effects in the 3D scene. Actions are performed at runtime, making the actuator's behaviour fully parametric and reconfigurable. For example, a generic button in the VR scene can operate in either bistable or monostable mode simply by modifying the state sequence in its associated JSON file, without any change to the JavaScript actuator code.

This architecture enables a data-driven paradigm in which the scene is built and controlled based on external configuration. The user or developer defines the scene's assets through a JSON file, specifying the actuator type, its position, orientation, and the reference to the related statechart.

This operational flow allows for the design and testing of complex interaction logic in a visual manner, leveraging *Stately.ai* [25], a web-based tool for creating, editing, and visualizing statecharts through an intuitive graphical interface.

In summary, the relationship between actuators, assets, and statecharts constitutes the core logic of the system. Each actuator becomes an autonomous functional unit capable of reacting to events, changing states, and triggering behaviours, fully aligned with the immersive Digital Twin paradigm.

5.2. From Concept to Implementation: Core Interactive Actuators

5.2.1. Button Actuator

The `ButtonActuator` class implements the interaction logic of a physical push button by extending the generic `Actuator` superclass. It is designed to support discrete state transitions and is capable of performing both simple and conditional actions based on user input.

Upon instantiation, the constructor initializes key properties: it stores the original position of the associated 3D mesh (`this.originalPosition = this.node.position.clone()`) and enables interaction by setting `isPickable = true`.

The main behavior is encapsulated in the `interpretAction()` method, which extends the logic of the base `Actuator` class by handling both string commands (e.g., `trigger`) and structured action objects. A `trigger` command directly invokes a state transition activating the associated state machine logic.

For physical simulation, the button supports `press` and `release` actions. When a `press` is received, the mesh is moved along a configurable offset vector (default: `{ x: 0, y: -0.1, z: 0 }`) using a `moveTo` animation handled by the base class. This visually simulates the downward displacement of a button. The `release` action restores the mesh to its original position, completing the tactile cycle. Both motions can be customized via a `payload.duration` parameter that defines the animation speed in milliseconds.

Auditory feedback is implemented through the `playSound` via the `playClickSound()` function. This function uses the Web Audio API to generate a short sine-wave sound, providing immediate feedback without external audio files. Parameters such as frequency, volume, and decay are defined within a `soundParams` object and applied dynamically at runtime.

Beyond basic motion and sound, the class enables more advanced control patterns:

- `send` dispatches events to other actuators by referencing their `id` and `send()` method.
- `if` evaluates the current state of a target actuator (via `getCurrentState()`) and conditionally executes an action.
- `sequence` performs a list of actions in a defined order, each optionally delayed via the `delay` parameter.
- `startLoopEvent` triggers periodic events at fixed intervals using `setInterval()`.

All these actions are defined externally through the statechart. For example, a button can transition from `idle` to `pressed` upon `trigger`, during which it executes a `playSound`, a `moveTo`, several `if` conditions, and a `sequence` of actions. After a timed delay (e.g., 300 ms), it transitions to `released` (return motion), and finally back to `idle`, completing the interaction cycle.

This architecture supports full separation between logic and structure: the actuator's behavior is entirely determined by the associated JSON configuration. As a result, the same `ButtonActuator` implementation can be instantiated in various contexts, ranging from simple toggles to complex decision-making hubs, without modifying the underlying code.

5.2.2. Rotary-Switch Actuator

The `RotarySwitchActuator` class implements a configurable rotary selector, allowing users to cycle through discrete operational states by triggering a `NEXT` event. It extends the generic `Actuator` base class and is designed to visually rotate a 3D mesh in response to user input, while simultaneously executing logic defined in an external statechart.

Upon instantiation, the component initializes key parameters including:

- a rotation axis (`x`, `y`, or `z`), determining the direction around which the knob rotates;

- a rotation direction (e.g., clockwise or counterclockwise);

To ensure correct behaviour, the class searches for a visible child mesh with geometry and defines it as the interaction and action target. The mesh is made `pickable` and associated with an `ActionManager` that listens for user input. On interaction (e.g., mouse click or VR controller trigger), the `send(NEXT)` function is invoked, prompting a transition in the associated statechart.

The logic for responding to state changes is handled inside the overridden `interpretAction()` method, which supports the following key commands:

- `rotateTo`: rotates the mesh to a specific angle, using the declared axis and maintaining continuity in mesh orientation;
- `playSound`: plays a synthetic click sound via the `playClickSound()` function to simulate mechanical feedback;
- `send`: dispatches events to other actuators, enabling coordination and interdependence between system components.

The associated JSON statechart defines the complete logic of the selector:

- Each state (e.g., `state0`, `state1`, etc.) includes entry actions such as `rotateTo`, `playSound`, and `send`.
- The `NEXT` event triggers transitions between states in a cyclic sequence, creating a loop through predefined operating modes.
- Rotation angles are specified per state and need not be evenly spaced, allowing maximum flexibility in visual and functional design.

Thanks to this architecture, the `RotarySwitchActuator` can be used to replicate real-world mode selectors or configuration dials in industrial VR applications. The component supports customization in both interaction logic and physical behaviour, making it suitable for a wide range of immersive control interfaces.

5.2.3. Visual Light Actuator

The `VisualLightActuator` is designed to simulate status indicators such as signal lights or panel LEDs, providing immediate visual feedback on system states. It extends the base `Actuator` class and operates by controlling the `emissiveColor` of associated 3D meshes, effectively creating the illusion of lights turning on or off without the need for real light sources.

The core behavior is handled in the `interpretAction()` method, which extends the base implementation with light-specific commands. These include:

- `lightOn` / `lightOff`: Sets the `emissiveColor` of the mesh to a bright or dim value, simulating illumination;
- `startBlinking` / `stopBlinking`: Initiates or halts an internal loop that alternates the light's state at a specified interval;
- `turnAllOff`: Disables all lights and ensures any active blinking is stopped.

Each light mesh operates based on color parameters explicitly defined in the external JSON configuration. The color to activate (e.g., `green`, `yellow`, `red`) is specified as an argument in each action, such as `lightOn` or `startBlinking`. This approach decouples the visual asset's name from its functional role, allowing greater flexibility in naming conventions and reusability. Blinking behaviors are managed through individual timers stored in `blinkIntervals`, enabling multiple lights to blink independently and concurrently based on their assigned configuration.

Visual feedback is tightly integrated with the system's logic via statecharts. For example, a light may transition from `off` to `on` upon receiving a `trigger` event, or enter a `blink` state that toggles visibility every 400 milliseconds.

Thanks to its lightweight structure and full separation of logic from visual rendering, the `VisualLightActuator` can be used in a wide range of applications, from simple binary indicators to complex multi, state visual systems.

5.3. From Concept to Implementation: Complementary Interactions

5.3.1. Physics-Based Grabbing

The `grabbing` interaction enables users to physically interact with virtual objects by picking them up, moving them around, and releasing them using the grip button on VR controllers. This feature is implemented through the `setupGripping()` function, which connects each Controller to the `xr-standard-squeeze` input component, the analog trigger located on the side of the VR controller. This trigger is ergonomically designed to simulate natural hand grasping and supports pressure sensitivity, allowing for a realistic and intuitive grabbing gesture.

`Grabbing` begins when the user points at an object using the controller's laser and presses the grip button. The system evaluates the mesh currently under the pointer (`scene.meshUnderPointer`) and checks whether it is both pickable and explicitly marked as `grippable`. This is done using the helper function `hasGripping()`, which recursively inspects the metadata of the target mesh and its parent nodes. A mesh is considered grabbable only if it (or one of its parents) has the flag `"gripping":true` in its configuration JSON.

Once a valid grabbable mesh is identified, it is reparented to the controller's grip transform, effectively binding its position and rotation to the user's hand in real time. This attachment enables smooth manipulation and direct spatial control of the object.

If the grabbed object includes a physics body, i.e., it has been assigned a dynamic physical behaviour in the scene via the HavokJS engine, it is temporarily switched to `PhysicsMotionType.STATIC` upon attachment to the controller. In this mode, the object is excluded from all physical simulations, meaning it no longer responds to forces such as gravity, collisions, or momentum.

This temporary suspension of physical behaviour serves two critical purposes:

1. **Eliminates instability during movement:** When an object is being manipulated by the user, real-time physics calculations (like collision resolution or inertia) can introduce jitter (unwanted oscillations) or slight shifts in position.
2. **Preserves alignment and responsiveness:** By freezing the object's physical simulation, it becomes perfectly synchronized with the controller's position and orientation. This ensures that the object visually follows the user's hand with no lag or offset.

The `STATIC` mode is therefore a crucial intermediary step: it guarantees full control and positional accuracy while the object is being held, without interference from the physics engine. The original physical behaviour will be restored immediately upon release.

When an object is grabbed, the system sends an `activate` event to all configured SnapZones present in the scene. This prepares specific areas to receive the object upon release, enabling precise placement and alignment in constrained environments. SnapZones represent optional destinations where the grabbed object can be automatically positioned or locked. Their structure and activation logic will be described in paragraph 5.3.2.

When the grip button is released, the object is detached from the controller while maintaining its world transformation. A new dynamic physics body is created using its dimensions. The mesh is assigned `PhysicsMotionType.DYNAMIC`, making it responsive to forces such as gravity and collisions.

Physical properties, like friction and restitution, are applied to ensure realistic post-release behaviour. Friction controls how much the object resists sliding on contact surfaces, while restitution determines how much it bounces after impact. These parameters allow the object to fall, slide, or rebound depending on the environment.

This implementation is completely data-driven. It uses runtime metadata (such as `gripping=true`) and controller inputs to determine behaviour, ensuring full modularity and reusability across a variety of scenes, asset types, and simulation scenarios.

Thanks to its flexible architecture and support for real-time physics manipulation, the grabbing interaction is suitable for a wide range of immersive applications, including assembly tasks, maintenance procedures, training modules, and educational simulations.

5.3.2. Constrained Object Placement

The `SnapZoneActuator` implements a reactive mechanism for constrained placement of grabbable objects, enabling automatic alignment to predefined spatial configurations within the virtual environment. Each instance is linked to a specific mesh and associated with a finite state machine, which governs its operational behavior.

Upon activation, the `SnapZone` becomes visible, instantiates a static physics body based on the associated mesh's bounding volume, and initiates a periodic overlap check against all scene meshes marked as `pickable` and `grippable`. This scan is executed at 600 ms intervals, maintaining responsiveness while preserving computational efficiency.

The snapping logic is triggered when a target mesh intersects the `SnapZone`'s bounding box. The `interpretAction(AttractTo,{mesh})` command activates the `attractObject()` method, which performs the following sequence:

- **Detachment:** The mesh is unparented from the gripping controller to eliminate local transform dependencies.
- **Physics suspension:** If the object has a physics body, it is temporarily set to `PhysicsMotionType.STATIC`, disabling dynamic interactions during alignment.
- **Transform override:** The object is relocated to the coordinates defined by `SnapPosition` and reoriented using `SnapRotation`, if these parameters are explicitly specified in the JSON configuration. If not provided, the object is positioned at the geometric center of the Snap-Zone mesh and adopts its current world orientation.
- **Physics reinitialization:** After a brief delay (typically 3 seconds), a new `DYNAMIC` physics body for the snapped object is created, preserving original friction and

restitution values to ensure continuity of physical behavior post-placement.

Finally, the `SnapZone` sends a `deactivate` event to its state machine after 2 seconds, reverting to an idle state and disposing of its physics body. This transition reduces collision computation overhead and prepares the actuator for subsequent use.

This mechanism supports both flexible and tightly constrained interaction paradigms, depending on the precision required by the scenario. Its modular architecture, based entirely on JSON-declared properties and state-driven logic, makes it highly reusable across various XR applications. For instance, snap zones can be used to guide the correct placement of components on an assembly line, to validate tool positioning in training simulations, or to trigger specific system responses when an object is docked in a predefined area.

5.4. From Concept to Implementation: Contextual Information Displays

5.4.1. Head-Up Display and Information Panels

To support real-time feedback and adaptive interaction in VR contexts, a modular panel system was developed and deployed. The system includes three complementary components: a central control panel (`mainPanel`), a head-anchored HUD panel (`hudPanel`), and a controller-mounted iTablet display (`iTablet`). Each component is instantiated as a 3D mesh with a dynamic texture, allowing runtime updates without significant rendering overhead.

The `mainPanel` is initialized via the `createSimplePanel()` function and parented to the XR camera. Positioned slightly below the field of view, it serves as the anchor for interactive VR buttons created with the `createVRButton()` function. Buttons enable control over key WebXR settings (e.g., speed, teleportation, flying, collision), with safeguards that prevent invalid transitions, for example, speed adjustments are disabled during teleportation. Button presses trigger updates via functions such as `updateXRMovement()`, `updateXRTeleportation()`, `updateXRCollisions()`, and `updateXRGravity()`, which modify the feature configuration of the WebXR session.

The `hudPanel` is rendered as a semi-transparent, horizontal bar anchored above the XR camera view. It uses a `DynamicTexture` updated via `updateHUDStatus()` to show current system parameters (movement speed, gravity, teleportation, collisions), displayed with emoji-based labels and colored status indicators. This design allows users to quickly verify the environment configuration while maintaining immersion. The HUD's position can be interactively adjusted using dedicated buttons (up/down, forward/back), with on-screen warnings triggered if the HUD is hidden during repositioning.

The `iTablet` provides an alternative wrist-mounted display attached to the left controller. Also created via `createSimplePanel()`, it uses a smaller dynamic texture (`updateITabletStatus()`) to present the same runtime parameters in a vertical layout optimized for close-range inspection. Its alignment to the controller makes it intuitive to consult on demand without disrupting spatial orientation.

All components are enabled or hidden programmatically and initialized as disabled to minimize distraction. Their materials feature emissive coloring to ensure readability across different lighting conditions.

This immersive display framework improves interaction reliability and accessibility by clearly exposing system state and responding dynamically to user inputs. Its architecture is reusable, non-blocking, and compatible with any WebXR scene that requires continuous feedback, adjustable parameters, and contextual warnings in real time.

5.5. Data Interface for Scene Configuration

To ensure modularity, scalability, and ease of reconfiguration, the entire virtual environment is defined through a spreadsheet-based configuration pipeline that transforms a human-readable Excel file into a structured JSON file consumed at runtime by the VEB.js application. This process enables seamless integration of all interactive actuators introduced in Chapter 5, allowing their behaviours to be defined without hardcoding logic in the source code.

The configuration process follows a clear pipeline:

Excel (.xlsm) → Macro Export → JSON (.json) → VEB.js Loader

This setup allows designers to control all aspects of the environment, from positioning and physical properties to logic and interactivity, using a structured table. The configuration file not only defines static 3D objects but includes dynamic actuators like buttons, switches and snapping zones.

The `Assets` sheet of the Excel file serves as the foundation of the configuration[1]. Each row corresponds to an object or actuator, with fields such as those shown in Table 5.1.

---

[1] Configuration file available at [28]:

Main configuration fields in the Excel-based setup

Excel Field	Description
`id`	Unique identifier for each asset. Used as reference in statecharts or commands.
`inScene`	Boolean or integer flag indicating whether the asset is initially present in the scene.
`type`	Ontology class defining the type of object or actuator (e.g., `ButtonActuator`, `SnapZoneActuator`).
`file`	Path to the model file or specific mesh node (e.g., `.glb`).
`unit`	Number of identical assets to instantiate from this configuration.
`position`	Spatial coordinates (x, y, z) defining the position of the asset.
`rotation`	Orientation of the asset in the scene (expressed in radians).
`placementRelTo`	Specifies to which other asset the current one is positioned relative to.
`parentObject`	Specifies the parent in a hierarchy, typically used for transformations.
`properties`	Defines physical and interaction behavior (e.g., mass, friction, restitution, gripping).
`statechart`	JSON block defining the asset's finite state machine and interaction logic.

Table 5.1: Key asset fields mapped from Excel to JSON for scene instantiation.

This structure is extensible and supports all actuator classes developed in this project. The JSON generated from the Excel file preserves this structure, enabling VEB.js to dynamically parse and instantiate each asset at runtime.

All actuators that will be presented in Chapter 6 are declared and fully configured through this system. For instance:

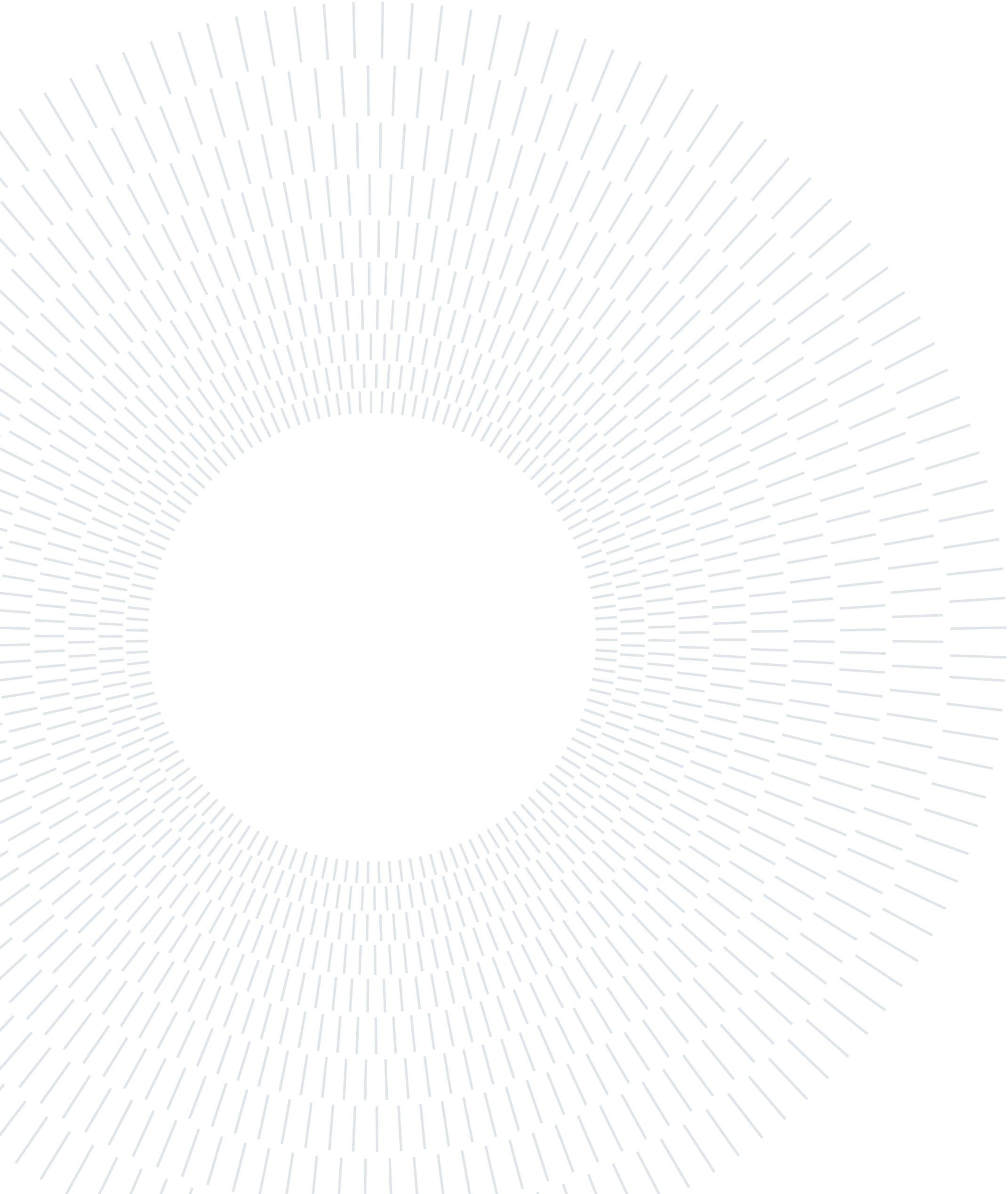
- **Push Buttons** (e.g., `CommandBox.GreenButton`) are identified as `ButtonActuator` and are connected to specific statecharts that define their press/release behaviour, visual feedback, and conditional event dispatching.

- **Rotary Switches** (e.g., `CommandBox.StateSelector`) are defined as `RotarySwitchActuator` with sequential state transitions and associated actions, such as sending messages or rotating the mesh.
- **Signal Pole** (e.g., `SignalPole.RedLight`) are defined as `VisualLightActuator` components, with action blocks that manage parameters such as `color`, `lightOn`, or `startBlinking`.
- **Grabbing-enabled Parts** (e.g., `part`, `box`) include a `gripping:true` property in their `properties` block, signaling that they can be physically picked up and manipulated.
- **Snap Zones** (e.g., `SnapZone_part`, `SnapZone_box`) are modeled as `SnapZoneActuator` types, with optional `SnapPosition` and `SnapRotation` parameters defined in the JSON. These zones support precise alignment and task validation.

The spreadsheet-based configuration system enables a high level of control over all components of the scene. Any mesh, regardless of how it was modeled or imported, can be turned into a fully functional interactive component simply by assigning it a semantic type and linking it to a coherent statechart.

This means that a custom 3D object can behave as a button, a rotary switch, a sensor, or any other actuator purely by configuring the right metadata, without modifying the underlying model. The logic is entirely encapsulated in the spreadsheet and JSON layers, allowing developers to declaratively assign behaviour while keeping assets reusable and modular.

This approach simplifies scene configuration and fosters scalability and maintainability across different industrial use cases.



6 | Validation of Immersive Interactions in the Pick & Place Use Case

This chapter demonstrates how the interactive actuators developed in the previous chapters have been instantiated and used within a realistic industrial simulation. The chosen use case, a Pick & Place cell, serves as a test environment for validating the modularity, reusability, and immersive potential of the framework. The scene includes configurable actuators, manipulable objects, and visual feedback systems, all orchestrated through declarative logic.

Through this implementation, the chapter explores the extent to which the designed components align with operational requirements and user expectations in terms of realism, usability, and responsiveness.

6.1. Overview of the Pick & Place Cell

The Pick & Place cell represents an application case developed by researchers Mathieu Roisin and Bernard Riera at the University of Reims Champagne-Ardenne (URCA), and it has been adopted as the industrial use case for this project. It is an automated manufacturing system designed to simulate handling, sorting, and transfer operations within a discrete production context. The cell consists of two conveyor belts, a Cartesian robotic arm, a control box equipped with physical buttons, and a network of presence sensors distributed along the conveyors.

The system is designed to operate in distinct production modes, each characterized by a specific logical sequence of actions. These operational modes, along with a detailed description of the main components of the cell, will be further explored in Section 6.2.1 to provide a complete functional understanding of the system.

The behaviour of the cell is governed by an event-driven logic, in which sensors (e.g., item presence, box detection, end-of-travel switches) and digital actuators (robot motion, conveyor activation, picking and releasing operations) interact to determine the system's response to operating conditions. Communication can be managed through industrial protocols such as Modbus TCP/IP or MQTT, ensuring compatibility with real production environments.

Thanks to its modular structure and intuitive interface, the cell is well-suited both for learning purposes and for the validation of industrial logic within immersive virtual environments, which can be configured as high-fidelity Digital Twins.

The foundational elements of the scene are the two conveyor belts (Conveyor 1 and Conveyor 2), on which the system's main operations take place. Both are equipped with detection barriers, consisting of a sensor-receiver pair positioned transversely to the direction of the belt. These virtual sensors are designed to detect the passage, presence, and correct positioning of objects moving along the line. The detection system allows the robot and control logic to determine when an item is ready to be handled or transferred.

At the centre of the entire scene operates a Cartesian robotic arm, programmed to execute pick and place sequences based on operating conditions defined via statecharts. Its primary task is to pick individual items from Conveyor 2 and place them inside boxes traveling on Conveyor 1. This interaction requires coordination between sensors, control logic, and the timing of the production cycle, simulating realistic conditions of an automated assembly line.

The scene includes the objects manipulated by the system:

- **Items:** simple-shaped elements transported on the Conveyor 2 and recognized by the robot to be placed into containers.
- **Boxes:** empty containers moving along the Conveyor 1, which are filled with the items picked from Conveyor 2.

At the end of both conveyors, unloading slides are positioned to simulate the physical evacuation of materials once processing is complete. These components enhance the consistency and realism of the simulation, replicating the continuous flow of objects through the system.

The Control Panel serves as the main user interface, enabling direct interaction with the cell. It is equipped with:

- Four physical buttons: **Start**, **Stop**, **Initialization (BP Init)**, **Emergency**
- A three-position **State Selector**: for selecting the operational state (State 0, State

1, State 2)

- A **Potentiometer**: used to select the number of items to be placed into each box.

This interface, combined with visual feedback provided by a vertical Signal Pole with three lights (Red, Yellow and Green), allows real-time monitoring of the system's operational status. Each light is associated with one of the three operating modes that can be activated via the Control Panel, contributing to operational clarity and safe interaction.

To complete the scene, protective grids are also present, modelled in two variants:

- **Solid grids**, placed to protect critical areas and moving mechanisms
- **Perforated grids**, used to host and support functional elements such as the Control Panel

Table 6.1 provides a complete overview of all virtual components that constitute the Pick & Place cell, summarizing their roles and characteristics within the simulated environment.

Virtual components of the Pick & Place cell

Component	Description
Conveyor 1	Conveyor that carries boxes to receive items. Also equipped with detection sensors for synchronization with the robotic arm.
Conveyor 2	Conveyor belt where items are initially transported. Equipped with virtual sensors for object detection and position monitoring.
Detection Barriers	Sensor-receiver pairs placed across both conveyors to detect item presence, passage, and positioning.
Cartesian Robot	Programmable robotic arm performing pick & place tasks. Picks items from Conveyor 2 and places them into boxes on Conveyor 1.
Items	Simple-shaped parts that are picked from Conveyor 2 and placed into boxes. Represent handled components.
Boxes	Containers that move along Conveyor 1 to be filled with items by the robot.
Unloading Slides	Slides located at the end of each conveyor to simulate material evacuation after processing.
Control Panel	User interface for interacting with the cell. Includes buttons, selector, potentiometer, and is placed on a perforated grid.
Signal Pole	Vertical signal light with three indicators showing the current operating state for real-time feedback.
Solid Grids	Protective barriers simulating the shielding of sensitive parts and moving elements from user interaction.
Perforated Grids	Structural supports that hold components like the Control Panel in place and ensure spatial organization.

Table 6.1: List and description of virtual components composing the Pick & Place cell.

The whole scene and each of its assets are illustrated below with an explanatory image and a corresponding descriptive caption.

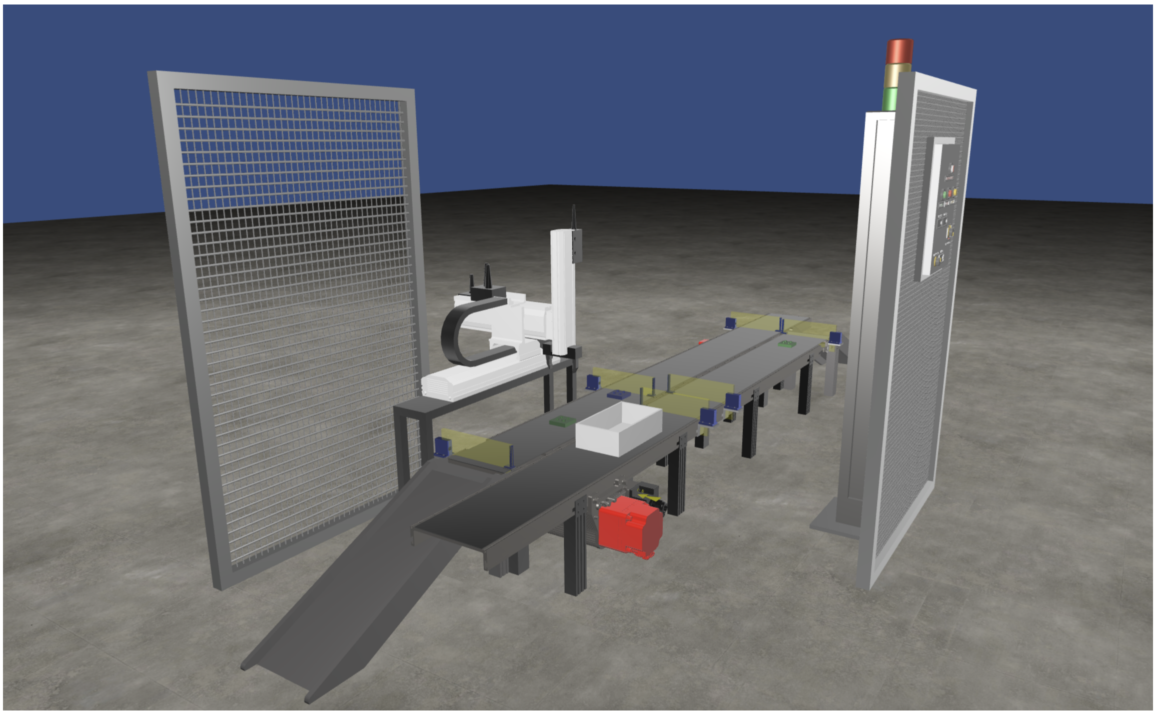


Figure 6.1: Full view of the Pick & Place virtual manufacturing cell.

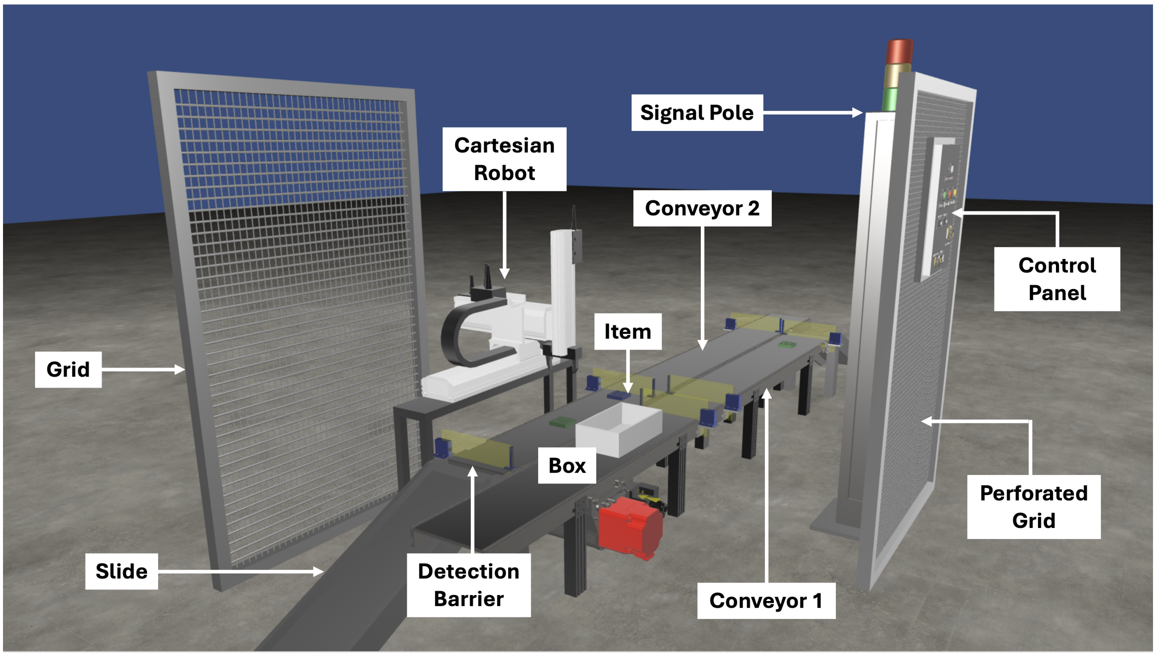
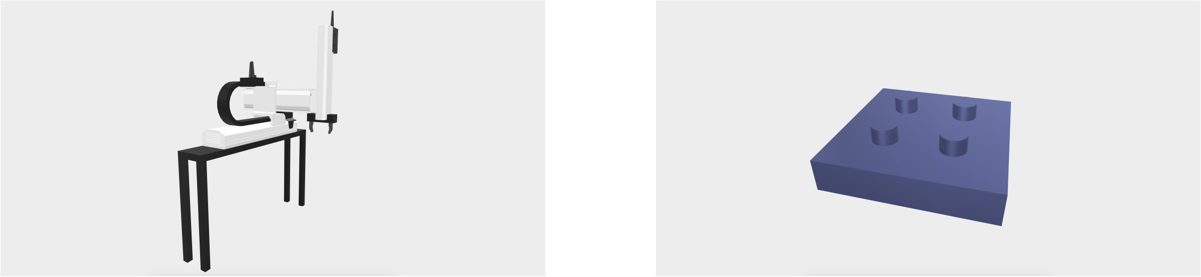


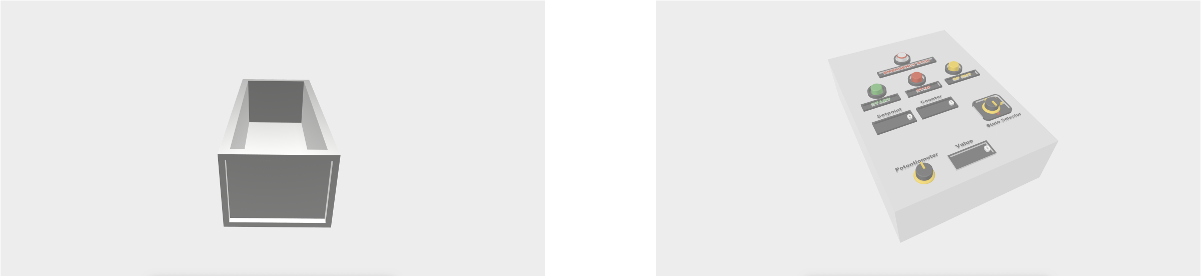
Figure 6.2: Full view of the Pick & Place virtual manufacturing cell, including the labels of each asset.



(a) Cartesian robotic arm used for pick and place operations.

(b) Basic item shape manipulated by the robot.

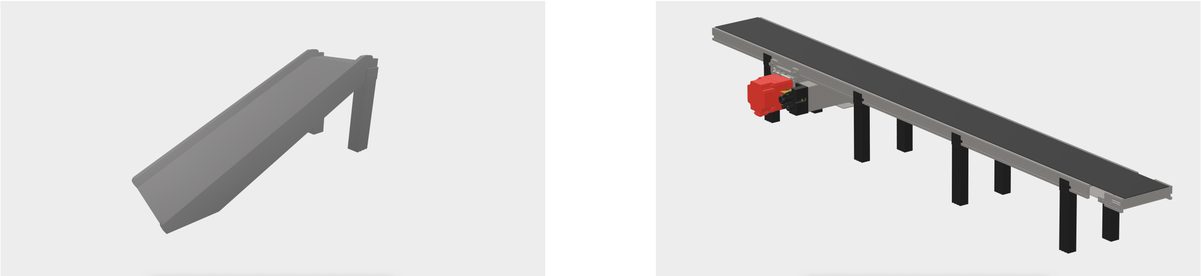
Figure 6.3: (a) Robot and (b) item used in the Pick & Place scenario.



(a) Transport box used for item collection.

(b) Control Panel with physical interface: buttons, selectors.

Figure 6.4: (a) Box and (b) Control Panel used in the Pick & Place scenario.



(a) Slide used for unloading processed items.

(b) Conveyor belt with motor for item transport.

Figure 6.5: (a) Final slide and (b) Conveyor belt used in the Pick & Place scenario.

(a) Signal pole indicating system state.

(b) Perforated protective grid supporting the Control Panel.

Figure 6.6: (a) Signal pole and (b) Protective grid used in the Pick & Place scenario.

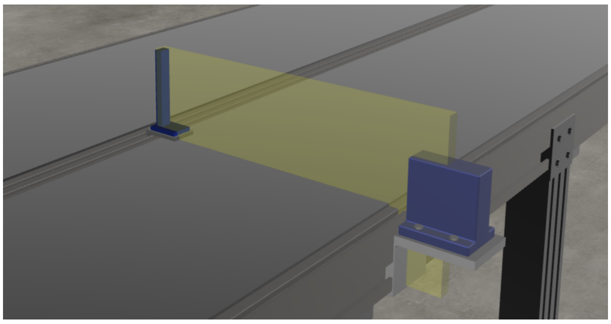


Figure 6.7: Barrier sensors for object detection on conveyors.

6.2. Application of Core Interactive Actuators

This section illustrates how each interactive component developed in this thesis has been applied within the Pick & Place simulation cell. For every element, such as push buttons, rotary switches, status lights, or grabbable objects, a direct link is established between its conceptual design (presented in Chapter 4) and its actual implementation in code (described in Chapter 5).

The goal is to demonstrate how the modular logic introduced in the earlier chapters can be concretely instantiated within a realistic and industrially inspired environment. Each subsection highlights a specific actuator or interaction mechanism, detailing its role within the simulation workflow and its behaviour in response to user input or system events.

6.2.1. Buttons and Discrete Control

In the Pick & Place scenario, push buttons represent the most immediate form of user interaction, triggering discrete control events through simple press-and-release actions.

These components are modeled using the `ButtonActuator` concept described in Section 4.2.1, and implemented via structured statecharts in Section 5.2.1.

The Control Panel serves as the main interaction hub for the user, featuring four distinct physical-style buttons:

- Start Button (Green)
- Stop Button (Red)
- BP_Init Button (Yellow)
- Emergency Button

Each button is defined as a separate mesh within the `CommandBox.glb` model and is configured in the scene by specifying its `type` as `ButtonActuator`, along with a dedicated statechart that describes its internal logic (including the `idle`, `pressed`, and `released` states) and associated behaviour.

The resulting behaviours, summarised in Table 6.2, reflect how each button performs different actions depending on the active mode, enabling modular and context-aware control of the system.

Logic and effects of button presses based on system state

State	Button	Action
`0`	`Start`	Triggers the Green Light of the Signal Pole and starts Conveyor 2, initiating the flow of parts toward the robot pickup area.
`1`	`Start`	Triggers the Yellow Light of the Signal Pole and instructs the Cartesian robot (PP) to execute a three-step sequence: `goPick` → `goPlace` → `goRest`, simulating the robotic pickup and placement of a part into the deposit box on Conveyor 1.
`2`	`Start`	Triggers the Red Light of the Signal Pole and starts Conveyor 1, allowing the completed boxes to exit the robot zone toward the unloading area.
`0 / 1 / 2`	`Stop`	Provides a mirrored logic for stopping the same elements: it halts conveyors, robot and lights depending on the active state selected.
`0 / 1 / 2`	`BP_Init`	Provides initialization functionality (visual and audio feedback), without directly affecting machine behaviour.
`0 / 1 / 2`	`Emergency Stop`	Performs an immediate and comprehensive shutdown: it stops both conveyors, resets the robot to its neutral position, and turns off all lights, offering a clear emergency response mechanism.

Table 6.2: System behaviour resulting from button interactions under different states.

From a user experience perspective, the interaction is immersive and intuitive. Users freely navigate the virtual cell using standard locomotion or teleportation, approach the Control Panel, and activate buttons via the laser pointer and trigger on the VR controller. This enables operational control in a natural way, effectively simulating real-world machine interfaces within a virtual environment. Figure 6.8 illustrates a typical interaction in which the user aims the right controller's laser at the Start button on the Control Panel, initiating the system's operational sequence.

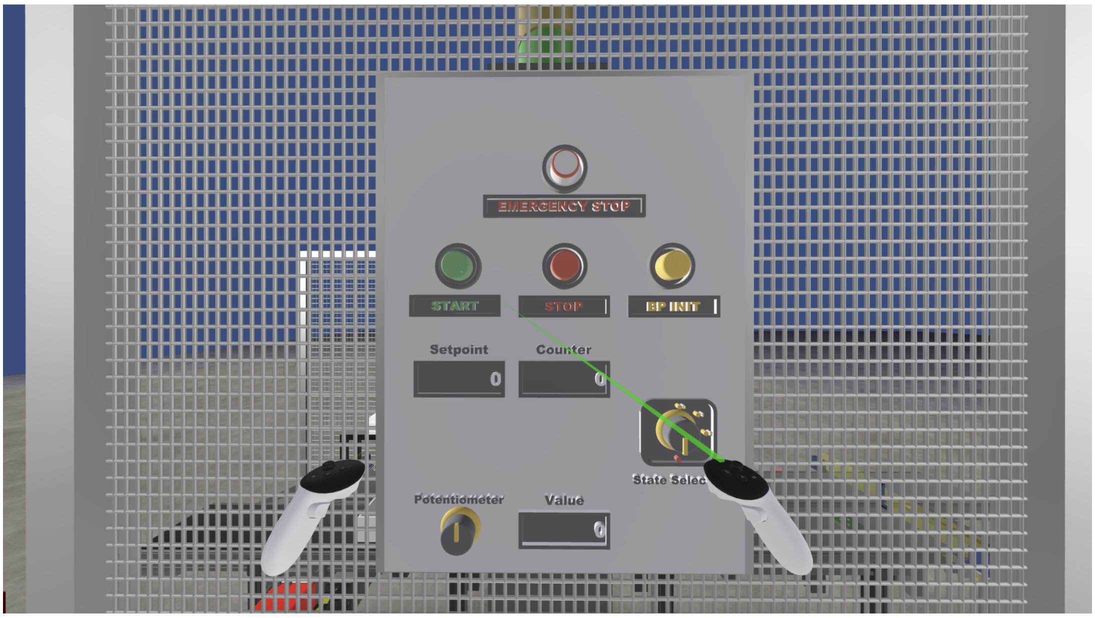


Figure 6.8: User aiming with the right controller's laser at the Start button of the Control Panel.

6.2.2. Rotary Switches and State Selection

The State Selector is a crucial component in the Pick & Place scenario as it enables users to switch between distinct operational modes. This component is modeled using the `RotarySwitchActuator` class defined in described in Section 4.2.2 and Section 5.2.2.

In this specific implementation, the rotary switch is embedded in the Control Panel and is configured by assigning it the type `RotarySwitchActuator`, along with a statechart composed of four cyclic states: `state0`, `state1`, `state2`, and `state3`.

Each activation of the rotary switch, triggered by user input, sends a `NEXT` event, causing a sequential transition to the next state and triggering the corresponding logic block. These transitions are accompanied by visual and auditory feedback, reinforcing the interaction. The following table 6.3 summarizes the semantics of each operational state defined by the rotary switch actuator.

Operational Modes Defined by the Rotary Switch

State	Description
`state0`	**Start Conveyor 2.** Represents the loading mode. The green light starts blinking, signaling the activation of Conveyor 1, which moves parts toward the robot area upon pressing the Start Button.
`state1`	**Trigger Robot Sequence.** Marks the pick-and-place mode. The yellow light blinks, and the green light is turned off. This state enables the execution of the robot cycle upon pressing the Start Button.
`state2`	**Start Conveyor 1.** Activates Conveyor 1 upon pressing the Start Button, moving completed boxes to the unloading area. The red light blinks while the yellow light is turned off.
`state3`	**Neutral State.** All lights are turned off. Represents the inactive or reset mode.

Table 6.3: Summary of the four operational modes controlled by the rotary switch actuator.

Each state transition is also accompanied by a `rotateTo` action, which animates the mesh to the corresponding angular position. This provides visual coherence between the virtual behavior and physical expectations of a mechanical dial.

From a user perspective, the rotary switch can be operated in VR by pointing the controller's laser beam at the selector and pressing the trigger button. Upon interaction, the dial rotates to the next position and the system provides immediate auditory feedback. This design fosters intuitive exploration and reinforces the perception of cause-effect relations, closely mimicking physical machine interfaces. Figure 6.9 shows the user interacting with the rotary State Selector by aiming the right controller's laser at it, thereby enabling mode transitions through trigger input.

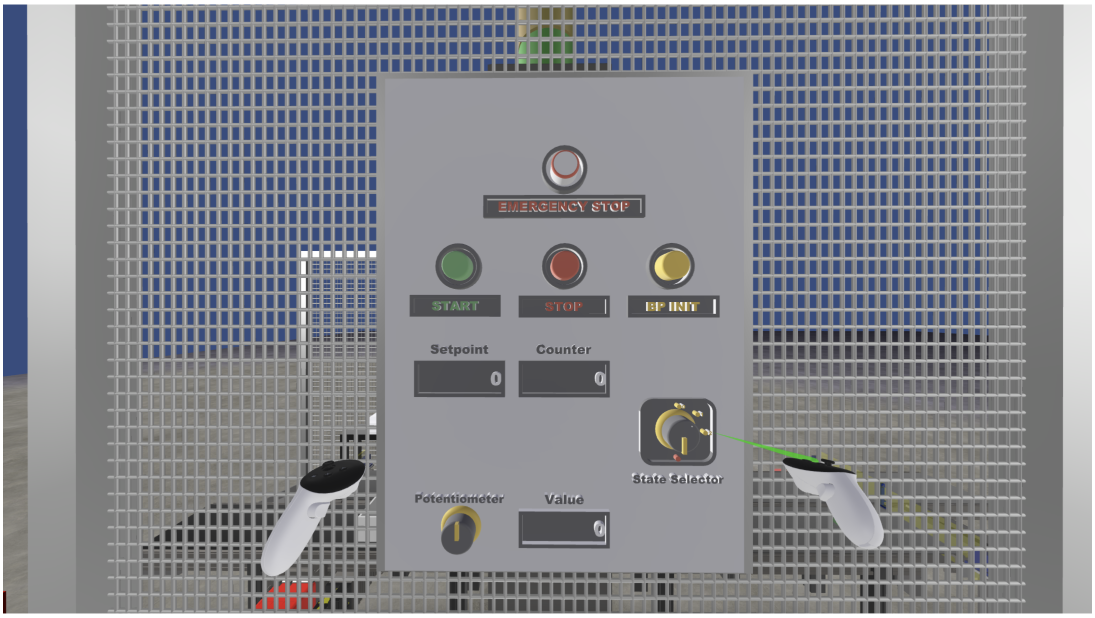


Figure 6.9: User pointing the right controller's laser at the rotary State Selector to cycle through operating modes.

6.2.3. Signal Pole System and Visual Feedback

In the Pick & Place cell, a set of three virtual signal lights provides immediate and intuitive visual feedback on the system's operational state. These components are defined using the `VisualLightActuator` class (Section 4.2.3 and 5.2.3), each supporting three core states: `off`, `on`, and `blink`.

The lights (green, yellow, and red) are integrated into a shared 3D model (`SignalPole.glb`) and are each configured as individual actuators. They are assigned the type `VisualLightActuator` and are controlled through a dedicated `statechart` that defines their behaviour, including blinking intervals and state transitions.

Each light responds to high-level events (`on`, `blink`, `turnAllOff`) triggered by other actuators such as the rotary switch or push buttons, creating a cohesive and modular interaction ecosystem.

Table 6.4 summarizes the correspondence between the operational modes selected via the `State Selector` and the visual feedback provided by the signal pole in the current implementation.

Mode-to-Light Feedback Mapping

State	Signal Pole Behaviour
`state 0`	Green Light blinks to indicate the system is in loading mode.
`state 1`	Yellow Light blinks to signal the pick-and-place mode is active.
`state 2`	Red Light blinks to represent that the box unloading mode is selected.
`state 3`	All lights are turned off, indicating a reset or standby state.

Table 6.4: Visual feedback provided by the signal pole for each operational mode.

This blinking pattern only signals state selection, i.e., the system is awaiting user confirmation. When the user presses the Start button (Green), the corresponding blinking light transitions to a solid `on` state, indicating that the system component has been activated.

Example:

- If the rotary switch is set to `state 0`, the Green Light blinks.
- When the Start Button (Green) is pressed, the system starts Conveyor 2 and the Green Light turns solid `on`.
- If the Stop Button (Red) is pressed, the conveyor stops and the Green Light returns to blinking, reflecting that the state is still selected but inactive.

Such feedback supports clear interaction transparency, helping users monitor system status at a glance and preventing ambiguity during multi-step tasks.

This color-to-state mapping is arbitrary by design and can be easily reconfigured. For example, in different scenarios the lights could instead encode Yellow as the selection phase, Green as system active, and Red as error or emergency.

This flexibility is a direct result of the declarative design: behaviors are encoded via statecharts and can be reassigned or extended without modifying the 3D models or logic source code.

By embedding these signaling elements within the VR environment, users gain clear visual cues about what part of the system is currently active, in a format that is both non-intrusive and highly intuitive, enhancing immersion and situational awareness.

Figure 6.10 illustrates the Signal Pole with its green light switched on, confirming that Mode 0 has been selected via the State Selector and activated by pressing the Start button.

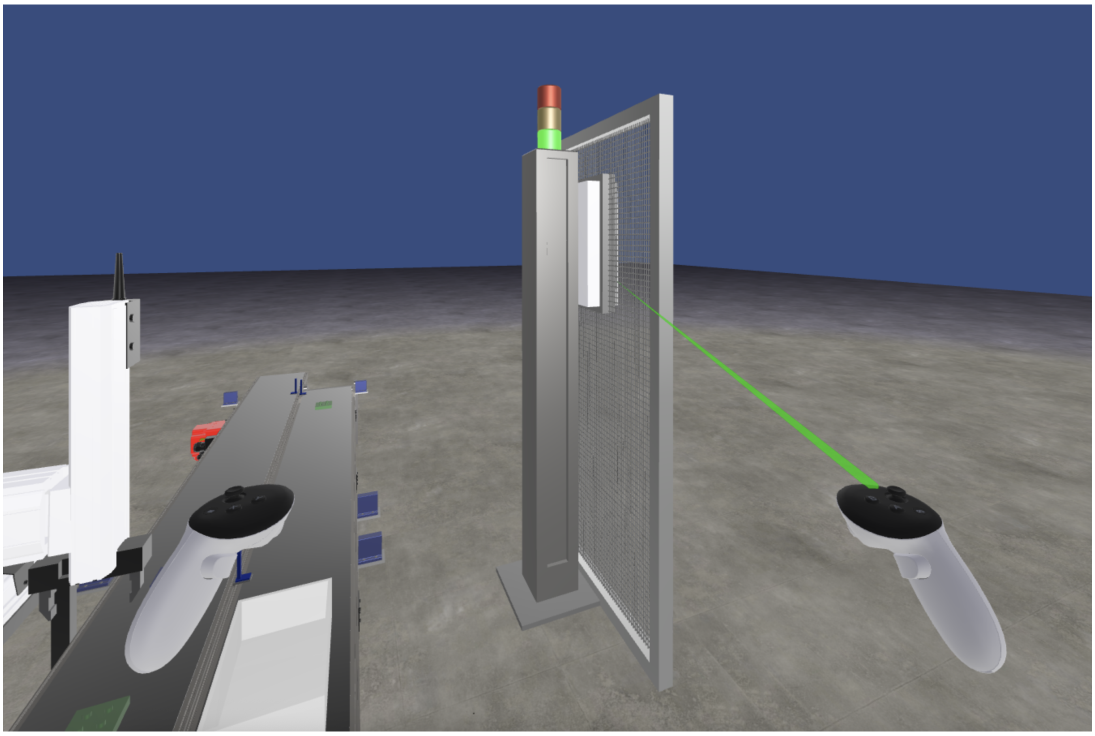


Figure 6.10: Signal Pole with the green light turned on, indicating that Mode 0 is active and the system has been started.

6.3. Application of Complementary Interaction Modalities

6.3.1. Object Grabbing and Guided Placement

In the Pick & Place scenario, users are allowed physically interact with virtual objects, such as boxes and items, using VR controllers. Among their properties, these objects include `gripping = true`, which enables them to respond to the `grip` button on the controller. This allows them to be picked up, moved freely, and placed within the environment in an intuitive and immersive manner.

When a object is selected via laser pointer targeting and grabbed, it becomes attached to the controller, synchronizing its position and rotation with the user's hand movements.

To facilitate structured interactions and accurate placement, the environment includes special `SnapZoneActuator` components, dedicated 3D regions that automatically detect and align objects upon release. These snap zones are positioned near the two conveyors, aligning with the expected workflow of the robotic cell:

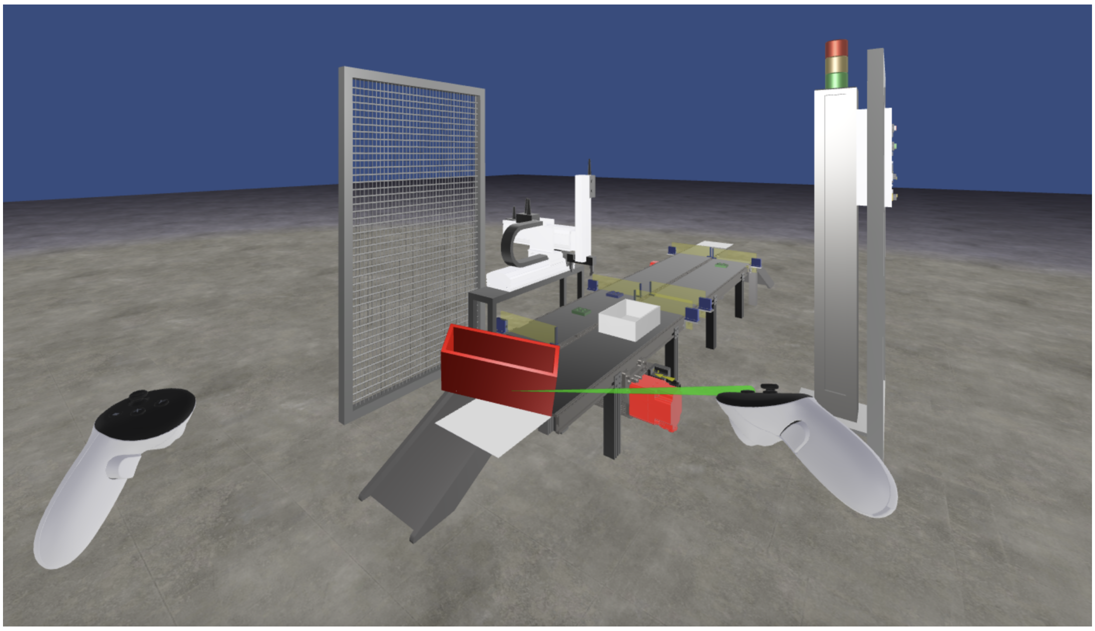
- `SnapZone_box` is located near Conveyor 1, designed to receive and orient boxes correctly before they are moved downstream.
- `SnapZone_part` is placed near Conveyor 2, meant to receive parts that will later be picked up by the robot.

Each snap zone is defined by a 3D mesh representation (a white thin surface), a target `SnapPosition` and `SnapRotation` that specify how the object should align upon release, and a `statechart` with two states: `inactive` and `active`.

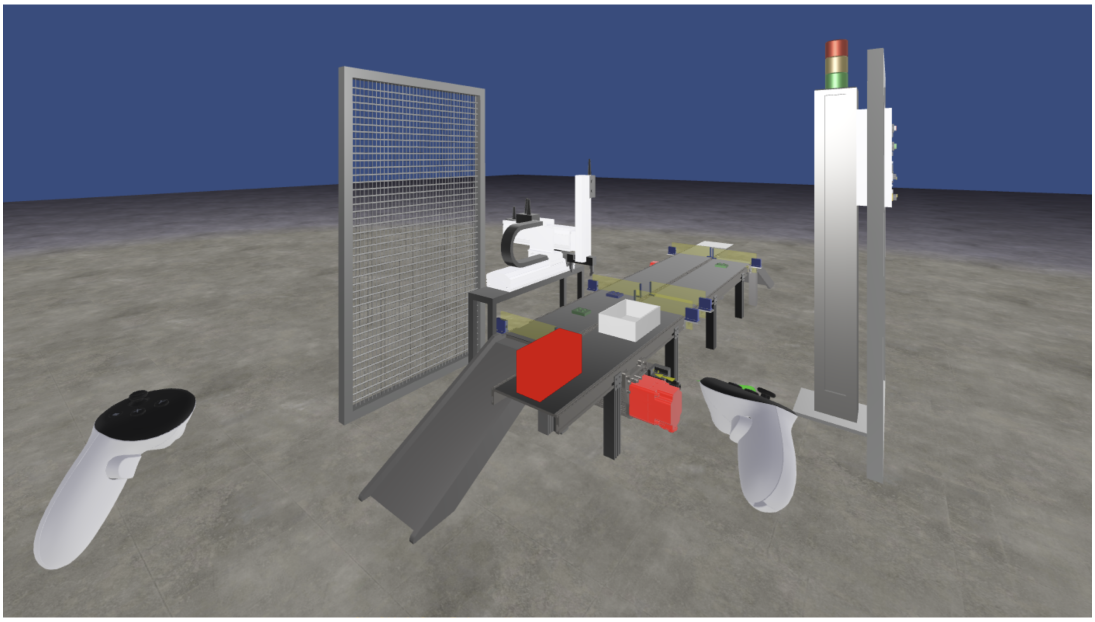
When an object is grabbed, the system programmatically activates and makes visible all Snap-Zones in the scene. Once the user releases the object near a valid Snap-Zone, it automatically snaps into the defined position and orientation, locking in place with precision.

This mechanism is essential for ensuring consistency and correctness in downstream tasks, such as ensuring objects are properly aligned for simulation logic and reducing user error in free placement

By decoupling interaction from exact manual positioning, Snap-Zones contribute to a more natural and robust experience while preserving the technical constraints of the virtual production environment.



(a) The user grabs a red box and moves it freely towards the SnapZone with arbitrary orientation.



(b) As soon as the box enters the SnapZone, it is repositioned and reoriented precisely on Conveyor 1.

Figure 6.11: Snapping interaction with a virtual box:(a) free positioning before contact and (b) automatic alignment upon release.

6.4. Application of Contextual Information Displays

6.4.1. Head Up Display and Information Panels

The information panels developed in this thesis, including the Main Control Panel, Head-Up Display (HUD), and iTablet, are designed as reusable components. Their logic, behavior, and visual layout are entirely decoupled from the geometry or structure of any given scene. For this reason, they are fully applicable within the *Pick & Place* environment, where they provide real-time feedback, user-controlled parameters, and interaction awareness during task execution. More importantly, the same panel system can be integrated into any other WebXR-based scene that requires immersive configuration or adaptive user feedback. This general-purpose architecture ensures continuity in the user experience while promoting scalability and cross-scene compatibility. Figure 6.12 shows an example of the immersive interface system in use within the *Pick & Place* scenario, with both the Head-Up Display and the iTablet visible during navigation and interaction.

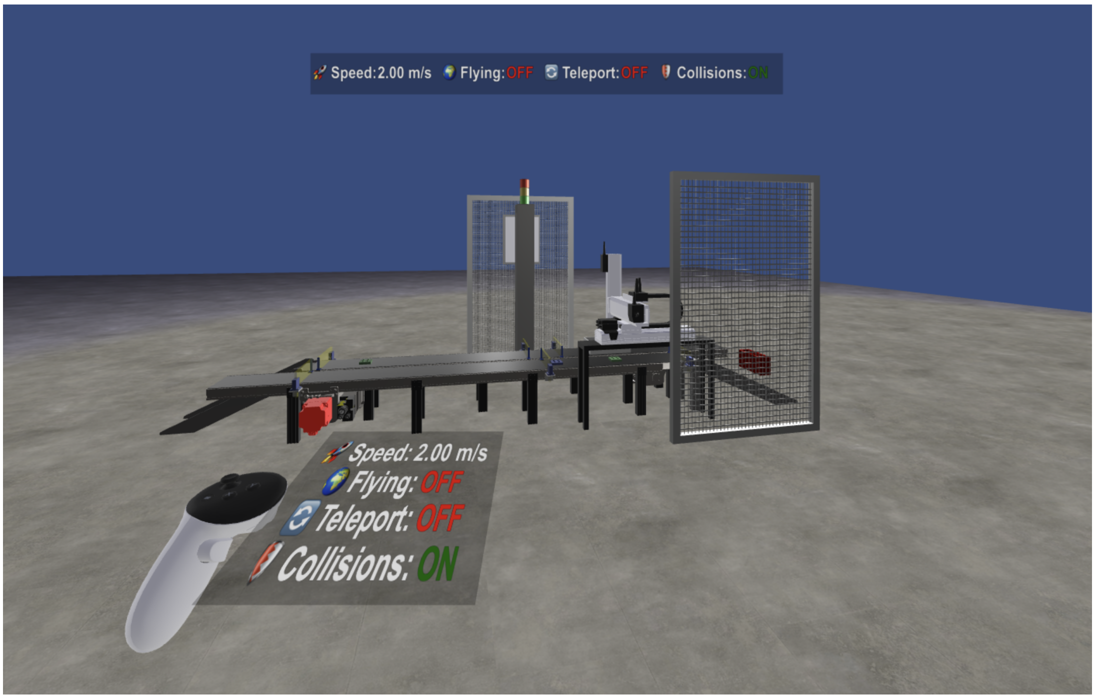


Figure 6.12: Example of user experience using immersive information panels. The Head-Up Display (HUD) is visible at the top of the field of view, while the iTablet is anchored to the left controller and appears when the user lifts their hand, simulating a watch-like motion.

6.5. User Testing of Immersive Interactions in VR

This section provides a structured evaluation of the proposed interaction framework, focusing on its effectiveness in terms of both usability and educational value. Building on the objectives defined in Chapter 3, the analysis explores how the system enables an intuitive and immersive user experience while promoting active learning and engagement. Particular attention is given to how the interactive features support a deeper understanding of industrial processes, reinforcing the role of virtual reality as a didactic tool for engineering education and enhancing the educational potential of digital twin environments.

6.5.1. Validation Approach and Methodological Basis

The validation process adopted an approach based on direct feedback collected from a group of 8 university students. Each participant took part in an individual immersive session in the *Pick & Place* scenario, executed through a Meta Quest 3S headset, which served as a representative environment for testing the interaction framework in a realistic VR setting.

During the experience, users were first guided through the navigation setup, configuring their preferred movement style, such as teleportation or free locomotion, via the Main Panel. This panel also allowed them to adjust gravity and collision settings in real time, testing the effectiveness and responsiveness of the Head-Up Display (HUD) and iTablet, which provided continuous visual feedback on active parameters.

The core of the test focused on direct interaction with industrial-style actuators, including push buttons and rotary selectors embedded in the Control Panel. These components were tested for responsiveness, clarity of feedback, and ease of use. Participants could observe the Signal Pole, which provided visual confirmation of state changes in response to user interactions with other components.

The session also involved complementary interaction modes such as grabbing and snapping. Users were asked to grab and move boxes or items within the virtual environment, including positioning them accurately onto predefined Snap-Zones, which automatically realigned the objects to simulate constrained placement. This part of the test aimed to assess the intuitiveness and physicality of the interaction, as well as the realism and usability of the supporting mechanics.

At the end of the session, each participant completed a set of validated questionnaires designed to assess various dimensions of their experience, including interest, sense of presence, realism, involvement, and flow state. These measures allowed for a comprehensive evaluation of the interaction framework, addressing both usability aspects and its didactic potential within an educational context. Table 6.5 summarizes the validated instruments adopted in the evaluation process, detailing the specific metrics used to assess both user experience and perceived educational value.

Validated Instruments for User Experience and Educational Impact

Perspective	Instrument / Source	Measured Dimension
User Experience	Intrinsic Motivation Inventory (IMI) [22]	Interest and Perceived Usefulness
User Experience	ITC-Sense of Presence Inventory (ITC-SOPI) [16]	Spatial Presence, Realism, Involvement
User Experience	Short Flow State Scale (SFSS) [14]	Flow Experience (focus, absorption, engagement)
Perceived Educational Value	Open-ended Feedback	Perceived impact on understanding of industrial systems

Table 6.5: Validated instruments used to evaluate user experience and educational value.

6.5.2. User Experience Evaluation

Table 6.6 summarizes the average scores reported by participants across the main evaluation metrics[1], along with their respective standard deviations. Each dimension was assessed using standardized instruments from the literature previously mentioned, enabling a structured analysis of user engagement, immersion, and perceived interaction quality within the VR environment. To complement the numerical overview, Figure 6.13 presents a bar chart visualization of the same data, offering an immediate comparison across the different experiential dimensions.

---

[1] Full questionnaire responses and dashboard available at [3]:

Summary of Average Scores and Standard Deviations

Metric (min=1 ; max=5)	**Mean ± SD**	**Instrument / Source**
Interest / Enjoyment	3.893 ± 0.544	IMI [22]
Sense of Spatial Presence	3.750 ± 0.518	ITC-SOPI [16]
Involvement	3.833 ± 0.642	ITC-SOPI [16]
Realism	3.750 ± 0.388	ITC-SOPI [16]
Flow State	3.778 ± 0.444	SFSS [14]

Table 6.6: Average participant scores on selected experiential metrics, based on validated instruments.

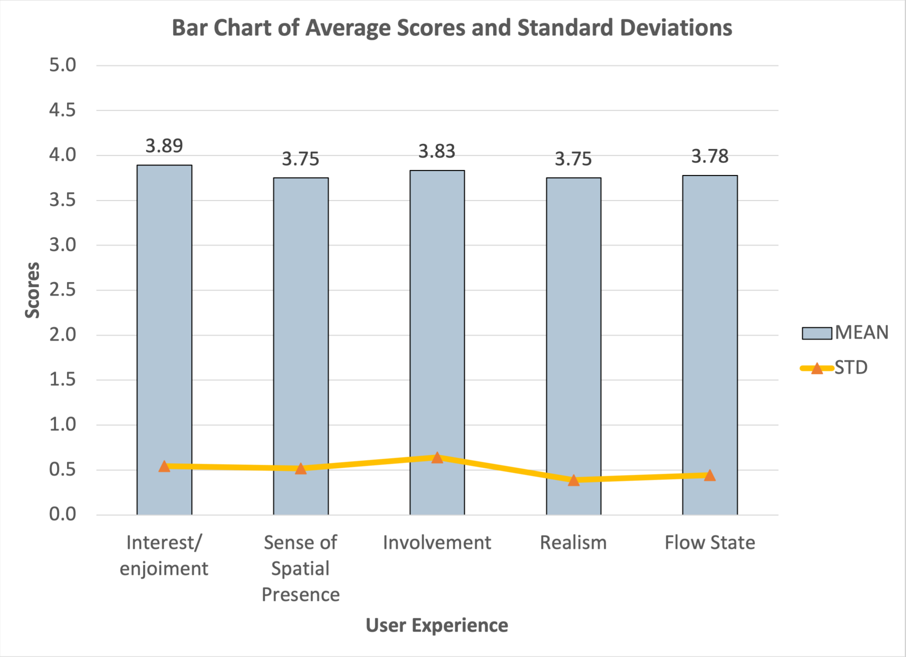


Figure 6.13: Bar chart visualization of average scores and standard deviations across experiential metrics.

Overall, the results reveal a consistently positive user experience, with all metrics scoring well above the midpoint of the scale (min = 1; max = 5). The highest score was observed in *Interest / Enjoyment* (M = 3.893 ± 0.544), indicating that the VR session was perceived as engaging and motivating. This high rating can be attributed to the interactive features implemented in the virtual environment, including the ability to press buttons, rotate selectors, manipulate physical-like components, and adjust movement parameters via real-time control panels. These actions promoted active participation and allowed users to feel in control of the experience, rather than observing predefined animations.

Similarly, the dimensions of *Flow State* (M = 3.778 ± 0.444) and *Involvement* (M = 3.833 ± 0.642) showed strong results, reflecting the participants' ability to remain focused and immersed throughout the session. These scores reinforce the educational value of the scene's interaction design: by enabling hands-on manipulation of core system components, the framework supports continuous cognitive engagement and promotes a sense of control. These dynamics are essential for fostering task-oriented attention and facilitating deeper conceptual understanding in engineering education.

Slightly lower, yet still satisfactory, scores were reported in *Sense of Spatial Presence* (M = 3.750 ± 0.518) and *Realism* (M = 3.750 ± 0.388). This suggests that while users felt immersed and the environment was generally believable, there may be room for improvement in terms of visual fidelity, environmental richness, or multisensory cues, which were often mentioned in qualitative feedback.

Importantly, the low standard deviations across all categories indicate a consistent perception among users, reinforcing the robustness of the design and the usability of the system across diverse user profiles. These results collectively validate the interaction framework's ability to deliver an intuitive and immersive experience, aligning well with the educational goals of promoting exploration, interaction, and understanding within a digital twin context.

6.5.3. Participant Reflections on the Usability and Learning Value

Participants generally expressed a positive impression of the immersive experience, emphasizing the clarity of feedback and the value of interacting directly with virtual components. Core elements such as buttons, selectors, and snapping mechanics were mentioned as engaging and effective in supporting user control. The opportunity to observe immediate visual responses, like the signal lights changing state or boxes snapping smoothly into position, helped reinforce a sense of participation and involvement. As one student noted, *"I liked how the system responded to my actions, especially with the animations of buttons, robots, conveyors, etc."*

The flexibility of the navigation system was also well received. Many participants appreciated being able to switch between teleportation and free movement, allowing them to adapt their experience according to comfort and preference. This was perceived as a helpful feature, especially in educational settings where user familiarity with VR tools may vary. One participant remarked that *"testing different ways to move around helped* *me understand what feels better in VR."*

Despite the overall positive feedback, several participants encountered moments of confusion or friction. For example, the teleportation feature was sometimes unclear at first, and the snapping behavior was perceived as too immediate in specific situations. One student commented that *"the snapping sometimes worked a bit too fast, I wasn't expecting the box to jump into place that quickly."* These issues underline the importance of including a brief in-VR tutorial or contextual hints to guide users, particularly beginners, through key functionalities such as movement configuration, grabbing mechanics, and control panel operations.

From an educational standpoint, the simulation was recognized as a valuable learning tool. Students highlighted the benefit of directly engaging with system logic in a safe and controlled environment, bridging the gap between theoretical content and applied understanding. The possibility to explore, test, and observe outcomes in real time offered a dynamic alternative to traditional lectures or textbook-based explanations.

Participants also suggested ways to improve the sense of immersion and realism. Several proposed enhancing the visual quality of the environment, introducing ambient background sounds, and incorporating haptic feedback during key interactions such as object grabbing. These additions, they argued, could further strengthen the sense of presence and make the experience more immersive and intuitive.

In conclusion, while the system was praised for its interactivity, clarity, and educational potential, the feedback also revealed opportunities for refinement, particularly in the initial approach and sensory feedback. These insights highlight the importance of balancing technical realism with intuitive design, especially when developing virtual environments for didactic purposes.

7 | Conclusions and Future Developments

7.1. Summary of Achieved Results

This thesis introduced a modular and configurable framework for designing and deploying immersive control systems in Virtual Reality, specifically aimed at supporting the education and training of industrial engineers within realistic digital twin environments. At its core, the proposed system addresses a critical challenge in engineering education: how to provide students with interactive, hands-on experiences that replicate the operational complexity of modern industrial settings, without relying on physical equipment or scripted simulations.

The approach is grounded in the use of finite state machines (FSMs) as a formalism to define the dynamic behaviour of interactive components. FSMs enable a structured and transparent modelling of stateful logic, where the evolution of each component is governed by discrete states, event-driven transitions, and declarative actions. By externalizing this logic through JSON configuration files, the system achieves a strong decoupling between behavioural control and visual rendering, allowing the same logic to be reused and remapped across different assets, scenes, and educational scenarios.

A layered class architecture was implemented to support modularity and reuse, starting from a general base class that encapsulates core functionality such as event interpretation and state subscription. This is extended by specialized actuator classes that define context-specific interactions like object movement, visibility toggling, or physical response. Components such as push buttons, rotary switches, indicator lights, and snapping zones were developed and integrated, each governed by its own FSM and capable of reacting to, or propagating, events across the system.

The coherence of this architecture enables the orchestration of complex behaviour sequences that mimic industrial workflows, where user actions on one element can propagate through state transitions to affect other devices. This event-driven, modular logic provides not only behavioural depth but also high configurability, empowering educators and developers to model varied scenarios without modifying the core implementation.

The full framework was validated within a functional Pick & Place simulation cell. In this context, all developed components were instantiated and coordinated under realistic spatial and logical constraints. The cell reproduced basic features of a manufacturing process, allowing users to interact with virtual machines through logic-driven interfaces. This setup served both as a technical testbed and a pedagogical demonstrator, enabling real-time interaction, exploration, and reflection.

Empirical validation through user testing further confirmed the system's effectiveness. Participants consistently reported high levels of usability, involvement, and enjoyment, highlighting the platform's potential for enhancing engagement and learning outcomes in engineering training.

In summary, this work represents a step toward the development of immersive educational tools that combine configurability, operational realism, and pedagogical relevance, aiming to help bridge the gap between abstract logic and embodied interaction within the evolving context of Industry 4.0.

7.2. Technical Limitations and Open Issues

Despite the encouraging results and the robustness of the proposed framework, several limitations and open issues still need to be addressed to fully exploit its potential:

- **Partial support for dynamic reconfiguration**: Currently, modifying the logic or replacing an asset requires direct intervention in the configuration files and partial adjustments to the codebase. The system lacks a user-friendly runtime interface or graphical dashboard that would enable real-time reconfiguration or scenario editing, limiting its accessibility for non-technical users and educators.

- **Gesture-based interaction and hand tracking:** While the system supports basic controller-based input, more advanced interaction modes such as hand tracking and gesture recognition are not yet implemented. This limits the expressiveness and naturalness of the interaction, especially in scenarios where fine motion skills or direct hand-based manipulation would enhance realism and training effectiveness.

- **Limited visual and auditory immersion**: The visual environment, while functional, is relatively minimalistic. Similarly, the absence of auditory feedback (e.g., machine noise, ambient cues) can reduce the sense of realism and presence, especially in industrial simulations where multisensory cues play a key role in perception and safety.

- **No support for collaborative or multi-user sessions**: The current architecture is designed for single-user deployment. In modern industrial training contexts, collaborative work is often essential. The lack of multi-user synchronization or shared simulation environments limits the framework's potential for team-based scenarios or remote collaborative learning.
- **Scalability to more complex systems**: While the Pick & Place cell demonstrates the validity of the approach, extending the framework to larger systems with dozens of interacting machines and parallel control flows might reveal new integration and performance challenges, especially concerning state machine orchestration and event propagation.
- **Lack of onboarding and in-VR guidance**: Although the system is aimed at educational contexts, there is currently no integrated onboarding mechanism or tutorial within the VR environment. First-time users may struggle to understand the interaction logic, movement systems, or control options without prior instruction.

7.3. Future Developments

Building upon the current results, several directions for future development can be envisioned to extend the system's capabilities and address its limitations:

- **Development of a visual configuration dashboard**: To make the system more accessible and reduce the reliance on manual editing, a graphical interface could be implemented for configuring and binding interactive components. This would allow users, especially non-technical stakeholders, to modify logic structures, assign behaviours, or test new scenarios at runtime, without touching the underlying code.
- **Integration of hand tracking and gesture-based controls:** Future developments could focus on enabling hand tracking and gesture recognition to allow more natural and intuitive interactions. Replacing or complementing controller input with direct hand gestures would enhance realism, support more expressive training tasks, and improve accessibility for users less familiar with VR controllers.
- **Improved environmental fidelity**: Adding environmental details such as ambient industrial soundscapes, machine operation noises, and subtle visual effects (e.g., smoke, wear) could greatly enhance presence and sensory immersion. These addi-

tions would also support the simulation of real-world conditions and improve the detection of anomalies.

- **Support for multi-user and collaborative scenarios**: Extending the platform to support multiple users within the same virtual environment would enable collaborative learning, peer interaction, and role-based simulations. This functionality could be achieved through networked synchronization of state charts and shared event propagation mechanisms.
- **In-VR onboarding and guided training**: The inclusion of contextual tutorials, help overlays, or animated assistants within the VR environment would help first-time users understand movement systems, control modes, and task objectives. Such support would be essential for improving usability and maximizing the system's educational impact.

These developments would significantly expand the platform's versatility, making it more immersive, accessible, and adaptable to a broader range of industrial training scenarios.

Bibliography

[1] Amazon.it. Meta quest 3s – all-in-one vr headset (128 gb), 2025. URL `https://www.amazon.it/dp/B09MJRPRR4`. Accessed: 8 June 2025.

[2] Babylon.js Team. Babylon.js documentation, 2025. URL `https://doc.babylonjs.com/`. Accessed: 8 June 2025.

[3] S. Benedet. Immersive vr simulation questionnaire – aggregated results, 2025. URL `https://forms.cloud.microsoft/Pages/AnalysisPage.aspx?AnalyzerToken=xBY3Bh6fiz1saJCzbETlsfweV3MuaKqL&id=K3EXCvNtXUKAjjCd8ope648M_8vPbjNDrzeJmrlER2RUOVlOWDdKVjFGWjlaTVQ4NOZURldZSDY4WC4u`. Accessed: 2025-06-21.

[4] F. Berardinucci, G. Colombo, M. Lorusso, M. Manzini, W. Terkaj, and M. Urgo. A learning workflow based on an integrated digital toolkit to support education in manufacturing system engineering. *Journal of Manufacturing Systems*, 63:411–423, 2022. ISSN 0278-6125. doi: 10.1016/j.jmsy.2022.04.003. URL `https://www.sciencedirect.com/science/article/pii/S027861252200053X`.

[5] Blender Foundation. *Blender – a 3D Modelling and Rendering Software*. Blender Foundation, 2023. Version 3.6. Available at: `https://www.blender.org` (Accessed: 8 June 2025).

[6] D. A. Bowman, D. B. Johnson, and L. F. Hodges. Testbed evaluation of virtual environment interaction techniques. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 26–33, 1999. doi: 10.1145/323663.323667. URL `https://doi.org/10.1145/323663.323667`.

[7] CGTrader. CNC Lathe SMTCL KE80-2000, n.d. URL `https://www.cgtrader.com/3d-models/industrial/industrial-machine/cnc-lathe-smtcl-ke80-2000`. Accessed: 8 June 2025.

[8] L. Evans. *Barriers to VR Use in HE*, pages 3–13. 5 2019. ISBN 9781906715281. doi: 10.1255/vrar2018.ch2. URL `https://www.researchgate.net/publication/333502677_Barriers_to_VR_use_in_HE`.

[9] R. Ferdig and B. Winn. The design, play, and experience framework. In S. Ferris, editor, *Handbook of Research on Effective Electronic Gaming in Education*, pages 1010–1024. IGI Global, Jan. 2009. doi: 10.4018/978-1-59904-808-6.ch058. URL `https://scispace.com/pdf/the-design-play-and-experience-framework-5501m7nkj6.pdf`.

[10] S. Frees and G. D. Kessler. Precise and rapid interaction through scaled manipulation in immersive virtual environments. In *Proceedings of IEEE Virtual Reality 2005*, pages 99–106, 2005. doi: 10.1109/VR.2005.1492759. URL `https://doi.org/10.1109/VR.2005.1492759`.

[11] N. Gavish, T. Gutierrez, S. Webel, J. Rodríguez-Arce, M. Peveri, and F. Tecchia. Evaluating virtual reality and augmented reality training for industrial maintenance and assembly tasks. *Interactive Learning Environments*, 23:1–21, Dec. 2013. doi: 10.1080/10494820.2013.815221. URL `https://doi.org/10.1080/10494820.2013.815221`.

[12] GrabCAD Community. Festo g 1/4 analogue pressure gauge 10bar back entry Ø40mm, 2021. URL `https://grabcad.com/library/festo-g-1-4-analogue-pressure-gauge-10bar-back-entry-40mm-outside-diameter-1`. Accessed: 2025-06-13.

[13] T. Hoang, S. Greuter, and S. Taylor. An evaluation of virtual reality maintenance training for industrial hydraulic machines. In *Proceedings of the 2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 573–581, March 2022. doi: 10.1109/VR51125.2022.00077. URL `https://doi.org/10.1109/VR51125.2022.00077`.

[14] S. A. Jackson, A. J. Martin, and R. C. Eklund. Long and short measures of flow: The construct validity of the fss-2, dfs-2, and new brief counterparts. *Journal of Sport Exercise Psychology*, 30(5):561–587, 2008. doi: 10.1123/jsep.30.5.561. URL `https://doi.org/10.1123/jsep.30.5.561`.

[15] C. Johnson, N. Fraulini, E. Peterson, J. Entinger, and D. Whitmer. Exploring hand tracking and controller-based interactions in a vr object manipulation task. In *Human Interaction and Emerging Technologies: Proceedings of the 6th International Conference on Human Interaction and Emerging Technologies (IHIET 2023)*, pages 64–81, December 2023. ISBN 978-3-031-48049-2. doi: 10.1007/978-3-031-48050-8_5. URL `https://doi.org/10.1007/978-3-031-48050-8_5`.

[16] J. Lessiter, J. Freeman, E. Keogh, and J. Davidoff. A cross-media pres-

ence questionnaire: The itc-sense of presence inventory. *Presence*, 10:282–297, 06 2001. doi: 10.1162/105474601300343612. URL `https://doi.org/10.1162/105474601300343612`.

[17] Meta Platforms Inc. *Meta XR Design Handbook*, 2023. URL `https://xrdesignhandbook.com/docs/Meta/User%20Input.html`. Accessed: 21 June 2025.

[18] Meta Platforms Inc. Meta quest 3s – technical specifications, 2024. URL `https://www.meta.com/it/quest/quest-3s/`. Accessed: 8 June 2025.

[19] A. Murillo, S. Subramanian, and D. Martinez Plasencia. Erg-o: Ergonomic optimization of immersive virtual environments. Conference contribution, University of Sussex, 2017. URL `https://hdl.handle.net/10779/uos.23448065.v1`. Accessed: June 2025.

[20] Politecnico di Milano. Digital factory – official course page (politecnico di milano), 2025. URL `https://www11.ceda.polimi.it/schedaincarico/schedaincarico/controller/scheda_pubblica/SchedaPublic.do?&evn_default=evento&c_classe=767359&__pj0=0&__pj1=adf72b6235aaa53500eff8425f3723c4`. Accessed June 2025.

[21] I. Poupyrev, T. Ichikawa, S. Weghorst, and M. Billinghurst. Egocentric object manipulation in virtual environments: Empirical evaluation of interaction techniques. *Computer Graphics Forum*, 17:41–52, 1998. doi: 10.1111/1467-8659.00252. URL `http://www.ivanpoupyrev.com/wp-content/uploads/2016/10/EG98.pdf`.

[22] R. M. Ryan and E. L. Deci. Intrinsic motivation inventory (imi). `https://selfdeterminationtheory.org/intrinsic-motivation-inventory/`, 2023. Accessed: 2025-06-21.

[23] S. Samsuri, N. N. Misman, A. Borhan, W. Z. N. Yahya, and W. N. A. Wan Osman. Power plant tour: From physical field trip to virtual reality. *Jurnal Penelitian Dan Pengkajian Ilmu Pendidikan: E-Saintika*, 8(3):352–372, 2024. doi: 10.36312/e-saintika.v8i3.2294. URL `https://doi.org/10.36312/e-saintika.v8i3.2294`.

[24] G. Sayeg, N. Amado-Moranchel, and A. Acero. Use of virtual reality to improve learning experience on a lean manufacturing course. In *Proceedings of the 2024 ASEE Annual Conference Exposition*, 06 2024. doi: 10.18260/1-2--48221. URL `https://peer.asee.org/48221`.

[25] Stately. Stately.ai – visual editor for state machines and statecharts. `https://stately.ai`, 2025. Accessed: 8 June 2025.

[26] R. T. Stone, K. P. Watts, P. Zhong, and C.-S. Wei. Physical and cognitive effects of virtual reality integrated training. *Human Factors*, 53(5):558–572, 2011. doi: 10.1177/0018720811413389. URL `https://doi.org/10.1177/0018720811413389`.

[27] A. Susilo, L. Barra, and Y. Wang. Use of virtual reality technology for learning mechanical skills. *Journal of Computer Science Advancements*, 2:259–272, 10 2024. doi: 10.70177/jsca.v2i5.1323. URL `https://doi.org/10.70177/jsca.v2i5.1323`.

[28] W. Terkaj. Pick and place cell configuration file, 2025. URL `https://raw.githubusercontent.com/BenedetSergio/Tesi/main/PickPlaceCell.xlsm`. Accessed: 2025-06-21.

[29] W. Terkaj. *VEB.js Documentation*, 2025. URL `https://virtualfactory.gitbook.io/vlft/tools/vebjs`. Accessed: 2025-06-28.

[30] W. Terkaj and G. P. Viganò. Semantic giove-vf: An ontology-based virtual factory tool. In *Proceedings of the Workshop Data Meets Applied Ontologies, Joint Ontology Workshops 2017 (JOWO)*, volume 2050 of *CEUR Workshop Proceedings*, Bolzano, Italy, Sept. 2017. CEUR-WS.org. URL `https://www.researchgate.net/publication/320078018_Semantic_GIOVE-VF_an_Ontology-based_Virtual_Factory_Tool`.

[31] W. Terkaj, M. Urgo, P. Kovács, et al. A framework for virtual learning in industrial engineering education: Development of a reconfigurable virtual learning factory application. *Virtual Reality*, 28:148, 2024. doi: 10.1007/s10055-024-01042-8. URL `https://doi.org/10.1007/s10055-024-01042-8`.

[32] W3C. Webxr device api, 2025. URL `https://www.w3.org/TR/webxr/`. Accessed: 8 June 2025.

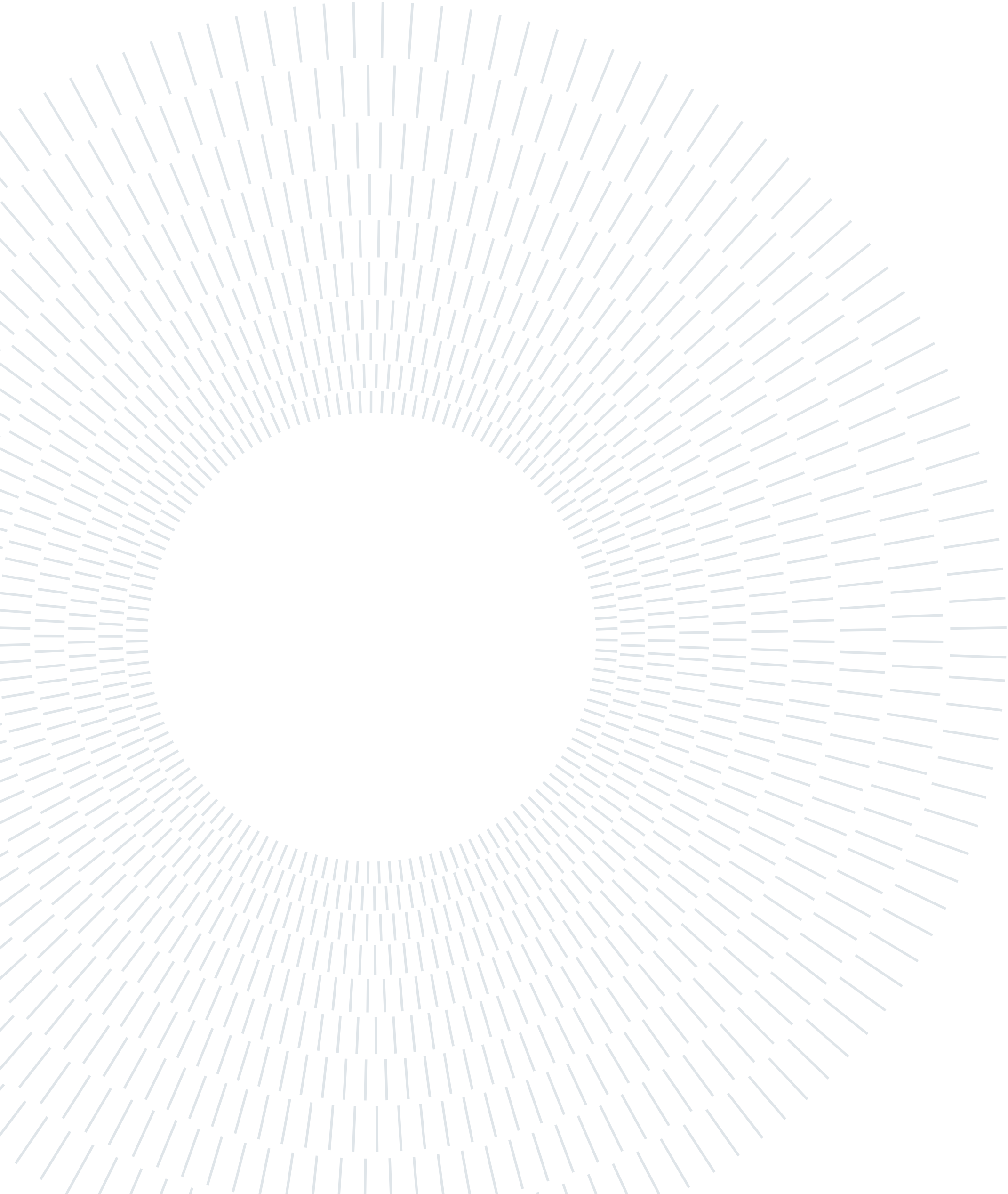
[33] WebXR Emulator Extension. Webxr emulator extension for chrome, 2024. URL `https://chromewebstore.google.com/detail/immersive-web-emulator/cgffilbpcibhmcfbgggfhfolhkfbhmik`. Accessed: 8 June 2025.

[34] M. Weise, R. Zender, and U. Lucke. How can i grab that?: Solving issues of interaction in vr by choosing suitable selection and manipulation techniques. *i-com*, 19: 67–85, Aug. 2020. doi: 10.1515/icom-2020-0011. URL `https://doi.org/10.1515/icom-2020-0011`.

[35] F. Wu, W. Lu, X. Wang, J. Yang, C. Li, L. Cheng, and Z. Xie. Enhanced virtual reality plant: Development and application in chemical engineering education. *International Journal of Emerging Technologies in Learning (iJET)*, 17(14):205–

220, 2022. doi: 10.3991/ijet.v17i14.23407. URL `https://doi.org/10.3991/ijet.v17i14.23407`.

[36] D. Yu, H.-N. Liang, F. Lu, V. Nanjappan, K. Papangelis, and W. Wang. Target selection in head-mounted display virtual reality environments. *Journal of Universal Computer Science*, 24(9):1217–1243, Sept. 2018. doi: 10.3217/jucs-024-09-1217. URL `https://doi.org/10.3217/jucs-024-09-1217`.



A | Appendix A

Mesh Optimization and Pivot Adjustment in Blender

A critical aspect of integrating 3D assets into the immersive WebXR scene developed in this thesis was the proper configuration of their internal hierarchy and pivot points. While most CAD tools are optimized for mechanical design and engineering accuracy, their export formats, especially when converted into `.glb` for visualization, tend to include complex and deeply nested node trees. These structures, although technically valid, are not always compatible with real-time environments where interactive transformations must be applied efficiently and predictably.

In particular, assets such as interactive components (e.g., rotary knobs, push buttons, switches) are frequently built in CAD software using assemblies and sub-assemblies, each with its own local reference frame. As a result, when such models are imported into Blender or directly into a Babylon.js scene, they may include redundant or unnecessary intermediary nodes. For instance, a simple component like a toggle switch might appear in the scene hierarchy as:

`ToggleSwitch` → `SubAssembly_A` → `Shell_Group` → `ShellMesh`

In this example, only `ShellMesh` represents the final geometry to be manipulated, while the upper nodes are purely structural and do not contribute directly to the visual or interactive behavior of the object.

To address this complexity, Blender was used as a pre-processing and optimization tool. The first step involved identifying the leaf mesh nodes within the imported `.glb` file, those that carry actual geometry, and separating them from their parent containers. This was done using the operation:

`Object` → `Parent` → `Clear and Keep Transform`

This operation detaches the mesh while preserving its current position, orientation, and scale in world space. Once isolated, the mesh was either renamed (e.g., `ToggleSwitchShell`) or reassigned directly to a top-level logical node (e.g., `ToggleSwitch`) that would serve as the unique identifier within the Veb.js configuration framework.

This flattening process brought several advantages: it simplified the scene graph, reduced hierarchy depth, minimized transformation inheritance issues, and enabled one-to-one mapping between the graphical asset and its logical representation in the JSON-based scene description.

In addition to hierarchy simplification, Blender proved essential for correcting the pivot points (origins) of individual mesh elements. CAD-exported models frequently include global origin references that reflect the positioning of the entire device, rather than the local interaction point of a specific component. This misalignment is problematic in immersive environments, where actions such as `rotateTo`, `moveTo`, or grabbing rely on an accurate pivot point.

To resolve this, Blender was employed to manually reposition the pivot points of all interactive elements. The procedure was:

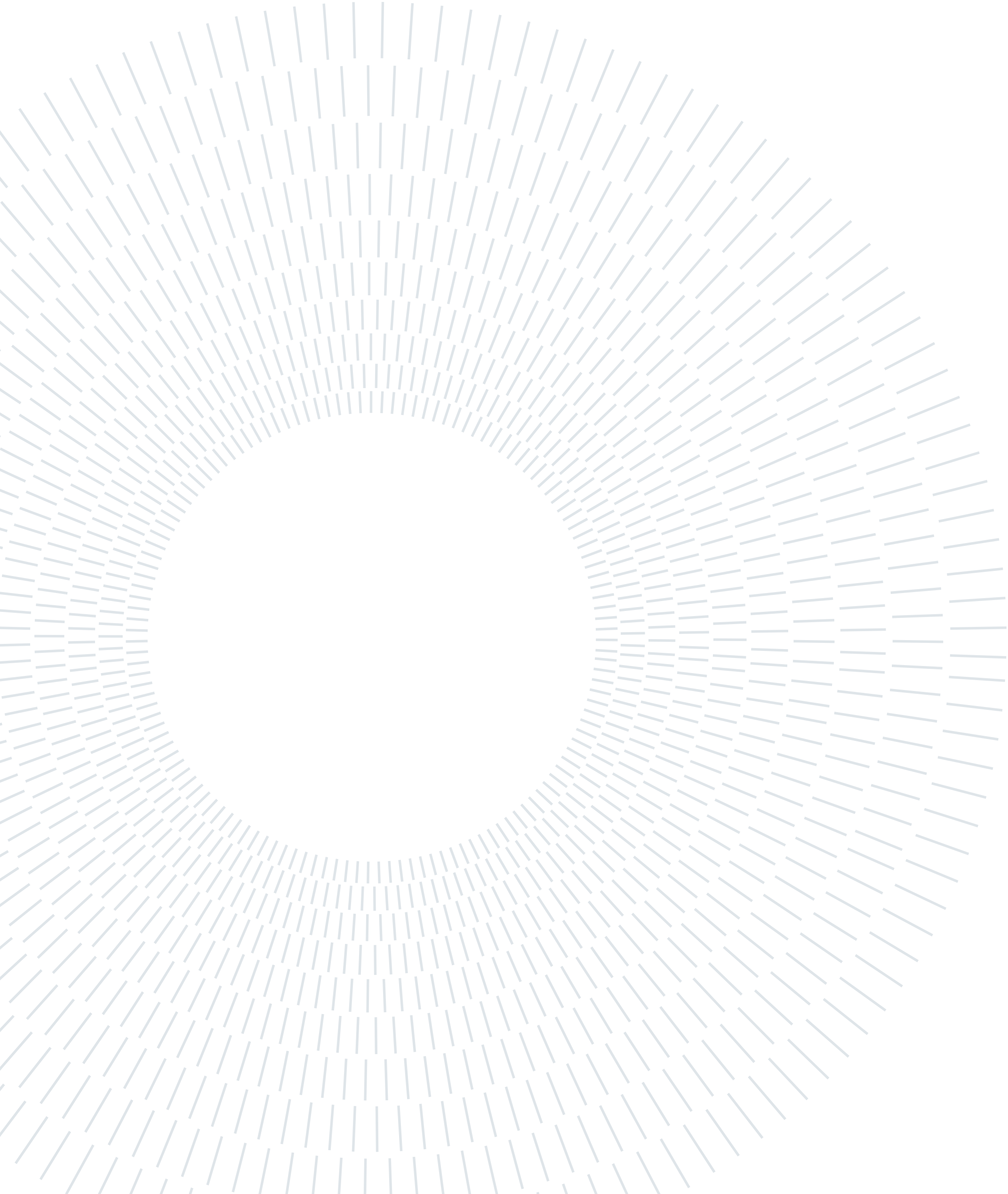
1. Enter `Edit Mode` for the selected mesh.
2. Select all geometry using `A`.
3. Move the 3D cursor to the center using `Shift + S` → `Cursor to Selected`.
4. Exit `Edit Mode`.
5. Set the object origin with `Object` → `Set Origin` → `Origin to 3D Cursor`.

This ensured that each mesh behaved coherently in the VR scene: rotations occurred around expected axes, snapping effects were precisely aligned, and visual feedback matched the physical affordances of the virtual interface. Without this adjustment, the scene would have exhibited unintuitive or unrealistic behavior, reducing the effectiveness of the simulation.

In summary, Blender played a foundational role in ensuring that all imported 3D models were structurally clean, geometrically centered, and properly configured for the modular animation and interaction system implemented in this thesis.

List of Figures

List of Tables



Acknowledgements

I would like to express my sincere gratitude to Dr. Marcello Urgo and Dr. Walter Terkaj for their continuous support, availability, and valuable guidance throughout the development of this thesis. Their mentorship has been instrumental in achieving the objectives of this work.