Department of Enterprise Engineering

Doctor of Philosophy

in Design, Manufactoring and Operations Egineering

Development of Virtual and Augmented Reality Methodologies for the Building and the Interactive Interrogation of Digital Twins

PON R&I
XXXIX Cycle

Supervisor
Prof. Pier Paolo Valentini

Co-Supervisor
Ing. Marco Cirelli

Coordinator
Prof. Francesco Vivio

Candidate
Alessio Cellupica

A.Y. 2025/2026

A Miryam, Miriana e ai miei genitori...

Ringraziamenti

Giunto al termine di questo percorso, sento il desiderio di rivolgere un sincero ringraziamento a tutti coloro che, direttamente o indirettamente, hanno reso possibile il raggiungimento di questo traguardo.

Un ringraziamento speciale è rivolto alla mia compagna, Miryam, che ha camminato al mio fianco durante tutto questo percorso, condividendone ogni gioia, ogni difficoltà e ogni traguardo. La sua vicinanza è stata una presenza costante, capace di sostenermi nelle decisioni più importanti e di infondermi fiducia anche nei momenti in cui tutto sembrava più complicato.Le sono profondamente grato per il tempo, l'affetto e la pazienza che mi ha dedicato, ma soprattutto per aver scelto di esserci ogni giorno, senza mai farmi mancare il suo sostegno.

Desidero esprimere la mia più sincera gratitudine a mia sorella Miriana e a mio cognato Marco, la cui presenza costante ha rappresentato per me un punto di riferimento fondamentale. Con il loro affetto, la loro fiducia e il loro incoraggiamento, mi hanno sostenuto nei momenti di incertezza e accompagnato lungo il percorso delle mie decisioni. Grazie alla loro vicinanza, ho trovato la determinazione per affrontare gli ostacoli, rialzarmi dopo le difficoltà e proseguire con maggiore serenità verso i miei obiettivi.

Un ringraziamento speciale va ai miei genitori, che mi hanno accompagnato lungo questo percorso con una presenza silenziosa, ma costante. Con i loro sacrifici hanno contribuito al raggiungimento di questo traguardo.

Desidero rivolgere un sentito ringraziamento ai miei amici, che con la loro spensieratezza e la loro capacità di rendere più leggeri anche i momenti più impegnativi hanno reso questo percorso più sereno.

Desidero esprimere la mia sincera riconoscenza al Professor Pier Paolo Valentini per il costante supporto, la grande professionalità e la disponibilità dimostrata durante lo sviluppo di questo lavoro. La sua guida attenta, i suoi consigli puntuali e la sua competenza sono stati fondamentali per affrontare questo percorso con maggiore consapevolezza e determinazione.

Un pensiero speciale al Prof. Pennestrì, una figura che ha avuto un ruolo significativo nel mio percorso di formazione e crescita accademica. La sua passione per l'insegnamento, la sua umanità e la sua capacità di ispirare continueranno a essere un punto di riferimento vivo e prezioso per me.

Un ringraziamento speciale è rivolto al Professor e amico Marco Cirelli, per essere stato una presenza costante durante questo percorso, offrendo non solo un prezioso sostegno professionale, ma anche un importante supporto umano.

Desidero rivolgere un sincero ringraziamento ai miei colleghi Luca, Matteo e Gio-

vanni, con i quali ho condiviso una parte fondamentale del mio percorso di ricerca. Il confronto quotidiano, la collaborazione e la disponibilità reciproca hanno reso ogni sfida più affrontabile e questa esperienza ancora più stimolante e significativa. La loro presenza ha contribuito a rendere questo percorso un'esperienza non soltanto formativa dal punto di vista professionale, ma anche ricca sotto il profilo umano. Un ringraziamento particolare è dedicato a Pietro, compagno di laboratorio e prezioso punto di riferimento durante questi anni. La sua presenza, la sua disponibilità e il suo sostegno nei momenti più impegnativi sono stati un aiuto fondamentale. La capacità di affrontare insieme le difficoltà quotidiane, trasformando anche le situazioni più complesse in occasioni di confronto e crescita, ha reso questo percorso più leggero e sicuramente più significativo.

Grazie di cuore a tutti voi.

Contents

1 Introduction **3**
1.1 Problem of interest and scope . . . 3
1.2 Main contributions of the dissertation . . . 4
1.3 Organization of the dissertation . . . 5

2 Digital twin and extended reality **7**
2.1 Digital twin . . . 7
2.1.1 Digital twin architectures . . . 9
2.1.2 Data synchronization and interoperability . . . 10
2.1.3 Digital twin modelling and validation . . . 10
2.1.4 Digital twins and Industry 5.0 . . . 11
2.2 Extended reality . . . 12
2.2.1 Historical evolution of extended reality . . . 12
2.2.2 XR devices and tracking technologies . . . 15
2.2.3 Visualization of 3D virtual objects . . . 21

3 Objects tracking methodologies **26**
3.1 Vision-based tracking . . . 28
3.1.1 Image-based tracking . . . 29
3.1.2 Model-based tracking . . . 33
3.2 Sensor-based tracking . . . 35

4 Interaction methodologies **39**
4.1 Interaction detection . . . 40
4.1.1 Analytic distance and closest-point methods . . . 41
4.1.1.1 Sphere-sphere interaction . . . 41
4.1.1.2 Capsule-capsule interaction . . . 41
4.1.1.3 Sphere-AABB interaction . . . 42
4.1.1.4 Sphere-OBB interaction . . . 42
4.1.2 Separating axis theorem . . . 42
4.1.3 Support mapping methods . . . 44
4.1.3.1 Minkowski difference . . . 44
4.1.3.2 The support function . . . 45
4.1.3.3 Gilbert-Johnson-Keerthi algorithm . . . 46
4.1.3.4 Expanding polytope algorithm . . . 48

4.2 Interaction response . . . 50
4.2.1 Multibody contact formulation . . . 50
4.2.2 Real-time XR engines . . . 51
4.3 Interaction methods . . . 52
4.3.1 Controller-based interaction . . . 52
4.3.2 Hand tracking . . . 53
4.3.2.1 Grasping active features/object active features methodology . . . 59
4.3.3 Haptic devices . . . 61
4.3.3.1 Geomagic Touch X . . . 63
4.3.3.2 SenseGlove Nova 2 . . . 64

5 Interactive modelling 65
5.1 AR interactive modelling framework based on subdivision surfaces . . . 68
5.1.1 Case study: interactive modelling using Varjo XR-4 and Ultraleap Motion Controller . . . 76

6 Virtual assembly and industrial applications 81
6.1 Evaluation of VR device usability in industrial applications . . . 83
6.1.1 Description of usability tests in human–system Interaction . . . 85
6.1.2 Usability analysis results . . . 99
6.2 Case study: industrial virtual assembly . . . 112

7 Real-time interactive simulations 123
7.1 Real-time dynamic analysis . . . 125
7.1.1 Real-time solution strategy in Project Chrono . . . 126
7.1.2 Sequential impulse solver . . . 129
7.1.3 Case study: interactive multibody simulations . . . 132
7.1.3.1 Four-bar mechanism . . . 135
7.1.3.2 Slider-crank mechanism . . . 140
7.1.3.3 Flexible beam . . . 141
7.2 Real-time structural analysis . . . 144
7.2.1 Reduced order model . . . 145

8 Digital twin case studies 153
8.1 Digital twin for VR prosthesis training . . . 153
8.1.1 The sensorized prosthesis . . . 154
8.1.2 Digital twin and physical twin integration . . . 154
8.1.2.1 Virtual prosthesis dynamic model and trackers interface . 156
8.1.2.2 Arm and hand mesh skinning . . . 158
8.1.2.3 Implementation of stable grasping actions . . . 161
8.1.3 Training tasks . . . 161
8.1.4 Usability assessment results . . . 165
8.2 AR multibody digital twin exploration and interrogation . . . 167
8.2.1 Interaction in AR and digital twin integration . . . 168
8.2.2 Experimental test bench . . . 169

8.2.3 Real-time coupling between digital twin model and physical system 170
8.2.4 AR interaction framework and holographic data visualization . . . 171
8.2.5 Communication architecture and synchronization strategy 172
8.2.6 Evaluation of the real-time simulation accuracy 173
8.3 ROM-based AR structural digital twin exploration and interrogation . . . 177
8.3.1 Experimental test bench 177
8.3.2 From simulative model to graphic model 180
8.3.3 Real-time accuracy assessment and performance evaluation of the experimental digital twin 183
8.4 ROM-driven AR digital twin for structural vehicle chassis monitoring . . . 186
8.4.1 Architecture of a sensor-driven vehicle digital twin 187
8.4.1.1 Sensor integration and real-time data acquisition 188
8.4.1.2 Real-time reduced-order finite element model 192
8.4.1.3 Augmented reality interface and digital twin integration . 195

9 Conclusions and future perspectives 198

Appendices 202

A Rigid body transformations in space 203
A.1 Properties of the rotation matrix 203
A.2 Displacement of a rigid body 204
A.3 Orientation of a rigid body 206
A.3.1 Composition of rotations about the coordinate axes 208
A.3.2 Euler angles 209
A.3.3 Rodrigues' formula 212
A.3.3.1 Cayley's formula 213
A.3.3.2 Rodrigues parameters 214
A.3.4 Euler parameters 216
A.3.5 Quaternions 219
A.3.5.1 Elements of quaternion algebra 220
A.3.5.2 Alternative notations 223
A.3.5.3 Rotations 224
A.3.5.4 Geometric interpretation of quaternions 226

B Operational foundations of the Unity Engine 229
B.1 The Unity development environment 230
B.2 Unity scene architecture 231
B.3 Scripting in Unity 235
B.4 Input management 239
B.5 XR project configuration and development framework 241
B.5.1 XR project configuration 241
B.5.2 XR application deployment 245
B.6 XR packages and platform extensions 245
B.6.1 XR Interaction Toolkit 245
B.6.2 XR Hands 249

B.6.2.1 Custom gestures 250
B.6.3 VIVE OpenXR plugin 250
B.6.4 Varjo XR plugin 253
B.6.5 Mixed Reality Toolkit (MRTK) 253
B.6.6 Haptic Direct plugin 254

C Vuforia Engine configuration for Unity Engine 256
C.1 Image, multi and cylinder targets configuration 262
C.2 Model target configuration 262
C.2.1 Neural networks and deep learning 264

D Basic and kinematics constraints 265
D.1 First orthogonality constraint 265
D.2 Second orthogonality constraint 266
D.3 Spherical constraint 267
D.4 Distance constraint 267
D.5 First parallelism constraint 268
D.6 Second parallelism constraint 269
D.7 Revolute joint 269
D.8 Cylindrical joint 270
D.9 Prismatic joint 271
D.10 Spherical joint 272
D.11 Screw joint 272

E Catmull–Clark refinement algorithm 274

F Delaunay triangulation 277

G C# source code for the industrial virtual assembly case study 283

List of Figures 288

List of Tables 297

Bibliography 300

Abstract

The increasing complexity of modern engineering systems and the digital transformation promoted by Industry 4.0 and Industry 5.0 have accelerated the development of Digital Twins as key technologies for design, monitoring, simulation, and decision support. At the same time, advances in Extended Reality technologies have created new opportunities for intuitive human–machine interaction through immersive visualization and natural user interfaces. This thesis investigates the convergence of these technological domains by proposing an integrated methodological framework for the construction and interactive interrogation of immersive digital twins.

The research combines numerical simulation, real-time interaction, and immersive visualization to extend the role of digital twins beyond static digital representations, turning them into interactive engineering tools that support analysis, design, training, and maintenance. The proposed framework covers the entire development process, from digital twin modelling and real-time object tracking to interaction techniques, immersive geometric modelling, virtual assembly, industrial validation, and real-time physical simulation. A key contribution of this work is the integration of multibody dynamics, reduced-order modelling, and finite element analysis within extended reality environments. This combination enables physically consistent real-time simulations while keeping the computational cost suitable for interactive applications. The research also introduces an immersive modelling approach based on subdivision surfaces, allowing users to create and modify geometries directly in augmented reality using hand gestures. Finally, the work presents a reproducible methodology for evaluating virtual reality hardware in industrial scenarios through virtual assembly case studies. The proposed approach provides objective metrics for assessing usability, performance, and ergonomics, and demonstrates its applicability through an industrial virtual assembly example.

The proposed methodologies are validated through a series of engineering case studies, including interactive multibody simulations, reduced-order structural models, immersive prosthesis training, and augmented reality visualization of structural quantities on physical automotive components. These applications demonstrate the versatility of the proposed framework across different engineering domains and confirm its ability to combine numerical simulation, real-time interaction, and immersive visualization within a unified environment.

Overall, this research presents different frameworks for immersive digital twins, showing how extended reality can become an effective interface for interacting with high-fidelity engineering models rather than serving solely as a visualization tool. By integrating simulation, spatial computing, and natural user interaction, the proposed approach

broadens the role of digital twins in engineering practice and provides a basis for future developments involving artificial intelligence, multiphysics simulation, and increasingly interactive cyber-physical systems.

Keywords: Digital twin; extended reality; augmented reality; mixed reality; virtual reality; industry 4.0; industry 5.0; object tracking; interaction methodologies; natural interaction; interactive modelling; assembly simulation; interactive simulation; reduced-order model; multibody; sequential impulse solver

Chapter 1

Introduction

1.1 Problem of interest and scope

The increasing digitalization of industrial processes and the growing complexity of engineering systems are substantially changing the ways in which products are designed, validated, and managed throughout their lifecycle. Mechanical systems are no longer treated as isolated entities, but rather as components of highly interconnected architectures in which information from sensors and computational models is continuously exchanged to describe and predict system behavior under varying operating conditions. In this context, the *Digital Twin* (DT) paradigm has emerged as one of the most relevant solutions within Industry 4.0 and, more recently, Industry 5.0. The core idea consists in building a digital representation of the physical system capable of integrating physical models, experimental data, numerical simulations, and in some cases artificial intelligence, with the aim of supporting analysis, prediction, and decision-making activities throughout the system lifecycle [1].

The interest in this class of approaches is mainly related to the possibility of overcoming the traditional separation between design, simulation, and operational monitoring. In conventional workflows, high-fidelity models are typically used in the design phase, while data collected during system operation are analyzed offline, mainly for maintenance or diagnostic purposes. This separation prevents continuous updates of the digital representation and limits the use of simulations as a real-time operational tool.

In parallel, the increasing heterogeneity of modern engineering systems, integrating mechanical, electrical, electronic, and software components, requires frameworks capable not only of accurately reproducing physical behavior, but also of making large amounts of heterogeneous information accessible and interpretable. From this perspective, the Digital Twin evolves from a simple virtual replica into a dynamic computational environment in which models, experimental data, and operational information coexist to support design, monitoring, and predictive analysis tasks.

Despite significant progress in recent years, the large-scale adoption of digital twins is still limited by several fundamental issues. On the one hand, many industrial implementations are predominantly data-driven and focus on monitoring and visualization aspects, with only marginal integration of physics-based models. On the other hand, model-driven approaches based on high-fidelity simulations are often incompatible with

real-time requirements due to the computational cost associated with methods such as FEM, CFD, and multibody simulations [2].

At the same time, *Extended Reality* (XR) technologies, including Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR), have introduced new ways of interacting with engineering data. Immersive visualization enables a more direct understanding of complex three-dimensional phenomena, improving spatial perception and reducing cognitive load compared to traditional two-dimensional representations. However, in this case as well, current usage remains mainly limited to visualization. XR environments are often used as passive interfaces, where system behavior is precomputed and not modifiable by the user. Interaction with the underlying physical model, as well as the ability to query or modify the system in real time, is generally absent [3].

This highlights a disconnection between simulation capability and interaction capability. On one side, physical models are accurate but too computationally expensive for real-time execution; on the other, immersive interfaces are perceptually efficient but not tightly coupled with the system's dynamic model. Bridging this gap requires the joint development of robust methodologies and real-time interaction techniques to integrate these two domains [4].

This research is positioned precisely within this context. The objective is the development of methodologies for the realization of interactive Digital Twins based on Extended Reality technologies, in which the immersive component is not limited to visualization but becomes a computational interface through which the user can naturally and consistently interact with the physical system. The framework integrates heterogeneous technologies including computer vision, object and device tracking, collision detection, real-time physics simulation, multibody dynamics, reduced order modeling, sensor integration, and advanced human–machine interaction paradigms in immersive environments.

Validation of the framework is carried out through a series of case studies developed during the research, covering progressively more complex scenarios, including immersive training and geometric modeling environments, digital twins for prosthesis usage training, augmented reality systems for manipulation and querying of dynamic digital twins based on real-time multibody simulations, and structural digital twins based on reduced models.

Taken together, these contributions show how the integration of numerical simulation, physical models, and immersive technologies enables overcoming the traditional separation between digital representation and interaction, paving the way for a new class of interactive digital twins for complex engineering systems.

1.2 Main contributions of the dissertation

The work presented in this dissertation investigates the integration of Extended Reality, real-time simulation, and physical models for the development of interactive Digital Twins in engineering applications. The main contributions can be summarized in the following aspects.

First, the dissertation investigates the use of Extended Reality as an active interface for engineering systems rather than only as a means of visualization. Different interaction techniques are developed and evaluated to establish a spatial and functional correspon-

dence between the physical and virtual environments. These include object and device tracking, collision detection, physical interaction, and immersive interaction techniques.

Second, different approaches are investigated to make physical simulation suitable for interactive applications. In particular, real-time multibody dynamics and reduced-order modeling are considered as alternatives to computationally demanding high-fidelity simulations. The objective is to retain the physical information required by the application while meeting the computational requirements imposed by real-time interaction.

Third, the dissertation develops and evaluates methodologies for integrating these technologies into different Digital Twin applications. The considered scenarios range from immersive training and interactive geometric modeling to Digital Twins for prosthesis usage, augmented reality systems for the manipulation and querying of dynamic mechanical models, and structural Digital Twins based on reduced-order models. These applications provide different perspectives on the same general problem, namely the interaction with engineering models through an immersive environment.

A further contribution concerns the experimental evaluation of the proposed approaches. The developed systems are tested in different application contexts, considering aspects such as tracking accuracy, real-time performance, interaction capabilities, user performance, and cognitive load. This experimental activity provides a basis for assessing the applicability of immersive and simulation-based approaches to engineering tasks.

Taken together, the results presented in the dissertation contribute to the development of Digital Twins in which simulation and interaction are more closely integrated. The work therefore considers Extended Reality not simply as a visualization layer, but as an interface through which users can access and interact with dynamic engineering models in real time.

1.3 Organization of the dissertation

The thesis follows the progressive development of the proposed methodological framework. Each chapter introduces the theoretical elements and technologies required for the subsequent ones, up to the realization and validation of the different interactive digital twins.

- **Chapter 2** introduces the fundamental concepts of digital twins and Extended Reality, tracing their evolution and comparing data-driven and model-driven approaches. It also presents the main XR technologies and the characteristics of immersive display devices, with reference to HMD hardware architectures and the geometric principles of real-time rendering pipelines.

- **Chapter 3** addresses methodologies for estimating spatial correspondence between the physical world and the virtual environment, with particular attention to determining the pose of real objects in the virtual system. Vision-based and sensor-based tracking techniques are analyzed, including image-based, model-based approaches and commercial tracking systems.

- **Chapter 4** focuses on real-time interaction methodologies, with reference to modeling physical interactions between users and virtual objects. The main collision

detection algorithms are introduced, along with response models based on contact constraints and impulse-based schemes within real-time physics engines.

- **Chapter 5** introduces the problem of geometric modeling in XR, starting from a brief state-of-the-art review, and presents a methodology for interactive modeling in augmented reality based on Subdivision Surfaces.
- **Chapter 6** presents the use of immersive technologies in industrial contexts, with particular reference to virtual assembly processes. The chapter includes a comparative evaluation of different VR headsets and an analysis of the impact of immersive interaction on operational performance and user cognitive load.
- **Chapter 7** addresses the integration of physical simulation in interactive digital twins. Real-time multibody dynamics models based on sequential impulse solvers or iterative methods are described, together with the use of Reduced Order Models for real-time structural analysis, aiming to reduce computational complexity while preserving physical consistency.
- **Chapter 8** collects and integrates the main case studies developed during the PhD, including immersive training applications, augmented reality digital twins for mechanical systems, ROM-based structural models, and vehicular applications with sensor integration and immersive visualization.
- **Chapter 9** concludes the thesis by summarizing the scientific contributions, discussing the main limitations encountered during experimental validation, and outlining possible future developments in the field of interactive digital twins and physics-informed engineering systems.

Overall, the thesis aims to confirm how the integration of numerical simulation, real-time interaction, and immersive visualization can extend the capabilities of digital twins towards truly interactive engineering systems.

Chapter 2

Digital twin and extended reality

This chapter introduces the fundamental concepts of Digital Twins and Extended Reality technologies. The chapter first examines the definition of the Digital Twin concept, tracing its evolution from its original formulation to its current role as a key enabling technology across multiple application domains. Particular attention is devoted to the main characteristics of Digital Twins, their integration of predictive models and real-time data, and the benefits they provide across a wide range of fields. The advantages associated with the integration of Digital Twins with Extended Reality technologies are also discussed.

The chapter then presents the concept of Extended Reality, introducing its main paradigms, namely Virtual Reality, Augmented Reality, and Mixed Reality. A historical overview of XR development is provided, from early immersive systems to contemporary devices and platforms. Subsequently, the XR devices available at the Virtual Prototyping Laboratory of the University of Rome Tor Vergata and employed throughout this research project are described, with emphasis on tracking architectures, display technologies, and hardware characteristics.

Finally, the chapter discusses the visualization of three-dimensional virtual objects in XR environments, focusing on the geometric representations and rendering principles that enable real-time applications.

2.1 Digital twin

The term "digital twin" was coined by Dr. Michael Grieves, a Professor at the University of Florida, in 2002 [5]. Grieves introduced the concept of a digital twin during a presentation to the National Aeronautics and Space Administration (NASA) of the United States, describing it as a digital replica of a physical object or complex system [6]. It relies on the notion that a digital representation of a physical system can be developed as an independent entity. This digital information would be a "twin" of the information embedded within the physical system itself and would be linked to that physical system through the system's lifecycle [7]. In addition to this clear definition, the concept of digital twins is often confused with a complex virtual prototype built for challenging simulations such as optimization, off-design investigation, and so on [8]. A DT is not a simple virtual prototype, as it communicates with its physical counterpart,

known as physical twin (PT), preferably in a bidirectional manner [9], by acquiring data from real sensors and enabling control of physical actuators. The purpose of a digital twin is to enhance the information associated with the PT through mathematical modelling, providing real-time insights into relevant aspects that cannot be directly measured or controlled by the physical system, and to support the PT throughout its entire lifecycle, rather than being limited to design or optimization stages. At the core of the DT there are high-fidelity predictive models that can be data-driven or model-driven (Figure 2.1).

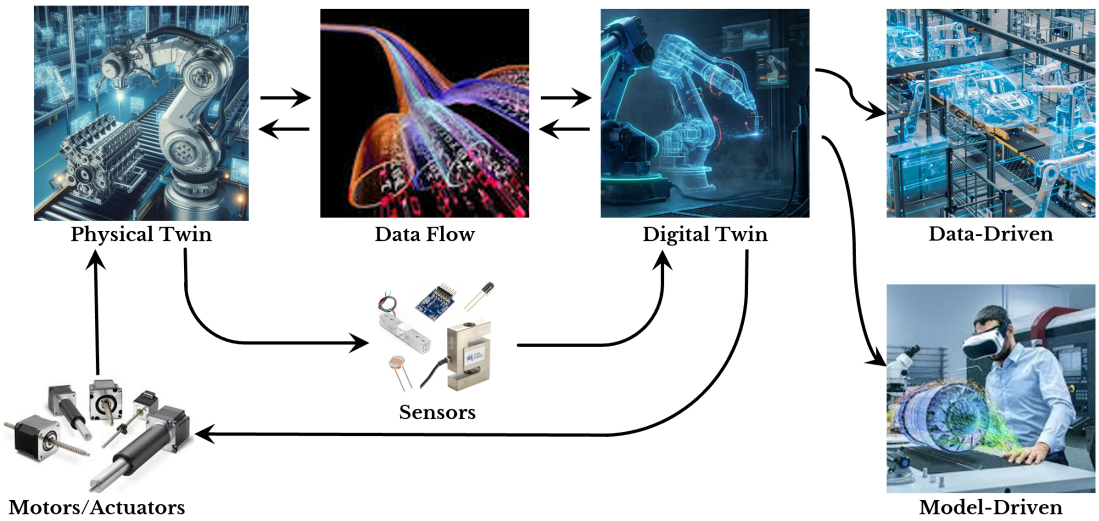


Figure 2.1: A general data flow architecture of a digital twin and its embedded predictive models.

In data-driven digital twins, the system behaviour is inferred directly from data, typically using statistical approaches and machine learning techniques [10]. These models rely on large amounts of historical and real-time data acquired from sensors embedded in the physical twin [11]. Methods such as neural networks, regression models, and deep learning architectures enable data-driven DTs to capture complex and highly nonlinear relationships, even when the underlying physical mechanisms are not fully understood [12]. For this reason, they are particularly well suited to complex systems where first-principles modeling is difficult.

Model-driven digital twins, on the other hand, are based on first-principles or physics-based descriptions of the system [13]. In this case, the system behaviour is governed by equations derived from fundamental laws and rely on numerical techniques, such as finite element methods [14], computational fluid dynamics [15], and multibody dynamics [16]. Because they are grounded in physical theory, model-driven DTs offer a high level of interpretability and are generally more reliable when extrapolating beyond observed conditions. They are therefore particularly effective for systems that are well understood from a theoretical perspective. To enable real-time execution and reduce computational complexity, these approaches often rely on reduced-order models [17, 18].

On this basis, digital twins have progressively moved beyond their original conceptual formulation and are now widely used across a variety of application domains. Their ability

to integrate real-time data, predictive modeling, and lifecycle information has led to significant developments in areas such as manufacturing [19,20], where they are employed for process optimization and predictive maintenance; engineering design [21], where they support virtual prototyping and performance assessment; the automotive sector [22], enabling advanced monitoring and control of vehicles; and the aerospace field [23, 24], where they are used for aircraft health monitoring, mission simulation, and structural performance analysis. More broadly, digital twins are increasingly being adopted in fields such as healthcare [25,26], transportation [27], and energy systems [28,29], reflecting their growing importance in both industrial and scientific contexts.

2.1.1 Digital twin architectures

The physical and digital components of a Digital Twin can be organized according to different architectural frameworks. Among the most widely adopted is the five-dimension model proposed by Tao et al. [30], which describes a Digital Twin in terms of a physical entity, a virtual entity, digital twin data, a set of services, and the connections between these components. Other reference models, such as the Reference Architecture Model Industry 4.0 (RAMI 4.0) [31] and the Asset Administration Shell [32], provide similar architectural views with a stronger focus on industrial integration and interoperability.

From a functional point of view, these components can be grouped into three main domains: data acquisition and exchange, modeling and computation, and services and interfaces. The first domain includes the mechanisms required to collect and transfer information between the physical and digital entities. The second comprises the models used to represent the system and reproduce its behaviour, including both physics-based and data-driven approaches. The third provides the functions through which the Digital Twin is used, such as monitoring, prediction, simulation, diagnosis, and optimization.

The connections between these domains are a key aspect of the Digital Twin architecture. Depending on the application, information can flow from the physical system to the digital representation, or in both directions when the Digital Twin is also used to support or implement control actions. The latter configuration requires the different components to operate in a coordinated manner and places greater demands on the communication infrastructure and on the consistency of the exchanged information.

Another important aspect of the architecture is the interaction between heterogeneous components. A Digital Twin may involve sensors, databases, simulation software, control systems, and visualization platforms developed using different technologies and data structures. Their integration therefore requires suitable interfaces and mechanisms for exchanging and interpreting information across the different components. This issue becomes particularly relevant when Digital Twins are integrated into larger industrial systems, where interoperability is required not only at the communication level but also at the data and semantic levels.

The architectural aspects introduced in this section provide the basis for discussing two closely related issues: the synchronization of the physical and digital representations and the interoperability of the systems involved. These aspects are particularly relevant to the integration of Digital Twins with Extended Reality technologies and are addressed in the following sections.

2.1.2 Data synchronization and interoperability

Data synchronization is required to keep the digital representation consistent with the current state of the physical system. The frequency at which the Digital Twin is updated depends on the application. Monitoring and lifecycle management generally do not require very frequent updates, whereas applications involving dynamic behaviour, real-time interaction, or control require higher update rates and low communication latency.

Maintaining this consistency can be challenging because the data used to update the Digital Twin may be acquired at different sampling rates and may contain noise, missing measurements, or delays. The digital state is therefore an estimate of the physical state, and a certain discrepancy between the two may be unavoidable. The extent of this discrepancy depends on the characteristics of the acquisition and communication processes and on the time required to process the acquired data.

For Digital Twins integrated with XR, synchronization also has a spatial dimension. Delays between changes in the physical system and their visualization can produce a mismatch perceived by the user, while errors in spatial registration can lead to an incorrect alignment between virtual and physical objects. Both temporal and spatial consistency are therefore important for reliable interaction with the Digital Twin.

Interoperability concerns the ability of the different components of a Digital Twin to exchange and correctly interpret information. This is particularly relevant when systems based on different software platforms, data structures, and communication technologies have to be integrated. Standards such as OPC UA and the Asset Administration Shell provide mechanisms for structured data exchange and standardized information models [32]. However, interoperability also requires the meaning of the exchanged information to be preserved across different systems. Semantic interoperability is therefore important when data from different sources have to be combined and interpreted consistently.

Synchronization and interoperability are closely related to the practical implementation of a Digital Twin. The former concerns the consistency of the digital representation with the physical system over time, while the latter enables its integration with the heterogeneous systems involved in the application.

2.1.3 Digital twin modelling and validation

The reliability of a Digital Twin depends not only on the quality of its underlying models, but also on their ability to reproduce the behaviour of the corresponding physical system under relevant operating conditions. Model validation is therefore required to assess the agreement between the predicted and observed behaviour of the physical system. This is typically performed by comparing simulation results with experimental or measured data using suitable error metrics.

Validation should consider representative operating conditions and, where possible, different regimes of system operation. A model that provides accurate results under a limited set of conditions may not maintain the same level of accuracy when the operating conditions change. Uncertainty in the measurements, modelling assumptions, and numerical approximations should therefore also be considered when assessing the reliability of the Digital Twin.

In addition to the validation of the individual models, the complete Digital Twin should be evaluated with respect to the consistency of its digital representation with the physical system. Changes in the physical system, degradation, or variations in operating conditions may progressively reduce the accuracy of the model. Model updating and recalibration can therefore be required during the lifecycle of the Digital Twin to maintain an adequate level of correspondence with the physical system.

Validation is consequently not a one-time procedure, but an activity that can be repeated throughout the lifecycle of the Digital Twin. This is particularly important for applications in which the Digital Twin is used for prediction, diagnosis, or decision support, where the reliability of its outputs directly affects the decisions based on the digital representation.

2.1.4 Digital twins and Industry 5.0

The transition from Industry 4.0 to Industry 5.0 broadens the scope of digital transformation in industrial and engineering domains. While Industry 4.0 centers primarily on automation, data exchange, and cyber-physical connectivity, Industry 5.0 reintroduces human agency, sustainability, and operational resilience as core architectural priorities [33].

In this framework, Digital Twins evolve beyond process monitoring and optimization to support human-centric operations [34]. Rather than automating decision-making entirely, a Digital Twin integrates heterogeneous sensor telemetry and computational models into a unified state representation, exposing system behavior that would otherwise be unobservable. This provides operators with actionable insights while preserving human supervision and control.

Extended Reality (XR) technologies act as the dedicated interaction layer within this human-centric paradigm [35,36]. In these environments, information is mediated through graphical elements and three-dimensional representations, offering intuitive interfaces for visualizing, manipulating, and interacting with Digital Twins [37].

By projecting Digital Twin analytics and simulation outputs directly onto physical assets via AR or VR interfaces [38–40], XR reduces cognitive friction in complex tasks such as maintenance, remote operation, and training [41]. This is achieved by leveraging spatial mapping, object recognition, haptic feedback, and natural user interfaces based on gesture tracking technologies [42, 43].

Ultimately, this contextualized visualization translates abstract predictive data into spatial information, streamlining human-system interaction [44, 45].

Concurrently, Digital Twins address the environmental and structural objectives of Industry 5.0. Continuous lifecycle monitoring and virtual prototyping lower material consumption, mitigate unplanned downtime, and reduce physical testing iterations. From a resilience perspective, predictive diagnostics enable early detection of operational anomalies, allowing systems to adapt dynamically to unexpected disturbances [46].

Combined with Extended Reality, Digital Twins thus form a foundational technology for human-centric cyber-physical environments, blending real-time numerical processing with spatial interfaces for transparent decision-making.

2.2 Extended reality

The term Extended Reality refers to a spectrum of immersive technologies aimed at integrating physical environments with interactive digital content [47]. Within engineering applications, XR is increasingly adopted not only as a visualization tool, but as an advanced interface for interacting with complex models, simulations, and data-driven systems such as digital twins.

XR encompasses three main paradigms: Virtual Reality, Augmented Reality, and Mixed Reality, which can be distinguished according to the level of immersion and the degree of integration between real and virtual elements [48].

Virtual Reality represents the fully immersive end of the spectrum, in which the user is entirely embedded within a completely digital environment. In this paradigm, physical reality is fully replaced by a virtual simulation, with which the user can interact in real time [49].

Augmented Reality, in contrast, preserves the physical environment as the primary frame of reference while superimposing digital content onto it. In this paradigm, the user continues to perceive the real world, which is enriched by virtual elements, such as text, images, or three-dimensional models, that enhance the information available [50].

Mixed Reality represents a more advanced form of integration, in which virtual objects are not merely superimposed onto the real environment but are spatially anchored within it and capable of interacting dynamically with it. In this paradigm, digital elements behave consistently with the physical context, requiring an understanding of the surrounding space and enabling them to adapt and respond to real-world conditions [51].

2.2.1 Historical evolution of extended reality

The conceptual foundations of Extended Reality precede the advent of modern digital computing and can be understood as emerging from the progressive convergence of perceptual science, representational techniques, and early experimental media systems. Rather than constituting a linear technological lineage, XR development reflects a gradual expansion of mechanisms for mediating, augmenting, and reconstructing human perceptual experience.

One of the earliest foundational contributions to this trajectory is the stereoscope introduced by Wheatstone in 1838 [52]. Its significance lies not merely in its technological novelty, but in its demonstration that depth perception can be systematically induced through controlled binocular disparity. This insight established a fundamental principle for later stereoscopic imaging systems and, more broadly, for virtual and augmented reality display technologies.

In parallel to these perceptual investigations, early theoretical and artistic contributions began to articulate the notion of mediated reality as a construct rather than a fixed given. In 1938, Antonin Artaud, in “Le Théâtre et son Double”, proposed a theatrical framework in which fictional elements acquire perceptual immediacy, thereby anticipating later formulations of immersive experience in computational environments [53]. Although not grounded in computational systems, this work can be interpreted as an early conceptualization of reality as externally modifiable and experientially reconstructible.

A more direct antecedent to immersive computational systems is represented by Morton Heilig's Sensorama (1962), a multisensory apparatus integrating stereoscopic visualization, spatial audio, haptic feedback, and olfactory stimulation [54]. Beyond its technical configuration, Sensorama is significant as an early attempt to systematically engineer full-sensory immersion, anticipating later virtual reality paradigms centered on multisensory congruence.

This trajectory continued with Ivan Sutherland's "Sword of Damocles" in 1968 (Figure 2.2a), widely regarded as the first operational head-mounted display system for immersive visualization [55]. Its importance lies in the shift from passive sensory stimulation to computationally generated, viewpoint-dependent visual synthesis. By combining real-time wireframe rendering with optical superimposition and head tracking via electromechanical linkage, the system established the foundational architecture of computer-mediated spatial visualization.

During the early 1970s, research shifted toward embodied interaction paradigms that decoupled immersion from wearable interfaces [56]. Myron Krueger's Videoplace introduced one of the earliest implementations of artificial reality based on vision-driven interaction. By extracting user silhouettes through real-time image processing techniques such as background subtraction, the system enabled direct interaction between human motion and computational environments projected on large displays, thereby redefining immersion as a relational rather than device-bound phenomenon.

In parallel, alternative forms of immersive engagement and spatial navigation were explored. The Aspen Movie Map (1978), developed at MIT, enabled structured, interactive exploration of a real urban environment through a database of photographic imagery [57]. Conceptually, this system anticipated later developments in spatial information systems and digital geographies. Similarly, NASA's VIEW project, led by Scott Fisher, investigated early immersive virtual environments integrating head tracking, stereoscopic rendering, and manual interaction devices for telepresence applications.

Within aerospace and defense contexts, Thomas Furness developed the "Super Cockpit" system during the late 1970s and early 1980s [58–60]. This system represents an early instance of perceptually integrated augmented information systems, in which head-mounted and head-up displays were coupled with real-time sensor data to spatially register flight information within the pilot's perceptual field. This approach introduced a fundamental shift toward cognitive offloading through perceptual alignment of digital and physical information.

During the 1980s, the concept of "virtual reality" emerged as a coherent research and cultural paradigm, most prominently through the work of Jaron Lanier and VPL Research [61]. VPL's development of data gloves, head-mounted displays, and interactive three-dimensional environments established early forms of embodied interaction with synthetic worlds. Despite severe computational constraints, these systems articulated the foundational principles of interactive spatial computing.

In the early 1990s, immersive visualization research expanded toward room-scale environments with the introduction of the Cave Automatic Virtual Environment (CAVE) by Cruz-Neira et al [62, 63]. This system introduced a fundamentally different modality of immersion based on projection-based stereoscopy within a physically navigable space, thereby decoupling immersion from head-mounted systems and emphasizing environmen-

tal enclosure as the primary medium of presence (Figure 2.2b).

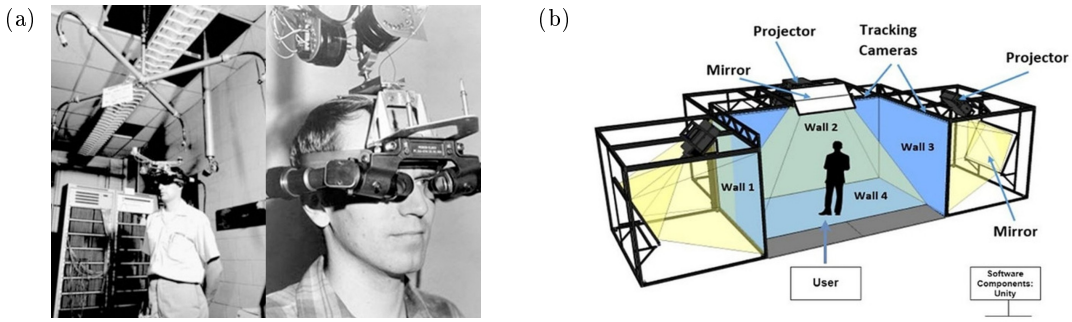


Figure 2.2: (a) Ivan Sutherland's "Sword of Damocles" and (b) example of CAVE structure [64].

The term Augmented Reality emerged in the early 1990s within an industrial context, introduced by Tom Caudell at Boeing [65]. Rather than constituting a theoretical construct, AR initially functioned as an applied solution for embedding digital instructions within physical assembly workflows. Its significance lies in the transition from simulation-centric immersive systems toward task-oriented augmentation of real-world environments, thereby linking perceptual computing with industrial efficiency.

Shortly thereafter, Milgram and Kishino (1994) formalized the Reality–Virtuality Continuum, providing a unifying representational framework for understanding immersive systems. Within this model, AR and Augmented Virtuality are not discrete categories but regions within a continuous spectrum spanning from purely physical to fully virtual environments. Mixed Reality, in this formulation, does not represent a temporal successor to AR but rather a structural characterization of intermediate coupling between real and virtual domains, while Virtual Reality occupies the extreme limit of full synthetic immersion. This taxonomy remains a foundational reference for XR classification [66].

Following this theoretical formalization, augmented reality evolved into a distinct research domain driven by advances in computer vision, tracking, and real-time rendering. Early frameworks such as ARToolKit enabled marker-based AR prototyping, significantly reducing the barrier to experimental development and accelerating academic dissemination [67]. Concurrently, Ronald Azuma provided a widely adopted operational definition of AR as systems characterized by real-time interaction, registration of real and virtual elements, and three-dimensional alignment, thereby consolidating AR as a formalized research discipline [68].

Subsequently, augmented reality transitioned from laboratory prototypes to early consumer and mobile deployments during the 2010s, albeit with significant technological constraints and limited commercial adoption, as exemplified by wearable systems such as Google Glass (2013) [69, 70].

While originally defined within the Reality–Virtuality Continuum, Mixed Reality has later been reinterpreted within technological and industrial contexts as a class of systems characterized by tighter and more persistent integration between physical and digital environments than conventional AR implementations. This reinterpretation is exemplified by systems such as Microsoft HoloLens (2016), which introduced spatial mapping, gesture-based interaction, and persistent holographic anchoring, enabling digital enti-

ties to maintain contextual coherence with physical space. This development reflects a broader shift toward spatial computing as a paradigm in which computation is embedded within spatial experience rather than overlaid upon it.

In parallel, Virtual Reality experienced a significant resurgence in the early 2010s with the introduction of the Oculus Rift prototype, which demonstrated low-latency head tracking and wide field-of-view stereoscopic rendering. This milestone marked the re-emergence of VR as a viable computational paradigm, enabled by advances in GPU architectures, display technologies, and sensor miniaturization.

Despite these advancements, XR technologies remained constrained for several decades due to limitations in computational capacity, display resolution, and sensing fidelity. The convergence of microelectronics, real-time rendering pipelines, and advanced tracking systems during the 2000s and 2010s enabled the transition from experimental prototypes to scalable immersive platforms.

In recent years, XR has progressively consolidated as an umbrella conceptual framework encompassing Virtual Reality, Augmented Reality, and Mixed Reality. This consolidation has been driven by major industrial actors such as Meta, Microsoft, and Apple, who have developed increasingly integrated systems characterized by high-resolution displays, spatial understanding, and natural interaction paradigms.

2.2.2 XR devices and tracking technologies

Head-Mounted Displays (HMDs) constitute a key enabling technology for immersive XR systems, providing stereoscopic visual output through near-eye displays that cover the user's field of view and support real-time three-dimensional rendering. By integrating display subsystems with motion sensing and tracking capabilities, HMDs establish a head-referenced rendering pipeline that ensures spatial coherence between user motion and virtual content.

Within the XR framework introduced previously, HMDs represent the primary interface through which VR, AR and MR experiences are delivered. Fully occlusive configurations are typically associated with VR, where the physical environment is entirely replaced by a synthetic one, whereas see-through configurations are characteristic of AR and MR systems, where digital content is integrated with the real world.

From a system perspective, HMD architectures can be distinguished based on the distribution of computational resources. Tethered devices delegate rendering and processing tasks to an external unit, enabling high graphical performance at the expense of mobility. Conversely, standalone HMDs integrate all computational, sensing, and rendering components within the device itself, leveraging embedded system-on-chip architectures to enable fully self-contained operation.

A complementary classification concerns the tracking architecture used to estimate the device pose. Outside-in systems rely on external sensors to determine the position and orientation of the HMD, typically achieving high accuracy and low latency but requiring dedicated infrastructure (Figure 2.3a). Inside-out approaches instead perform onboard estimation through embedded sensors and vision-based algorithms, thereby enabling greater deployment flexibility while introducing higher computational demands and sensitivity to environmental conditions (Figure 2.3b).

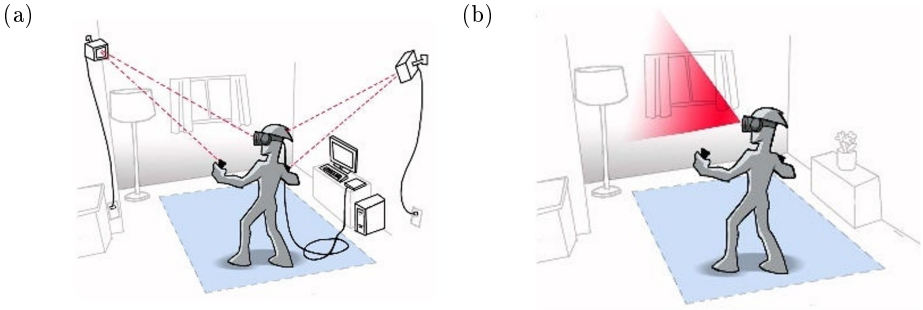


Figure 2.3: (a) Outside-In and (b) Inside-Out tracking systems.

The relationship between virtual content and real-world perception further differentiates HMD configurations into video see-through (VST) and optical see-through (OST) systems. VST approaches acquire the surrounding environment through onboard cameras and present a composited image in which virtual elements are integrated with the captured scene. This mediated representation enables precise registration and consistent occlusion handling, and is commonly implemented in modern passthrough systems employing stereo imaging to improve depth cues and reduce perceptual discrepancies. In contrast, OST systems rely on transparent optical elements, such as waveguide-based combiners [71] and partially reflective optical combiners, typically implemented through beam splitters [72,73], to directly superimpose virtual imagery onto the user's view of the physical world. While this preserves real-world fidelity and eliminates camera-induced latency, it introduces challenges related to brightness consistency, occlusion management, and depth alignment.

The visual subsystem is based on near-eye display technologies, which differ in their emission and modulation mechanisms. Liquid Crystal Displays (LCDs) remain widely used due to their technological maturity and cost efficiency, although their transmissive operation limits contrast performance. Organic Light Emitting Diode (OLED) displays, based on self-emissive pixels, provide higher contrast, improved black levels, and faster response times, making them more suitable for dynamic immersive scenarios. Micro-OLED technology extends these advantages by enabling significantly higher pixel densities, which are particularly beneficial in compact optical systems. MicroLED displays represent a promising alternative, offering high brightness and efficiency, although their adoption is currently constrained by manufacturing complexity.

The XR device ecosystem available at the Virtual Prototyping Laboratory of the University of Rome Tor Vergata reflects the main architectural categories discussed in the previous sections, including both outside-in and inside-out tracking systems, as well as VR and mixed reality configurations.

Among the devices employing outside-in tracking, the laboratory includes the HTC Vive Pro and HTC Vive Pro 2 headsets. Both systems are based on the lighthouse tracking architecture, which enables external optical pose estimation through infrared laser scanning and distributed photodiode sensing. The tracking system relies on external base stations that emit infrared light in a structured and time-controlled manner (Figure 2.4). Each base station first emits a brief synchronization pulse that is used to establish temporal alignment between the headset and the tracking system. This is followed by

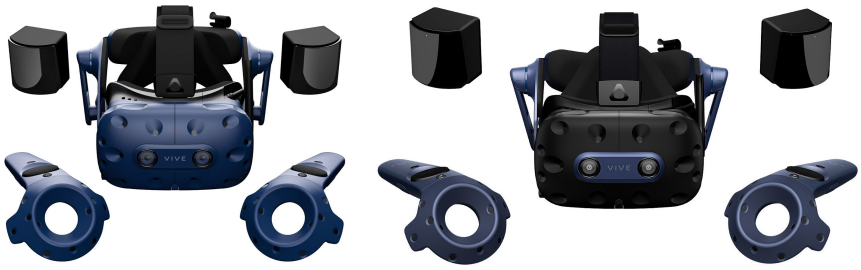
rapidly rotating infrared laser sweeps that scan the tracking volume along predefined axes. The headsets are equipped with multiple photodiodes distributed across their surface. These sensors detect the precise moment in which the infrared laser beam passes over them. By combining the timing of these detections with the known motion of the laser sweeps, the system can infer the direction of each sensor relative to the base stations. The final head pose is obtained by combining the information from all sensors to estimate the position and orientation of the headset in space. From a geometric perspective, the tracking process can be interpreted as a triangulation problem in which angular measurements from at least two spatially separated base stations are combined to estimate the three-dimensional position of the tracker. Let θ_i and φ_i denote the horizontal and vertical angles measured by a given photodiode with respect to the lighthouse i. Each angular measurement defines a ray in space originating from the known lighthouse position. The intersection (or least-squares minimization of distances) of rays associated with multiple base stations produces the estimated tracker position $\mathbf{p} \in \mathbb{R}^3$.

Figure 2.4: Vive base station 1.0 (left) and 2.0 (right).

The HTC Vive Pro 1 is a tethered VR headset equipped with dual OLED displays with a resolution of 2880 × 1600 pixels (1440 × 1600 per eye), a refresh rate of 90 Hz, and a field of view of approximately 110 degrees (Figure 2.5a). The device is designed primarily for fully immersive virtual reality applications. Although front-facing cameras are available, their use for mixed or augmented reality is not natively supported and requires custom development without an integrated spatial mapping pipeline.

The HTC Vive Pro 2 maintains the same tracking architecture while significantly improving visual performance (Figure 2.5b). It features dual LCD panels with a resolution of 2448 × 2448 pixels per eye, supports refresh rates up to 120 Hz, and provides an expanded field of view of approximately 120 degrees. Camera-based passthrough is available at a hardware level but is not integrated into a complete mixed reality system, limiting its use for MR applications.

Among the devices adopting inside-out tracking, the laboratory includes the HTC Vive XR Elite, Meta Quest Pro, Meta Quest 3, Microsoft HoloLens 2, and Varjo XR-4. These systems estimate head pose using onboard sensors, typically combining RGB cameras and inertial measurement units, without requiring external tracking infrastructure.



(a) HTC Vive Pro 1 (b) HTC Vive Pro 2

Figure 2.5: Devices based on outside-in tracking.

(a) HTC Vive XR Elite (b) Meta Quest Pro (c) Meta Quest 3

(d) Microsoft Hololens 2 (e) Varjo XR-4

Figure 2.6: Devices based on inside-out tracking.

The HTC Vive XR Elite is a standalone mixed reality headset supporting both immersive VR and camera-based passthrough MR (Figure 2.6a). It features dual displays with a resolution of 1920 × 1920 pixels per eye, a refresh rate of 90 Hz, and a field of view of approximately 110 degrees. The system is powered by a Qualcomm Snapdragon XR2 platform with 12 GB of RAM and 128 GB of internal storage, enabling onboard processing for visual-inertial tracking and spatial reconstruction.

The Meta Quest Pro is a standalone mixed reality headset designed for professional and collaborative applications (Figure 2.6b). It uses inside-out tracking based on RGB cameras and IMU sensors to achieve six-degrees-of-freedom pose estimation without external infrastructure. The display system provides a resolution of 1800 × 1920 pixels per eye, a refresh rate of 90 Hz, and a field of view of approximately 106 degrees. It additionally integrates eye tracking, hand tracking, spatial audio, and wireless connectivity, supporting collaborative XR scenarios.

The Meta Quest 3 is a standalone mixed reality headset that extends VR capabilities with improved camera-based passthrough (Figure 2.6c). It adopts an inside-out tracking

system based on RGB cameras and IMU sensors. The display subsystem provides a resolution of 2064 × 2208 pixels per eye, a refresh rate of up to 120 Hz, and a field of view of approximately 110 degrees. Compared to previous-generation devices, it significantly improves color passthrough quality and spatial reconstruction, although accuracy remains constrained by camera-based sensing.

The Microsoft HoloLens 2 is an optical see-through mixed reality headset designed for industrial and research applications (Figure 2.6d). It uses waveguide-based transparent displays to overlay holographic content onto the real environment. The tracking system combines RGB cameras, depth sensors, inertial measurement units, and eye tracking within a unified spatial mapping framework. The display provides a resolution of 2048 × 1080 pixels per eye, a refresh rate of 60 Hz, and a field of view of approximately 52 degrees. A dedicated Holographic Processing Unit (HPU) performs real-time sensor fusion and spatial mapping.

The Varjo XR-4 is a high-end mixed reality headset designed for professional simulation and engineering applications (Figure 2.6e). It combines high-fidelity video passthrough with inside-out tracking based on RGB cameras and inertial sensors. The display system features dual high-resolution panels with a resolution of 3840 × 3744 pixels per eye and a field of view of 120° × 105°, enabling extremely high visual fidelity for demanding XR applications. Integrated eye tracking and spatial sensing systems support precise alignment between virtual and physical content, positioning the device at the highest performance level within the laboratory ecosystem.

Table 2.1 summarizes the technical specifications of some of the headsets used in this research.

Table 2.1: Technical specifications of the devices under investigation.

	HTC Vive Pro 1	**HTC Vive Pro 2**	**Meta Quest Pro**	**HTC Vive XR Elite**	**Meta Quest 3**
General Info					
Manufacturer	HTC	HTC	Meta	HTC	Meta
Device Type	PC-Powered VR	PC-Powered VR	Standalone VR	Standalone VR	Standalone VR
Platform	SteamVR, Viveport	SteamVR, Viveport	Meta Quest	Viveport	Meta Quest
Release Date	4 April 2018	3 June 2021	25 October 2022	31 March 2023	10 October 2023
Optics					
Optics		Dual-Element Fresnel Lenses	Pancake Lenses	Pancake Lenses	Pancake Lenses
IPD Range	61–72 mm Hardware Adjustable (manual)	57–70 mm Hardware Adjustable (manual)	55–75 mm Hardware Adjustable (manual)	54–73 mm Hardware Adjustable (manual)	58–71 mm Hardware Adjustable (manual)
Passthrough	Dual Passthrough Cameras	Dual Passthrough Cameras	Color Passthrough	16MP RGB Camera	Dual 10 PPD Color Passthrough Cameras
Display					
Display Type	2 × AMOLED	2 × LCD Binocular	2 × LCD Binocular Local Dimming	2 × LCD Binocular	2 × LCD Binocular
Resolution	1440 × 1600 Per-Eye	2448 × 2448 Per-Eye	1800 × 1920 Per-Eye	1920 × 1920 Per-Eye	2064 × 2208 Per-Eye

	HTC Vive Pro 1	**HTC Vive Pro 2**	**Meta Quest Pro**	**HTC Vive XR Elite**	**Meta Quest 3**
Refresh Rate	90 Hz	120 Hz	90 Hz (72 Hz Mode Available)	90 Hz	120 Hz
Subpixel Layout	PenTile Diamond (2 Subpixels Per Pixel)	RGB Stripe (3 Subpixels Per Pixel)	RGB Stripe (3 Subpixels Per Pixel)	RGB Stripe (3 Subpixels Per Pixel)	RGB Stripe (3 Subpixels Per Pixel)
Image					
Visible FoV	98° Horizontal 98° Vertical 110.48° Diagonal	116° Horizontal 96° Vertical 113° Diagonal	106° Horizontal 95.57° Diagonal	110° Diagonal	110° Horizontal 96° Vertical
Rendered FoV	107.06° Horizontal 107.71° Vertical 110.48° Diagonal	116.52° Horizontal 96.49° Vertical 113.3° Diagonal	108° Horizontal 95.57° Vertical 111.24° Diagonal	102.13° Horizontal 91.27° Vertical 116.04° Diagonal	
Binocular Overlap	90.46°	79.83°	79.72°	80.93°	
Average Pixel Density	14.58 PPD Horizontal 13.36 PPD Vertical	24.93 PPD Horizontal 25.37 PPD Vertical	19.17 PPD Horizontal 18.83 PPD Vertical	20.97 PPD Horizontal 21.03 PPD Vertical	
Foveated Rendering	X	X	V	X	X
Device Embodiment					
Weight	550 G Without Headstrap 800 G With Headstrap	850 G With Headstrap	722 G With Headstrap	625 G With Headstrap (With "Battery Cradle" Headstrap. With Glasses-Style Arms: 273 G)	515 G With Headstrap
Material	Plastic, Foam Facial Interface	Plastic, Foam Facial Interface	Plastic, Foam Facial Interface	Plastic, Foam Facial Interface	Plastic, Foam Facial Interface
Headstrap	Hard Padded Retractable Strap	Hard Padded Retractable Strap	Hard Padded Retractable Strap	Replaceable Headstrap With Hot-Swappable Battery	Flexible Fabric Strap
Tracking					
Tracking Type	6 DoF Marker-Based	6 DoF Marker-Based	6 DoF Inside-Out Via 5 Integrated Cameras	6 DoF Inside-Out Via 4 Integrated Cameras. Also Includes Depth Sensor	6 DoF Inside-Out Via 4 Integrated Cameras. Also Includes Depth Sensor
Tracking Frequency	1000 Hz (Rotational Tracking Freq. Positional Freq: 100 Hz)	1000 Hz (Rotational Tracking Freq. Positional Freq: 100 Hz)	X	X	X
Base Stations	2 × Vive Base Station	2 × SteamVR 2.0	X	X	X
Eye Tracking	X	X	V	X	X
Face Tracking	X	X	V	X	X
Hand Tracking	X	X	V	V	V
Body Tracking	X	X	X	X	X

	HTC Vive Pro 1	**HTC Vive Pro 2**	**Meta Quest Pro**	**HTC Vive XR Elite**	**Meta Quest 3**
Finger Tracking	Partial Thumb And Index Finger Tracking	Partial Thumb And Index Finger Tracking	Partial Finger And Thumb Tracking Via Capacitive Sensors	Partial Finger And Thumb Tracking Via Capacitive Sensors	Partial Finger And Thumb Tracking Via Capacitive Sensors
Haptics	V	V	V	V	V
Batteries	Rechargeable 6 h Battery Life	Rechargeable 6 h Battery Life	Rechargeable 8 h Battery Life	Rechargeable 15 h Battery Life	AA
Sound					
Speakers	Integrated Stereo Headphones	Removable Stereo Headphones	Integrated Stereo Speakers	Integrated Stereo Speakers	Integrated Stereo Speakers
Microphone	V	V	V	V	V
3.5mm Audio Jack	X	V	V	X	V
Connectivity					
Ports	X	USB-C	USB Type-C, Charging Contacts	2 × USB 3.2 Gen 1 Type-C	USB Type-C, Charging Contacts
Wired Video	HDMI, USB-C 3.0	DisplayPort 1.2, USB 3.0	USB Type-C Oculus Link	USB-C	USB Type-C Oculus Link
Wireless Video	X	Available Via VIVE Wireless Adapter, Sold Separately	WiFi Streaming (Virtual Desktop, AirLink)	WiFi Streaming	WiFi Streaming (Virtual Desktop, AirLink)
WiFi	X	X	WiFi 6E	WiFi 6E	WiFi 6E
Bluetooth	X	Bluetooth	Bluetooth 5.2	Bluetooth 5.2 LE	Bluetooth 5.2
System					
Chipset	X	X	Qualcomm Snapdragon XR2+	Qualcomm Snapdragon XR2	Qualcomm Snapdragon XR2 Gen2
CPU	X	X	Octa-Core Kryo 585 (1 × 2.84 GHz, 3 × 2.42 GHz, 4 × 1.8 GHz)	Octa-Core Kryo 585 (1 × 2.84 GHz, 3 × 2.42 GHz, 4 × 1.8 GHz)	Octa-Core Kryo (1 × 3.19 GHz, 4 × 2.8 GHz, 3 × 2 GHz)
GPU	X	X	Adreno 650	Adreno 650	Adreno 740
Memory	X	X	12 GB LPDDR5	12 GB	8 GB

2.2.3 Visualization of 3D virtual objects

A fundamental challenge in XR systems lies in the efficient real-time visualization of three-dimensional geometries, where rendering constraints directly shape the underlying geometric representations and define their role beyond mere descriptive abstraction. The suitability of a given geometric representation for XR applications is therefore primarily determined by its compatibility with real-time GPU rendering pipelines, which impose strict constraints on latency, memory bandwidth, and computational complexity [74]. As a consequence, three-dimensional geometry is represented through a variety of formalisms originating from different domains, including computer-aided design (CAD), computer graphics, and scientific visualization [75].

Parametric representations, such as Bézier curves, B-splines, and Non-Uniform Rational B-Splines (NURBS), define geometry analytically through sets of control points

and basis functions (Figure 2.7) [76]. These formulations provide high geometric accuracy and smoothness and are widely adopted in CAD applications, where accuracy is a primary requirement. However, their evaluation requires continuous computation of basis functions, making them computationally expensive for real-time rendering. For this reason, they are typically discretized into polygonal approximations prior to visualization in interactive systems.

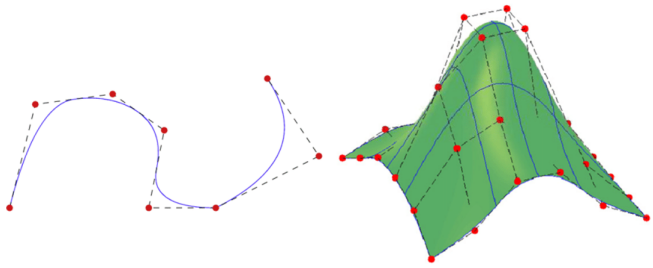


Figure 2.7: Example of a NURBS surface representation [77].

Subdivision surfaces generate smooth geometries through iterative refinement of an initial coarse mesh (Figure 2.8) [78] . While they are extensively used in modeling and animation due to their ability to produce visually high-quality surfaces, the recursive refinement process introduces additional computational overhead. In real-time applications, subdivision surfaces are therefore commonly evaluated offline or converted into fixed polygonal representations.

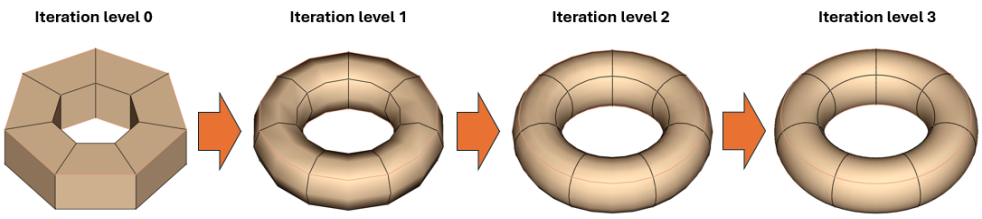


Figure 2.8: Example of a subdivision surface representation.

Implicit surfaces describe geometry not through an explicit mesh of vertices and edges, but as a continuous scalar field defined over space. In this representation, a surface is implicitly defined as the set of points where the scalar field evaluates to zero (the zero-level set). Points inside and outside the object are typically distinguished by the sign of the field, enabling the representation of complex and organic shapes (Figure 2.9) [79]. Despite their expressive flexibility, direct rendering requires computationally expensive procedures such as surface extraction via algorithms like marching cubes, which limits their direct applicability in latency-sensitive XR contexts.

Volumetric representations, such as voxel grids, represent objects by discretizing the 3D space into a regular lattice of volumetric elements, each storing scalar or vector values that describe the presence or properties of the object within the corresponding region of space (Figure 2.10) [80]. These representations are particularly effective for data-centric applications such as medical imaging and scientific visualization. However, their memory

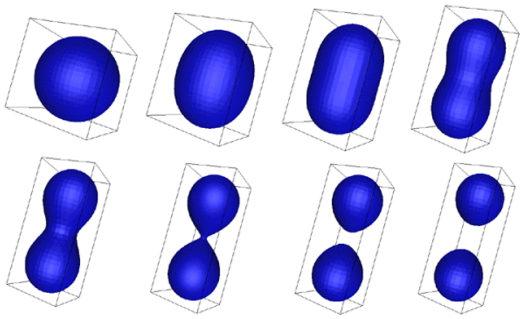


Figure 2.9: Example of an implicit surface representation.

footprint and the cost of rendering techniques such as ray casting or voxel traversal [81] reduce their efficiency in interactive rendering scenarios [82].

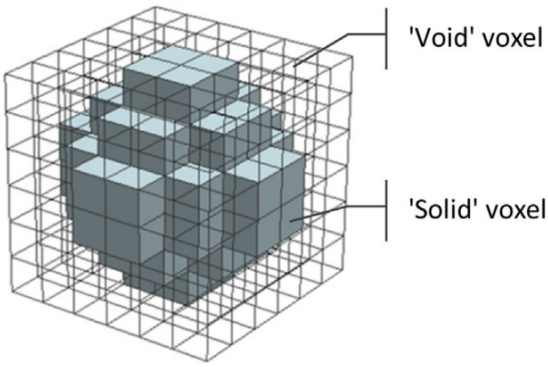


Figure 2.10: Example of voxel grids representation [83].

Point-based representations, including point clouds, provide an efficient means of capturing and storing spatial data, particularly through 3D scanning technologies [84]. Nevertheless, the absence of explicit connectivity information complicates surface reconstruction, shading, and lighting computation, requiring additional processing stages that may conflict with strict real-time constraints.

Taken together, these characteristics indicate that geometric representations exhibit different trade-offs in terms of their suitability for interactive visualization. In real-time XR systems, this suitability is strongly influenced by compatibility with GPU-oriented rendering architectures, which impose constraints on computational cost. Consequently, most non-polygonal representations must be transformed into GPU-efficient formats prior to rendering. Within this context, polygonal meshes constitute the dominant representation in current XR systems [85]. They approximate continuous surfaces using discrete planar primitives, typically triangles, which are directly supported by hardware rasterization pipelines. This alignment between data structure and GPU execution model enables an efficient balance between geometric fidelity and computational performance, which explains their widespread adoption in real-time interactive visualization [86].

A polygonal mesh can be formally interpreted as a discrete piecewise-linear approximation of a continuous surface, defined by a finite set of vertices, edges, and faces.

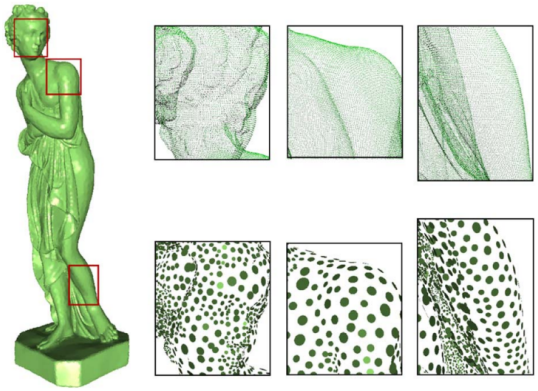


Figure 2.11: Example of a point-based representation.

Vertices represent sampled positions in three-dimensional Euclidean space, while edges encode adjacency relationships and faces constitute oriented surface elements. In practice, faces are typically restricted to triangular primitives, which ensure planarity, avoid geometric ambiguities, and provide a consistent representation for arbitrarily complex surfaces (Figure 2.12).

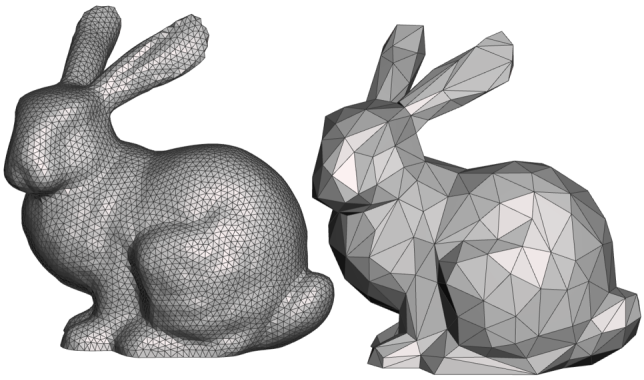


Figure 2.12: Example of a polygonal mesh representation.

In addition to their geometric structure, polygonal meshes are enriched with per-vertex attributes that define surface appearance during rendering. Each vertex encodes not only a spatial position but also auxiliary information describing local surface properties. Among the most relevant attributes are normal vectors, which specify the local surface orientation and determine light–surface interaction, and texture coordinates, which establish a mapping between surface points and a two-dimensional image space, enabling the application of detailed visual patterns [87]. Additional parameters may encode material properties that influence shading behavior. During rendering, these attributes are interpolated across triangle surfaces and evaluated within shading models, allowing the renderer to compute the combined effect of illumination, material response, and viewing direction in producing the final appearance of the object [88].

In real-time XR rendering, the efficiency of polygonal mesh visualization is influenced

not only by geometric complexity, but also by conventions governing surface visibility. Since polygonal models are typically represented through triangular primitives, modern graphics pipelines employ optimization strategies to avoid processing surfaces that cannot contribute to the final image. Among these, one of the most widely adopted techniques is back-face culling (Figure 2.13a), which discards polygons whose orientation indicates that they are facing away from the observer [89].

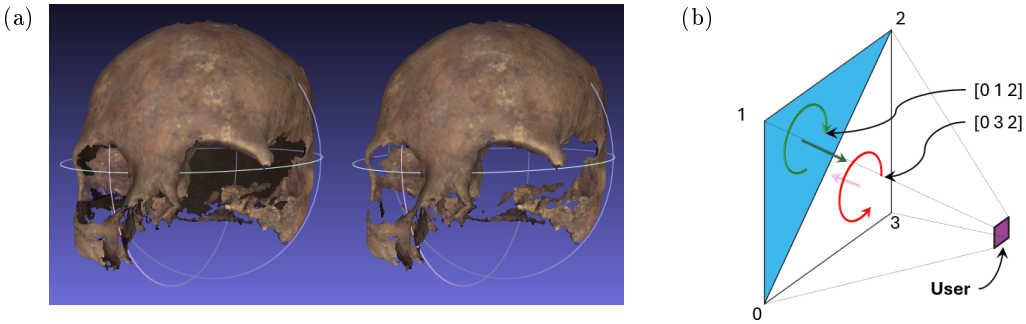


Figure 2.13: (a) Back-face culling and (b) Clockwise winding order in a left-handed reference system [22].

This operation relies on the ordered definition of triangular primitives. A triangle is specified by an ordered triplet of vertices, whose permutation defines the winding order and enables the rendering pipeline to infer the corresponding surface normal through the cross product of two triangle edges. This information is then used to classify primitives as front-facing or back-facing with respect to the viewing direction [90]. Since the sign of the cross product depends on the adopted reference frame, the interpretation of winding order is inherently linked to the handedness of the global coordinate system. Consequently, clockwise and counterclockwise conventions acquire meaning only relative to the spatial convention defined for the scene (Figure 2.13b).

Within XR development, real-time applications are predominantly implemented using engines such as Unity and Unreal Engine, which commonly adopt left-handed coordinate systems. This choice is motivated by compatibility with screen-space conventions and simplifies projection and camera transformations in real-time rendering pipelines, contributing to the widespread use of left-handed spatial conventions in XR software ecosystems [91].

Chapter 3

Objects tracking methodologies

The integration of Digital Twins with XR technologies enables immersive visualization and intuitive interaction with physical systems through their virtual counterparts. The effectiveness of such integration can be improved by introducing spatial alignment between physical and digital entities. To achieve an accurate overlay, real objects must be mapped into the virtual environment through tracking systems. The estimation of a real object's pose can be defined as the process of continuously determining its six-degree-of-freedom (6-DoF) state, consisting of its three-dimensional position and orientation within a reference coordinate system.

In particular, the orientation of a rigid body, as explained in Appendices A.2 and A.3, is described by the rotation matrix $\mathbf{R}$, which defines the components of the body-fixed reference frame relative to the absolute reference frame. This matrix can be parameterized using different representations, such as Euler angles (Appendix A.3.2), Rodrigues parameters (Appendix A.3.3), Euler parameters (Appendix A.3.4), and quaternions (Appendix A.3.5). Among these, quaternions are widely adopted in computer graphics software, such as Unity Engine (Appendix B), due to their computational efficiency and their ability to avoid singularity issues, such as gimbal lock [92], that may occur when using Euler-angle-based representations.

A quaternion is conventionally expressed as the sum of one real component and three imaginary components. Thus, given $\mathbf{q} \in \mathbb{H}$ and $q_0, q_1, q_2, q_3 \in \mathbb{R}$:

$$\mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k \tag{3.1}$$

An alternative notation emphasizes the decomposition into a real scalar part $s \in \mathbb{R}$ and an imaginary vector part $\mathbf{v} \in \operatorname{Im}\mathbb{H}$, where $\mathbb{H} = \mathbb{R} \oplus \operatorname{Im}\mathbb{H}$:

$$\mathbf{q} = (s, \mathbf{v}) \tag{3.2}$$

Quaternions $\mathbf{q} \in \operatorname{Im}\mathbb{H}$ with zero scalar part, i.e. of the form $\mathbf{q} = (0, \mathbf{v})$, can be directly associated with vectors in Euclidean space and are commonly used to represent rotations. In this setting, the endomorphism $\mathbf{p} \in \mathbb{H} \to \mathbf{p}' \in \mathbb{H}$, where $\mathbf{p}$ is defined as $\mathbf{p} = (0, \mathbf{v})$, induces a rotation of the vector $\mathbf{v}$ into the rotated vector $\mathbf{v}'$:

$$\begin{aligned} \mathbf{p}' &= \mathbf{q}\mathbf{p}\mathbf{q}^* \\ (0, \mathbf{v}') &= \mathbf{q}(0, \mathbf{v})\mathbf{q}^* \end{aligned} \tag{3.3}$$

This formulation naturally allows sequences of rotations to be expressed in a compact and computationally convenient way. For instance, applying first a rotation that maps $\mathbf{p}$ into $\mathbf{p}'$ via $\mathbf{q}_a$, and subsequently rotating $\mathbf{p}'$ into $\mathbf{p}''$ via $\mathbf{q}_b$, yields:

$$\mathbf{p}' = \mathbf{q}_a \mathbf{p} \mathbf{q}_a^* \tag{3.4a}$$
$$\mathbf{p}'' = \mathbf{q}_b \mathbf{p}' \mathbf{q}_b^* \tag{3.4b}$$
$$\mathbf{p}'' = (\mathbf{q}_b \mathbf{q}_a)\mathbf{p}(\mathbf{q}_a^* \mathbf{q}_b^*) \tag{3.4c}$$

More generally, given a sequence of n rotations represented by quaternions $\mathbf{q}_1, \ldots, \mathbf{q}_n$, the following compact expression can be written:

$$\mathbf{p}' = \mathbf{q}_s \mathbf{p} \mathbf{q}_s^* \tag{3.5}$$

where $\mathbf{q}_s = (\mathbf{q}_n, \ldots, \mathbf{q}_2, \mathbf{q}_1)$. It is important to emphasize that, since quaternion multiplication is not commutative, the order of application of $\mathbf{q}_1, \ldots, \mathbf{q}_n$ cannot be arbitrarily permuted without altering the resulting rotation.

The equivalence between the quaternion formulation and classical rotation representations can be established by introducing the rotation matrix $\mathbf{R}$ such that $\mathbf{v}' = \mathbf{R}\mathbf{v}$, and by identifying the quaternion $\mathbf{q}$ that generates the same transformation through $(0, \mathbf{v}') = \mathbf{q}(0, \mathbf{v})\mathbf{q}^*$.

To obtain a compact matrix representation of quaternion multiplication, the matrices $\mathbf{M}_\oplus$ and $\mathbf{M}_\ominus$ are introduced. These operators allow the quaternion product to be expressed as an equivalent linear operation in $\mathbb{R}^4$, thereby simplifying its implementation in computational frameworks:

$$\mathbf{ab} = \mathbf{c} \iff \mathbf{M}(\mathbf{a})_\oplus \mathbf{b} = \mathbf{M}(\mathbf{b})_\ominus \mathbf{a} \tag{3.6a}$$

$$\mathbf{M}(\mathbf{a})_\oplus = \begin{bmatrix} +a_0 & -a_1 & -a_2 & -a_3 \\ +a_1 & +a_0 & -a_3 & +a_2 \\ +a_2 & +a_3 & +a_0 & -a_1 \\ +a_3 & -a_2 & +a_1 & +a_0 \end{bmatrix} \tag{3.6b}$$

$$\mathbf{M}(\mathbf{b})_\ominus = \begin{bmatrix} +b_0 & -b_1 & -b_2 & -b_3 \\ +b_1 & +b_0 & +b_3 & -b_2 \\ +b_2 & -b_3 & +b_0 & +b_1 \\ +b_3 & +b_2 & -b_1 & +b_0 \end{bmatrix} \tag{3.6c}$$

where the quaternions are defined as $\mathbf{a} = a_0 + a_1 i + a_2 j + a_3 k$, $\mathbf{b} = b_0 + b_1 i + b_2 j + b_3 k$, and $\mathbf{c} = c_0 + c_1 i + c_2 j + c_3 k$. The product 3.3 can be expressed in matrix form using 3.6, yielding:

$$\begin{aligned} \mathbf{p}' &= \mathbf{M}(\mathbf{q})_\oplus(\mathbf{p}\mathbf{q}^*) = \mathbf{M}(\mathbf{q})_\oplus \mathbf{M}(\mathbf{q}^*)_\ominus \mathbf{p} \\ &= \begin{bmatrix} +q_0 & -q_1 & -q_2 & -q_3 \\ +q_1 & +q_0 & -q_3 & +q_2 \\ +q_2 & +q_3 & +q_0 & -q_1 \\ +q_3 & -q_2 & +q_1 & +q_0 \end{bmatrix} \begin{bmatrix} +q_0 & +q_1 & +q_2 & +q_3 \\ -q_1 & +q_0 & -q_3 & +q_2 \\ -q_2 & +q_3 & +q_0 & -q_1 \\ -q_3 & -q_2 & +q_1 & +q_0 \end{bmatrix} \mathbf{p} \\ &= \begin{bmatrix} q_0^2 + q_1^2 + q_2^2 + q_3^2 & 0 & 0 & 0 \\ 0 & q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_3 q_0) & 2(q_1 q_3 + q_2 q_0) \\ 0 & 2(q_1 q_2 + q_3 q_0) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(-q_1 q_0 + q_2 q_3) \\ 0 & 2(q_1 q_3 - q_2 q_0) & 2(q_1 q_0 + q_2 q_3) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \mathbf{p} \end{aligned} \tag{3.7}$$

Since only the imaginary part of $\mathbf{p}' = (0, \mathbf{v}')$ is relevant, and the real part of $\mathbf{p} = (0, \mathbf{v})$ is identically zero, the first row and first column of the previous expression can be discarded. This leads to the standard 3×3 rotation matrix formulation:

$$\mathbf{v}' = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_3q_0) & 2(q_1q_3 + q_2q_0) \\ 2(q_1q_2 + q_3q_0) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(-q_1q_0 + q_2q_3) \\ 2(q_1q_3 - q_2q_0) & 2(q_1q_0 + q_2q_3) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \mathbf{v} = \mathbf{R}(\mathbf{q})\mathbf{v} \tag{3.8}$$

In order to obtain a more intuitive representation of rigid body orientation, Euler angles can be used to construct the corresponding rotation matrix to transform from the body-fixed reference frame to the absolute reference frame. In Unity Engine, a 213 rotation sequence is adopted, i.e., a rotation about the z-axis is applied first, followed by a rotation about the x-axis, and finally a rotation about the y-axis, leading to the following formulation:

$$\mathbf{R}_{213} = \mathbf{R}_{\varphi}^{y}\mathbf{R}_{\theta}^{x}\mathbf{R}_{\psi}^{z} = \begin{bmatrix} c_\varphi c_\psi + s_\theta s_\varphi s_\psi & -c_\varphi s_\psi + s_\theta s_\varphi c_\psi & c_\theta s_\varphi \\ c_\theta s_\psi & c_\theta c_\psi & -s_\theta \\ -s_\varphi c_\psi + s_\theta c_\varphi s_\psi & s_\varphi s_\psi + s_\theta c_\varphi c_\psi & c_\theta c_\varphi \end{bmatrix} \tag{3.9}$$

where $\mathbf{R}_{\varphi}^{y}$, $\mathbf{R}_{\theta}^{x}$, and $\mathbf{R}_{\psi}^{z}$ denote the rotation matrices about the y, x, and z axes by angles φ, θ, and ψ, respectively, while c and s denote the cosine and sine of the corresponding angle. Using the Equation A.115a, the corresponding quaternion can be derived, which represents the rotation in the Unity Engine.

Depending on the sensing technology and application requirements, several tracking approaches can be employed to estimate object pose and ensure robust alignment between real and virtual content. The following sections provide an overview of the main tracking strategies adopted in XR applications, including vision-based approaches such as marker-based tracking and software development kit (SDK) based solutions implemented through Vuforia Engine, as well as sensors-based tracking systems developed for the HTC VIVE ecosystem.

3.1 Vision-based tracking

Vision-based tracking systems estimate the pose of an object through the processing of images acquired by one or more cameras. These approaches are widely used in Augmented Reality applications because they can operate without dedicated external infrastructures and can be implemented using standard imaging devices.

One of the most common methodologies is marker-based tracking, where artificial fiducial markers are detected within the camera frame to estimate the relative pose between the camera and the observed object. Fiducial markers are designed to maximize detection robustness and pose estimation accuracy under different environmental conditions. For this reason, they generally adopt high contrast patterns, such as black and white, and their internal structure is composed of regular geometric regions arranged according to predefined binary patterns.
They have a square geometry because the four corners provide well defined reference points that allow the position and orientation of the marker relative to the camera to be

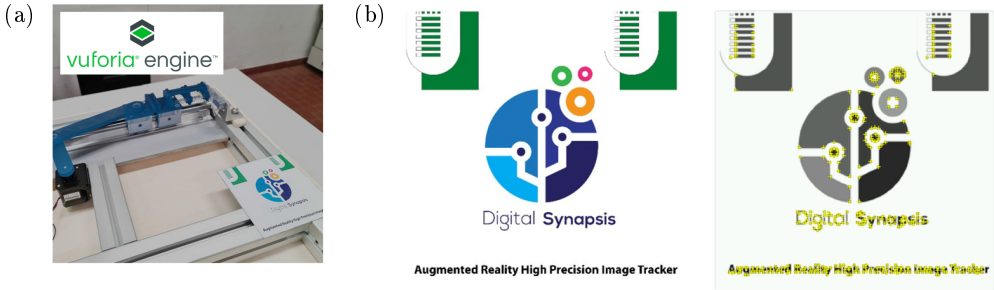


Figure 3.1: (a) Image Target application example and (b) real 2D image target (left) and corresponding tracked features (right).

estimated from its perspective projection in the image, by solving the Perspective-n-Point (PnP) problem [93, 94].

In contrast, markerless image-based approaches exploit natural image features such as corners, edges, or texture patterns extracted directly from the observed scene [95, 96], while model-based tracking systems rely on the three-dimensional geometry of the object, typically derived from CAD models, to estimate the pose by matching projected model features with image observations [97, 98].

In this work, both image-based and model-based tracking approaches were implemented using Vuforia Engine, a computer vision–based SDK for Augmented Reality applications. Vuforia performs real-time detection and tracking of visual features extracted from the camera stream to estimate the 6-DoF pose of predefined targets, abstracting low-level processes such as feature extraction, matching, and pose estimation. The framework supports different tracking paradigms tailored to specific classes of objects, including Image Targeting, Multi Targeting, Cylinder Targeting, and Model Targeting. For the configuration of these tools in Unity Engine, please refer to Appendix C.

Despite their flexibility, vision-based methodologies may suffer from limitations related to lighting variations, motion blur, occlusions, or the lack of sufficiently distinctive visual features, and their performance can significantly degrade in high-speed motion scenarios, where image acquisition becomes unstable and reliable feature tracking is no longer guaranteed.

3.1.1 Image-based tracking

Regarding image-based tracking approaches, Vuforia provides several tracking paradigms, each designed for different types of physical objects and interaction scenarios. These approaches vary in their geometric assumptions, robustness to occlusions, and achievable accuracy. Among them, planar image-based targets are the simplest and most commonly used solution. At the same time, Vuforia also extends image-based tracking to curved and multi-faceted objects, which makes it possible to handle a wider range of real-world components. The main supported types are the following:

- **Image target**
  Image Targets are essentially flat, two-dimensional surfaces that contain enough visual detail to allow reliable feature detection. Elements such as corners, edges, and

textured regions are identified and tracked from different viewpoints in a consistent way (Figure 3.1b). Once the target is detected, its pose is estimated assuming a planar model, and it is continuously updated as either the camera or the target moves (Figure 3.1a). This type of target works particularly well for rigid, flat, and well-textured surfaces such as printed images, technical drawings, or specially designed patterns. Because they are easy to create and integrate, Image Targets are widely used in Augmented Reality applications, especially during prototyping and proof-of-concept development.

- **Multi target**

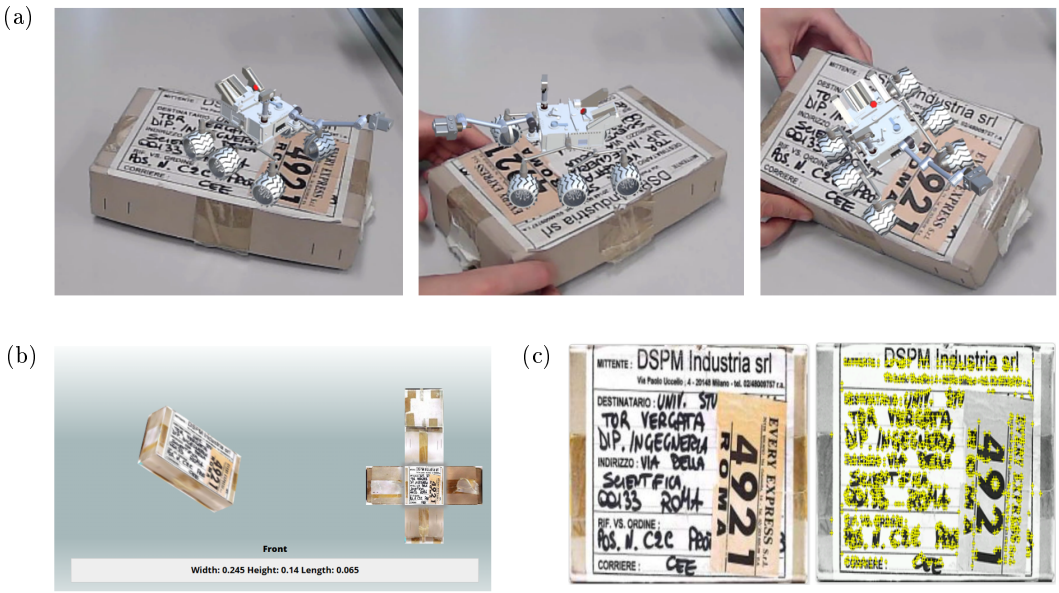


Figure 3.2: (a) Multi Target application example, (b) parallelepiped unwrapping and (c) real front image target (left) and corresponding tracked features (right).

Multi-Targets are made up of multiple Image Targets arranged according to a known spatial configuration, such as the faces of a parallelepiped (Figure 3.2b). Like Image Targets, they rely on natural visual features (Figure 3.2c). Each face is tracked separately, while the overall pose is reconstructed by enforcing the predefined geometric relationships between them. This approach improves robustness and helps reduce pose jitter, since several visual constraints contribute to the tracking result at the same time. On the other hand, it requires careful physical assembly and accurate alignment between the real object and its virtual representation (Figure 3.2a).

- **Cylinder target**

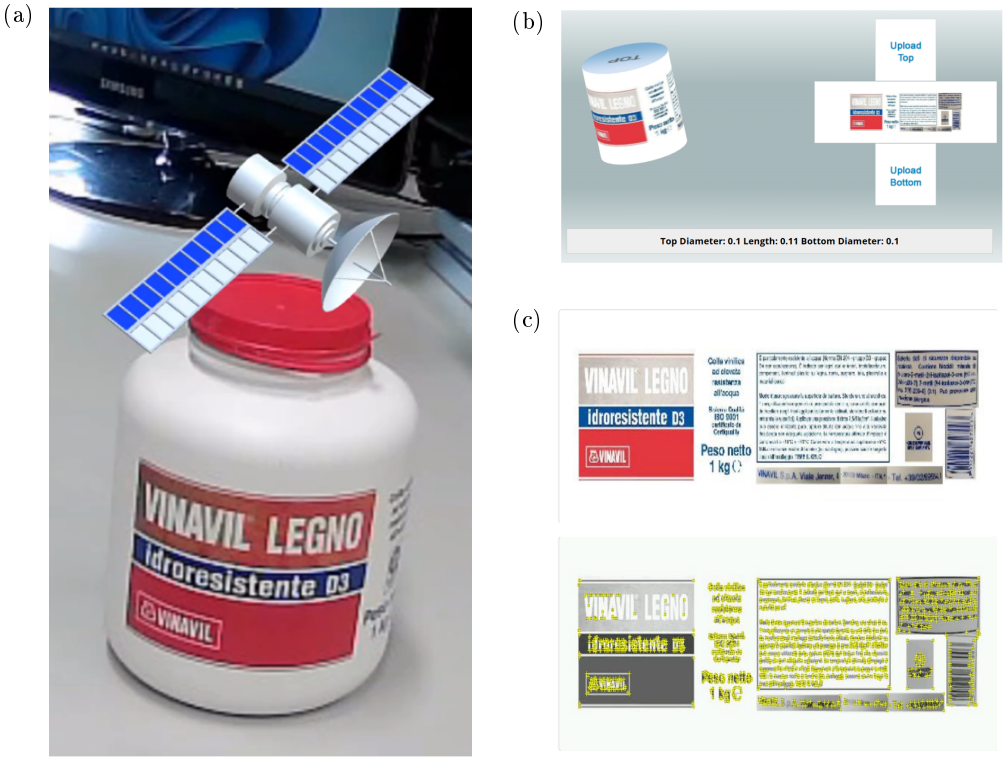


Figure 3.3: (a) Example of a Cylinder Target application, (b) cylindrical unwrapping, and (c) real side image target (top) and corresponding tracked features (bottom).

Cylinder Targets extend planar tracking to objects with cylindrical geometry. In this case, the target is defined by an image wrapped around a virtual cylinder, whose geometric parameters are specified during the setup phase (Figure 3.3b). This allows the system to track the object as long as sufficient visual detail is available along the curved surface. This approach improves tracking robustness for objects that are frequently rotated around their main axis (Figure 3.3a). Although Cylinder Targets provide better pose continuity compared to planar targets in some situations, their accuracy still depends on natural features, such as image target. and correct geometric parametrization (Figure 3.3c).

The quality of an Image Target strongly influences the performance of image-based AR tracking systems. To provide an indication of the expected tracking robustness, Vuforia assigns each target a rating from zero to five stars. In general, highly rated targets provide more stable pose estimation under different viewing angles, distances, and illumination conditions, while low-rated targets are more likely to suffer from tracking instability or target loss (Figure 3.4).

This rating mainly depends on the number and distribution of the visual features detected in the image. In Vuforia, features correspond to distinctive local details, such as corners, edges, or textured regions, that can be reliably recognized across consecutive camera frames. Images containing many well-distributed and non-repetitive features usually lead to better tracking performance, since they reduce ambiguities during feature

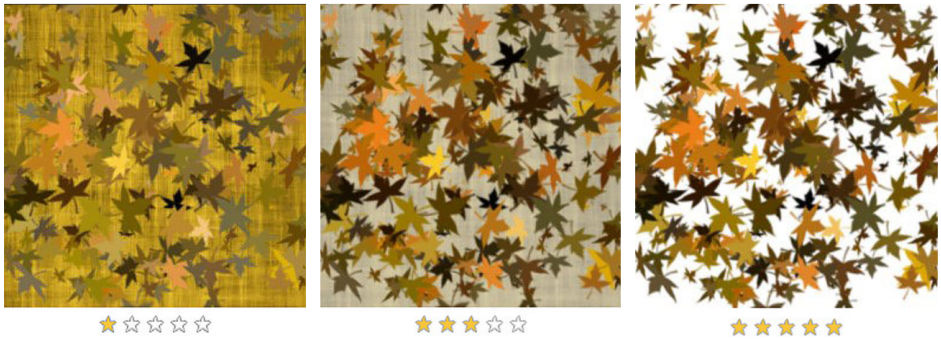


Figure 3.4: Vuforia Engine rating system for visual features detected.

matching and improve pose stability.

Images with high local contrast and rich texture content generally produce better results because they allow the extraction of a larger number of salient feature points. On the contrary, homogeneous regions, repetitive patterns, strong symmetries, or blurred areas tend to reduce tracking quality, as they provide fewer distinctive details for reliable matching. Reflective materials and low-resolution textures can also negatively affect the detection process.

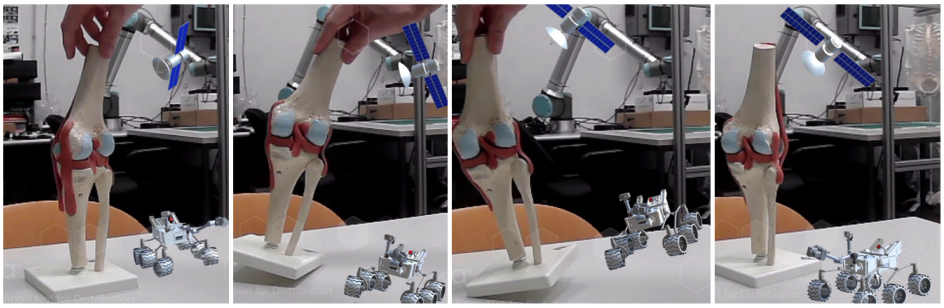
Tracking performance is additionally influenced by acquisition and environmental conditions. Variations in illumination, reflections, partial occlusions, and poor printing quality may alter the visibility of the extracted features and reduce tracking robustness. Motion blur represents another critical aspect, especially during high-speed movements, where rapid camera displacements can compromise feature detection and temporal consistency between frames.

For this reason, Vuforia recommends using targets with ratings of at least four stars in applications requiring stable and continuous tracking. The feature visualization provided by the Vuforia Target Manager can also be used to verify that the extracted feature points are sufficiently distributed across the entire target surface rather than concentrated in limited regions.

The same considerations apply to Multi Targets, where each surface should contain enough distinctive visual information to maintain stable tracking during viewpoint changes or partial occlusions. Similarly, Cylinder Targets require textures with sufficiently rich and uniformly distributed features along the curved surface, while reflective, transparent, or poorly textured materials should generally be avoided. The configuration of image-based targets is provided in Appendix C.1.

3.1.2 Model-based tracking

(a)



(b)

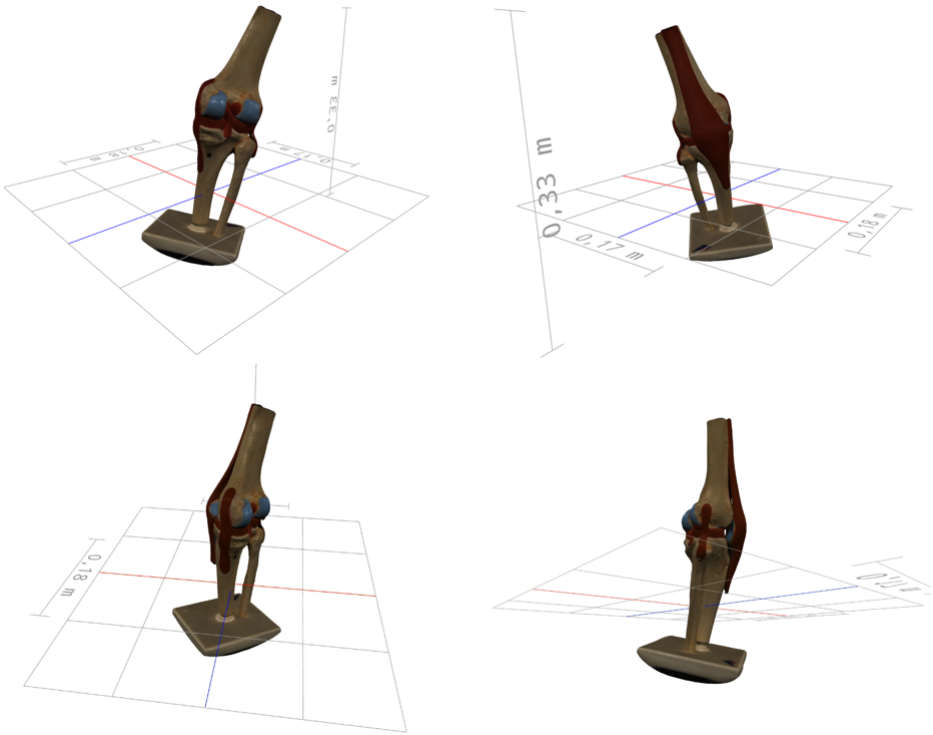


Figure 3.5: (a) Model Target application example and (b) guide views used for the tracking.

The Vuforia Engine system, through its model targets functionality, enables the recognition and tracking of three-dimensional objects in the real world based exclusively on their geometry. This technology relies on pre-existing 3D models, typically derived from CAD files or 3D scans, which are used as a reference to identify a physical object and maintain its tracking during interactions in augmented reality. This approach is applicable to a wide range of contexts, including small-scale objects, complex industrial devices, vehicles, and large-scale structures.

In order to ensure reliable performance, it is essential that the digital model accurately represents the real-world object. The fundamental principle of tracking requires a consis-

tent geometric correspondence between the virtual model and its physical counterpart. For this reason, the technology is particularly suitable for rigid and static objects, while it becomes less effective in the case of deformable structures or those subject to changes in their geometric configuration. Similarly, surfaces characterized by high transparency or reflectivity may compromise the recognition process, as they reduce the amount of visual information available to the tracking algorithms.

A significant role is played by the geometric complexity of the model. The system tends to perform more robustly when the object exhibits distinctive structural features such as edges, surface discontinuities, and depth variations, which enable a clear differentiation from the surrounding environment. Conversely, overly simple geometries or those characterized by high symmetry may introduce ambiguities in the recognition process, making it more difficult to determine the correct spatial orientation. Repetitive structural patterns can also negatively affect tracking stability due to the similarity between different regions of the model.

Another critical aspect concerns the dimensional consistency between the digital model and the physical object. The correct scale of the CAD model is essential to ensure accurate spatial alignment. Significant differences in size may compromise tracking precision and the stability of the overlay between virtual content and the real object. Although the system can tolerate minor discrepancies, these must remain within limited margins to avoid performance degradation.

The surface properties of the object further influence recognition effectiveness. The presence of textures, color variations, or visual details enriches the available information, facilitating the identification process. In contrast, uniform surfaces lacking texture or characterized by extreme colors such as pure white or absolute black are generally more difficult to track, especially under suboptimal lighting conditions. Reflective or transparent materials can also reduce system reliability, as they alter or diminish the perception of geometric features.

Although color correspondence between the digital model and the real object is not a strict requirement, the use of coherent textures can significantly improve tracking robustness. In particular, textures derived from real images or photogrammetric scans can contribute to more stable recognition, especially in variable or partially occluded environments.

Finally, the preparation of the CAD model plays a decisive role in the overall efficiency of the system. Excessively complex geometric models may be inefficient in real-time execution scenarios, particularly on mobile devices. It is therefore often necessary to simplify the mesh by removing non-essential details and non-visible internal components, in order to reduce computational load without compromising the overall fidelity of the shape. It is also essential to ensure the correctness of the geometric representation, including the consistency of surface normals, in order to avoid artifacts that could negatively affect the recognition process.

Figure 3.5a shows an example of a model target developed from a CAD model generated using an Artec EVA scanner, while Figure 3.5b illustrates the guide views used to facilitate the recognition of the real object. The configuration of model target is provided in Appendix C.2.

3.2 Sensor-based tracking

Sensor-based tracking systems estimate the pose of an object using dedicated hardware devices capable of directly measuring motion and orientation. Among the commercially available tracking systems are the HTC VIVE Tracker 2.0 and 3.0, the HTC VIVE Wrist Tracker, and the HTC VIVE Ultimate Tracker. Compared to vision-based tracking approaches, the VIVE tracking ecosystem provides higher spatial accuracy, greater robustness, and improved temporal stability for moving objects.

These devices rely on different tracking methodologies. The HTC VIVE Tracker 2.0 and 3.0 are based on an outside-in tracking paradigm, in which the position and orientation of each tracker are computed relative to a set of fixed infrared reference units, known as "Lighthouse" base stations. As previously described for the HTC VIVE Pro 1 and 2 systems (Section 2.2.2), these trackers are equipped with photodiodes that detect infrared signals emitted by the base stations, enabling spatial triangulation within the real-world environment.

Let $\mathbf{r}_1$ denote the coordinates of a point P of the real object relative to the reference frame associated with the tracker. Since the tracker is rigidly attached to the object, these relative coordinates remain constant over time. Knowing the relative pose of both the tracker and the HMD with respect to the Lighthouse base stations, and exploiting the rotation matrix defined in Equation 3.9, it is possible to determine the rotation matrix $\mathbf{R}$ and the position vector $\mathbf{d}$ of the tracker with respect to the absolute reference frame. Consequently, the coordinates of the point P of the real object in the absolute reference frame of the virtual environment, denoted by $\mathbf{r}_0$, can be determined according to:

$$\mathbf{r}_0 = \mathbf{R}\mathbf{r}_1 + \mathbf{d} \tag{3.10}$$

According to this equation, the pose of a real object, considered as a rigid body, in a virtual environment is fully defined.

(a) HTC VIVE Tracker 2.0 (b) HTC VIVE Tracker 3.0

Figure 3.6: HTC VIVE tracker left-handed reference systems.

The HTC VIVE Tracker 2.0 is shown in Figure 3.6a, while the HTC VIVE Tracker 3.0 is illustrated in Figure 3.6b, where the reference coordinate systems used to determine the pose of the physical object within the global reference frame are reported. Although both devices rely on the same tracking principle and provide comparable accuracy, the

Tracker 3.0 features a more compact and lightweight design, making it more suitable for handheld tools and dynamic interactions without compromising usability or tracking performance.

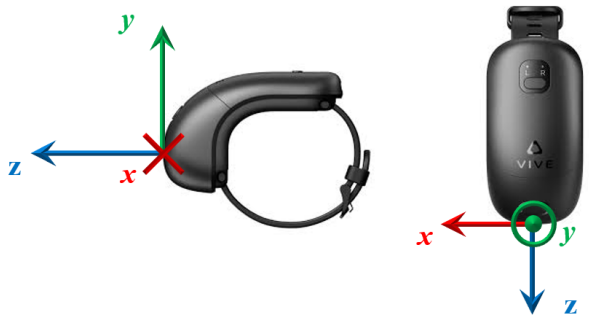


Figure 3.7: HTC VIVE Wrist tracker left-handed reference system.

The HTC VIVE Wrist Tracker (Figure 3.7) is based on an inside-out tracking system, in which the position and orientation of the device are estimated using an internal Inertial Measurement Unit (IMU), combined with visual information provided by cameras mounted on the HMD, such as the VIVE XR Elite. This sensor fusion approach allows the system to continuously track the tracker's motion in real time without requiring external base stations, improving portability and ease of setup while maintaining accurate spatial awareness.

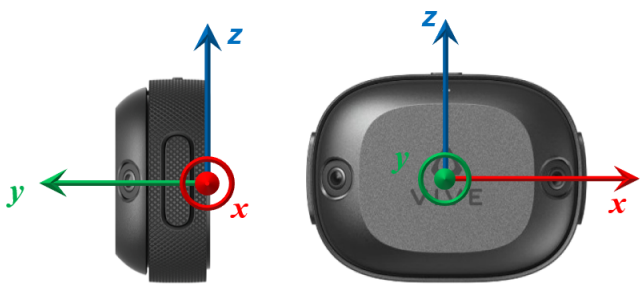


Figure 3.8: HTC VIVE Ultimate tracker left-handed reference system.

At last, the HTC VIVE Ultimate Tracker (Figure 3.8) is also based on an inside-out tracking system; however, unlike other solutions, it estimates its position and orientation using onboard 6-DoF cameras integrated directly into the device, in combination with an internal IMU. This self-contained sensing approach allows the tracker to perform real-time spatial localization independently, without relying on an HMD or external lighthouse base stations. As a result, the Ultimate Tracker can operate as a fully standalone tracking device, offering greater flexibility in deployment across different scenarios.

The integration of all these trackers within the Unity Engine is achieved through the SteamVR tracking ecosystem. SteamVR is a software framework developed by Valve Corporation for managing virtual reality devices and spatial tracking systems. It provides a unified interface for communication between VR hardware components, such as HMDs, controllers, trackers, and Lighthouse base stations, and software applications developed in

Figure 3.9: Tracker application example.

game engines such as Unity. Through the SteamVR plugin and the OpenVR or OpenXR APIs, tracking data can be acquired in real time and directly mapped into the virtual environment (Figure 3.9). Listings 3.1 and 3.2 show, respectively, code excerpts where raw tracking data from the devices is processed and converted into usable transformation information within Unity.

Listing 3.1: OpenVR Unity script to directly access the tracker poses at runtime.

```
using UnityEngine;
using Valve.VR;

namespace Valve.VR
{
    public class SteamVR_TrackedObject : MonoBehaviour
    {
        [...]

        // Called when new tracking data is received
        private void OnNewPoses(TrackedDevicePose_t[] poses)
        {
            if (index == EIndex.None) return;

            int i = (int)index;

            isValid = false;
            if (poses.Length <= i) return;
            if (!poses[i].bDeviceIsConnected) return;
            if (!poses[i].bPoseIsValid) return;

            isValid = true;

            // Convert SteamVR pose to Unity transform
            var pose = new SteamVR_Utils.RigidTransform(poses[i].
                mDeviceToAbsoluteTracking);

            if (origin != null)
            {
                transform.position = origin.TransformPoint(pose.pos);
                transform.rotation = origin.rotation * pose.rot;
            }
            else
            {
                transform.localPosition = pose.pos;
```

```
                    transform.localRotation = pose.rot;
                }
            }

            [...]
        }
    }
```

Listing 3.2: OpenXR Unity script to directly access the tracker poses at runtime.

```
using System;
using UnityEngine.InputSystem.LowLevel;

namespace UnityEngine.InputSystem.XR
{
    public class TrackedPoseDriver : MonoBehaviour,
        ISerializationCallbackReceiver
    {
        [...]

        // Applies the latest tracked pose to the Unity Transform
        protected virtual void PerformUpdate()
        {
            SetLocalTransform(m_CurrentPosition, m_CurrentRotation);
        }

        // Updates the Transform position and rotation based on tracking
             data and configuration
        protected virtual void SetLocalTransform(Vector3 newPosition,
            Quaternion newRotation)
        {
            var positionValid = m_IgnoreTrackingState || (
                m_CurrentTrackingState & TrackingStates.Position) != 0;
            var rotationValid = m_IgnoreTrackingState || (
                m_CurrentTrackingState & TrackingStates.Rotation) != 0;

            // Apply rotation if valid and enabled by tracking type
            if (rotationValid &&
            (m_TrackingType == TrackingType.RotationAndPosition ||
            m_TrackingType == TrackingType.RotationOnly))
            {
                transform.localRotation = newRotation;
            }

            // Apply position if valid and enabled by tracking type
            if (positionValid &&
            (m_TrackingType == TrackingType.RotationAndPosition ||
            m_TrackingType == TrackingType.PositionOnly))
            {
                transform.localPosition = newPosition;
            }
        }

        [...]
    }
}
```

Chapter 4

Interaction methodologies

Digital Twins involve interaction between the user and both the physical asset and its digital representation. For Digital Twins integrated with XR technologies, it is necessary to develop or integrate methodologies that enable the user to interrogate and interactively manipulate the digital model. To this end, the main solvers used for the creation and management of XR environments have been analysed, with particular attention to the mathematical modelling of interactions between virtual entities.

The analysis focuses mainly on interactions between virtual objects and on user-virtual object interactions. Interactions between the user and the user interface are addressed more briefly, since the underlying principles are substantially similar to those governing the previous cases.

Interactions between virtual objects fall within the scope of contact modeling in multibody solvers. For this reason, a brief theoretical review of contact models and of the related implementation techniques in multibody dynamics solvers will first be presented. Subsequently, attention will be directed towards the strategies adopted by real-time physics engines, commonly used in the development of XR applications.

User-virtual object interactions share part of the mechanisms used for object-object interaction. In particular, interaction detection is based on the same detection principles used for contact handling, although in many cases it is not necessary to compute full contact forces. The main element distinguishing user-object interaction from object-object interaction is the presence of physical devices capable of capturing the user's intent and converting it into information that can be interpreted by the simulation system. Among the most widely used technologies are dedicated controllers, hand tracking systems with gesture recognition, and haptic devices.

In general, interactions within engines used for the development of XR applications are based on three fundamental steps:

- associating bounding volumes to the objects involved in the interaction [99],
- inserting the interacting objects into the physics engine (real-time Multibody Dynamics [100], such as a sequential impulse solver [101]),
- using event-driven programming [102], where interaction logic is triggered only when specific events occur, instead of being continuously checked through polling, reducing unnecessary load on the simulation loop.

4.1 Interaction detection

The first phase of interaction, in general, is the identification of pairs of objects that may potentially interact and the detection of whether a collision has occurred. Since the use of full three-dimensional object geometry is computationally expensive, especially in scenarios characterized by a large number of entities, simplified geometric representations known as bounding volumes are used.

Bounding volumes are volumetric approximations that fully enclose the object geometry, with the aim of making spatial interaction computations more efficient.

This simplification is related to the nature of the contact problem in multibody systems [103]. Contact between bodies cannot be treated as a simple geometric intersection, but involves physical phenomena such as local material deformation and the generation of reaction forces. Its modelling requires non-penetration constraints and nonlinear interaction laws, whose solution can be computationally expensive, especially in systems with multiple interacting bodies [104]. In development environments such as the Unity Engine, these volumes are implemented through physical components known as *colliders*.

Bounding volumes can be implemented using different geometric shapes, depending on the trade-off between approximation accuracy and computational cost:

- Bounding Sphere: a sphere defined by a center $\mathbf{C}$ and a radius r.
- Axis-Aligned Bounding Box (AABB): a box aligned with the axes of the global reference frame. This volume can be represented either by the extreme coordinates along the three directions, $\mathbf{b}_{\min} = \{x_{\min}, y_{\min}, z_{\min}\}$ and $\mathbf{b}_{\max} = \{x_{\max}, y_{\max}, z_{\max}\}$, or equivalently by its center $\mathbf{C}$ and the vector $\mathbf{e}$ of three half-extents, which define its spatial extension along each axis [105].
- Oriented Bounding Box (OBB): a box arbitrarily oriented in three-dimensional space. This volume can be represented by the center $\mathbf{C}$, a rotation matrix $\mathbf{R} = [\mathbf{u}_1\ \mathbf{u}_2\ \mathbf{u}_3]$, where $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$ form an orthonormal basis defining the local frame of the box, and by the scalar half-extents e_1, e_2, e_3, which define its extension along the respective local axes [106].
- Capsule: a three-dimensional solid formed by a cylinder with hemispherical ends, defined by two points $\mathbf{P}$ and $\mathbf{Q}$ and a radius r [107].
- k-Discrete Oriented Polytope (k-DOP): a convex polyhedron obtained as the intersection of k pairs of parallel planes. Each pair is defined by a direction $\mathbf{d}_i$ and two scalar values l_i, u_i, which bound the solid between two parallel planes orthogonal to $\mathbf{d}_i$ [108].
- Convex Hull: the minimal convex envelope of the set of points defining a geometry.

In general, real-time collision detection pipelines are organized in a hierarchical structure that distinguishes a *broad-phase*, in which candidate pairs of potentially interacting objects are rapidly identified, typically through the use of 3D grids, and a *narrow-phase*, in which more accurate geometric tests are performed on the actual primitives [109]. In this latter phase, collision detection algorithms depend on the type of bounding volume

associated with the colliding objects and consist of verifying specific geometric intersection conditions.

4.1.1 Analytic distance and closest-point methods

Analytic distance and closest-point methods are used when collision detection is based on geometric queries between simple primitives. Typical examples include Sphere-Sphere, Capsule-Capsule, Sphere-AABB, and Sphere-OBB.

4.1.1.1 Sphere-sphere interaction

In the case of sphere-sphere interaction, the overlap test is based on the Euclidean distance between the sphere centers $\mathbf{C}_1$ and $\mathbf{C}_2$, which is computed and compared against the sum of the sphere radii r_1 and r_2.

$$\|\mathbf{C}_2 - \mathbf{C}_1\|^2 \leq (r_1 + r_2)^2 \tag{4.1}$$

4.1.1.2 Capsule-capsule interaction

In the case of capsule-capsule interaction, a capsule K_i is defined as the set of points obtained as the union of spheres of radius r_i, whose center moves along a line segment in space. This segment is described by the vector function:

$$\mathbf{L}_i(t) = \mathbf{P}_i + t(\mathbf{Q}_i - \mathbf{P}_i), \quad t \in [0, 1] \tag{4.2}$$

where $\mathbf{P}_i$ is the starting point of the segment, $\mathbf{Q}_i$ is the endpoint of the segment, and t is the scalar parameter of linear interpolation. The collision detection problem between two capsules K_1 and K_2 reduces to determining the minimum distance between their respective axis segments $\mathbf{L}_1(s)$ and $\mathbf{L}_2(t)$, with $s, t \in [0, 1]$. The squared distance between two points belonging to the segments is defined as:

$$d(s, t) = \|\mathbf{L}_1(s) - \mathbf{L}_2(t)\|^2. \tag{4.3}$$

The parameters s^* and t^* that identify the pair of closest points are defined as the solution of the constrained optimization problem:

$$(s^*, t^*) = \arg \min_{s,t \in [0,1]} d(s, t) \tag{4.4}$$

These parameters are obtained by enforcing the stationarity conditions:

$$\frac{\partial d}{\partial s} = 0 \tag{4.5a}$$

$$\frac{\partial d}{\partial t} = 0 \tag{4.5b}$$

and subsequently projecting the resulting values onto the interval $[0, 1]$. The points corresponding to the minimum distance between the segments are:

$$\mathbf{M}_1 = \mathbf{L}_1(s^*) \tag{4.6a}$$

$$\mathbf{M}_2 = \mathbf{L}_2(t^*) \tag{4.6b}$$

The intersection condition between the two capsules is satisfied if:

$$\|\mathbf{M}_1 - \mathbf{M}_2\|^2 \leq (r_1 + r_2)^2 \tag{4.7}$$

4.1.1.3 Sphere-AABB interaction

In the case of sphere-AABB interaction, the intersection between an AABB and a sphere $S = (\mathbf{C}, r)$ can be tested by identifying the point $\mathbf{P}$ of the AABB that is closest to the sphere center $\mathbf{C}$, without explicitly considering the box faces.

The point $\mathbf{P}$ is obtained by projecting the point $\mathbf{C}$ onto the interval defined by the AABB along each axis. In particular, each coordinate of $\mathbf{P}$ is constrained between the corresponding coordinates of the AABB extreme points. Formally, the operation is expressed as:

$$P_i = \max\left(b_{\min,i}, \min(C_i, b_{\max,i})\right), \quad i \in \{x, y, z\} \tag{4.8}$$

where C_i is the i-th component of the sphere center $\mathbf{C}$, while $b_{\min,i}$ and $b_{\max,i}$ are the components of the extreme points $\mathbf{b}_{\min}$ and $\mathbf{b}_{\max}$ along the same direction, respectively. Once the point $\mathbf{P}$ has been determined, the intersection condition between the sphere and the AABB is verified by comparing the Euclidean distance between $\mathbf{C}$ and $\mathbf{P}$ with the sphere radius r:

$$\|\mathbf{C} - \mathbf{P}\|^2 \leq r^2 \tag{4.9}$$

4.1.1.4 Sphere-OBB interaction

In the case of sphere-OBB interaction, the intersection between a sphere $S = (\mathbf{C}, r)$ and an OBB O can be tested by identifying the point $\mathbf{P}$ of the OBB that is closest to the sphere center $\mathbf{C}$, without explicitly analysing the box faces.

To determine the closest point, the difference vector between the sphere center and the OBB center is considered:

$$\mathbf{v} = \mathbf{C} - \mathbf{O}_c \tag{4.10}$$

This vector is projected onto the local axes of the OBB $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$ and constrained by the half-extents e_1, e_2, e_3. The closest point $\mathbf{P}$ on the OBB to the sphere is then expressed as:

$$\mathbf{P} = \mathbf{O}_c + \sum_{i=1}^{3} \min(e_i, \max(-e_i, \mathbf{v} \cdot \mathbf{u}_i))\, \mathbf{u}_i \tag{4.11}$$

Once the point $\mathbf{P}$ has been determined, the intersection condition between the sphere and the OBB is verified by comparing the Euclidean distance between $\mathbf{C}$ and $\mathbf{P}$ with the sphere radius r:

$$\|\mathbf{C} - \mathbf{P}\|^2 \leq r^2 \tag{4.12}$$

4.1.2 Separating axis theorem

The Separating Axis Theorem (SAT) [110] constitutes a discrete formulation, in $\mathbb{R}^2$ and $\mathbb{R}^3$, of the Hyperplane Separation Theorem [111], and forms the basis of many collision detection algorithms for convex sets.

Formally, let $A, B \subset \mathbb{R}^3$ be two convex and compact sets. The two domains are mutually disjoint ($A \cap B = \emptyset$) if and only if there exists at least one direction in space, represented by a non-zero vector $\mathbf{s}$, along which the orthogonal projections of A and B do not overlap. Therefore, if such a direction $\mathbf{s}$ exists, then A and B do not intersect. Otherwise, the two sets overlap.

The SAT is commonly used for intersection tests between polyhedral bounding volumes such as AABB, OBB, and k-DOPs, where the set of separating axes can be reduced to a finite set of candidate directions.

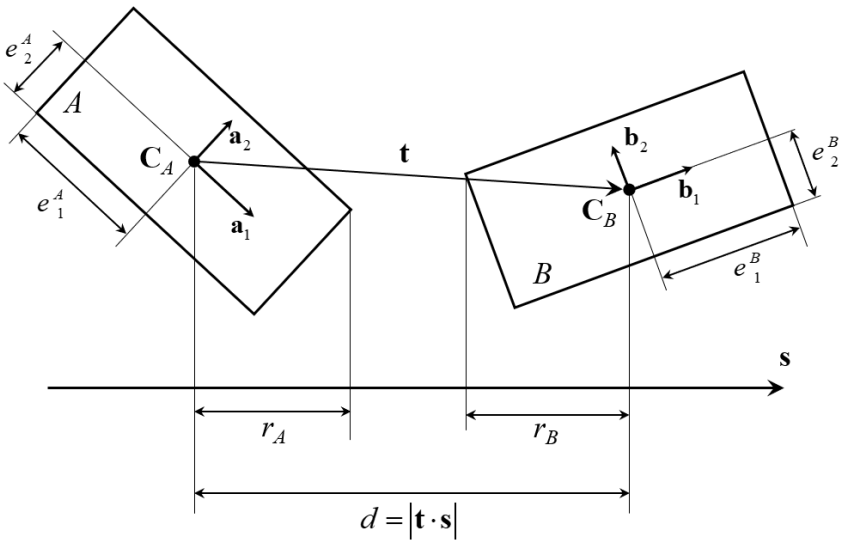


Figure 4.1: Separating Axis Theorem for 2D OBB interaction along a candidate axis $\mathbf{s}$.

In the specific case of OBB–OBB interaction (Figure 4.1), the SAT takes a fully explicit computational form [112]. Let A and B be two OBBs in $\mathbb{R}^3$, each defined by a center, an orthonormal frame, and three half-extents along its local directions. Let $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ denote the local axes of OBB A and $\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\}$ those of OBB B. According to the SAT, a potential separating axis can be found within the following finite set of candidate directions:

- the three axes $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$;
- the three axes $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$;
- the nine vectors $\mathbf{a}_i \times \mathbf{b}_j$, with $i, j \in \{1, 2, 3\}$.

The set of candidate directions therefore consists of 15 distinct axes. For each candidate axis $\mathbf{s}$, the projection of an OBB onto a given direction forms an interval whose position is determined by projecting the OBB center. Let $\mathbf{C}_A$ and $\mathbf{C}_B$ be the centers of OBBs A and B, and define the separation vector:

$$\mathbf{t} = \mathbf{C}_B - \mathbf{C}_A. \tag{4.13}$$

The projections of the centers along $\mathbf{s}$ are:

$$c_A = \mathbf{C}_A \cdot \mathbf{s}, \qquad c_B = \mathbf{C}_B \cdot \mathbf{s}. \tag{4.14}$$

The scalar distance between the projected centers is therefore:

$$d = |c_B - c_A| = |\mathbf{t} \cdot \mathbf{s}|. \tag{4.15}$$

Let a generic point of OBB A be defined as:

$$\mathbf{P}_A = \mathbf{C}_A + \sum_{i=1}^{3} x_i \mathbf{a}_i, \quad x_i \in [-e_i^A, e_i^A], \tag{4.16}$$

where $\mathbf{a}_i$ are the local axes and e_i^A the corresponding half-extents. Its projection along $\mathbf{s}$ is:

$$\mathbf{P}_A \cdot \mathbf{s} = \mathbf{C}_A \cdot \mathbf{s} + \sum_{i=1}^{3} x_i (\mathbf{a}_i \cdot \mathbf{s}). \tag{4.17}$$

Since the parameters x_i vary independently, the maximal deviation from the projected center is:

$$r_A = \sum_{i=1}^{3} e_i^A \, |\mathbf{a}_i \cdot \mathbf{s}|. \tag{4.18}$$

Analogously:

$$r_B = \sum_{i=1}^{3} e_i^B \, |\mathbf{b}_i \cdot \mathbf{s}|. \tag{4.19}$$

Each OBB projected onto $\mathbf{s}$ is thus represented as an interval centered at c_A and c_B with half-widths r_A and r_B, respectively. The separation condition along $\mathbf{s}$ is therefore:

$$|\mathbf{t} \cdot \mathbf{s}| > r_A + r_B. \tag{4.20}$$

It follows that two OBBs are disjoint if and only if there exists at least one candidate axis $\mathbf{s}$ for which this inequality holds.

4.1.3 Support mapping methods

Unlike discrete geometric approaches, which are strictly bound to explicit primitive representations, support-mapping-based methods such as the Gilbert–Johnson–Keerthi (GJK) algorithm and the Expanding Polytope Algorithm (EPA) [113] allow for a unified treatment of general convex geometries. These algorithms do not operate directly on the surfaces of the objects; instead, they rely on an implicit representation of their shape through support functions.

The central idea behind these methods is to reformulate the geometric problem of collision detection between two bodies into an equivalent problem defined in Minkowski space [114]. In this formulation, both intersection testing and minimum distance computation are reduced to the analysis of a single convex set, known as the Minkowski difference.

4.1.3.1 Minkowski difference

Let $A, B \subset \mathbb{R}^3$ be two compact convex sets representing the spatial domains of two interacting bounding volumes. The Minkowski difference, denoted by the operator $\ominus$, is defined as:

$$A \ominus B = \left\{ \mathbf{z} \in \mathbb{R}^3 \;\middle|\; \mathbf{z} = \mathbf{a} - \mathbf{b},\ \mathbf{a} \in A,\ \mathbf{b} \in B \right\}. \tag{4.21}$$

Geometrically, the set $A \ominus B$ can be interpreted as the collection of all vectors connecting a point in B to a point in A. In this sense, it compactly encodes the relative configuration of the two bodies in space.

The relevance of the Minkowski difference in collision detection stems from the fact that the intersection problem between two objects can be transformed into a point-inclusion test of the origin $\mathbf{O}$ in this set. Indeed, a collision occurs if and only if there exists at least one pair of coincident points $\mathbf{a} \in A$ and $\mathbf{b} \in B$, i.e.:

$$\mathbf{a} - \mathbf{b} = \mathbf{0}. \tag{4.22}$$

This leads to the condition:

$$A \cap B \neq \emptyset \iff \mathbf{O} \in A \ominus B. \tag{4.23}$$

From a geometric standpoint, this relationship implies that the position of the origin with respect to $A \ominus B$ fully encodes the interaction state between the two bodies: the objects are disjoint if the origin lies outside the set, they are in a tangential contact state if the origin lies on its boundary, and they are in penetration if the origin lies in its interior.

4.1.3.2 The support function

In general, the explicit construction of the Minkowski difference $A \ominus B$ is computationally expensive. Its complexity grows rapidly with the number of vertices of the involved geometries, since all possible pairwise combinations between vertices of the two sets must be considered. This makes this approach unsuitable for real-time simulations.

To overcome this limitation, the GJK and EPA algorithms do not rely on an explicit representation of the geometry. Instead, they adopt an implicit description based on the so-called support function. This operator characterises a convex set $\mathcal{X}$ through a directional query that returns the point in the set that maximises the projection along a given direction $\mathbf{d}$. The support function is defined as:

$$\mathbf{s}_{\mathcal{X}}(\mathbf{d}) = \arg\max_{\mathbf{x} \in \mathcal{X}} (\mathbf{x} \cdot \mathbf{d}), \tag{4.24}$$

where $\mathbf{x}$ denotes a generic point belonging to the set $\mathcal{X}$.

With this formulation, the operation corresponds to a directional query of the geometry that returns the extreme point along the considered direction. In this way, geometric information is reduced to the component relevant along $\mathbf{d}$, avoiding the need to explicitly describe the entire surface of the object and making the approach suitable for iterative collision-detection procedures.

For this purpose, since $\mathbf{s}_{\mathcal{X}}(\mathbf{d})$ denotes the support function of a convex set $\mathcal{X}$ evaluated along the direction $\mathbf{d}$, and letting A and B be two convex sets in three-dimensional space, the support function of their Minkowski difference $\mathbf{s}_{A \ominus B}(\mathbf{d})$ can be expressed as:

$$\mathbf{s}_{A \ominus B}(\mathbf{d}) = \mathbf{s}_A(\mathbf{d}) - \mathbf{s}_B(-\mathbf{d}), \tag{4.25}$$

The vector $\mathbf{d}$ represents the query direction of the support function, while $-\mathbf{d}$ denotes the opposite direction used for the second set.

4.1.3.3 Gilbert-Johnson-Keerthi algorithm

As previously discussed, instead of explicitly representing the full geometry of $A \ominus B$, the GJK algorithm is based on an iterative process in which, at each iteration, a simplex is constructed[1], which may consist of a point, a line segment, a triangle, or a tetrahedron.

At each iteration, the algorithm considers a search direction $\mathbf{d}_k$ and, through the support function, determines a new point of the Minkowski difference:

$$\mathbf{P}_k = \mathbf{s}_{A \ominus B}(\mathbf{d}_k). \tag{4.26}$$

This point corresponds to the extreme point of $A \ominus B$ along the considered direction and is used to update the current simplex.

The iteration is governed by a fundamental condition, known as the *progress test*, which verifies whether the newly generated point effectively brings the construction closer to the origin. If this condition is not satisfied,

$$\mathbf{P}_k \cdot \mathbf{d}_k \leq 0, \tag{4.27}$$

it is concluded that the origin cannot belong to the Minkowski difference, and therefore that the two convex sets are disjoint. Otherwise, the point is added to the simplex.

At this stage, the simplex is not treated as a simple set of points, but rather as a geometric structure describing the region of space currently relevant to the search for the origin. Depending on its configuration (point, segment, triangle, or tetrahedron), the sub-region closest to the origin is identified, providing the geometric information required to proceed with the iteration.

From this analysis, a new search direction $\mathbf{d}_{k+1}$ is constructed, consistent with the geometry of the current simplex and oriented toward progressively reducing the distance to the origin.

The procedure is repeated iteratively until one of two stopping conditions is met: either it is proven that the two sets are disjoint, or the origin is found to lie inside the current simplex.

To clarify rigorously the construction of the first iterations of the GJK algorithm, assuming that the progress test is satisfied at each step, the sequential generation of simplex points and corresponding search directions is described in geometric terms. An arbitrary non-zero initial direction $\mathbf{d}_0$ is chosen, for instance the vector connecting the centroids of the two bodies. Using the support function 4.25, the first vertex $\mathbf{P}_0$ is computed, which corresponds to the extreme point of the Minkowski difference along the chosen direction:

$$\mathbf{P}_0 = \mathbf{s}_{A \ominus B}(\mathbf{d}_0) \tag{4.28}$$

The initial simplex is then defined as $\mathcal{W}_0 = \{\mathbf{p}_0\}$. To obtain the second simplex point, a new search direction must be determined. The first update direction is chosen

[1] An n-dimensional simplex is the convex hull of $n+1$ affinely independent points in the Euclidean space $\mathbb{R}^n$. In other words, it represents the simplest possible polytope in a given dimension, where the term polytope denotes the convex hull of a finite set of points in $\mathbb{R}^n$. A zero-dimensional simplex is a point, a one-dimensional simplex is a segment, a two-dimensional simplex is a triangle, and a three-dimensional simplex is a tetrahedron.

as the opposite of the position vector of $\mathbf{P}_0$, ensuring that the search is directed toward the origin $\mathbf{O}$:

$$\mathbf{d}_1 = -\mathbf{P}_0. \tag{4.29}$$

Applying the support function 4.25 along $\mathbf{d}_1$ yields the second simplex point:

$$\mathbf{P}_1 = \mathbf{s}_{A \ominus B}(\mathbf{d}_1). \tag{4.30}$$

The simplex becomes the segment $\mathcal{W}_1 = \{\mathbf{P}_0, \mathbf{P}_1\}$. The search direction $\mathbf{d}_2$ is determined by analyzing the orthogonal projection of the origin onto this segment. A generic point $X(t)$ on the segment can be written as:

$$\mathbf{X}(t) = \mathbf{P}_0 + t\,(\mathbf{P}_1 - \mathbf{P}_0). \tag{4.31}$$

The parameter t measures the position along the segment starting from $\mathbf{P}_0$. The value corresponding to the orthogonal projection of the origin is obtained by imposing orthogonality between the residual vector and the segment direction, yielding:

$$t^* = \frac{(\mathbf{P}_1 - \mathbf{P}_0) \cdot (-\mathbf{P}_0)}{\|(\mathbf{P}_1 - \mathbf{P}_0)\|^2}. \tag{4.32}$$

The numerator represents the projection of $\mathbf{P}_0\mathbf{O}$ onto the segment direction, while the denominator normalizes this quantity with respect to the squared length of the segment, yielding a dimensionless parameter t^* that identifies the projection point.

The value of t^* determines the location of the projection:

- if $t^* < 0$, the projection lies beyond $\mathbf{P}_0$, hence $\mathbf{d}_2 = -\mathbf{P}_0$;
- if $t^* > 1$, the projection lies beyond $\mathbf{P}_1$, hence $\mathbf{d}_2 = -\mathbf{P}_1$;
- if $0 \leq t^* \leq 1$, the projection lies inside the segment. In this case, the closest point is $\mathbf{X}^* = \mathbf{P}_0 + t^*(\mathbf{P}_1 - \mathbf{P}_0)$, and the search direction is $\mathbf{d}_2 = -\mathbf{X}^*$.

Equivalently, the same direction can be expressed as:

$$\mathbf{d}_2 = ((\mathbf{P}_1 - \mathbf{P}_0) \times (-\mathbf{P}_0)) \times (\mathbf{P}_1 - \mathbf{P}_0). \tag{4.33}$$

The third simplex point is obtained as:

$$\mathbf{P}_2 = \mathbf{s}_{A \ominus B}(\mathbf{d}_2), \tag{4.34}$$

and the simplex becomes a triangle $\mathcal{W}_2 = \{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2\}$. The corresponding search direction is derived from the triangle geometry. The normal vector is defined as:

$$\mathbf{n} = (\mathbf{P}_1 - \mathbf{P}_0) \times (\mathbf{P}_2 - \mathbf{P}_0). \tag{4.35}$$

The normal is oriented toward the origin:

$$\mathbf{d}_3 = \operatorname{sgn}\left((-\mathbf{P}_0) \cdot \mathbf{n}\right) \mathbf{n}. \tag{4.36}$$

The fourth simplex point is:

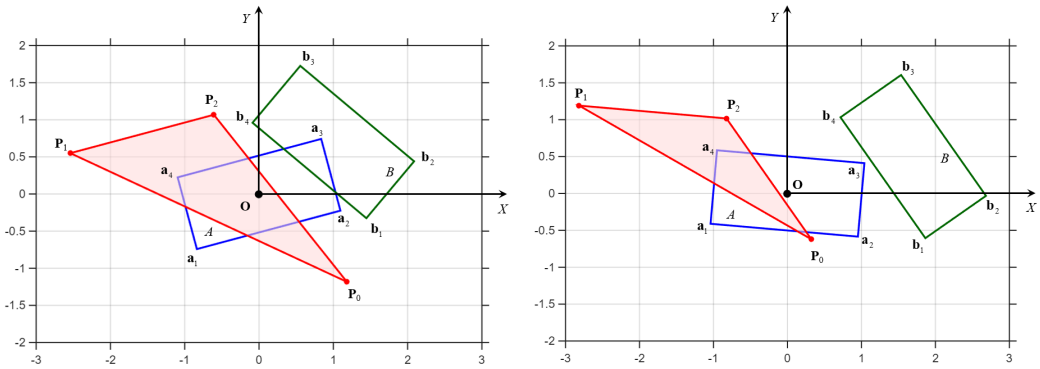
$$\mathbf{P}_3 = \mathbf{s}_{A \ominus B}(\mathbf{d}_3) \tag{4.37}$$

and the simplex becomes a tetrahedron $\mathcal{W}_3 = \{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3\}$. At this stage, the inclusion of the origin is tested by examining each triangular face of the tetrahedron. For each face i, an outward normal $\mathbf{n}_i$ is defined, and the sign of the scalar products is evaluated:

$$(\mathbf{O} - \mathbf{P}_i) \cdot \mathbf{n}_i, \quad i \in \{1, 2, 3, 4\} \tag{4.38}$$

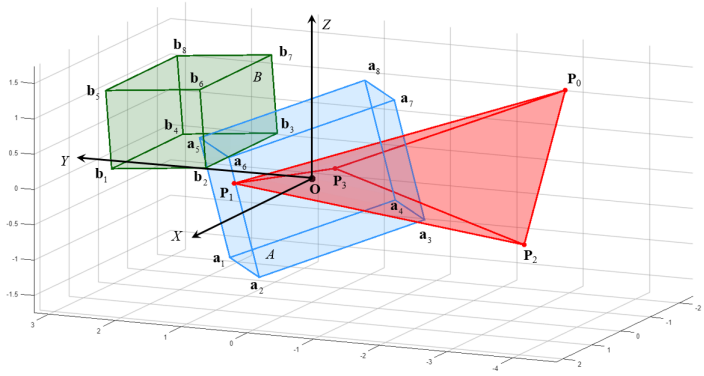
The sign of this quantity determines the position of the origin relative to the half-space defined by each face. The origin lies inside the tetrahedron if and only if it is contained simultaneously in all such half-spaces.

The Figures 4.2 show some examples of the application of the GJK algorithm in the case of OBB–OBB interaction. In particular, Figures 4.2a and 4.2b illustrate the two-dimensional case, in which the algorithm is able to robustly discriminate between colliding and non-colliding configurations, while Figure 4.2c shows the extension of the method to the three-dimensional case.



(a) Collision detected in the 2D case.

(b) No collision detected in the 2D case.



(c) Collision detected in the 3D case.

Figure 4.2: Application of the GJK algorithm for OBB-OBB interaction in both 2D and 3D cases.

4.1.3.4 Expanding polytope algorithm

When the GJK algorithm terminates with a positive result, it indicates that the two bodies are intersecting. However, the final simplex does not yet provide quantitative

information such as penetration depth or contact direction. To extract these quantities, the EPA is employed. The main idea behind EPA is to start from the simplex returned by GJK, which can be interpreted as a first approximation of the intersection region within the Minkowski difference $A \ominus B$, and to iteratively refine this approximation. More precisely, EPA treats the initial simplex as a small convex polytope and expands it outward. At each iteration, the polytope is updated so as to progressively approach the true boundary $\partial(A \ominus B)$ in the direction of the origin, until a sufficiently accurate approximation is achieved.

The initial polytope is constructed from the final simplex returned by GJK. In the case of intersection, this simplex is a convex tetrahedron denoted by $\mathcal{W} = \{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3\}$. The triangular faces of the polytope are interpreted as oriented geometric elements f_i, each associated with an outward normal vector $\mathbf{n}_i$. At each iteration, the face $f_{\min}$ is selected, defined as the face of the current polytope that has the minimum distance from the origin $\mathbf{O}$. This distance is evaluated along the face normal and identifies which region of the polytope provides the best local approximation of the Minkowski difference boundary.

Formally, the distance of a face f_i from the origin is defined as

$$\mathrm{dist}(f_i, \mathbf{0}) = \mathbf{P}_i \cdot \mathbf{n}_i, \tag{4.39}$$

where $\mathbf{P}_i$ is any of the three vertices of the face f_i, and $\mathbf{n}_i$ is its outward unit normal vector. The normal associated with the selected face, denoted by $\mathbf{n}_f$, defines the direction along which the support function is queried. In particular, the new support point $\mathbf{P}'_0$ of the Minkowski difference in direction $\mathbf{n}_f$ is computed as

$$\mathbf{P}'_0 = \mathbf{s}_{A \ominus B}(\mathbf{n}_f). \tag{4.40}$$

The termination criterion compares the new support point with the current face approximation, checking whether the support mapping provides a significant expansion of the polytope in the considered direction:

$$(\mathbf{P}'_0 \cdot \mathbf{n}_f) - \mathrm{dist}(f_{\min}, \mathbf{0}) < \epsilon, \tag{4.41}$$

where $\mathrm{dist}(f_{\min}, \mathbf{0})$ denotes the distance of the selected face from the origin, and $\epsilon > 0$ is a numerical tolerance accounting for floating-point errors and finite precision.

If the convergence condition is satisfied, the algorithm terminates and returns the contact kinematic quantities. In particular, the penetration depth d_p is obtained as the projection of the support point $\mathbf{P}'_0$ onto the face normal $\mathbf{n}_f$:

$$d_p = \mathbf{P}'_0 \cdot \mathbf{n}_f. \tag{4.42}$$

Otherwise, if the condition is not satisfied, the point $\mathbf{P}'_0$ is inserted into the current polytope $\mathcal{W} = \{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \mathbf{P}'_0\}$. Next, all faces of the polytope that are visible from $\mathbf{P}'_0$ are identified. This visibility condition is expressed as

$$(\mathbf{P}'_0 - \mathbf{P}_i) \cdot \mathbf{n}_i > 0. \tag{4.43}$$

Faces satisfying this condition are removed, since they no longer belong to the convex hull consistent with the new point. The set of edges shared between removed and non-removed faces forms the open boundary of the polytope. The point $\mathbf{P}'_0$ is then connected to all edges of this boundary, generating new triangular faces and producing the updated polytope. The procedure is repeated iteratively until condition (4.41) is satisfied.

4.2 Interaction response

Once a potential interaction between two bodies has been identified through collision detection, the system must compute the corresponding dynamic response. This step is responsible for enforcing physically admissible motion, preventing interpenetration and generating the effects associated with contact, such as reaction impulses and velocity changes.

Unlike collision detection, which is based on geometric queries, interaction response is formulated within the framework of multibody dynamics. In this setting, contact is not treated as a geometric event, but as a set of constraints acting on the relative motion of the bodies, which must be satisfied during the time evolution of the system.

4.2.1 Multibody contact formulation

In multibody dynamics, contact between rigid bodies is typically modeled through unilateral constraints acting on the relative motion at the contact point. Let $\mathbf{n}$ denote the contact normal and let $\mathbf{v}_r$ be the relative velocity between the two bodies at the contact location [115]. The component of this velocity along the normal direction is given by:

$$v_n = \mathbf{n} \cdot \mathbf{v}_r. \tag{4.44}$$

The non-penetration condition requires that, once contact is established, the bodies cannot move towards each other along the normal direction:

$$v_n \geq 0. \tag{4.45}$$

To enforce this constraint, a normal contact impulse (or force in a continuous formulation) λ_n is introduced, with the additional condition:

$$\lambda_n \geq 0. \tag{4.46}$$

The interaction between constraint and reaction is expressed through the complementarity condition:

$$\lambda_n \, v_n = 0. \tag{4.47}$$

This relation enforces that contact is active only when the bodies are in a non-separating state. In particular, either $v_n > 0$ and the bodies move apart with no contact reaction ($\lambda_n = 0$), or $\lambda_n > 0$ and the contact impulse prevents penetration by enforcing $v_n = 0$.

In numerical multibody solvers, these constraints are typically enforced at the velocity or impulse level, which leads to complementarity formulations of LCP or NCP type [116].

Friction is introduced by noting that contact interactions act not only along the normal direction, but also within the tangent plane at the contact interface. For this reason, the relative velocity is decomposed into normal and tangential components:

$$\mathbf{v}_t = \mathbf{v}_r - v_n \mathbf{n}. \tag{4.48}$$

The vector $\mathbf{v}_t$ represents the relative sliding velocity along the contact surface, i.e., the motion that friction tends to oppose.

According to Coulomb's friction model, the tangential contact impulse $\boldsymbol{\lambda}_t$ is constrained in magnitude by the normal impulse:

$$\|\boldsymbol{\lambda}_t\| \leq \mu\lambda_n, \tag{4.49}$$

where μ is the friction coefficient.

Although conceptually straightforward, the numerical solution of these constraints becomes computationally demanding when multiple contacts are present. The main reason is that each contact introduces additional unknowns (normal and tangential impulses) that are coupled through the global dynamics of the system. As a consequence, the equations of motion and all contact constraints must be solved simultaneously, resulting in a large coupled system.

In dense contact scenarios, which are typical of real-time simulations and XR applications, the set of active constraints also changes over time. The solver must therefore repeatedly assemble and resolve a high-dimensional constrained problem at every time step, which makes a direct solution of the full system unsuitable for interactive performance.

4.2.2 Real-time XR engines

The contact formulation presented in the previous section provides a rigorous framework for describing interactions in multibody systems. However, its direct solution is often impractical in XR applications, where simulations must be executed at interactive rates.

For a rigid body, the unconstrained dynamics are governed by the Newton-Euler equations [117]:

$$m\dot{\mathbf{v}} = \mathbf{f}, \tag{4.50a}$$

$$\mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\mathbf{I}\boldsymbol{\omega}) = \boldsymbol{\tau}, \tag{4.50b}$$

where m is the body mass, $\mathbf{v}$ the linear velocity, $\mathbf{I}$ the inertia tensor, $\mathbf{f}$ the resultant external force, and $\boldsymbol{\tau}$ the resultant external torque.

When contacts occur, additional reactions must be introduced to satisfy the non-penetration and friction constraints discussed previously. Since each contact affects the motion of the involved bodies, the corresponding constraints are generally coupled and must be solved together with the equations of motion. As the number of active contacts increases, the resulting problem rapidly becomes computationally expensive.

For this reason, real-time physics engines usually avoid solving the full contact formulation. Instead, interaction response is expressed in terms of impulses $\mathbf{p}$, defined as:

$$\mathbf{p} = \int_t^{t+\Delta t} \mathbf{f}, dt. \tag{4.51}$$

An impulse produces an instantaneous variation of momentum and, consequently, of velocity. Contact constraints can therefore be enforced directly at the velocity level, avoiding the explicit computation of contact forces during the simulation step [118].

Most physics engines adopted in XR applications rely on iterative impulse-based solvers, where contacts are resolved through successive local velocity corrections. Among

these approaches, the Sequential Impulse method is one of the most widely used and will be discussed in detail in Chapter 7.

4.3 Interaction methods

While object-to-object interaction is completely defined by the mechanisms introduced in the previous sections, user-to-object interaction requires a means of transmitting the user's intentions to the system. For this reason, input devices must be introduced, through which signals generated by the user are translated into actions applied to virtual objects. Among the most widely used technologies are dedicated controllers, hand tracking systems with gesture recognition, and haptic devices.

4.3.1 Controller-based interaction

(a) HTC VIVE Pro 1 and 2 controllers.

(b) HTC Vive XR Elite controllers.

(c) Varjo XR-4 controllers.

(d) Meta Quest 3 controllers.

(e) Meta Quest Pro controllers.

Figure 4.3: Controllers of the devices in Figures 2.5 and 2.6.

Controllers (Figure 4.3) still represent one of the primary interaction paradigms in extended reality environments, owing to their tracking stability, high sampling frequency, and relative robustness against occlusion phenomena typically affecting computer vision systems. From a data acquisition perspective, these devices produce a heterogeneous set of raw signals, including the kinematic pose of the device, expressed through position and orientation, as well as discrete and continuous signals such as button states and analog actuators.In modern software architectures, these signals are not used directly by the application logic. Instead, they are reinterpreted through an intermediate abstraction layer, commonly referred to as the *input mapping layer*. Within this layer, the information flow is translated from physical device quantities into system commands, such as *select*, *grab*, *teleport*, and so forth.

This process can be schematized as the transformation:

$$\mathcal{F} : \mathcal{S}_{dev} \rightarrow \mathcal{S}_{sim}, \tag{4.52}$$

where $\mathcal{S}_{dev}$ represents the observation space of the device and $\mathcal{S}_{sim}$ the space of actions interpreted by the simulation system. The instantaneous configuration of the controller is described by the pose:

$$\mathbf{T}_c(t) = \begin{bmatrix} \mathbf{R}_c(t) & \mathbf{x}_c(t) \\ 0 & 1 \end{bmatrix}, \tag{4.53}$$

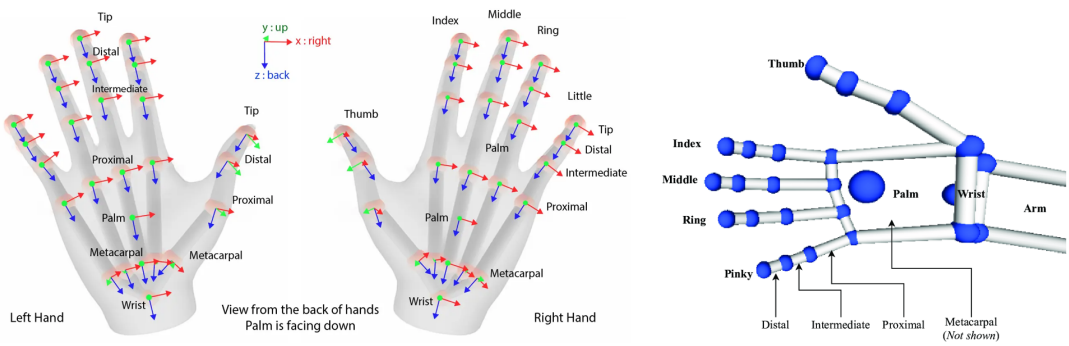
where $\mathbf{x}_c(t)$ denotes the position and $\mathbf{R}_c(t)$ represents the orientation of the device in the global reference frame.

From the perspective of physical simulation, the controller pose is not directly transferred to virtual objects, since this would impose rigid kinematic constraints that are incompatible with the dynamic evolution of the system. Interaction is therefore mediated by the physics engine, which translates it into temporary constraints or equivalent coupling models between degrees of freedom.

In practice, the interaction pipeline is primarily based on the detection of a contact condition through bounding volumes. When the controller enters the proximity of, or collides with, an object, the system generates an interaction event. At this point, event-driven logic comes into play: if the active state of the grab command is simultaneously detected (for example, through the press of the associated button), the system initiates the grasping procedure.

The object is then constrained to the controller through a temporary constraint, which can be interpreted either as a kinematic coupling or as a constraint resolved by the physics solver through impulses or equivalent forces. In this way, the motion of the controller does not directly impose the configuration of the rigid body, but rather influences its evolution while respecting the governing dynamic equations and contact constraints.

4.3.2 Hand tracking



(a) OpenXR hand tracking joints.

(b) Ultraleap hand tracking joints.

Figure 4.4: Hand tracking joints in different framework.

Hand tracking systems enable the user to interact with virtual environments without the need for dedicated controllers, reconstructing in real time the configuration of the hands from observations provided by the headset sensors. Depending on the platform, such observations may be obtained through RGB cameras, infrared sensors, or depth cameras.

The goal of the process is to determine position, orientation, and velocity for several points on a user's hand. Unlike controller-based interaction, where the device is represented by a single pose in space, hand tracking provides an articulated representation of the hand, capable of describing the independent motion of the fingers. The implementation of natural interfaces based on hand tracking and advanced gesture recognition techniques improves both the intuitiveness and accuracy of the system [119] (Figure 4.5).

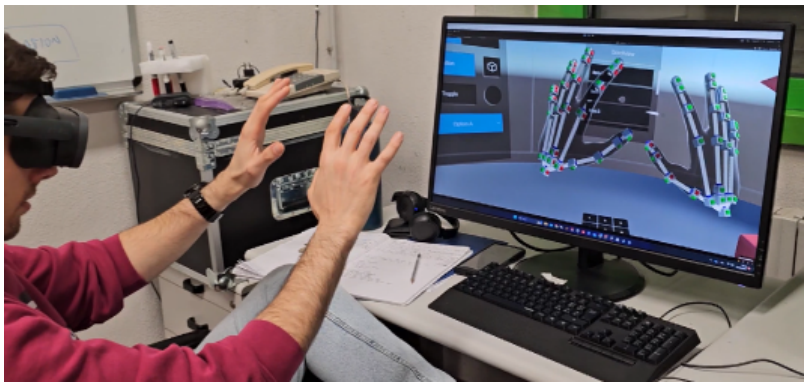


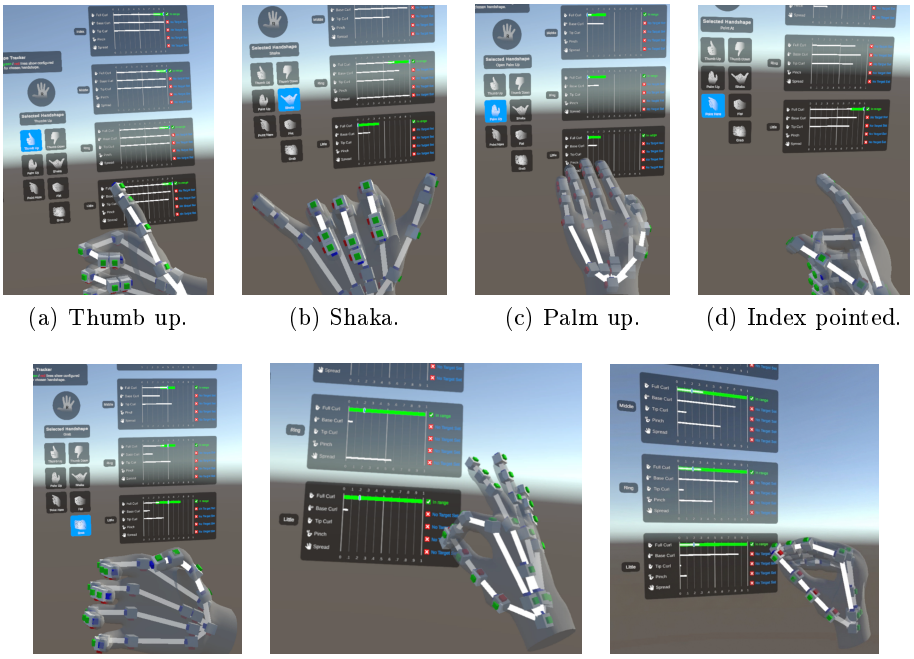
Figure 4.5: Hand tracking application example.

From a computational standpoint, the hand is typically represented using an articulated digital model (*rig*). This skeleton is composed of a set of characteristic points, called *joints*, and the connections between them, called *bones*. Joints represent the articulation points used to describe the configuration of the hand, while bones define the segments connecting two consecutive joints and describe the kinematic structure. This representation allows for an approximation of the anatomical behavior of the hand and enables tracking of its motion over time. The arrangement and number of joints depend on the tracking system model (Figure 4.4). In general, the skeleton includes the wrist, the palm, and the main finger joints. For each finger, the metacarpal, proximal, intermediate, and distal phalanges are typically represented, as well as the fingertip used to describe the extremity.

The state of a tracked hand can be represented as the set of poses associated with its joints:

$$\mathcal{H}(t) = \{T_1(t), T_2(t), \ldots, T_N(t)\}\,, \tag{4.54}$$

where $T_i(t)$ denotes the pose of the i-th joint and N is the total number of joints in the model. From an operational perspective, the hand tracking process can be divided into three main stages. In the first stage, images from the sensors are acquired and pre-processed. Subsequently, computer vision and deep learning algorithms detect the presence of hands and estimate the configuration of the corresponding joints. Finally, the joint poses are used to update the virtual skeleton, reconstructing the configuration of the open kinematic chain within the virtual environment.



(a) Thumb up. (b) Shaka. (c) Palm up. (d) Index pointed.

(e) Grab. (f) Pinch. (g) Custom pinch.

Figure 4.6: Customized hand gestures.

User–object interactions and, more generally, user intentions are triggered through *gestures*, that is predefined configurations of the hands (Figure 4.6). The use of hands as an input device introduces a more natural interaction paradigm [120].

The definition of a gesture is typically based on two main components. The first concerns the *hand shape*, i.e., the configuration of the fingers, while the second concerns the *hand pose*, which describes the orientation of the hand.

Regarding hand shape, instead of directly using all finger joint poses, reference configurations are introduced and finger postures are defined through normalized coefficients in the range $[0, 1]$. The reference configurations include:

- Full curl: the overall curve of a finger.
- Base curl: the angle between the hand and the base of the finger.
- Tip curl: the curve of the outer portions of the finger.
- Pinch: whether the finger is in a pinching posture based on the proximity between the fingertip and the thumb tip.
- Spread: the spread between two adjacent fingers.

For example, for the *full curl* parameter, a value of 0 corresponds to a fully extended finger, while a value of 1 corresponds to a fully flexed finger (Table 4.1). Hand pose, on the other hand, complements hand shape by adding a specific spatial orientation, thus defining gesture subclasses within the broader set of hand shapes. For instance, to define

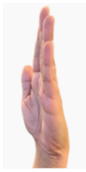
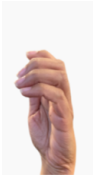
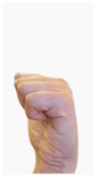
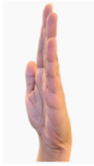
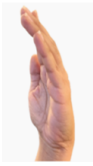
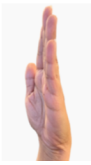
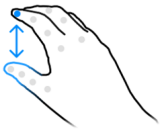
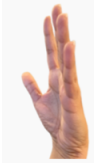
Type	Joints	Value = 0	Value = 0.5	Value = 1
Full Curl (All Fingers)				
Base Curl (All Fingers)				
Tip Curl (All Fingers)				
Pinch				
Spread				

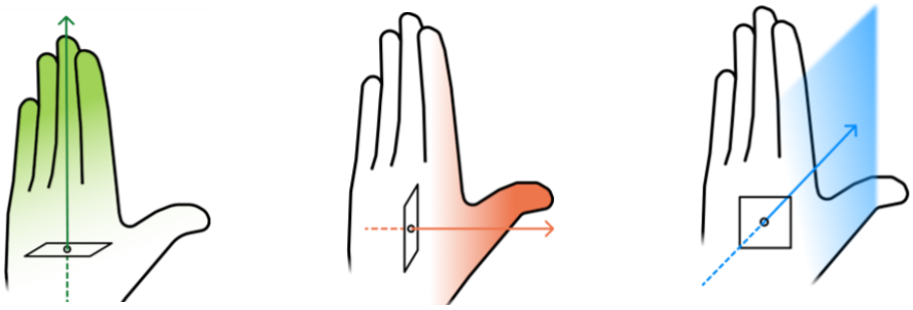
Table 4.1: Reference configurations of hand shape and corresponding normalized values.

a “thumbs up” gesture, a hand shape with an extended thumb and folded remaining fingers can be used. Ensuring that the thumb is oriented upward requires associating the corresponding orientation constraints to that hand shape.

A hand pose is defined by three main elements:

- Hand axis: the reference axis of the hand.
- Reference direction: a direction relative to the user, another object, or the world origin.
- Alignment: how the two directions must be related.

For specifying orientation, the hand axes are defined with respect to the extended fingers, the extended thumb, and the palm (Figure 4.7). The first hand axis points along the direction of the fully extended fingers (proximal–distal direction, Figure 4.7a). The second hand axis points along the extended thumb direction (radial–ulnar direction,



(a) Fingers Extended Direction.

(b) Thumb Extended Direction.

(c) Palm Direction.

Figure 4.7: Hand axis.

Figure 4.7b). The third hand axis corresponds to the direction in which the palm is facing (palmar–dorsal direction, Figure 4.7c).

The reference direction defines the vector against which the selected hand axis is compared in order to determine whether the gesture satisfies the required orientation constraints. Three user-relative reference directions are considered (Table 4.2): Origin, Hand to Head, and Hand to Target. The *Origin* corresponds to the positive axis of the global reference frame. The Hand-to-Head direction is defined as the vector pointing from the hand toward the user's head, while the Hand-to-Target direction points from the hand toward the interaction target. In the *Hand to Head* case, additional sub-variants are introduced to disambiguate orientation, including the nose direction (forward head direction), the chin direction (downward direction), and the ear direction (lateral direction depending on the active hand). Similarly, for *Hand to Target*, the reference direction is expressed in the target local coordinate system and aligned with its principal axes, namely x, y, and z, corresponding to right, up, and forward directions.

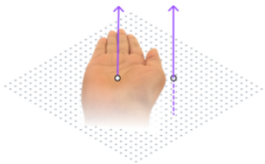
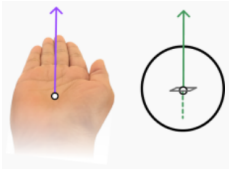
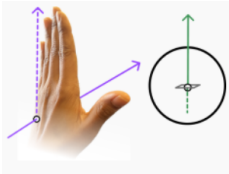
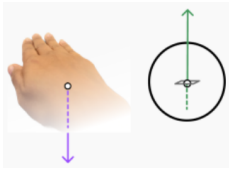
	Align with	Perpendicular To	Opposite To
Origin Up			
Target Up			

Table 4.3: Alignment configurations.

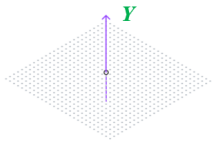
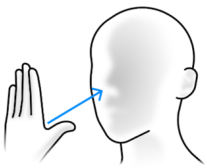
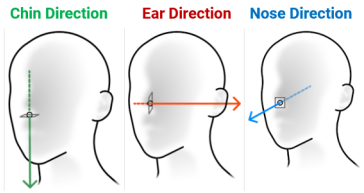
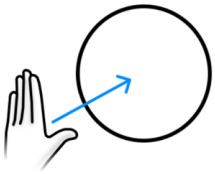
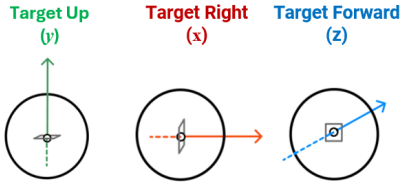
Origin Up		-
Hand to Head		
Hand to Target		

Table 4.2: Reference directions.

Regarding alignment, three configurations are typically defined. In the *Aligns with* case, the hand axis and reference direction must be aligned. In the *Perpendicular to* case, they must be orthogonal. In the *Opposite to* case, they must be collinear but with opposite orientation. An angular tolerance in the range 0.1° to 180° is defined, and a gesture is recognized only if the angular distance between the hand axis and the reference direction satisfies the selected constraint (Table 4.3, showing the palm direction case).

Once the hand configuration is identified, the system proceeds to the mapping phase toward virtual objects. Object selection follows a logic similar to controller-based interaction, based on bounding volumes. In this case, the hand (or a significant portion of its

articulated skeleton) is used as a geometric proxy for collision detection with scene objects. When a contact condition is detected and the associated grab gesture is active, the system generates an interaction event enabling object selection. The selected object is then coupled to the hand kinematic structure through a temporary constraint, enforcing motion following while preserving the physical simulation.

4.3.2.1 Grasping active features/object active features methodology

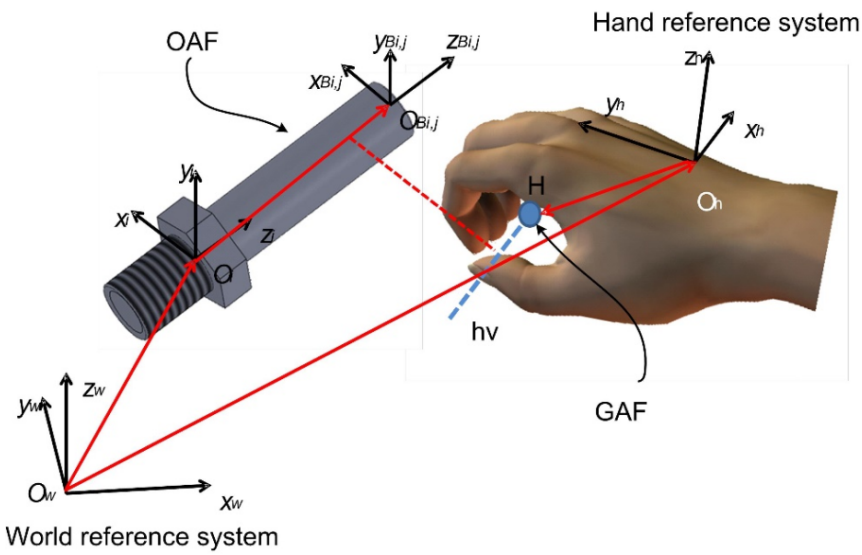


Figure 4.8: An example of the OAF/GAF methodology for the cylindrical grip [121].

A widely used approach for modelling grasping operations consists in introducing geometric entities capable of representing the interaction between the hand and virtual objects [122]. These entities, called *Grasping Active Features* (GAF), describe geometric features associated with the hand and are related to analogous entities defined on virtual objects, called *Object Active Features* (OAF) [123]. According to the GAF/OAF methodology, each grasp configuration is associated with two reference frames: one rigidly attached to the hand (GAF) and one attached to the object (OAF), as shown in Figure 4.8. Their interaction is described through constraint equations defined between the corresponding reference frames (Appendix D). Each equation removes a specific relative degree of freedom, enabling different types of coupling. As in multibody system topology modeling, such constraints can be expressed as algebraic relations between geometric entities defining the active feature frames [124]. The operational procedure is structured as follows:

- when the user performs a hand configuration corresponding to a grasp, the associated GAF is activated;
- the relative position and orientation between the active GAF and the OAFs of scene objects are evaluated;

- if proximity and alignment criteria are satisfied within predefined tolerances, the grasp is confirmed and the corresponding constraint between the two reference frames is imposed;
- otherwise, the grasp is rejected and numerical integration proceeds without modifying the system equations.

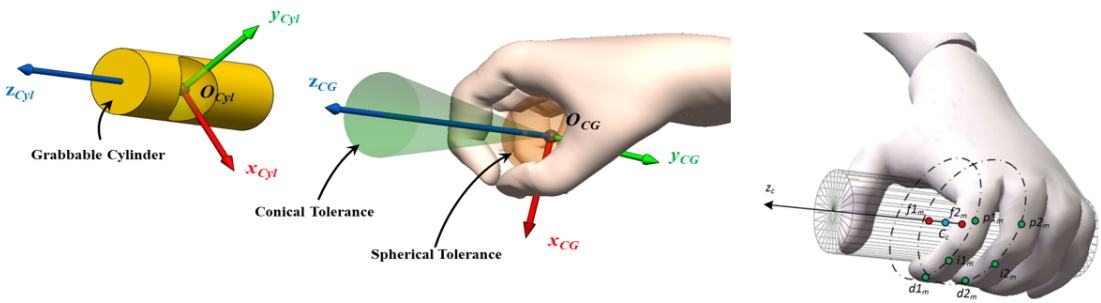


Figure 4.9: GAF/OAF methodology adapted for cylindrical grasp [42, 90].

For the cylindrical grasp, the reference frame attached to the hand includes the origin $\mathbf{O}_{CG}$ and the axes $\mathbf{x}_{CG}$, $\mathbf{y}_{CG}$, and $\mathbf{z}_{CG}$, where $\mathbf{z}_{CG}$ defines the grasp axis. The object reference frame includes the origin $\mathbf{O}_{OAF}$ and the axes $\mathbf{x}_{OAF}$, $\mathbf{y}_{OAF}$, and $\mathbf{z}_{OAF}$, where $\mathbf{z}_{OAF}$ corresponds to the cylinder axis. As shown in Figure 4.9, the cylindrical grasp constraint can be imposed if both the origin coincidence condition (Equation 4.55a) and the axis alignment condition (Equation 4.55b) are satisfied within a tolerance ε.

$$\|\mathbf{O}_{CG} - \mathbf{O}_{OAF}\| \leq \varepsilon \tag{4.55a}$$

$$1 - \|\mathbf{z}_{CG} \cdot \mathbf{z}_{OAF}\| \leq \varepsilon \tag{4.55b}$$

Equation 4.55a describes a spherical tolerance region, while Equation 4.55b defines a conical angular tolerance. These constraints differ from the original GAF/OAF formulation since they explicitly encode angular tolerance between cylindrical axes, enabling more efficient real-time evaluation. An alternative formulation of the cylindrical grasp is given by:

$$\mathbf{z}_{CG} \cdot \mathbf{x}_{OAF} \leq \varepsilon \tag{4.56a}$$

$$\mathbf{z}_{CG} \cdot \mathbf{y}_{OAF} \leq \varepsilon \tag{4.56b}$$

$$\|O_{CG} - \mathbf{O}_{OAF}\| \leq \varepsilon \tag{4.56c}$$

For the pinch grasp, the fundamental elements of the two reference frames on the hand and on the virtual object are the origins $\mathbf{O}_{PG}$ and $\mathbf{O}_{OAF}$, respectively. Figure 4.10 shows the spherical tolerance described by Equation 4.57, within which the constraint can be activated.

$$\|\mathbf{O}_{PG} - \mathbf{O}_{OAF}\| \leq \varepsilon \tag{4.57}$$

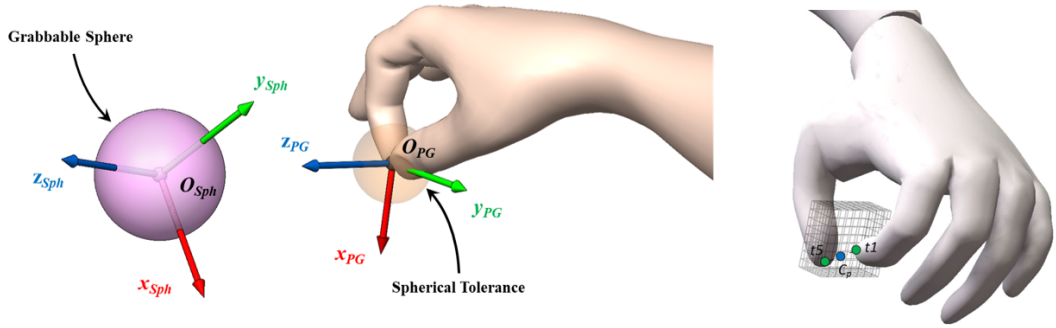


Figure 4.10: GAF/OAF methodology adapted for pinch grasp [42, 90].

In case of constraint and orientation lock, the following equations are added:

$$\mathbf{x}_{OAF} \cdot (\mathbf{t}_1 - \mathbf{t}_5) \leq \varepsilon \tag{4.58a}$$

$$\mathbf{y}_{OAF} \cdot (\mathbf{t}_1 - \mathbf{t}_5) \leq \varepsilon \tag{4.58b}$$

where $\mathbf{t}_1$ and $\mathbf{t}_5$ are the fingertips of index and thumb, respectively, and $\mathbf{x}_{OAF}$ and $\mathbf{y}_{OAF}$ are the x and y axes of the object frame.

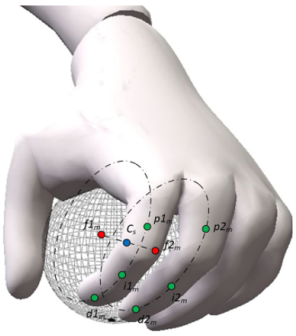


Figure 4.11: GAF/OAF methodology adapted for spherical grasp [42].

For the spherical grip (Figure 4.11), the fundamental elements of the two reference frames on the hand and on the virtual object are the origins $\mathbf{O}_{SG}$ and $\mathbf{O}_{OAF}$, respectively. The spherical tolerance is defined by Equation 4.59.

$$\|\mathbf{O}_{SG} - \mathbf{O}_{OAF}\| \leq \varepsilon \tag{4.59}$$

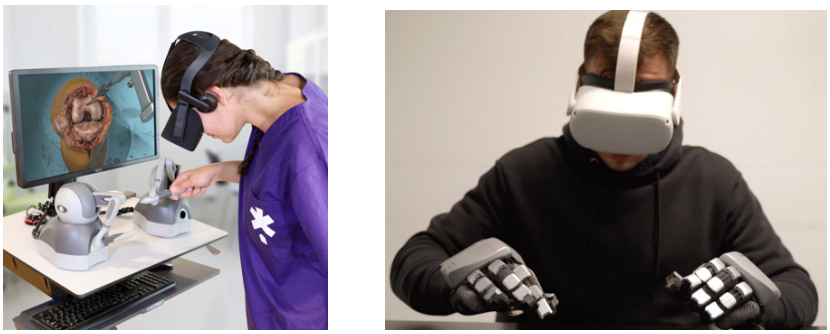
4.3.3 Haptic devices

Haptic devices constitute a class of human–machine interfaces designed to enable interaction through the sense of touch. The term *haptic* derives from the Greek *haptikos*, meaning "related to touch", and refers to the set of technologies developed to convey sensory information through controlled mechanical stimuli. The main objective of such systems is to enrich the user experience and increase the level of immersion during interaction with virtual environments, numerical simulations, or robotic systems.

Unlike traditional input and output interfaces, which rely predominantly on the visual and auditory channels, haptic devices introduce an additional layer of sensory communication based on tactile and kinesthetic perception. Feedback is generally provided through the controlled application of forces, vibrations, or motions that act directly on the user. These stimuli involve both the tactile system, through the activation of mechanoreceptors located on the skin surface, and the kinesthetic or proprioceptive system, through the stimulation of receptors located in muscles, tendons, and joints.

A fundamental characteristic of haptic systems is their intrinsically bidirectional nature. In addition to receiving information from the virtual environment, the user continuously provides kinematic input to the system through their movements. This therefore establishes a closed interaction cycle (*two-way interaction loop*) in which the operator's behavior influences the evolution of the simulation and, simultaneously, the model's response is returned in the form of perceivable physical sensations.

The use of haptic devices has gained increasing relevance in recent years across a wide range of application domains, including professional training and education, ergonomic simulation, robotics, as well as virtual and augmented reality systems. In general, existing applications can be classified into three main categories. The first category concerns the interaction with virtual tools kinematically driven by the haptic device, such as mechanical instruments, chisels, knives, drills, or surgical tools operating within rigid or deformable environments. In these scenarios, the user perceives contact forces generated by the interaction with the virtual environment, including viscous effects, static and dynamic friction, and surface adhesion phenomena. The second category includes the direct manipulation of virtual objects, in which the user grasps and moves the object through the haptic interface. The rendered force feedback reflects the physical properties of the manipulated object, such as weight, inertia, and dynamic reaction forces, enabling a realistic perception of its physical behavior. Finally, a third class of applications exploits the generation of synthetic or virtual forces that do not correspond to any direct physical counterpart. These forces are typically used for assistive or corrective purposes: through localized vibrations, artificial force fields, or virtual constraints (e.g., virtual walls), the system guides the user during task execution, reducing errors and improving training efficiency.



(a) Grounded haptic devices (b) Internal-force haptic devices

Figure 4.12: Types of haptic devices.

Haptic devices can be divided into two main categories based on the way forces are generated and transmitted to the user: grounded haptic devices and internal-force haptic devices.

Grounded devices are characterized by the presence of a mechanical structure rigidly connected to an external base, which allows the forces generated by the actuators to be grounded. This configuration enables the effective rendering of resistive forces, weight-like effects, contact reactions, and mechanical interactions with virtual tools. From a structural standpoint, these systems are typically implemented using articulated kinematic chains and multi-linkage arrangements, which ensure high precision and stability of the feedback (Figure 4.12a).

Internal-force devices, on the other hand, rely on a different principle. In this case, forces are applied directly to the user's fingers, hands, or other body parts without being grounded to an external structure. The main objective is to reproduce the shape, texture, and geometric properties of virtual objects, without fully simulating their global weight. This category mainly includes haptic gloves and haptic suits, in which small mechanical or vibrotactile actuators generate localized sensations associated with contact with the virtual environment (Figure 4.12b).

The two categories are therefore complementary. Grounded devices generally offer higher force output capability and greater accuracy in simulating mechanical interactions, whereas wearable devices prioritize freedom of movement and immersion in complex virtual environments.

4.3.3.1 Geomagic Touch X

(a) (b)

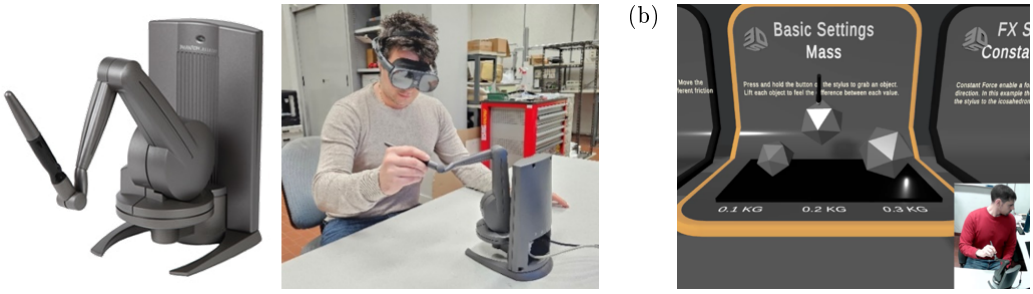


Figure 4.13: ((a)) Geomagic Touch X, ((b)) example of an application.

Among the most widely used grounded haptic devices in industrial and scientific contexts is the Geomagic Touch X (Figure 4.13a). The system consists of an articulated mechanical structure that allows the user to manipulate a stylus with six degrees of freedom of motion, while simultaneously providing active force feedback along three spatial directions.

The device features an acquisition resolution of 0.023 mm and a maximum output force of 7.9 N, making it particularly suitable for integration with interactive physics simulations and high-fidelity virtual environments. Communication with external software applications is typically achieved through the OpenHaptics SDK, which provides both low-level and high-level functionalities for device control and haptic rendering.

Thanks to its combination of accuracy, stability, and ease of integration, the Geomagic Touch X represents one of the most widely used devices for applications in dynamic simulation, operational training, and research in the field of interactive virtual reality (Figure 4.13a).

4.3.3.2 SenseGlove Nova 2

(a) (b)

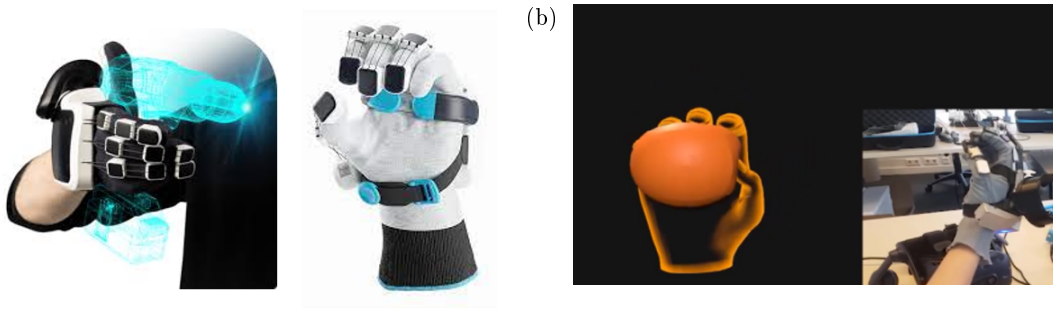


Figure 4.14: ((a)) SenseGlove Nova 2, ((b)) example of an application.

A different approach is represented by the SenseGlove Nova 2, which belongs to the category of wearable internal-force haptic devices (Figure 4.14a). The system is designed as a haptic glove and enables the simulation of contact with virtual objects through localized force feedback applied to the fingers, as well as distributed vibrotactile feedback (Figure 4.14b).

Unlike grounded devices, the SenseGlove Nova 2 is not intended to accurately simulate the global weight of objects, but rather to reproduce their shape, stiffness, and grasping characteristics. This feature is particularly advantageous in immersive virtual reality applications, where user freedom of movement is a fundamental requirement.

Integration with virtual environments is typically achieved through the SenseGlove Unity Plugin, which enables synchronization between virtual hand tracking and the haptic feedback system. This solution is commonly used in training scenarios, virtual assembly, ergonomics studies, and evaluation of human–product interaction.

Chapter 5

Interactive modelling

In recent years, with the introduction of Industry 4.0 and the emergence of the fifth industrial revolution, which aims to create a more human-centered working environment [125] and to promote collaboration between humans and machines [33, 126], technologies such as VR and AR have been playing an increasingly strategic role. In this context, VR and AR represent key tools for enhancing human–machine interaction [127, 128], enabling immersive experiences that support advanced training [129], assisted maintenance [130, 131], real-time problem solving [132], and interactive modelling [133].

A particularly promising application domain for VR and AR is their integration with Computer-Aided Design, where immersive technologies enhance the way designers and engineers create, interact with, and refine complex 3D models. The interest in integrating 3D modelling tools, particularly with AR [134], stems from the numerous advantages they offer [135], especially when multimodal interfaces are available for interacting with the software [136]. Direct interaction with the real world provides users with immediate spatial awareness of the size and scale of the model being created. Moreover, enabling simultaneous access to the same 3D model for multiple users fosters collaboration among designers, engineers, and other stakeholders, allowing them to visualize and manipulate the design in real time, even from remote locations. Design meetings and reviews thus become more efficient, as all participants can share a common understanding of the model and propose modifications instantly. The ability to link physical objects to the virtual model allows designers to observe how products or structures integrate into the physical context. This helps to identify spatial, proportional, or environmental interaction issues more intuitively than traditional screen-based visualization, thereby reducing the likelihood of design errors.

The current state of the art includes several implementations of 3D modelling in VR/AR environments, which differ in the underlying solid modelling mathematics, modelling paradigms, and interaction frameworks. The use of a tilted high-resolution interactive display with multi-touch and pen input has enabled immersive 3D modelling, allowing users to modify primitive geometries through standard operations or to create objects via extrusion or revolution from 2D sketches [137] (Figure 5.1).

Thanks to significant advances in consumer AR and VR hardware and software, it is now possible to draw 3D curves directly in space. Several tools have been developed for surface generation based on 3D sketching, using controllers [138, 139], motion-capture

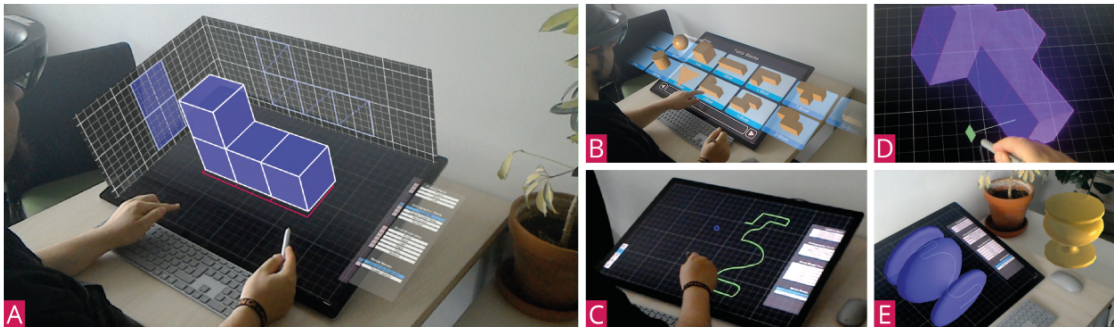


Figure 5.1: A general overview of DesignAR [137].

pens [140], hand tracking [141], or specialized VR trackers and gloves to monitor finger trajectories or hand motion [142]. Many of these conceptual design approaches, such as sketch-based interfaces, require designers to construct all surface boundaries before visualizing the final surface, following a Boundary Representation (B-Rep) approach.

Other approaches rely on mesh sculpting techniques [143], enabling users to shape 3D objects from coarse geometry as if it were clay [144] (Figure 5.2). These applications employ techniques such as octrees [145], voxels [146], and polygonal meshes [147]. In addition to these algorithms, further methods have been developed to facilitate sculpting, including mass–spring systems [148], mesh morphing [149], and formulations based on Non-Uniform Rational B-Splines (NURBS) [150]. Some of these systems also integrate haptic devices to enhance immersion in the virtual scene. Mesh-based modelling is well suited for creating complex and organic surfaces but is not ideal for high-precision modelling.

Figure 5.2: AiRSculpt environment [146].

To overcome this limitation, high-precision CAD models have been integrated with VR and AR tools using solid modelling algorithms such as B-Rep [151] (Figure 5.3) and Constructive Solid Geometry (CSG) [152]. B-Rep defines a solid through its geometric boundaries using curves and NURBS surfaces [153], while CSG constructs complex solids

by applying Boolean operations to primitive shapes such as cubes, spheres, and cylinders [154]. In Malik et al. [155], haptic gloves were introduced to provide force feedback, enabling a more immersive virtual experience, and deep learning techniques were applied to support more intuitive reverse engineering based on CSG representations. Fuge et al. [156] defined a workflow for conceptual design and freeform modelling based on hand gestures and NURBS surfaces. The use of hand gestures improves both interactivity and intuitiveness of the system [157, 158].

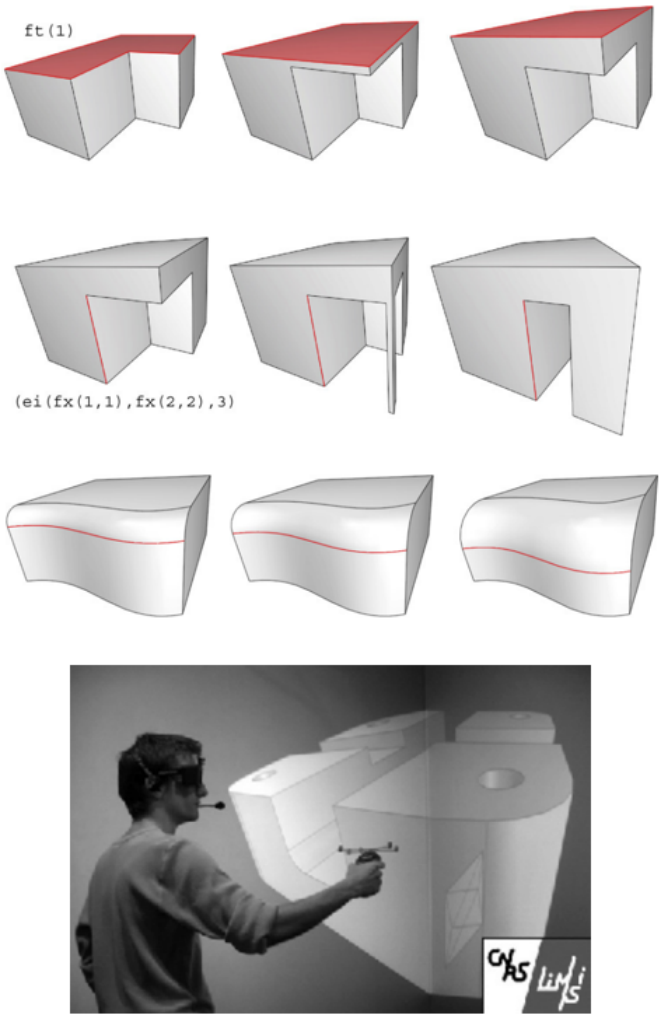


Figure 5.3: Immersive and multimodal user interface of the VRAD/OpenCASCADE application [151].

Despite the existing approaches in the state of the art for qualitative mesh-based sculpting and high-precision CAD modelling based on NURBS, these methodologies may be insufficient when complex organic shapes must be modeled while maintaining high geometric accuracy. To combine the flexibility of mesh-based sculpting with the mathematical precision and generalization capability of B-Rep approaches based on NURBS, freeform modelling using subdivision surfaces generated through refinement algorithms

can be adopted [78]. Subdivision surface methods generate smooth, high-resolution surfaces through iterative refinement of a coarse polygonal mesh, referred to as a control cage. The user interacts directly with this structure by modifying its elements (vertices, edges, and faces) or by adding and removing components, while the final surface is updated accordingly. These approaches are increasingly integrated into computer-aided modelling environments and represent a valid alternative to history-based parametric modelling. Subdivision-based modelling significantly simplifies the creation of complex organic shapes; however, it remains limited in representing exact functional features such as cylindrical holes, pins, or planar surfaces. Therefore, a complete modelling workflow typically includes a final refinement stage in which feature-based modelling techniques are used to detail the geometry.

A key advantage over polygonal mesh sculpting lies in the fact that recursive subdivision schemes can converge toward surfaces with high geometric continuity, up to G^2 continuity (curvature continuity). This property enables subsequent conversion into NURBS patches and their seamless integration with geometries modeled using the traditional B-Rep approach. Another important aspect in the modelling of complex shapes is the possibility of interacting with the geometry through direct gestures, without the use of wearable devices, controllers, or pointers, thus capturing the user's intent and translating it directly into modelling actions. This approach, commonly referred to as a natural interface, has gained increasing attention in both scientific and industrial contexts, particularly in applications related to assisted assembly and interactive simulation [42, 159].

5.1 AR interactive modelling framework based on subdivision surfaces

Based on the previous considerations, an innovative integrated methodology has been developed that implements an immersive augmented reality modelling environment based on freeform modelling using subdivision surfaces, combined with a purpose-designed natural interface relying on hand tracking systems and advanced gesture recognition. The implemented modelling algorithm derives from an adaptation of the well-known Catmull–Clark subdivision surface algorithm (Appendix E). This approach combines the effectiveness of direct modelling with the possibility of subsequently processing the obtained result through a B-Rep geometric kernel, in order to convert the final model into an exact representation based on NURBS patches. The interaction with the geometry to be modelled is implemented as an extension of the GAF/OAF methodology, suitably adapted for the selection and manipulation of the entities composing the control cage of subdivision surfaces. In this way, the user can interact naturally with such entities, while the subdivision algorithm updates the model shape in real time.

The interaction between the user and the entities of the control cage takes place through customised gestures. The control cage comprises three types of entities: points, edges, and faces; the latter can be identified by their centroid. For point-like or small-radius spherical entities, the pinch pose, where the pinch grasp is defined by Equation 4.57, is the most suitable, as it represents the natural gesture for picking them (Figure 4.10). For linear or cylindrical entities with a small base radius, where the cylindrical grasp is defined by Equation 4.55, the cylindrical pose is the most suitable

(Figure 4.9).

The strategy for implementing interactive manipulation of the various geometric entities is based on a procedural scheme in which the user first selects the entities and then modifies their geometric features. With reference to Figure 5.4, during the AR experience the hand-tracking system is always active. When the reference pinch pose is recognised and the proximity GAF/OAF constraints are satisfied for a geometrically grabbable entity (vertex, edge, or face), the entity is added to the selection list. The user can continue selecting other objects by repeating the pinch pose in proximity to other entities, while simultaneously satisfying the GAF/OAF constraints. The selection phase ends when the thumb-up pose is recognised. If the user performs a thumb-down pose, the selection is cancelled. Once the entities are selected, the user can act on them according to another chosen pose. If the user selects the pinch pose, the selected entities are translated according to the pinch point displacement. If the user selects the open palm pose, the entities are extruded following the tracked displacement of the palm centre. The modification ends when the user switches to the thumb-up pose, accepting the change, or to the thumb-down pose, rejecting it. In a similar manner, rotation of the selected entities can be performed by first selecting them using the fist pose while simultaneously satisfying the cylindrical GAF/OAF constraints, and then changing the palm orientation, which constrains the attitude of the selected entities. The modification ends when the user switches to the thumb-up pose, accepting the change, or to the thumb-down pose, rejecting it. In general, the recognition of the thumb-down pose always cancels any active command. This logic enables an extension of the sequence of operations while maintaining a fully natural and intuitive interaction with the geometry to be modelled. The procedure is compatible with the use of different hand-tracking devices and imple-

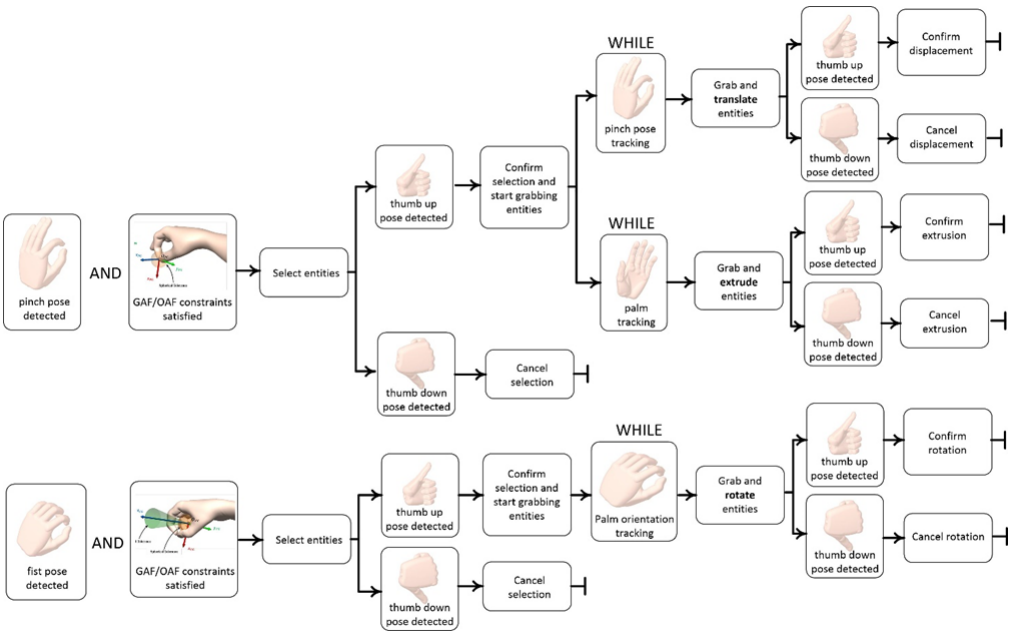


Figure 5.4: Pose recognition and interactive manipulation scheme.

mentations based on different libraries, and it can therefore be considered completely general. However, in the case study presented in the following section of the manuscript, hand tracking and pose recognition are specialised for a specific device.

The Catmull–Clark algorithm is optimised for polygonal meshes composed mainly of quadrilaterals, but it must be adapted to handle triangular meshes in order to achieve general and robust results. In VR and AR applications, triangular meshes are preferred due to their efficiency in real-time rendering [74]. In particular, triangular meshes offer superior computational performance and are better suited for models designed to run on standalone devices with limited hardware resources. For this purpose, the Catmull–Clark algorithm can be adapted for VR and AR applications by converting the triangular mesh into a quadrilateral mesh where possible. The subdivided quadrilateral mesh must then be reconverted into a triangular mesh, as triangular representations are optimised for faster rendering and improved performance in real-time applications (Listing 5.1). The conversion from triangular to quadrilateral mesh, as well as the subsequent reconversion from quadrilateral to triangular mesh after refinement, must be carefully controlled, with particular attention to the diagonals of quadrilateral faces. For this reason, the Delaunay triangulation approach (Appendix F) could not be employed directly, since it does not guarantee explicit control over the orientation of the diagonals (Figure 5.5).

During the transition from the initial triangular mesh to the quadrilateral mesh, each pair of triangles defines a diagonal of the corresponding quadrilateral face. After the subdivision process, the conversion back to triangular faces must ensure the alignment between the diagonal defined by each pair of final triangles and the diagonal of the initial generating pair (Figure 5.6). In particular, the control of diagonal directions, required

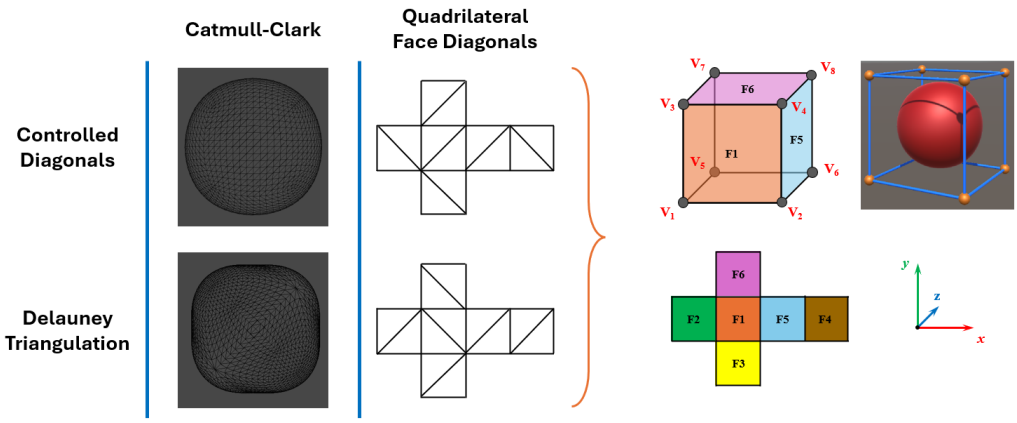


Figure 5.5: Catmull-Clark algorithm applied within both the controlled diagonals framework and Delaunay triangulation.

for the adapted Catmull–Clark algorithm on triangular meshes, is based on the use of base cubes with face indexing (Figure 5.7a). The face indexing F_{ID} allows identification of the reference frame RF of each face and its type (front-face or back-face). Adjacent faces belonging to different base cubes share the same F_{ID}. Each face is described by an F_{Num}, which consists of a set of indices corresponding to its vertices. The ordering of these indices is determined by the corresponding RF: for faces indexed F1 and F4, the order follows the x-axis and then the y-axis; for F2 and F5, the order follows the y-axis

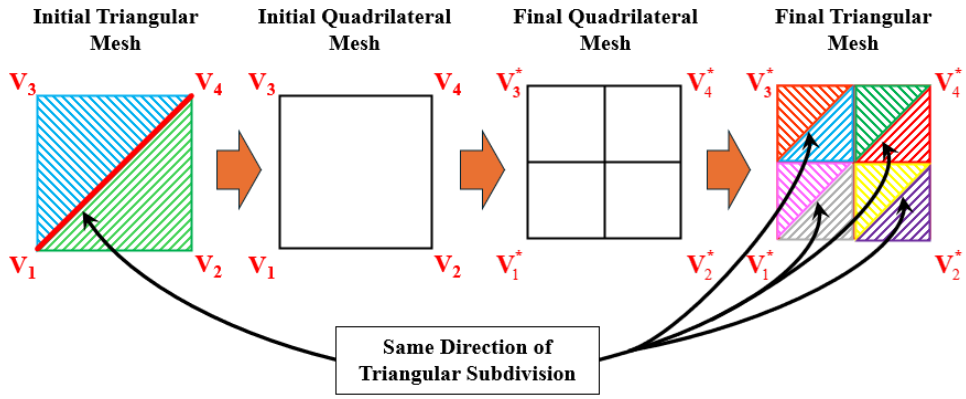


Figure 5.6: Catmull–Clark algorithm adapted for triangular meshes preserving the diagonal direction when transitioning between triangular and quadrilateral meshes.

and then the z-axis; and for F3 and F6, the order follows the z-axis and then the x-axis. For each quadrilateral face, pairs of triangles can be generated using the F_{Num} and the face type. Two lists of information describe each triangle mesh: the first contains the spatial coordinates of the three vertices, while the second specifies their ordering, which is essential for back-face culling [160]. To extract the second list of information, two integer arrays are used, containing the vertex positions in the F_{Num} (Listing 5.2). The array in Equation 5.1a is used for front-faces, while the array in Equation 5.1b is used for back-faces:

$$\mathbf{v}_{\text{Front}} = \{0, 2, 3, 0, 3, 1\} \tag{5.1a}$$

$$\mathbf{v}_{\text{Back}} = \{0, 3, 2, 0, 1, 3\} \tag{5.1b}$$

The first three indices identify the first triangle, and the last three identify the second triangle of the pair. These arrays are invariant under cyclic rotations for the two groups of three elements, as each cyclic permutation of each group preserves the geometric structure of the triangles defined by the specified vertices. By extracting from F_{Num} the indices at the positions specified by the vectors $\mathbf{v}_{\text{Front}}$ and $\mathbf{v}_{\text{Back}}$, a new array $\mathbf{T}_{\text{FV}}$ is obtained for each face if it is a Front-Face, or $\mathbf{T}_{\text{BV}}$ if it is a back-face. The arrays $\mathbf{T}_{\text{FV}}$ and $\mathbf{T}_{\text{BV}}$ contain the order of the vertices needed to generate each pair of front and back triangles, respec-tively. These arrays contain two pairs of repeating vertices: by using nested loops, it is possible to view all the combinations and generate the integer array that defines the quad-rilateral face. The conversion from a quadrilateral mesh to a triangular mesh is achieved by starting from new arrays of four integers that identify the indices of the new vertices describing each quadrilateral face. By using the integer array in Eq. 5.2, the quadrilateral face is de-composed into a pair of triangular faces.

$$\mathbf{v}_t^q = \{0, 1, 2, 0, 2, 3\} \tag{5.2}$$

The enumeration of the vertices of the initial mesh is done by iterating over all combinations of coordinates along the x, y, and z axes, using nested loops arranged in this

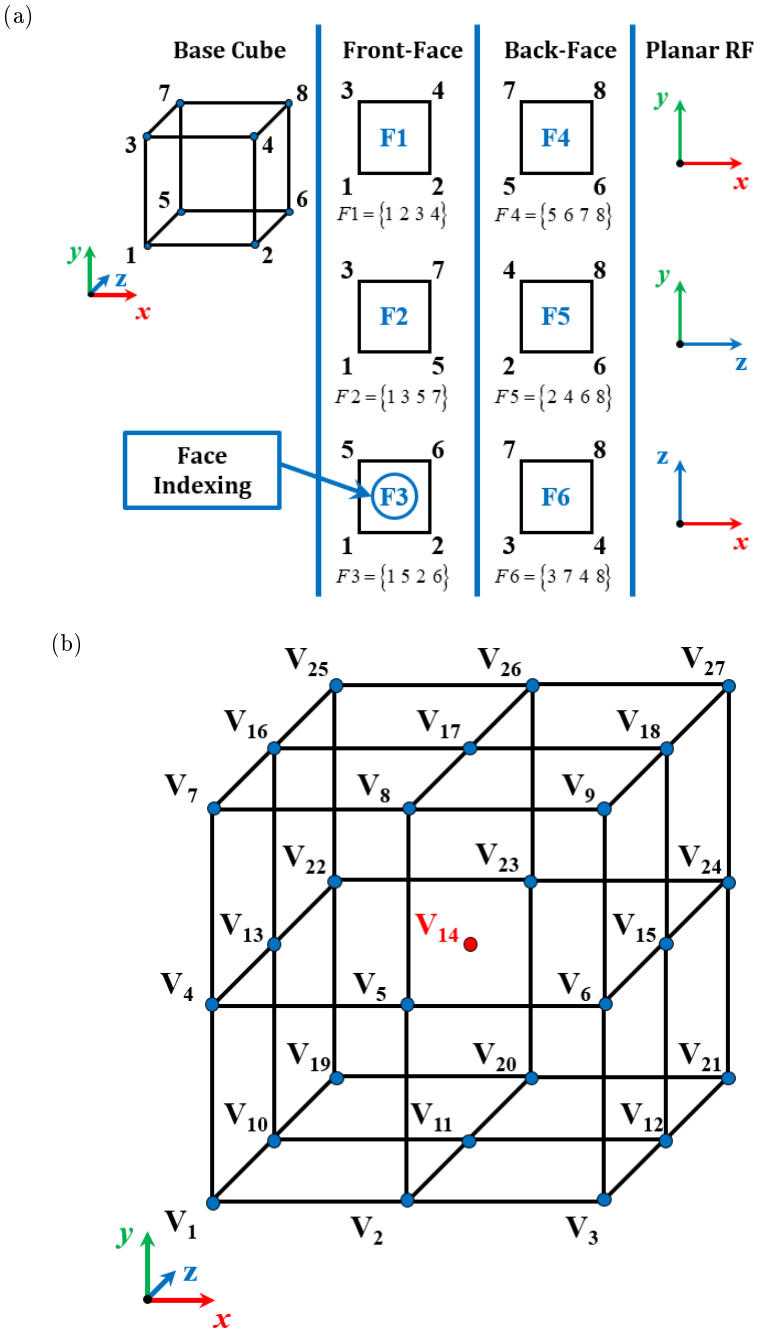


Figure 5.7: (a) Face indexing for each face of the base cube created in the application with the left-handed reference frame, and (b) the vertex naming order in the generic mesh, which allows the adaptation of the Catmull–Clark algorithm and the removal of not-showed vertices (red).

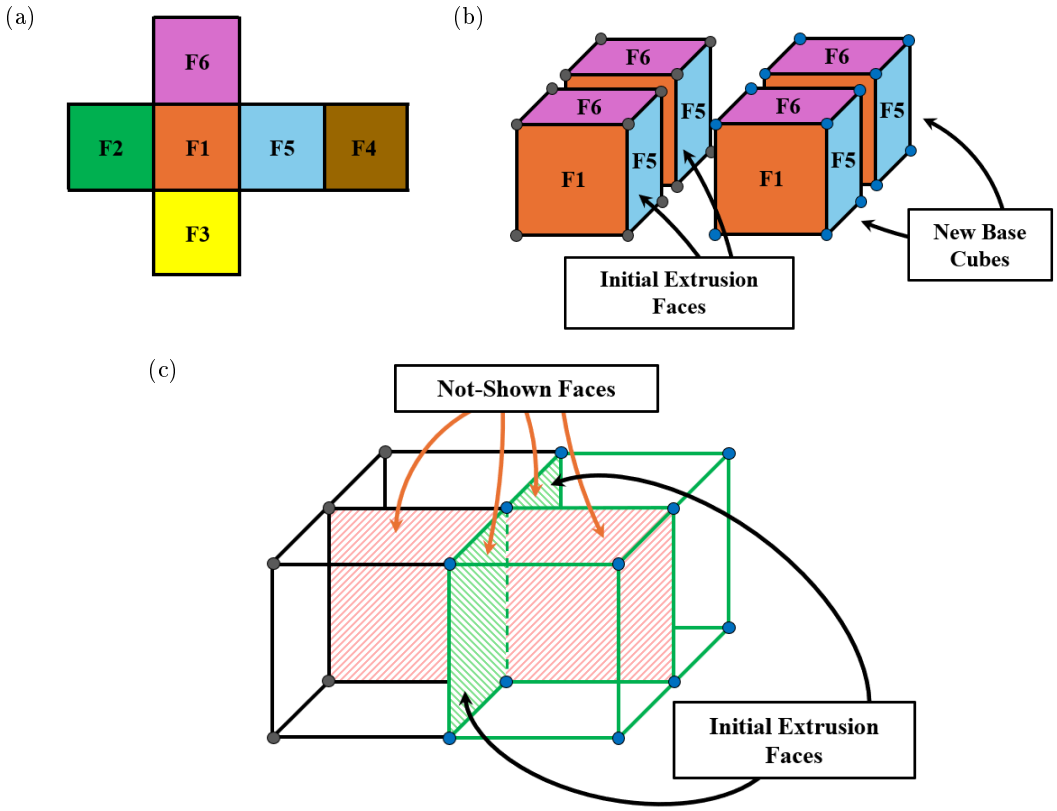


Figure 5.8: (a) Base cubes face indexing, (b) faces indexing of the new base cubes for an extrusion from F5 faces and (c) not-shown faces after the extrusion operation.

order, from the innermost to the outermost, generating a three-dimensional grid. The vertices inside the control solid are eliminated (red vertex in Figure 5.7b).

In addition to the manipulation of the initial mesh, a face extrusion function for one or more base cubes has been implemented. The main challenge consists in generating a triangular mesh after the extrusion of quadrilateral faces while preserving the nomenclature required by the adapted Catmull-Clark algorithm. The use of the F_{ID} values and face types enables a straightforward procedure for generating the triangular mesh following the extrusion operation. When new base cubes that do not belong to the initial mesh are added, the ordered vertex enumeration based on nested loops is no longer applicable. Consequently, a more general algorithm, independent of ordered vertex enumeration, is required. In this process, each F_{ID} must be considered individually (Figure 5.8a). The first step consists in generating four new vertices which, together with the four vertices of the selected face, define a new base cube. The face defined by these four new vertices retains the same F_{ID} and face type as the original face. The corresponding F_{Num} must be defined consistently with the reference frame associated with that F_{ID}. The opposite face, characterised by the opposite face orientation, is classified as a non-visible face since it coincides with the original face. As a consequence, the original face also becomes non-visible after the extrusion operation (Figure 5.8c). The F_{ID} and the face type of the remaining faces of the new base cube depend on their adjacency relationship with

the original base cube. In particular, adjacent faces belonging to different base cubes share the same F_{ID} (Figure 5.8b). The corresponding F_{Num} values must also be defined consistently with the reference frame associated with the corresponding F_{ID}. When multiple adjacent faces are extruded simultaneously, the procedure remains substantially unchanged, except for the treatment of the four lateral faces generated by the extrusion process. In this case, if two faces belonging to different newly generated base cubes have the same F_{Num} and opposite face types, they represent an internal interface between adjacent cubes and are therefore classified as non-visible faces. Consequently, such faces are excluded from the final mesh representation.

Listing 5.1: Implementation of the mesh generation procedure based on controlled face triangulation with front and back-face handling and indexed vertex mapping.

```
1
2   private Mesh GernericMesh(List<Vector3> new_vec3, List<int[]>
        Face_index_gen, List<string> Face_Type_gen)
3   {
4       var ts_gen = new List<int>();
5       for (int i = 0; i < Face_index_gen.Count; i++)
6       {
7           int[] q = Face_index_gen[i];
8           var ts_temp = new List<int>();
9
10          // For each face, generate the 6 vertices that define the
                triangles
11          if (Face_Type_gen[i] == "F1" || Face_Type_gen[i] == "F2" ||
                Face_Type_gen[i] == "F3")
12          {
13              ts_gen.AddRange(arr1.Select(a => q[a]));
14              ts_temp.AddRange(arr1.Select(a => q[a]));
15          } // Add front-face triangle vertices to the ts list
16
17          else if (Face_Type_gen[i] == "F4" || Face_Type_gen[i] == "F5" ||
                 Face_Type_gen[i] == "F6")
18          {
19              ts_gen.AddRange(arr2.Select(a => q[a]));
20              ts_temp.AddRange(arr2.Select(a => q[a]));
21          } // Add back-face triangle vertices to the ts list
22
23          Face_Mesh_Index[face[i]].Clear();
24          Face_Mesh_Index[face[i]] = ts_temp; // Assign to each face the
                list of integers used to generate the mesh
25
26          // Debug.Log(face[i].name + ": " + string.Join(" ",
                Face_Mesh_Index[face[i]].Select(a => (a + 1).ToString())));
27      }
28
29      Mesh mgen = new Mesh();
30      mgen.SetVertices(new_vec3);
31      mgen.SetTriangles(ts_gen, 0);
32      return mgen;
33  }
```

Listing 5.2: Implementation of the base control cage generation for a structured parallelepiped, using face based indexing and parametric vertex mapping.

```
1   private void Parallelepiped()
2   {
3       // Generation of the F1 face list
4       for (int i = 0; i < ny; i++)
5       {
```

```
            for (int j = 0; j < nx; j++)
            {
                int[] int1 = { j * dx + i * dy, (j + 1) * dx + i * dy, j *
                    dx + (i + 1) * dy, (j + 1) * dx + (i + 1) * dy };

                if (!Face_index.Contains(int1))
                {
                    Face_index.Add(int1);
                    Face_type.Add("F1");
                }
            }
        }

        // Generation of the F2 face list
        for (int i = 0; i < nz; i++)
        {
            for (int j = 0; j < ny; j++)
            {
                int[] int1 = { j * dy + i * dz, (j + 1) * dy + i * dz, j *
                    dy + (i + 1) * dz, (j + 1) * dy + (i + 1) * dz };

                if (!Face_index.Contains(int1))
                {
                    Face_index.Add(int1);
                    Face_type.Add("F2");
                }
            }
        }

        // Generation of the F3 face list
        for (int i = 0; i < nx; i++)
        {
            for (int j = 0; j < nz; j++)
            {
                int[] int1 = { j * dz + i * dx, (j + 1) * dz + i * dx, j *
                    dz + (i + 1) * dx, (j + 1) * dz + (i + 1) * dx };

                if (!Face_index.Contains(int1))
                {
                    Face_index.Add(int1);
                    Face_type.Add("F3");
                }
            }
        }

        // Generation of the F4 face list
        for (int i = 0; i < ny; i++)
        {
            for (int j = 0; j < nx; j++)
            {
                int[] int1 = { j * dx + i * dy + nz * dz, (j + 1) * dx + i *
                     dy + nz * dz, j * dx + (i + 1) * dy + nz * dz, (j + 1)
                    * dx + (i + 1) * dy + nz * dz };

                if (!Face_index.Contains(int1))
                {
                    Face_index.Add(int1);
                    Face_type.Add("F4");
                }
            }
        }

        // Generation of the F5 face list
        for (int i = 0; i < nz; i++)
        {
```

```
66            for (int j = 0; j < ny; j++)
67            {
68                int[] int1 = { j * dy + i * dz + nx * dx, (j + 1) * dy + i *
                      dz + nx * dx, j * dy + (i + 1) * dz + nx * dx, (j + 1)
                    * dy + (i + 1) * dz + nx * dx };
69
70                if (!Face_index.Contains(int1))
71                {
72                    Face_index.Add(int1);
73                    Face_type.Add("F5");
74                }
75            }
76        }
77
78        // Generation of the F6 face list
79        for (int i = 0; i < nx; i++)
80        {
81            for (int j = 0; j < nz; j++)
82            {
83                int[] int1 = { j * dz + i * dx + ny * dy, (j + 1) * dz + i *
                      dx + ny * dy, j * dz + (i + 1) * dx + ny * dy, (j + 1)
                    * dz + (i + 1) * dx + ny * dy };
84
85                if (!Face_index.Contains(int1))
86                {
87                    Face_index.Add(int1);
88                    Face_type.Add("F6");
89                }
90            }
91        }
92    }
```

5.1.1 Case study: interactive modelling using Varjo XR-4 and Ultraleap Motion Controller

As a case study, the aforementioned framework was implemented for the modelling of an organic shape in AR mode, developed using the Unity platform. The headset chosen for the implementation of the case study is the Varjo XR-4. It does not have its own hand-tracking system, but it can be achieved by using 2nd generation Ultraleap's Leap Motion Controller to be integrated with a proprietary mounting kit. This device has two infrared cameras and multiple infrared LEDs. For each hand, the controller is able to track 24 joints, returning the spatial coordinates of 4 finger joints for each of the 5 fingers (knuckle, proximal, intermediate, and distal) and 5 additional joints for the palmand wrist. The manufacturer provides development libraries that allow direct access to the tracking information that becomes necessary to be able to apply the algorithm and procedures described in the previous section.

The great advantage in using the Varjo XR-4 is not only in the high technical specifications but above all in the availability of support libraries (*Varjo XR Plugin* and *Varjo SDK*) made available by the manufacturer that allow extensive and complete access. The libraries have several extremely useful features for customising mixed-reality environments and allow almost complete control of the potential offered by the instrumentation. More specifically, the *Varjo XR Plugin* gives access to the `VarjoMixedReality` class to manage the Varjo interface and to the `VarjoRendering` class to manage rendering options. The AR functionality of the Varjo XR-4 can be enabled with the following

two scripted commands:

```
VarjoMixedReality.StartRender();
VarjoRendering.SetOpaque(false);
```

The first line starts the rendering of the scene on the two displays and the second line enables passthrough (rendering of the video frame acquired by the cameras). To ensure high visual realism, it is advisable to take into account depth occlusion [159], which allows the correct stacking between real objects and virtual objects, preventing virtual objects from incorrectly overlapping real ones when they are further from the observer's point of view. This verification can be managed directly from the libraries supplied with the Varjo by activating the calculation of the depth map, calling the respective methods in the `VarjoMixedReality` class using the following function:

```
VarjoMixedReality.EnableDepthEstimation();
```

The calculation of depth occlusions is performed using data from the LiDAR sensor. Although depth occlusion is computationally demanding, it significantly improves the realism of the scene. However, it should be noted that enabling mixed reality can vary from headset to headset, and not all headsets directly support depth map calculation, which is a time-consuming operation. As for hand tracking, the Ultraleap Motion Controller library allows full access to both raw data (position and orientation of hand bones and joints) and processed data (pose and gesture estimation). In this example, raw data were extracted and processed, as they are more suitable for implementing the adapted GAF/OAF methodology. Communication with the controller is managed by the Leap Service Provider, which is able to retrieve synchronous information that are stored in a *frame* data structure. For each acquisition frame, the frame object is updated and the information can then be extracted using the corresponding methods and properties. For example, it is possible to extract the vector of recognised hands, each containing five fingers, which in turn contain information on the connecting structures (bones and joints). Below are some of the C# commands that enable access to the data of the right and left recognised hands, using the Leap namespaces:

- To access the hand information from an acquired frame:

  ```
  Left hand → Hand leftHand = frame.GetHand(Chirality.Left);
  Right hand → Hand rightHand =
  frame.GetHand(Chirality.Right);
  ```

- To access the coordinates of the j-th finger of the right hand (0 = thumb, 1 = index, 2 = middle, 3 = ring, 4 = pinky):

  ```
  Vector3 tipCoordinate = rightHand.fingers[j].TipPosition;
  ```

- To access the coordinates of the joint of the k-th bone of the j-th finger of the right hand (0 = metacarpal, 1 = proximal, 2 = intermediate, 3 = distal):

  ```
  Vector3 jointCoordinates =
  rightHand.fingers[j].bones[k].NextJoint.ToVector3();
  ```

- To access the angle between two adjacent bones (bone1 and bone2) on the j-th finger of the right hand:

```
Bone bone1 = rightHand.fingers[j].bones[k1];
Bone bone2 = rightHand.fingers[j].bones[k2];
float angle = Vector3.Angle(
  bone1.Direction.ToVector3(),
  bone2.Direction.ToVector3()
);
```

- To access the coordinates of the middle of the palm of the right hand:

```
Vector3 palmPosition = rightHand.PalmPosition;
```

- To access the normal vector of the palm of the right hand:

```
Vector3 palmAxis = rightHand.PalmAxis;
```

- To access the coordinates of the pinch point of the right hand:

```
Vector3 pinchPoint = rightHand.GetPinchPosition();
```

The recognition of the hand pose is achieved by comparing the angle between adjacent bones with those of a reference pose, within a specific tolerance. For the case study, we used the *Pose Detector* object provided in the Ultraleap's library for direct comparison with standard poses (pinch, fist, thumb-up, thumb-down). The initial control cage is generated as a rectangular box of $0.5 \times 0.5 \times 0.5$ m dimensions. The parameters n_x, n_y, n_z, specifying the number of cage subdivisions along the three main axes, are chosen as $5, 5, 5$. The number of vertices N_{Vertex} defining the initial control cage is calculated as the difference between the total vertices $N^{\text{Tot}}_{\text{Vertex}}$ generated by the nested loops and the hidden vertices $N^{\text{NS}}_{\text{Vertex}}$, which are internal to the cage itself:

$$N_{\text{Vertex}} = N^{\text{Tot}}_{\text{Vertex}} - N^{\text{NS}}_{\text{Vertex}} = (n_x+1)(n_y+1)(n_z+1) - (n_x-1)(n_y-1)(n_z-1) \quad (5.3)$$

The number of edges N_{Edge} and the number of faces N_{Face} of the control cage are expressed by Equations 5.4 and 5.5, respectively:

$$\begin{aligned} N_{\text{Edge}} =& 2n_x(n_y-1) + 2n_x(n_z+1) + 2n_y(n_z-1) \\ &+ 2n_y(n_x+1) + 2n_z(n_x-1) + 2n_z(n_y+1) \end{aligned} \quad (5.4)$$

$$N_{\text{Face}} = 2n_xn_y + 2n_yn_z + 2n_xn_z \quad (5.5)$$

The vertices and centroids of the faces are represented by spheres, while the edges are identified by cylinders. The assignment of the F_{Num} and F_{ID} to each face is managed using the parameters n_x, n_y, n_z, and the parameters δ_x, δ_y, δ_z, which indicate the variation in the indices of adjacent vertices along the x, y, and z directions, respectively.

Figure 5.9 shows six frames captured by the Varjo XR-4 cameras during the AR experience of the case study. In the first step the user explores the virtual scene and

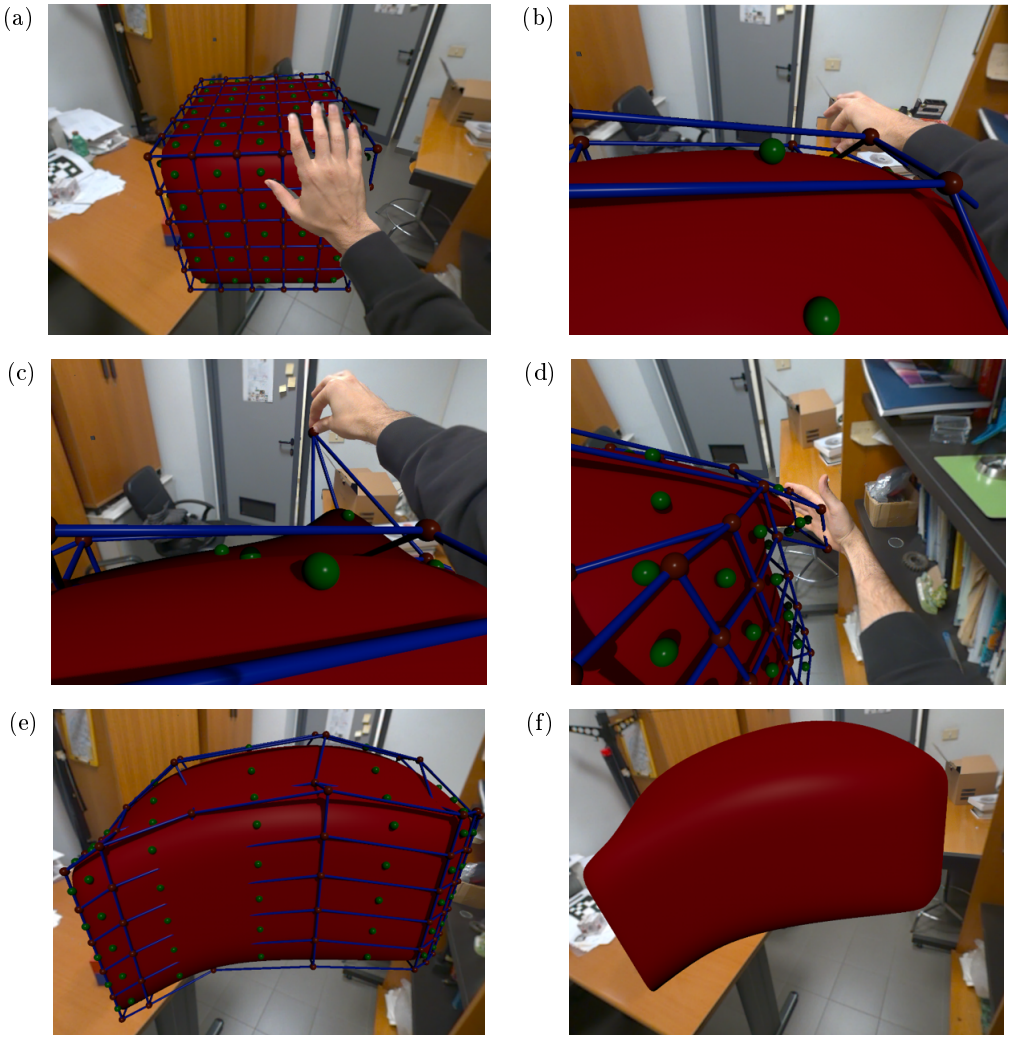


Figure 5.9: Snapshots from the interactive modelling experience using the adapted GAF/OAF methodology and the subdivision surfaces. Explanation of each snapshot is provided in the text.

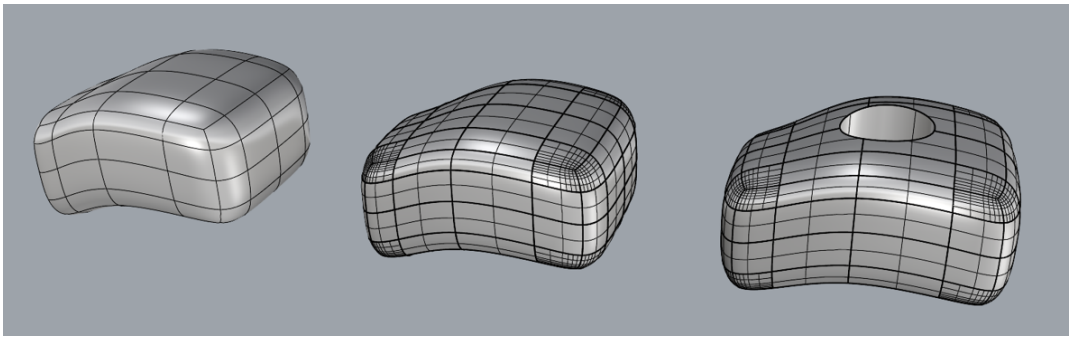


Figure 5.10: Finalising the modelling with the conversion of the SubD surface (on the left) into NURBS patches (in the middle) and then a perfect hole is subtracted from the resulting B-Rep solid (on the right). The final modelling is achieved into Rhinoceros.

locates the starting geometry, aligning it with the real environment using marker-based tracker. In the first frame Figure 5.9a, it is possible to see how depth occlusion with the user's hands is active, which provides a high level of realism. The control cage is depicted in blue, the control vertices in red, the edges of the cage in blue, and the subdivided surface in red. Points and lines are represented as three-dimensional geometries in order to enhance spatial awareness. Green spheres indicate the centres of the control cage faces, facilitating selection and interaction (grabbing). In the second step, the user selects various vertices of the control cage, applying different transformations to sculpt the final geometry. Frame Figure 5.9b shows an instant in which the user selects a vertex. The persistence of depth occlusion further improves the accuracy of interaction by supporting correct entity selection. The third frame Figure 5.9c captures the user performing an interactive movement of a vertex, and it is clearly visible how the geometry described by subdivision surfaces follows this motion in real time. The fourth frame Figure 5.9d shows an example of face extrusion of the control cage triggered by tracking the open palm gesture. In the subsequent frames Figure 5.9e and Figure 5.9f, the final geometry is shown, with and without the control cage, respectively.

After the modelling phase, the resulting geometry (mesh and SubD cage) can be exported into a feature-based CAD environment to convert mesh polygons into NURBS surfaces and complete the design with functional fit features. Figure 5.10 shows shows the exported resulting geometry (on the left) into Rhinoceros where it is converted to NURBS patches (in the middle) and a hole is added as a Boolean subtraction(on the right).

Chapter 6

Virtual assembly and industrial applications

Interactive assembly represents one of the most mature and relevant applications of Extended Reality technologies in the industrial domain. Since the 1990s, assembly simulation has been recognized as an effective tool for operator training and for the early validation of operational procedures [161]. With the evolution of Virtual Reality, Augmented Reality, and Mixed Reality technologies, these systems have achieved increasing levels of realism and interactivity, contributing to improvements in productivity, safety, and efficiency of industrial processes [162–167]. One of the main advantages of virtual assembly lies in the possibility of training operators in controlled environments, reproducing complex procedures without the need for physical prototypes or disruption of production activities (Figure 6.1). In this context, XR platforms enable the practice of operational skills in realistic scenarios, reducing the time, cost, and risks associated with traditional training approaches [168, 169]. For instance, Roldán et al. [170] demonstrated the effectiveness of a virtual environment with on-screen instructions in improving assembly speed and accuracy compared to conventional training methods. In parallel, numerous studies have addressed the challenge of interacting with virtual objects during assembly tasks. In particular, Valentini et al. [42, 122, 159] introduced and refined approaches based on natural object manipulation and the integration of physical models for simulating assembly constraints. These works highlighted how natural interaction techniques can significantly enhance procedural understanding and user experience quality.

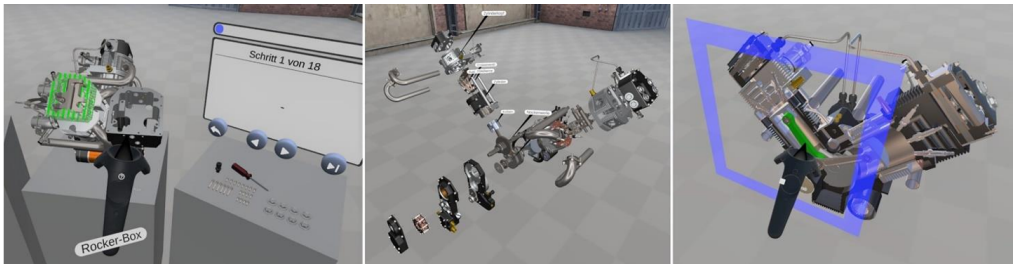


Figure 6.1: Training of an assembly procedure guided by textual instructions and visual cues (left); exploded view of the model to explore the individual components of the object (center); sectional view used to inspect the internal structure of the assembly (right) [171].

While early applications were mostly developed in fully virtual environments, Augmented Reality technologies have progressively taken a central role in industrial assembly activities. The ability to overlay instructions, 3D models, and operational information directly onto the real environment makes AR particularly suitable both for training and for on-site task support [172, 173]. In this context, optical see-through devices are especially effective, as they preserve direct visual contact with the operational environment, enabling a tighter integration between physical components and virtual content [174]. Smart glasses currently represent one of the most widely adopted solutions due to their practicality and suitability for prolonged use during operational sessions [175].

Several studies have also shown how Extended Reality can act as a spatial and collaborative interface for digital twins, improving the understanding of multiphysics phenomena and supporting shared decision-making among heterogeneous teams, including stakeholders from engineering, production, and maintenance [176, 177]. Recent industrial applications further confirm their relevance, ranging from engineering reviews and design validation to remote inspection and maintenance of critical assets through their digital replicas [178, 179]. In this context, Chheang et al. [180] demonstrated how collaborative VR can support component inspection in additive manufacturing through the overlay of digital twins onto physical objects. Similarly, Yang et al. [181] proposed a real-time digital twin system with VR interfaces for immersive monitoring and control of manufacturing processes, while Mahdi et al. [182] developed an OPC UA-based architecture for wire arc additive manufacturing, enabling real-time synchronization between physical and virtual assets. Overall, these examples highlight the role of VR and AR as advanced interaction layers for digital twins, while also emphasizing the importance of carefully evaluating hardware performance. Parameters such as latency, tracking accuracy, and usability significantly influence the effectiveness of integrating immersive environments with digital representations in operational industrial settings.

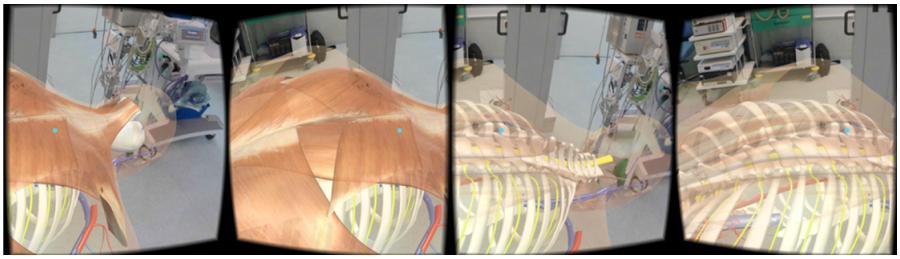


Figure 6.2: AR application as a training tool in medical education [183].

Within the industrial domain, XR-based approaches have also been widely applied beyond assembly training, including visual inspection tasks supported by VR solutions that enable detailed examination of components, defect detection, and quality assurance within controlled environments [184–186].

Beyond industrial contexts, similar XR paradigms have been extended to other application domains where human performance and training are central. In particular, the medical field represents one of the most prominent areas of XR adoption (Figure 6.2), where immersive simulations support surgical preparation, anatomical exploration, and skill acquisition under safe yet highly realistic conditions [183, 187–189]. In this context,

Avola et al. [190] developed an immersive VR endoscopy prototype capable of capturing fine motor actions and task completion times, demonstrating how quantitative performance metrics can be extracted from virtual simulations. Building on this performance-oriented perspective, VRheab [191] introduced a neural network-based motor rehabilitation system grounded in repeated user trials and objective movement accuracy recording, while Avola et al. [192] proposed a low-cost full-body rehabilitation framework based on serious games, highlighting the usefulness of weighted averaging across repeated trials to mitigate learning and fatigue effects. Although these studies were primarily focused on clinical applications, they established practical protocols for collecting and processing user performance data in VR environments, particularly through repeated task executions and the management of outlier trials.

Overall, these examples demonstrate the transversal relevance of Extended Reality for industrial and professional training, while also emphasizing its fundamental role in supporting human-centered activities.

6.1 Evaluation of VR device usability in industrial applications

Despite these advances,significant challenges remain in the implementation of realistic and interactive VR scenes. Achieving a convincing sense of immersion requires not only high-fidelity graphical rendering but also low-latency interaction, ergonomic usability, and seamless integration of hardware and software. These requirements impose constraints on both device selection and system design, often leading practitioners to choose VR headsets and controllers based primarily on technical specifications such as resolution, refresh rate, or tracking capabilities. However, such specifications do not always translate into optimal user interaction or usability in practice. Moreover, many VR environments are developed in a device-agnostic manner, which ensures portability across platforms but frequently results in suboptimal performance and interaction quality [193–195]. This gap between hardware capabilities andactual user experience underscores the need for systematic evaluations that extend beyond datasheets and specifications.

The concept of usability, defined according to the ISO 9241 standard of 1998 and integrated by the ISO/IEC 25010 standard, includes the effectiveness, efficiency, and satisfaction of specific users in achieving specific goals in a specific context of use. Usability testing is commonly adopted as an empirical evaluation method in which representative users interact with a system to provide direct evidence of its performance in realistic usage conditions. The primary objective is to assess whether a product satisfies its intended requirements and supports the expected user tasks.

In recent years, there has been a growing interest in the systematic evaluation of XR hardware for industrial applications. Barve and De Amicis [196] presented one of the first user-centered evaluations of a low-cost VR headset used in a harvesting simulator, quantifying cognitive load, usability, and simulator sickness through established measurement instruments. Gornall et al. [197] conducted a systematic review of XR technologies in Construction 4.0, analyzing the distribution of devices, rendering engines, and interaction paradigms, and identifying key barriers to adoption. Malva et al. [198] proposed a criteria-based comparative study of eleven commercial AR/MR devices for industrial

asset management, focusing on factors such as ergonomics, display quality, and spatial mapping performance. Collectively, these contributions highlight an increasing attention towards evidence-based assessment of XR systems in professional contexts. However, most existing approaches remain limited in scope, as they either focus on single-device evaluations, rely on qualitative assessments, or do not adopt a standardized, task-oriented benchmarking methodology specifically designed for controller-based VR interaction in assembly, inspection, and training scenarios. To address this limitation, the present study proposes a methodological framework that extends beyond conventional device comparison. First, a composite performance metric is introduced to normalize task execution time and accuracy scores onto a unified scale, enabling consistent and reproducible comparisons across heterogeneous VR headsets. Second, the task set is explicitly designed to isolate fundamental interaction primitives underlying industrial operations, including grasping, rotating, inserting, button pressing, and object manipulation, in accordance with established ergonomic principles. Third, a weighted scoring scheme is introduced to reduce the impact of learning and fatigue effects across repeated trials, thereby improving the robustness of the collected data. Finally, the complete evaluation pipeline, from task definition to statistical analysis, is formulated as a reusable benchmark intended to support future VR hardware assessments.

The objective of this work is to provide a comparative evaluation of commercial VR headsets with a focus on interaction performance and usability, relying on quantitative metrics rather than subjective impressions [199]. This differs from prior studies, which often rely on qualitative assessments, isolated device evaluations, or high-level system descriptions. For example, Kamm et al. [200] examined a single headset for dexterity training in multiple sclerosis rehabilitation; Khorasani et al. [201] compared interaction techniques without performing cross-device benchmarking; Chang et al. [202] investigated VR sickness measurement methods; Kaminska et al. [203] proposed general usability guidelines without defining quantitative Key Performance Indicators (KPIs); Nenna et al. [204] analyzed fatigue in virtual robotic control; and Luo et al. [205] studied scenario fidelity in VR-based safety training with a focus on usability and behavioral outcomes.

The experimental evaluation is based on a set of tasks designed to replicate representative industrial operations, enabling the comparison of five commercial VR headsets using objective metrics and inferential statistical analysis. The study focuses on controller-based interaction, which remains the dominant input modality in industrial VR applications. All experiments were conducted in a single controlled laboratory environment with consistent lighting and background conditions, reducing external variability while ensuring identical testing conditions across devices. Each headset was tested using the same software stack based on the *Unity XR Interaction Toolkit*, and a platform-independent virtual environment, ensuring that observed differences primarily reflect hardware characteristics rather than software optimization. Participants had heterogeneous levels of VR experience, reflecting realistic industrial training conditions where prior expertise is typically limited. To improve robustness, weighted averaging of trials (Equation 6.1) and a threshold-based KPI formulation (Equation 6.2) are used to mitigate the influence of outliers and inter-subject variability. Although only five headsets from two manufacturers are included, the methodology is not device-specific. The task set is intentionally based on simple geometries such as cubes, spheres, and prisms, allowing the evaluation

of core interaction properties—including tracking accuracy, input latency, and visual clarity—while minimizing the influence of scene complexity.

Overall, the proposed framework provides a structured basis for evaluating the usability of VR headsets in industrial assembly and inspection scenarios. It can also serve as a foundation for more complex applications, including large-scale CAD interaction, multi-user collaboration, and real-time integration with digital twins. The structure of the study is as follows. Initially it describes the proposed usability evaluation methodology for VR systems. A standardized testing protocol based on objective metrics is introduced for industrial assembly and inspection scenarios. Participants were recruited with diverse professional backgrounds and without specific expertise in VR technologies, and all interactions and feedback were systematically recorded and analyzed. Then, all the results are collected in order to highlight the strengths and weaknesses of the devices.

6.1.1 Description of usability tests in human–system Interaction

In virtual assembly and inspection scenarios, users are required to perform certain actions or recognize system issues through sensory perceptions (sight, hearing,touch, etc.) [35]. Typical operations involve moving, rotating, and inserting components into predefined locations, with the goal of achieving stable and geometrically correct assemblies. These actions demand a high level of precision and coordination, as even small errors can compromise the correctness of the final assembly. In this context, virtual environments are used not only for visualization but also as training and validation tools, enabling users to practice procedures that are later transferred to physical systems or to refine existing workflows.

The devices used in this study consist of head-mounted displays and handheld controllers. While the quality of the virtual environment plays a central role, the choice of hardware also significantly affects the overall user experience and interaction performance. For this reason, a systematic assessment of both visual performance and tracking accuracy is essential when comparing different devices. The commercial VR market offers a wide range of headsets with heterogeneous hardware specifications. Not all devices are equally suited for interaction-intensive tasks such as virtual assembly and inspection, making usability evaluation necessary to identify their respective strengths and limitations. In this study, five commercially available headsets are evaluated and compared in the context of VR-based assembly and inspection tasks: three devices manufactured by HTC (HTC VIVE Pro 1, HTC VIVE Pro 2, and HTC VIVE XR Elite) and two devices by Meta (Meta Quest 3 and Meta Quest Pro).

To ensure a fair comparison independent of computational constraints, all devices were evaluated under controlled and consistent conditions. First, all headsets were connected via wired PC streaming to the same high-performance workstation, ensuring identical rendering resources across all experiments. Second, each device was operated at its native refresh rate and resolution (e.g., 120 Hz for Meta Quest 3 and HTC VIVE Pro 2, and 90 Hz for the remaining headsets), in order to preserve the visual configuration intended by the manufacturers (see Table 2.1 for details). Third, the same virtual environment and interaction framework, based on the Unity XR Interaction Toolkit, was used across all platforms without modifications. These conditions are not intended as assumptions on system performance, but rather as experimental constraints designed to

ensure the validity of cross-device comparisons. Although differences in hardware specifications make it difficult to isolate individual performance factors, the adopted approach captures the overall usability of each system, which is the primary objective of this work.

The usability analysis of virtual reality devices is based on a statistical study conducted on a sample of sixty users (Table 6.1). To avoid learning and fatigue effects, the order in which the devices were used was randomized. All experiments were carried out in a large room with a background noise level below 48 dB SPL. For all devices, the same workspace, floor distance, and reference systems were kept constant in order to minimize potential effects related to lighting conditions; all other parameters were left at their default factory settings. To ensure the reliability and representativeness of the analysis, the participant sample was balanced in terms of gender (30 males and 30 females) and age (ranging from 25 to 50 years). Participation was voluntary, involving individuals with heterogeneous backgrounds to reflect a realistic distribution of potential industrial users. No formal tests of visual acuity or spatial abilities were administered. All participants had normal or corrected-to-normal vision and reported no motor impairments that could interfere with task execution. Participants were not necessarily VR experts or domain specialists. This choice reflects typical industrial conditions, where operators exhibit varying levels of prior experience with VR systems. The use of tasks based on generic primitives (grasping, placing, rotating, etc.) avoids dependence on domain-specific knowledge, while weighted averaging across repeated trials reduces the influence of individual outliers. As a result, the comparative outcomes are robust across different levels of user expertise. Although participants were not selected according to specific professional profiles, all of them were sufficiently familiar with basic technological tools and provided a self-assessed measure of VR experience on a four-level scale (0 = no experience; 3 = high experience). These data, reported in Table 6.1, ensured the inclusion of both novice and moderately experienced users, thereby reducing potential selection bias.

The study involved non-invasive usability testing of VR head-mounted displays with adult volunteers and did not include any clinical procedures, sensitive personal data, or vulnerable populations. In accordance with institutional policies, formal ethical approval was not required, as the study was classified as low-risk. All procedures were conducted in compliance with applicable guidelines and regulations. Participation was voluntary, and informed consent was obtained from all participants prior to their inclusion in the study.

For each user, 15 tasks were defined, of which 10 were quantitative and 5 qualitative. Participants performed these tasks and provided both quantitative and qualitative feedback for each device. The task order was kept constant across all participants to ensure a standardized learning curve and to reduce inter-subject variability in the initial interaction with the virtual environment. The tasks were designed to directly reflect real industrial workflows, as summarized in Table 6.2. The use of primitive geometries was intentional, in order to avoid confounding factors related to CAD model size or scene complexity, ensuring that performance differences can be primarily attributed to the headsets and controllers rather than rendering load.

User ID	Age	Gender	VR Exp (0–3)	User ID	Age	Gender	VR Exp (0–3)
1	29	Male	2	31	26	Male	3
2	27	Male	3	32	30	Male	3
3	26	Male	2	33	31	Male	2
4	28	Male	2	34	28	Female	0
5	35	Male	2	35	40	Male	2
6	27	Female	0	36	27	Female	1
7	29	Male	1	37	30	Male	1
8	31	Male	0	38	32	Male	0
9	27	Male	1	39	29	Female	3
10	28	Female	1	40	45	Female	2
11	30	Male	0	41	47	Female	1
12	46	Male	0	42	39	Female	2
13	25	Female	0	43	33	Male	0
14	26	Female	1	44	29	Male	1
15	33	Female	2	45	34	Female	3
16	40	Female	2	46	33	Female	1
17	30	Female	3	47	39	Male	0
18	42	Male	2	48	25	Male	2
19	50	Male	2	49	32	Female	0
20	32	Female	0	50	27	Female	3
21	29	Female	1	51	35	Female	0
22	33	Male	0	52	36	Male	1
23	46	Female	1	53	37	Male	2
24	50	Female	0	54	28	Male	0
25	37	Male	1	55	29	Male	3
26	36	Female	1	56	37	Female	1
27	40	Male	2	57	33	Male	2
28	28	Female	1	58	39	Male	1
29	36	Female	2	59	28	Female	3
30	25	Female	0	60	32	Male	0

Table 6.1: Users' list, with age, gender, and VR background experience.

Task (s)	Industrial Workflow
Tasks 1–2 (near/far pick and place)	Retrieving components from a parts tray or from a distant shelf (assembly line, warehouse)
Tasks 3–4 (pick, rotate and insert)	Inserting a pin, tightening a screw, coupling a connector (mechanical assembly)
Task 5 (two-hand dynamics)	Handling large or articulated parts (e.g., placing a car door, assembling a tool)
Task 6 (button interaction)	Operating a control panel, entering numeric codes on an industrial keypad
Task 7 (teleporting)	Navigating large industrial sites (factory floor, power plant) for inspection or maintenance
Task 8 (pick and throw)	Sorting defective items into a reject bin, disposing of waste materials (quality control)
Task 9 (reading canvas at distance)	Reading instructions, safety warnings, or instrument displays from several meters away
Task 10 (interactive dynamics—catching falling spheres)	Intercepting falling components in high-speed assembly, picking moving items from a conveyor belt
Tasks 11–15 (qualitative)	Visual inspection of high-resolution models, auditory anomaly detection, ergonomic assessment, low-light operation

Table 6.2: Mapping between the proposed tasks and concrete industrial workflows.

Although no formal spatial ability tests were conducted, the design of the tasks (especially those involving 3D manipulation, navigation, and spatial interaction) indirectly evaluated spatial reasoning and user adaptability. To reduce the influence of initial inexperience or fatigue, each task was performed in five repetitions per user, and a weighted average strategy was applied, assigning less significance to the first and last trials. The feedback is based on Key Performance Indicators (KPIs) [193] and on personal comments about the experiences. To ensure consistency in device handling, each participant was assisted by a trained technician responsible for verifying correct headset positioning and controller usage. The experiments were conducted in a controlled indoor environment, a laboratory space measuring 6×6 m, to ensure safe and repeatable movement conditions.

Following data collection, the KPIs were subjected to inferential statistical analysis to assess whether performance differences between the VR headsets were statistically significant across tasks. For each task, a Repeated Measures Analysis of Variance (RM-ANOVA) was performed, treating the headset as a within-subjects factor [193]. The assumption of sphericity was tested using Mauchly's test [206]; when violated ($p < 0.05$), the degrees of freedom were adjusted using the Greenhouse-Geisser correction [207]. However, if the Greenhouse-Geisser epsilon (GG-ε) exceeded 0.75, the more liberal Huynh-Feldt correction was considered more appropriate [208]. Statistical significance was evaluated based on p-values with a threshold of $\alpha = 0.05$. To assess the practical relevance of the results, generalized eta-squared (η_G^2) values were computed, representing the proportion of variance in user performance attributable to the headset. Finally, Tukey's Honest Significant Difference (HSD) [209] post hoc test was applied to determine which specific pairs of headsets showed significant differences.

All the tasks were implemented using the Unity3D 2022.3.10 engine and the Unity *XR Interaction Toolkit 2.5*, developing a single, platform-agnostic environment (the same scene executed on all devices without modifications). The Unity physics engine (gravity, rigid body dynamics, and collision detection) was active for all tasks. Primitive geometric shapes and basic interactions were chosen instead of realistic CAD models or full digital twins to isolate low-level hardware responsiveness (controller buttons, tracking, visual latency) from confounding factors such as scene complexity or software optimization. This reductionist approach allows direct attribution of performance differences to the headset and controllers. Tests were performed on a high-performance workstation at the Virtual Prototyping Laboratory of the University of Rome Tor Vergata with the following characteristics: Intel i9-12900KF CPU, 128 GB of RAM, Nvidia RTX 3070 8 GB. These specifications support the development and implementation of VR-based experiences and were selected to minimize delays due to insufficient hardware, thus enhancing the reliability of the proposed method.

As previously outlined, 15 tasks are defined, of which 10 were quantitative and 5 qualitative. The quantitative tasks are so named because the KPIs are based on ANOVA evaluation criteria or quantitative KPIs defined ad hoc for each task. The ANOVA criterion is based on variance analysis; therefore, statistical methods are used to quantitatively measure the differences between the devices [210–212].

To methodologically isolate hardware performance from individual differences, such as varying visuospatial aptitudes, motor coordination, or spatial memory, a rigorous within-subject, repeated-measures experimental design was deployed. Since every par-

ticipant evaluated all five devices under a fully randomized presentation order, intrinsic user traits and cognitive profiles were effectively controlled across all hardware conditions, thus eliminating inter-subject confounding. Furthermore, by aggregating multiple trials per task and utilizing a large, demographically balanced sample ($N = 60$), the protocol ensures high statistical power capable of distinguishing genuine hardware-driven performance indicators from user-dependent fluctuations.

The quantitative tasks cover aspects of kinematic manipulation (grab, pick, and place objects) [122], interactive real-time dynamics [159], and interface management [119]. The ten tasks are described in the following sections of this document.

For the qualitative tasks, a custom five-point Likert scale [213, 214] was adopted to collect subjective user evaluations regarding comfort, visual clarity, and perceived interaction quality. The method consists of a set of rapid-response questions, where participants assign a score ranging from 1 to 5, with 1 corresponding to "Strongly Disagree" and 5 corresponding to "Strongly Agree". The five qualitative tasks correspond to Tasks 11 through 15.

1. **Task 1: near-field manipulation—pick and place**
   Rationale: Assess the ability to pick and place objects at the user's fingertips (reachable with the touch of controllers).

   Description: The user is asked to grab three cubes of different colors on a surface and place them on platforms with corresponding colors, placed at a distance of 2 m from the user (see Figure 6.3).

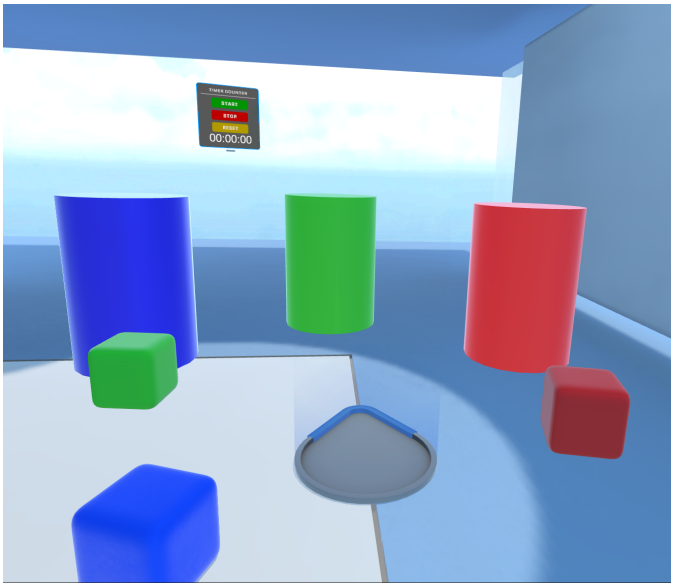


Figure 6.3: An example of the virtual environment of quantitative task 1.

2. **Task 2: far-field manipulation—pick and place**
   Rationale: Assess the ability to pick and place objects far from the user (reachable only with the controller's ray).

Description: The user is asked to grab three cubes of different colors at a distance of 7 m, using the controller's ray, and place them on platforms with corresponding colors, placed at 7 m from the user. A red barrier, placed at 1.5 m from the user, is included as a no-trespassing constraint (see Figure 6.4).

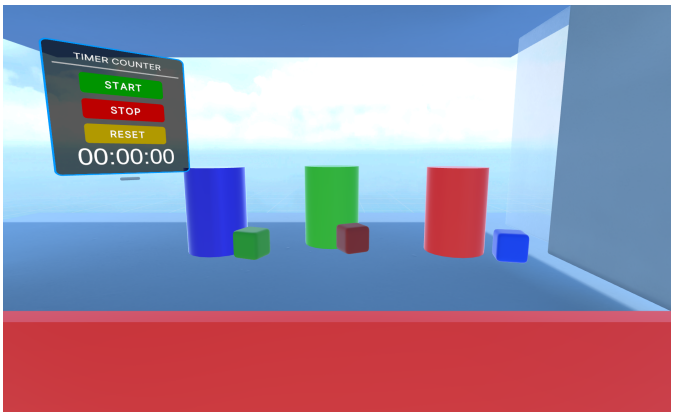


Figure 6.4: An example of the virtual environment of quantitative task 2.

3. **Task 3: near-field manipulation—pick, rotate and insert**
   Rationale: Assess the ability to manipulate objects at the user's fingertips (reachable with the touch of controllers).

   Description: The user is asked to take one prismatic object on the outer surface, rotate it, and insert it into a cavity. The table, on which both the prismatic object and the cavity are positioned, is located 1 m away from the user (see Figure 6.5).

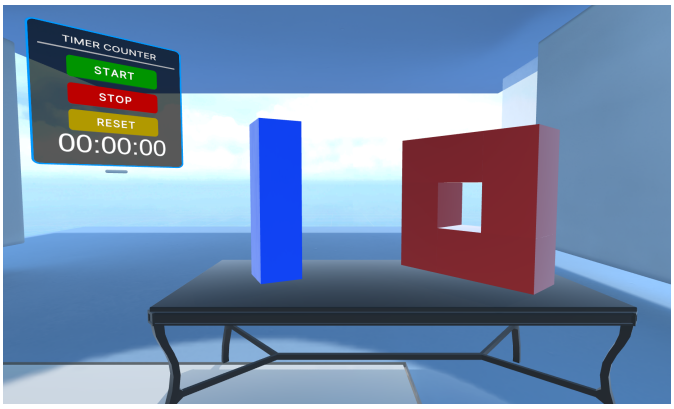


Figure 6.5: An example of the virtual environment of quantitative task 3

4. **Task 4: far-field manipulation—pick, rotate and insert**
   Rationale: Assess the ability to manipulate objects far from the user (reachable with the controller's ray).

   Description: The user is asked to take one prismatic object, rotate it, and insert it into a cavity, using the controller's ray. The table, on which the prismatic object

and the cavity are positioned, is located 3.5 m away from the user. A red barrier, placed 1.5 m from the user, is employed as a no-trespassing restrain (see Figure 6.6).

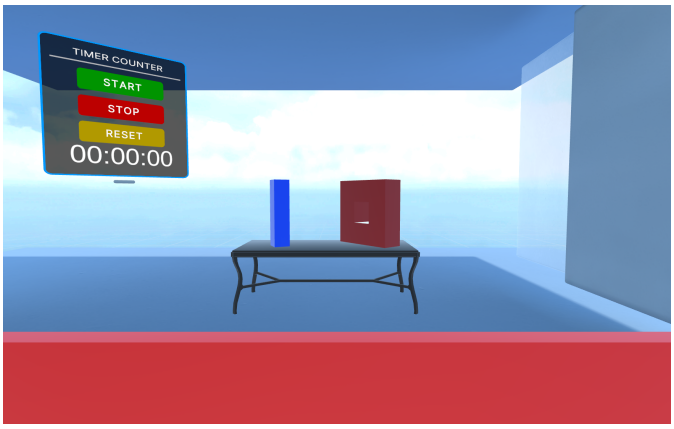


Figure 6.6: An example of the virtual environment of quantitative task 4.

5. **Task 5: two-hand dynamics**
   Rationale: Assess how the user can interact with both hand controllers with objects subjected to physics.

   Description: The user is asked to grasp two prismatic objects (one in each hand),placed on the user's left side, to pick up with them a third object of prismatic shape, located in front of the user's initial position, and move it onto another platform, placed on the user's initial right side (see Figure 6.7).

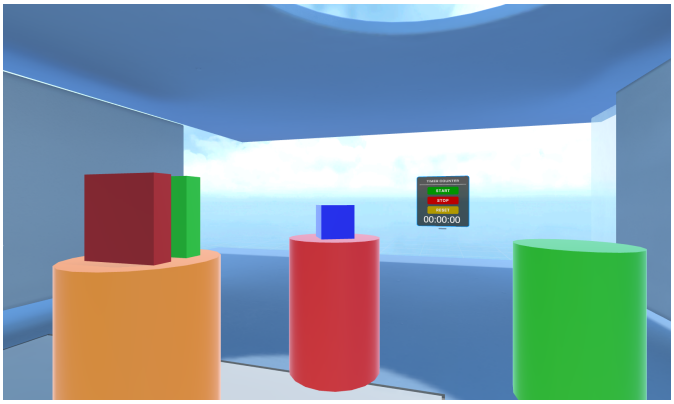


Figure 6.7: An example of the virtual environment of quantitative task 5.

6. **Task 6: button interaction**
   Rationale: Assess the interaction between the user and virtual buttons through multiple pressing.

   Description: The user is asked to type a combination of six randomly generated numbers on a vertical numeric keypad placed at a distance of 1 m (see Figure 6.8).

Figure 6.8: An example of the virtual environment of quantitative task 6.

7. **Task 7: teleporting**
   Rationale: Assess the ability in the recognition of teleporting and accuracy in pointing stations.

   Description: The user is asked to teleport in a sequence of six fixed stations placed at a relative distance of 3 m from each other (see Figure 6.9).

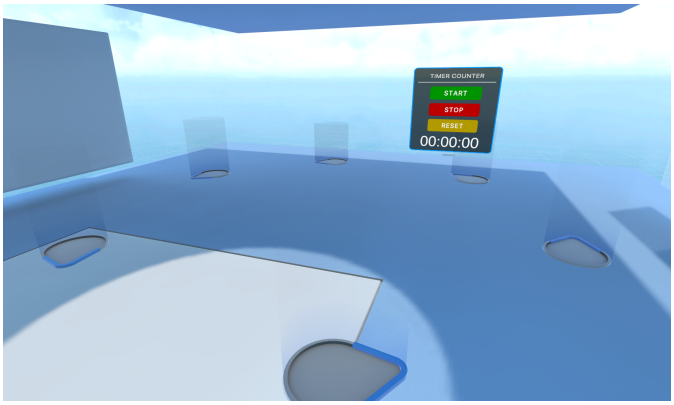


Figure 6.9: An example of the virtual environment of quantitative task 7.

8. **Task 8: dynamics—pick and throw**
   Rationale: Assess how the user can interact with objects subjected to physics and how the release action from the controller can influence the user's perception of physical behavior.

   Description: The user is asked to grab three spherical objects with a radius of 10 cm, placed at a distance of 1 m, and to throw them towards a vertical sticky

dartboard, placed at a distance of 3.5 m. The dartboard has four different scoring zones: black scores 100 points, yellow scores 75 points, red scores 50 points, and blue scores 25 points. A red barrier, placed 1.5 m from the user, is employed as a no-trespassing restraint (see Figure 6.10).

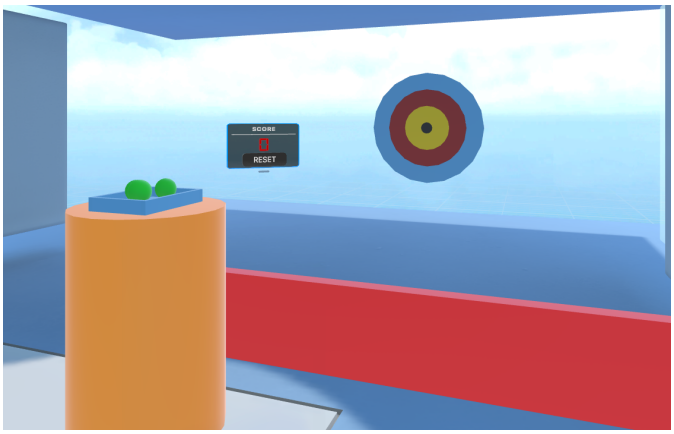


Figure 6.10: An example of the virtual environment of quantitative task 8.

9. **Task 9: reading canvas at different distances**
   Rationale: Assess the readability of text at different distances from the user.

   Description: The user is asked to read 21 letters displayed on three different panels, with 7 letters on each one. The letters are written in uppercase using the Inter-Regular SDF font with a font size of 25 units. These panels are placed at three distances: 4.5 m, 6.5 m and 8.5 m. The user can navigate through the panels using the UI panel on the left. A red barrier, placed 1.5 m from the user, is employed as a no-trespassing restraint (see Figure 6.11).

10. **Task 10: interactive dynamics**
    Rationale: Assess how the user can interact with moving objects in space.

    Description: The user is asked to grab a hollow object and to catch five free-falling spheres from 2.5 m of height in a randomized sequence before they reach the ground. To increase the difficulty, seven spherical objects are present instead of five, challenging the user to catch the last sphere (see Figure 6.12).

11. **Task 11: hi-res model inspection**
    Rationale: Assess the perception of the details of 3D models in the virtual environment.

    Description: The user navigates a high-resolution texturized mesh model (100,000 triangles and 4K resolution texture), making a judgment on the sharpness of the details (see Figure 6.13).

12. **Task 12: sound listening**
    Rationale: Assess the perception of the sound and listening.

Figure 6.11: An example of the virtual environment of quantitative task 9.

Figure 6.12: An example of the virtual environment of quantitative task 10.

Description: The user is asked to listen to a series of sounds and judge the clearness. There are three different panels located in front of the user: the left one is used to adjust the sound source intensity, the front panel allows the activation of the source coming from six different distances and location combinations and the right one allows the users to hear five different frequencies The sound sources are placed at 1 m, 5 m, and 20 m. The frequencies are 150 Hz, 300 Hz, 2 kHz, 10kHz, and 15 kHz (see Figure 6.14).

13. **Task 13: mid-exposure tolerability**
    Rationale: Assess the motion sickness and the sense of dizziness and alienation.

    Description: The user is immersed in the virtual environment for 20 min and, at

Figure 6.13: An example of the virtual environment of qualitative task 11.

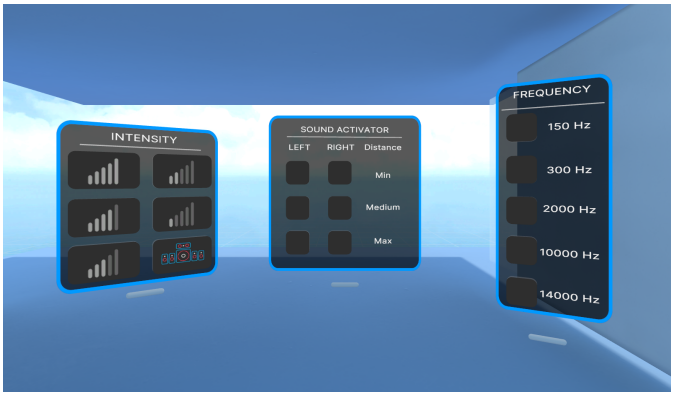


Figure 6.14: An example of the virtual environment of qualitative task 12.

the end, expresses a tolerability judgment to the device. As explained by Chang et al. [202], 20 min is enough to perceive feelings of sickness.

14. **Task 14: ergonomics and comfort in wearing**
    Rationale: Assess the ergonomics of head-mounted displays and controllers.

    Description: The user is asked to judge the ergonomics of wearing the head-mounted displays and controllers assessing the weight, the screen quality, the lens, the buttons, and the adjustments.

15. **Task 15: low-light environment sensibility**
    Rationale: Assess the head-mounted display performance in low-light conditions.

    Description: The user is asked to judge an ambient scene illuminated by four directional lights, positioned in the upper corners of the room, each with five different intensity levels ranging from 0 to 5 (Figure 6.15). The user can adjust the light intensities using a user interface (UI) panel. An overview of the virtual environment is shown in Figure 6.16.

For each quantitative task, to ensure statistically significant results and to mitigate erroneous data due to limited VR experience, each user is required to perform five trials.

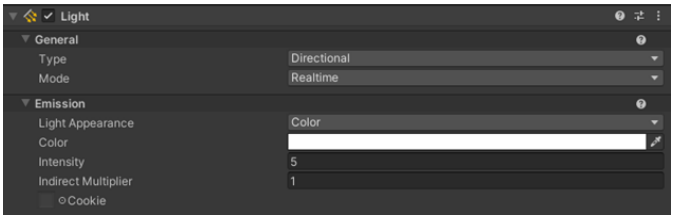


Figure 6.15: Properties of directional lights for the 5th level of intensity

Figure 6.16: An example of the virtual environment of qualitative task 15.

For each trial, a performance parameter (PP) is defined. To define a key performance parameter (KPP), a weighted average of the PP values obtained in each individual trial is computed using the following formula [215]:

$$KPP = \frac{\sum_{i=1}^{5} PP_i \cdot w_i}{\sum_{i=1}^{5} w_i} \tag{6.1}$$

where i is the trial, and w_i is the corresponding weight. The definition of the weights is linked to the experience of the user in the development of the task. For this reason, the first and last trials are considered to have low significance due to users being inexperienced and overly experienced in task development. Following this rationale, the weights have been distributed as follows:

- Trials 1 and 5: the weight is 0.1;
- Trials 2 and 4: the weight is 0.25;
- Trial 3: the weight is 0.3.

For the first seven quantitative tasks, temporal KPIs have been defined. The users need to click a button to start the timer, perform the task, and then click a second button to stop the timer. In these cases, the PPs consist of the user's time to complete the i-th trial. Once a reference time (t_{ref}) was defined as the mean value of the times for all users' trials for the j-th task, and the KPP was calculated with Equation 6.1, each KPI was defined with the following formula:

$$KPI_j = \begin{cases} 1 & & KPP_j < \frac{t_{\text{ref}}}{2} \\ \frac{3}{2} - \frac{KPP_j}{t_{\text{ref}}} & \frac{t_{\text{ref}}}{2} \leq & KPP_j \leq \frac{3}{2} t_{\text{ref}} \\ 0 & & KPP_j > \frac{3}{2} t_{\text{ref}} \end{cases} \tag{6.2}$$

The threshold values were selected to map performance into an intuitive normalized scale, where completion within half the reference time was considered optimal and performance exceeding 150% of the reference time was considered unacceptable for industrial usage. According to this rationale, outlier behaviors were addressed within the KPI computation itself: for any performance time exceeding 150% of the task's reference value, the KPI was automatically set to zero in accordance with Equation 6.2. This approach ensured that outliers did not affect the overall evaluation without requiring the exclusion of any participants.

Figure 6.17 shows the KPI assessment according to Equation 6.2. Note that a task is fully achieved with the maximum score when the time required for its completion is lower than half of its reference value.

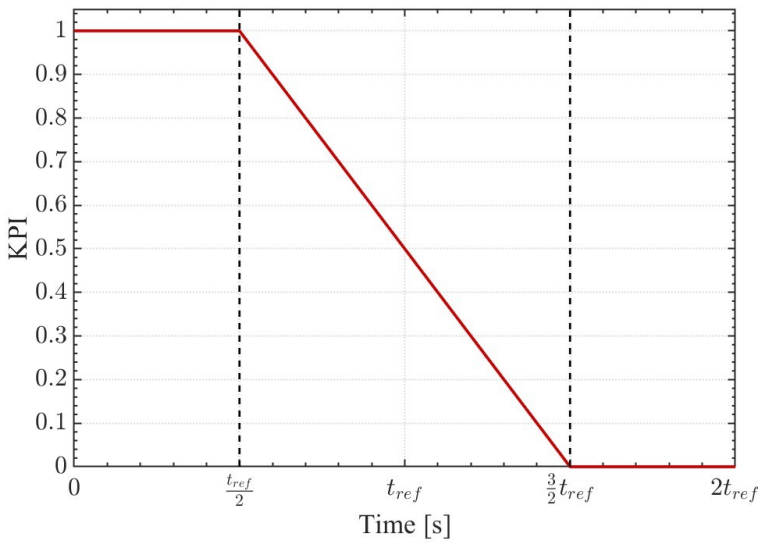


Figure 6.17: KPI assessment function for the first seven quantitative tasks.

Reference values, i.e., t_{ref}, are computed as the average value among all the trials and all the users. Table 6.3 reports the computed values of t_{ref} for the set of sixty users that have tested the experiences.

For the last three quantitative tasks (Tasks 8, 9, and 10), the PPs consist of the score of the respective task: in the eighth task, the PP is set equal to the target score result; for the ninth one, the PP is determined by the number of correct letters that users have read; meanwhile, for the tenth one, the PP is based on the number of balls the user has caught with the hollow box. The KPPs are equal to Equation 6.1. Meanwhile, the KPI

is derived from the normalized results of the KPPs: Equations 6.3, 6.4, 6.5 represent the KPIs for the eighth (T8), ninth (T9), and tenth tasks (T10), respectively.

$$KPI_{T8} = \frac{KPP_{T8}}{300} \tag{6.3}$$

$$KPI_{T9} = \frac{KPP_{T9}}{21} \tag{6.4}$$

$$KPI_{T10} = \frac{KPP_{T10}}{5} \tag{6.5}$$

Task	t_{ref}**[s]**
1	9.75
2	9.63
3	7.84
4	11.23
5	6.86
6	8.10
7	7.05

Table 6.3: Reference time for the first seven quantitative tasks.

In qualitative tasks, a 5-point Likert scale was used (1 = Very poor, 2 = Poor, 3 = Sufficient, 4 = Good, 5 = Very good). The user was asked to provide a personal judgment for each task by selecting the corresponding numerical score. The normalized KPI was then computed as follows:

$$KPI = \frac{score}{5} \tag{6.6}$$

This simple scale was preferred over comprehensive multi-item standardized questionnaires (such as the SUS, NASA-TLX, or SSQ) because each subjective task addressed a single, well-defined perceptual or ergonomic attribute (visual sharpness, sound clarity, motion sickness, ergonomics, low-light sensitivity) rather than broad, multidimensional latent constructs. From a methodological standpoint, multi-item scales were conceptually unnecessary and theoretically redundant for such atomistic evaluations [216, 217]. Furthermore, introducing extensive multi-item questionnaires across our large-scale protocol (60 users, 5 devices, 15 tasks) would have forced each participant to evaluate thousands of distinct query inputs. In human–computer interaction workflows, such an extreme response burden induces severe user fatigue, leading to automated straight-lining and significantly compromising data integrity through response bias [218]. Because each subjective hardware attribute is measured via an independent, single-item metric, multi-item psychometric internal consistency indicators such as Cronbach's α, McDonald's ω, or exploratory factor analysis (EFA) are mathematically inapplicable. Instead, measurement reliability and cross-study reproducibility are strictly ensured by explicit, discrete Likert anchors, identical environmental baselines, and a large sample size ($N = 60$), which is fully sufficient for the benchmarking purposes of this study [214].

6.1.2 Usability analysis results

The data collected during the tests offer an insightful perspective on the devices involved in the usability analysis, highlighting their strengths and weaknesses across various tasks. Numerical results from the tests conducted on 60 users are summarized in the following figures, showing, for each device, the mean value and standard deviation of users' KPIs for the corresponding task. Furthermore, the KPIs were used as dependent variables in a repeated-measures ANOVA to assess the statistical significance of the differences observed across tasks and devices.

Task	**p**	**GG-ε**	**Correction**
1	0.071	0.883	None
2	<0.001	0.744	Greenhouse–Geiser
3	<0.001	0.635	Greenhouse–Geiser
4	0.020	0.863	Huynh–Feldt
5	0.030	0.859	Huynh–Feldt
6	<0.001	0.583	Greenhouse–Geiser
7	0.037	0.866	Huynh–Feldt
8	<0.001	0.824	Huynh–Feldt
9	<0.001	0.507	Greenhouse–Geiser
10	<0.001	0.796	Huynh–Feldt
11	<0.001	0.801	Huynh–Feldt
12	<0.001	0.773	Huynh–Feldt
13	<0.001	0.618	Greenhouse–Geiser
14	<0.001	0.770	Huynh–Feldt
15	0.006	0.845	Huynh–Feldt

Table 6.4: Results of Mauchly's test for the sphericity assumption and the corresponding corrections.

Task	**p-Value**	η^2_G	**Task**	**p-Value**	η^2_G	**Task**	**p-Value**	η^2_G
1	<0.001	0.296	6	<0.001	0.603	11	<0.001	0.149
2	<0.001	0.156	7	<0.001	0.253	12	<0.001	0.338
3	<0.001	0.223	8	<0.001	0.271	13	<0.001	0.406
4	<0.001	0.456	9	<0.001	0.131	14	<0.001	0.467
5	<0.001	0.238	10	<0.001	0.452	15	<0.001	0.129

Table 6.5: Results of RM-ANOVA: p-values and generalized η^2.

Table 6.4 reports the results of Mauchly's test for each task, which was conducted to verify the assumption of sphericity. For all tasks except Task 1, the p-values are below the conventional threshold of 0.05, indicating that the sphericity assumption is violated. Therefore, correction methods were applied to adjust the degrees of freedom accordingly. Specifically, for Tasks 2, 3, 6, 9, and 13, the Greenhouse-Geisser correction was used, as the GG-ε was below the threshold of 0.75. For Tasks 4, 5, 7, 8, 10, 11, 12, 14, and 15, the Huynh-Feldt correction was applied, since the GG-ε exceeded the 0.75 threshold. Once the appropriate corrections were applied, the p-values and generalized η^2 were calculated for each task (Table 6.5). All p-values are below the threshold of 0.05, indicating that the effects observed for all tasks are statistically significant. The generalized η^2_G values provide an indication of effect size, i.e., how much of the variance in the dependent vari-

able (KPIs) can be attributed to the headset. More specifically, high η_G^2 values suggest that the choice of device plays a considerable role in performance for that task, whereas low values suggest a smaller impact. For instance, Tasks 4 ($\eta_G^2 = 0.456$), 6 ($\eta_G^2 = 0.603$), 10 ($\eta_G^2 = 0.452$), 13 ($\eta_G^2 = 0.406$), and 14 ($\eta_G^2 = 0.467$) exhibit large effect sizes. This indicates that, for these tasks, performance varies substantially depending on the headset used, making them particularly informative. Finally, the statistical procedure was completed with a post hoc analysis using Tukey's test. While the 15 tasks assess independent usability dimensions (meaning that cross-task alpha correction is not strictly mandatory under independent operational constructs), the robustness of the main effects against Type I error inflation was formally verified. For all 15 primary RM-ANOVAs, the original omnibus F-tests yielded an extremely high separation with $p < 0.001$. Consequently, even when applying the most conservative Bonferroni adjustment across all 15 tasks, which lowers the significance threshold to $\alpha_{\text{adj}} = 0.05/15 = 0.0033$, all 15 main macro-effects remain highly statistically significant ($p < 0.001$), leaving the scientific conclusions of the benchmark unchanged. Furthermore, a sensitivity check on the pairwise comparisons confirms that switching from Tukey's HSD to a Bonferroni adjustment yields identical results: highly significant pairs consistently retain $p < 0.001$, while non-significant pairs remain at $p = 1.000$. Within each individual task, the family-wise error rate (FWER) for the 10 pairwise comparisons was strictly controlled using Tukey's HSD post hoc test (see Tables (see Tables 6.7-6.21)).

In addition to numerical results, user comments and considerations were analyzed. From Figures 6.18, 6.20, and 6.22, it can be observed that the HTC VIVE Pro 2 device, followed by the two Meta headsets and the HTC VIVE Pro 1, appears to facilitate close-range object manipulation and handling. The manipulation of objects was performed using the trigger button. Despite the controllers of Meta and HTC VIVE XR Elite appearing more ergonomic and lighter compared to those of Pro 1 and Pro 2 devices, the majority of users found greater practicality in the latter, particularly due to more consistent input feedback. Furthermore, the controllers of Meta devices are lighter than those of the XR Elite. However, users noted in the Meta devices, particularly the Meta Quest 3, an impulsive release of objects, producing an unexpected bounce upon release. These bounces consist of an initial object velocity not congruent with the user's hand motion, altering the subsequent physical simulation. From Tables 6.7, 6.9, and 6.11, supported by Table 6.6, it is possible to identify statistically significant comparisons between headsets ($p < 0.05$). The HTC VIVE XR Elite proves to be statistically less suitable for close-range object manipulation compared to the other devices analyzed. Conversely, the HTC VIVE Pro 2 appears, on average, to be the most statistically suitable headset.

Task 8 (Figure 6.25, Table 6.14) provided deeper insights into release dynamics: despite the HTC VIVE XR Elite achieving higher scores in trajectory linearity due to its external tracking system, users perceived greater physical realism in object behavior with the HTC VIVE Pro 1 and Pro 2, suggesting that simulation fidelity is not solely dictated by tracking accuracy but also by integration with the physics engine. However, a marked improvement in release physics was observed from HTC VIVE Pro 1 to Pro 2.

Figures 6.19 and 6.21 and Tables 6.8 and 6.10 show tasks related to handling and movement of objects far from the user, where manipulation occurs through pointing rays

originating from the controllers. According to the test results, especially Task 4, the rays cast by the HTC VIVE Pro 2 were more accurate and stable than those from the other devices, facilitating more precise movements. This supports the observation that Tasks 3 and 4, which involve high-precision assembly operations, exhibited a clear performance advantage of the VIVE Pro 2 over the XR Elite.

Regarding the grip button, Task 6 (Figure 6.23, Table 6.12) reveals a significant performance gap between the HTC VIVE XR Elite and the other devices. An issue was identified concerning unintended double inputs from the grip button, often leading to errors when entering the six-number combination and causing prolonged task durations. This corroborates the decline in performance observed in Task 6, where users directly attributed delays to these interaction problems.

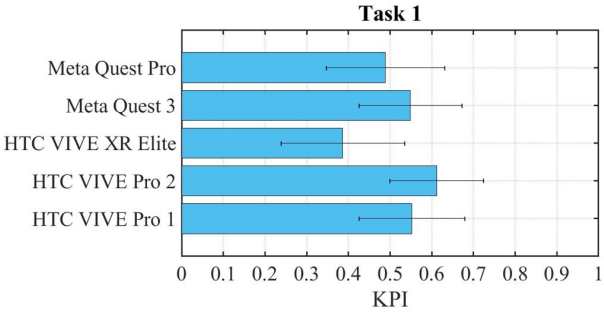
Task 7 involves a teleportation sequence using the thumbstick for Meta devices and HTC VIVE XR Elite, and the trackpad button for HTC VIVE Pro 1 and Pro 2 (Figure 6.24, Table 6.13). Users preferred the trackpad command for its faster recognition and execution, though they reported occasional unintended activations. This aligns with the limited performance differences in teleport-based locomotion tasks, where only marginal gaps were observed across headsets.

Task 9 (Figure 6.26, Table 6.15) reports results related to visual quality. All devices offered excellent rendering for the nearest panels, but differences were observed on the third distant panel. Based on both objective data and subjective feedback, the Meta Quest 3 and Meta Quest Pro delivered the most enjoyable visual quality, followed by the HTC VIVE Pro 2. However, the p-values between HTC VIVE Pro 2 and the two Meta devices are higher than 0.05, meaning these differences are not statistically significant. These results align with the low variance reported in Task 9, confirming the robustness of visual performance. The blurriness perceived at the top of the field of view in HTC VIVE Pro 1 is likely linked to its optical system, while the XR Elite showed a less comfortable visual experience despite its resolution, likely due to design limitations.

Task 10 (Figure 6.27, Table 6.16) evaluated spatial perception. The XR Elite was rated as providing a suboptimal perception experience: during the ball-falling sequence, users were required to quickly move their heads to follow the motion, and some reported perceiving a slight elliptical distortion of vertical space. These findings are consistent with those of Task 10, where the XR Elite ranked lower than the VIVE Pro 2, VIVE Pro 1, and Meta Quest Pro. In contrast, the Meta Quest 3 was reported to offer smoother scene transitions and improved visual comfort, enhancing spatial awareness.

Full name	**Acronym**
HTC VIVE Pro 1	VP1
HTC VIVE Pro 2	VP2
VIVE XR Elite	XR-E
Meta Quest 3	MQ3
Meta Quest Pro	MQ-Pro

Table 6.6: List of VR devices and their acronyms used throughout the tables.

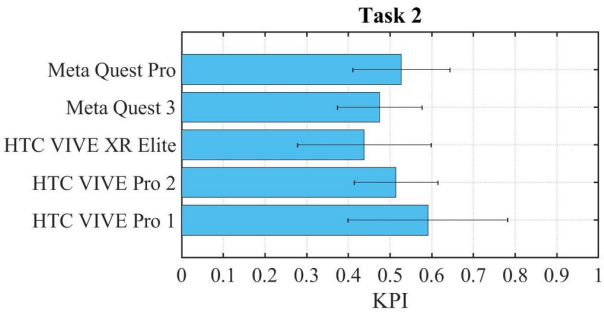


Average	Standard Deviation
0.488	0.142
0.549	0.124
0.386	0.148
0.612	0.113
0.551	0.127

Figure 6.18: Mean KPI and standard deviation for Task 1 (near-field manipulation—pick and place). For pairwise statistical comparisons, see Table 6.7.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	0.206	< 0.001	< 0.001	0.15
VP2	0.970	0.075	< 0.001	-
XR-E	< 0.001	0.144	-	
MQ3	0.055	-		

Table 6.7: Task 1 – Near field manipulation: pick and place. Tukey's post-hoc pairwise comparisons.



Average	Standard Deviation
0.527	0.116
0.475	0.102
0.438	0.160
0.514	0.100
0.591	0.192

Figure 6.19: Mean KPI and standard deviation for Task 2 (far-field manipulation—pick and place). For pairwise statistical comparisons, see Table 6.8.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	0.001	0.443	< 0.001	0.531
VP2	< 0.001	0.010	< 0.001	-
XR-E	0.003	< 0.001	-	
MQ3	0.073	-		

Table 6.8: Task 2 – Far field manipulation: pick and place. Tukey's post-hoc pairwise comparisons.

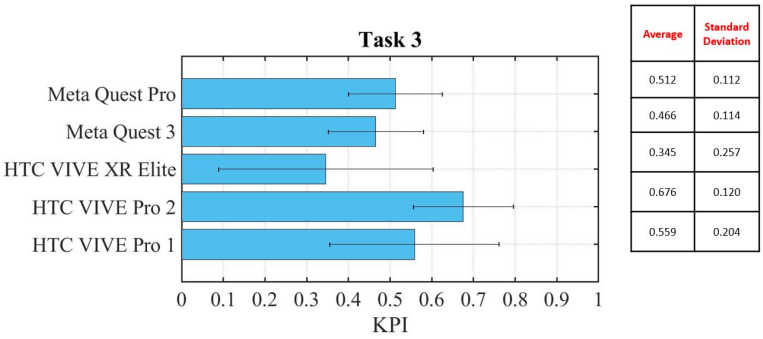


Figure 6.20: Mean KPI and standard deviation for Task 3 (near-field manipulation—pick, rotate and insert). For pairwise statistical comparisons, see Table 6.9.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	0.454	0.029	0.009	< 0.001
VP2	< 0.001	< 0.001	< 0.001	-
XR-E	0.058	0.677	-	
MQ3	0.236	-		

Table 6.9: Task 3 – Near field manipulation: pick, rotate and insert. Tukey's post-hoc pairwise comparisons.

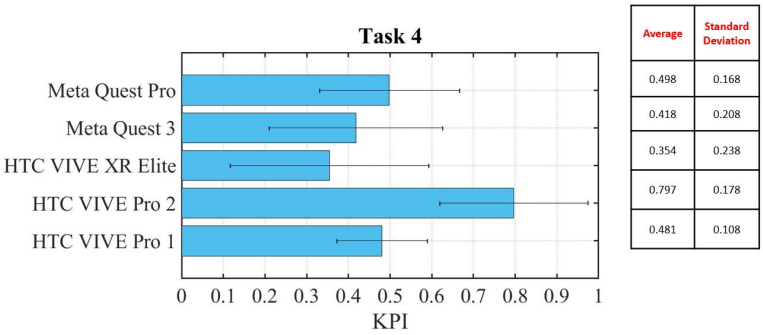


Figure 6.21: Mean KPI and standard deviation for Task 4 (far-field manipulation—pick, rotate and insert). For pairwise statistical comparisons, see Table 6.10.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	0.339	0.880	< 0.001	< 0.001
VP2	< 0.001	< 0.001	< 0.001	-
XR-E	< 0.001	0.013	-	
MQ3	0.160	-		

Table 6.10: Task 4 – Far field manipulation: pick, rotate and insert. Tukey's post-hoc pairwise comparisons.

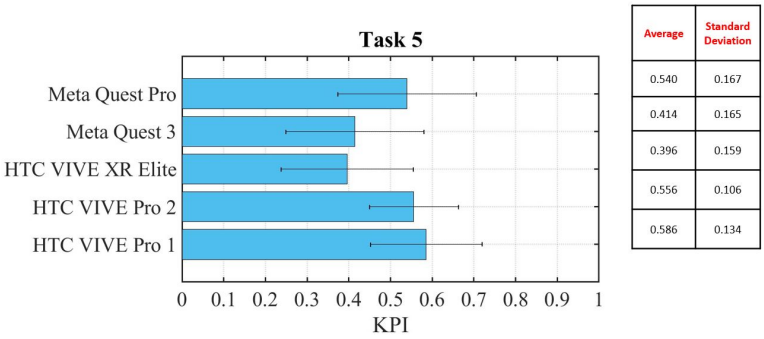


Figure 6.22: Mean KPI and standard deviation for Task 5 (two-hand dynamics). For pairwise statistical comparisons, see Table6.11.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	0.213	< 0.001	< 0.001	0.175
VP2	0.974	< 0.001	< 0.001	-
XR-E	< 0.001	0.874	-	
MQ3	< 0.001	-		

Table 6.11: Task 5 – Two hands dynamics. Tukey's post-hoc pairwise comparisons.

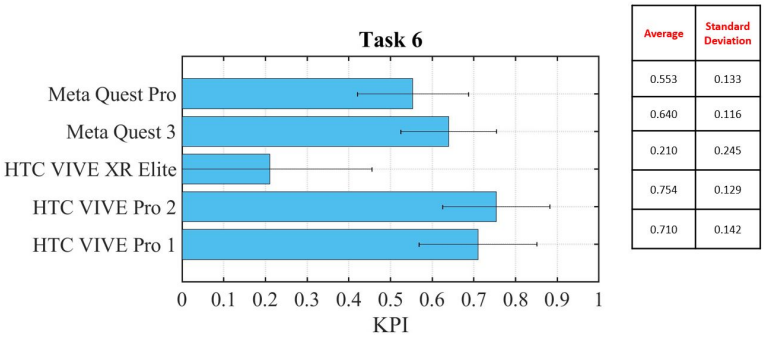


Figure 6.23: Mean KPI and standard deviation for Task 6 (button interaction). For pairwise statistical comparisons, see Table 6.12.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	< 0.001	< 0.001	< 0.001	0.298
VP2	< 0.001	< 0.001	< 0.001	-
XR-E	< 0.001	< 0.001	-	
MQ3	0.003	-		

Table 6.12: Task 6 – Button interaction. Tukey's post-hoc pairwise comparisons.

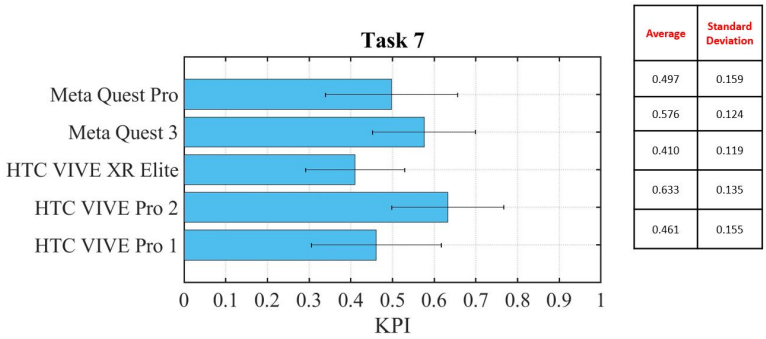


Figure 6.24: Mean KPI and standard deviation for Task 7 (teleporting). For pairwise statistical comparisons, see Table 6.13.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	0.672	< 0.001	0.120	< 0.001
VP2	< 0.001	0.142	< 0.001	-
XR-E	0.006	< 0.001	-	
MQ3	0.024	-		

Table 6.13: Task 7 – Teleporting. Tukey's post-hoc pairwise comparisons.

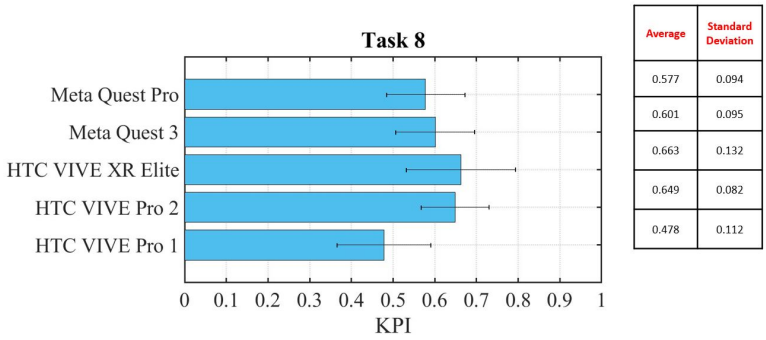


Figure 6.25: Mean KPI and standard deviation for Task 8 (dynamics—pick and throw). For pairwise statistical comparisons, see Table 6.14.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	< 0.001	< 0.001	< 0.001	< 0.001
VP2	0.004	0.002	0.928	-
XR-E	0.021	0.009	-	
MQ3	0.998	-		

Table 6.14: Task 8 – Dynamics: pick and throw. Tukey's post-hoc pairwise comparisons.

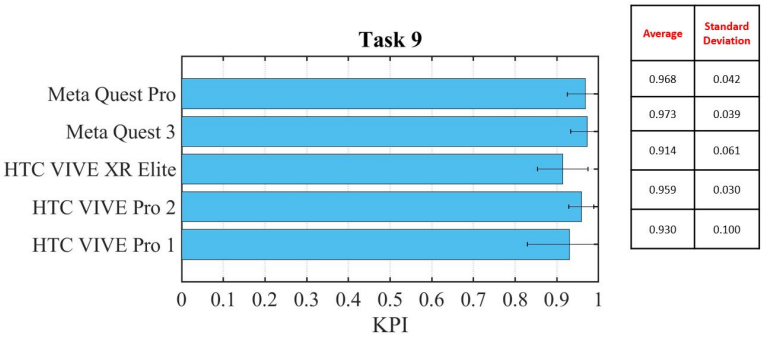


Figure 6.26: Mean KPI and standard deviation for Task 9 (reading canvas at different distances). For pairwise statistical comparisons, see Table 6.15.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	0.012	0.054	0.416	0.180
VP2	0.358	0.236	< 0.001	-
XR-E	< 0.001	< 0.001	-	
MQ3	0.980	-		

Table 6.15: Task 9 – Reading canvas at different distances. Tukey's post-hoc pairwise comparisons.

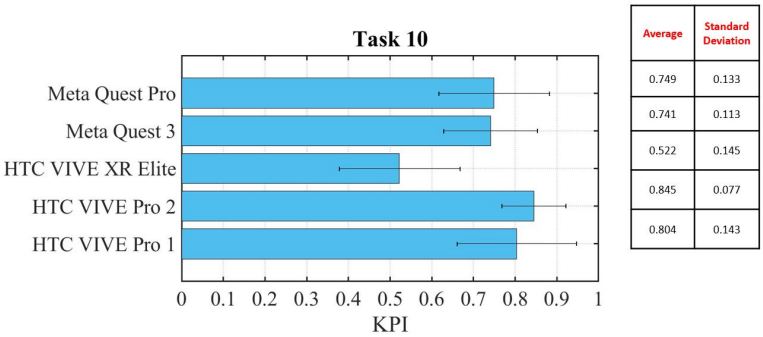


Figure 6.27: Mean KPI and standard deviation for Task 10 (interactive dynamics). For pairwise statistical comparisons, see Table 6.16.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	0.021	0.092	< 0.001	0.232
VP2	< 0.001	< 0.001	< 0.001	-
XR-E	< 0.001	< 0.001	-	
MQ3	0.994	-		

Table 6.16: Task 10 – Interactive dynamics (catching falling spheres). Tukey's post-hoc pairwise comparisons.

The qualitative assessment from Task 11 (Figure 6.28, Table 6.17) confirmed a clear user preference for the HTC VIVE Pro 2 in high-resolution inspection tasks, supported

by relatively low data dispersion. This observation reinforces its utility in precision-dependent virtual assembly environments.

Regarding device weight and comfort (Task 12, Figure 6.29, Table 6.18), all users consistently reported a clear advantage for the Meta Quest Pro and Meta Quest 3, followed by the XR Elite. The HTC VIVE Pro 1 and Pro 2 were perceived as significantly heavier and more cumbersome, especially during longer sessions. This perception matches the increase in reported fatigue and discomfort over time observed during the experimental protocol. Despite the visual superiority of the VIVE Pro 2 in high-resolution tasks, its headset structure appears to compromise extended usability, which may limit its adoption in immersive industrial simulations requiring prolonged engagement.

In Task 13 (Figure 6.30, Table 6.19), which evaluated interaction with the surrounding environment and the ability to identify scene boundaries, the Meta Quest Pro and Quest 3 demonstrated excellent performance, thanks to the integrated passthrough system and ergonomic design, which facilitated quick and natural physical orientation. The XR Elite, despite having a passthrough mode, showed limitations due to its reduced field of view and slight latency in scene rendering, especially during rapid head movements. This was consistent with subjective user reports, which expressed a preference for Meta devices in terms of contextual awareness and safety during transitions between virtual and physical environments.

Task 14 (Figure 6.31, Table 6.20) concerned the usability and precision of menu interaction using ray pointers. The results showed strong performance by the HTC VIVE Pro 2 and the Meta Quest Pro, confirming their superior pointing stability observed earlier in Tasks 2 and 4. In contrast, the XR Elite exhibited occasional jitter and pointer drift, particularly when menus were positioned toward the edges of the user's field of view. This reduced accuracy was reflected in slower completion times and increased user frustration reported during the test.

In the final task, Task 15 (Figure 6.32, Table 6.21), which involved the use of the thumbstick or trackpad for interacting with a virtual control panel, differences in device performance were less pronounced. However, some users noted that the trackpad on the HTC VIVE Pro 1 and Pro 2 provided more precise feedback compared to the thumbsticks of the Meta devices, especially when fine directional input was required. Nonetheless, overall ratings suggest that both input systems were acceptable for such interactions, with preferences largely influenced by familiarity and prior VR experience.

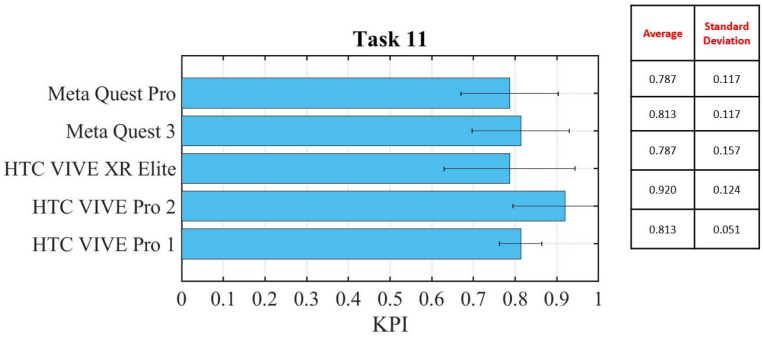


Figure 6.28: Mean KPI and standard deviation for Task 11 (high-resolution model inspection). For pairwise statistical comparisons, see Table 6.17.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	0.004	0.100	0.010	0.024
VP2	< 0.001	< 0.001	< 0.001	-
XR-E	1.000	0.745	-	
MQ3	0.254	-		

Table 6.17: Task 11 – Hi-res model inspection. Tukey's post-hoc pairwise comparisons.

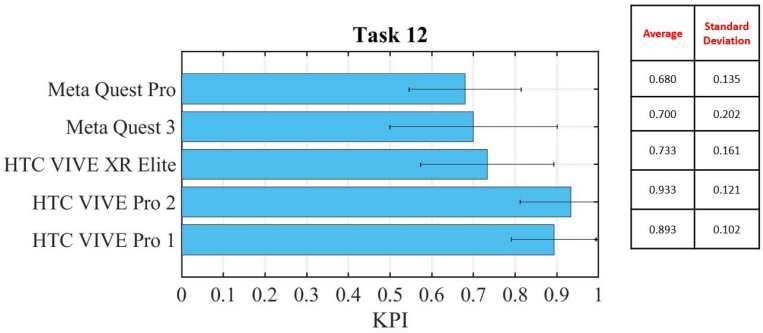
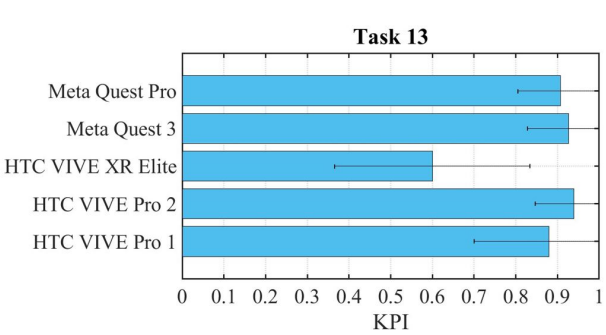


Figure 6.29: Mean KPI and standard deviation for Task 12 (sound listening). For pairwise statistical comparisons, see Table 6.18.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	< 0.001	< 0.001	< 0.001	0.143
VP2	< 0.001	< 0.001	< 0.001	-
XR-E	0.343	0.986	-	
MQ3	0.736	-		

Table 6.18: Task 12 – Sound listening. Tukey's post-hoc pairwise comparisons.

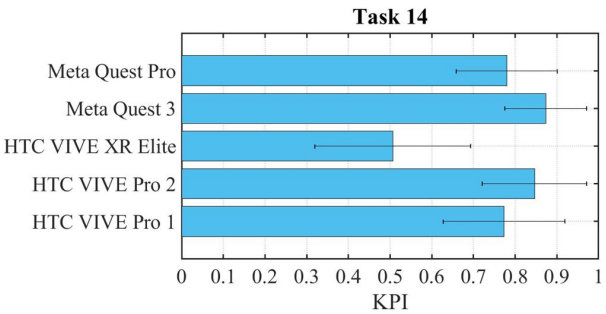


Average	Standard Deviation
0.907	0.102
0.927	0.098
0.600	0.235
0.940	0.093
0.880	0.179

Figure 6.30: Mean KPI and standard deviation for Task 13 (mid-exposure tolerability). For pairwise statistical comparisons, see Table 6.19.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	0.542	1.000	< 0.001	0.101
VP2	0.623	0.145	< 0.001	-
XR-E	< 0.001	< 0.001	-	
MQ3	0.512	-		

Table 6.19: Task 13 – Mid exposure tolerability Tukey's post-hoc pairwise comparisons.



Average	Standard Deviation
0.780	0.122
0.873	0.098
0.507	0.187
0.847	0.125
0.773	0.146

Figure 6.31: Mean KPI and standard deviation for Task 14 (ergonomics and comfort in wearing). For pairwise statistical comparisons, see Table 6.20.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	0.770	< 0.001	< 0.001	< 0.001
VP2	0.036	0.821	< 0.001	-
XR-E	< 0.001	< 0.001	-	
MQ3	< 0.001	-		

Table 6.20: Task 14 – Ergonomics and comfort in wearing. Tukey's post-hoc pairwise comparisons.

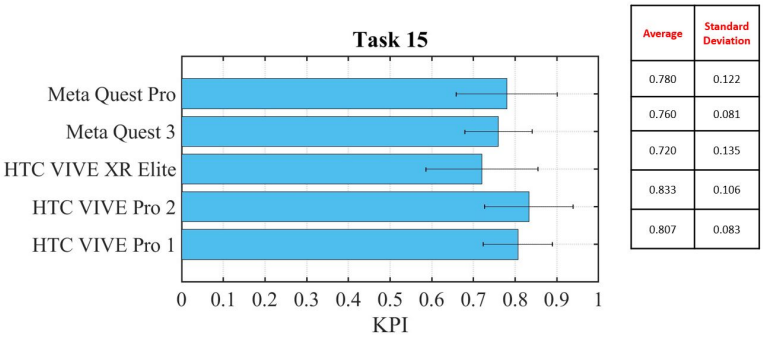


Figure 6.32: Mean KPI and standard deviation for Task 15 (low-light environment sensibility). For pairwise statistical comparisons, see Table 6.21.

	MQ-Pro	**MQ3**	**XR-E**	**VP2**
VP1	0.172	0.001	< 0.001	1.000
VP2	0.006	< 0.001	< 0.001	-
XR-E	0.070	0.308	-	
MQ3	0.762	-		

Table 6.21: Task 15 – Low light environment sensibility. Tukey's post-hoc pairwise comparisons.

	HTC VIVE Pro 1	HTC VIVE Pro 2	HTC VIVE XR Elite	Meta Quest 3	Meta Quest Pro
Task 1: Near field manipulation – pick and place	0.583	0.611	0.386	0.548	0.488
Task 2: Far field manipulation – pick and place	0.591	0.514	0.414	0.472	0.527
Task 3: Near field manipulation – pick, rotate and insert	0.559	0.691	0.407	0.465	0.512
Task 4: Far field manipulation – pick, rotate and insert	0.445	0.805	0.294	0.418	0.498
Task 5: Two hands Dynamics	0.598	0.556	0.396	0.414	0.540
Task 6: Button interaction	0.710	0.754	0.210	0.640	0.553
Task 7: Teleporting	0.461	0.633	0.410	0.576	0.497
Task 8: Dynamics – Pick and throw	0.482	0.649	0.663	0.601	0.596
Task 9: Reading canvas at different distances	0.930	0.959	0.914	0.973	0.968
Task 10: Interactive Dynamics	0.804	0.845	0.522	0.741	0.749
Task 11: Hi-res model inspection	0.857	0.920	0.787	0.813	0.787
Task 12: Sound listening	0.893	0.933	0.733	0.717	0.680
Task 13: Mid exposure tolerability	0.887	0.940	0.600	0.887	0.917
Task 14: Ergonomics and comfort in wearing	0.747	0.847	0.507	0.873	0.780
Task 15: Low light environment sensibility	0.830	0.833	0.720	0.760	0.780

Table 6.22: KPI mean values for each device and each task.

As shown by the results, as an overall assessment, the HTC VIVE Pro 2 device appears to be the most practical for virtual assembly applications, thanks to the ergonomics of the trigger button, excellent spatial perception, and accurate tracking and manipulation of both close and distant objects. The HTC VIVE Pro 1 and Meta Quest Pro headsets yield good results regarding object manipulation, slightly less so for the Meta Quest 3 and HTC VIVE XR Elite. Concerning the grip button (side button), the HTC VIVE XR Elite seems to present issues with reliable release. Headsets equipped with thumbsticks (HTC VIVE XR Elite, Meta Quest 3, and Meta Quest Pro) appear to facilitate user control in terms of stability, while headsets with trackpads (HTC VIVE Pro 1 and Pro 2) seem to speed up execution, at the risk of reduced control due to unintended touch inputs. Regarding visual tasks, Meta headsets and the HTC VIVE Pro 2 support more effective virtual inspection of the system, while the HTC VIVE XR Elite achieves slightly lower scores. The VIVE Pro devices can be used in applications where audio feedback is relevant, in contrast to the Meta Quest 3. However, the Meta Quest Pro does not appear

to be the best choice for applications involving high-frequency sounds. For long-term use, the Meta Quest Pro and Meta Quest 3 provide the best trade-off between physical and visual comfort. The physical comfort and sense of ease with the HTC VIVE XR Elite are highly user-dependent: experienced VR users without glasses may find this headset suitable for long-term use, whereas new users report lower comfort levels. The HTC VIVE Pro 1 and Pro 2 devices are better suited for limited exposure times due to their higher weight. The mean KPI values for each device and task are reported in Table 6.22. Despite highlighting average KPI values, the relatively wide standard deviations must be taken into account. This indicates that, despite the objective evaluation effort, certain aspects of performance are inherently user-dependent.

6.2 Case study: industrial virtual assembly

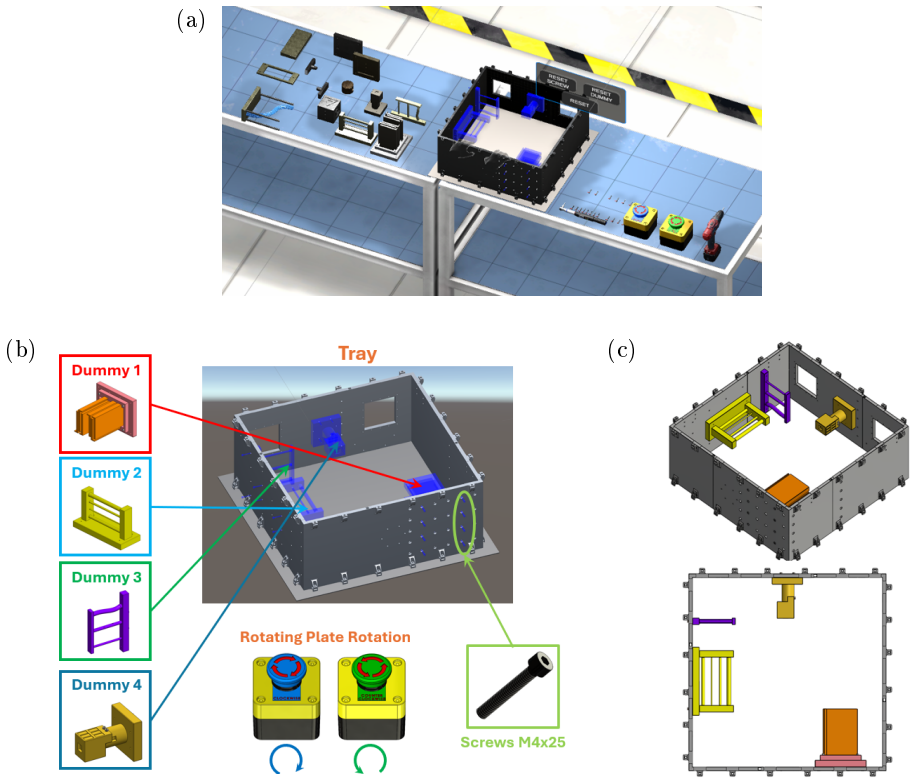


Figure 6.33: (b) Assembly of the Dummies on the Tray using M4×25 screws, (c) fully assembled Tray module, and (a) virtual assembly workstation.

Based on the literature presented in Section 6, a virtual reality assembly station was designed and implemented for training specialised personnel in the assembly of components on a dedicated module (Figure 6.33a). This module is referred to as the *Tray*, while the individual components to be assembled are denoted as *Dummy* elements (Figure 6.33b). The assembly process of the *Dummies* on the *Tray* consists of the correct positioning of the components followed by their fastening using M4×25 screws in the

corresponding slots (Figure 6.33c). To provide a more realistic haptic feedback related to tool weight during interaction, two physical tools were integrated into the virtual scene via the HTC VIVE Wrist Tracker: a screwdriver (Figure 6.34a) and a torque wrench (Figure 6.34b). On this basis, a dedicated framework for screw-driving operations was developed, applicable both to the screwdriver (Figures 6.34a) and to the torque wrench (Figures 6.34b).

Figure 6.34: (a) Real and virtual screwdriver, (b) real and virtual torque wrench.

The virtual environment was developed using the Unity3D engine (version 2022.3.49) and made compatible with the HTC VIVE XR Elite and Varjo XR-4 headsets. The *OpenXR* plugin was integrated to provide a unified, cross-platform access layer for device input and pose tracking. User interaction with virtual objects is enabled through hand tracking and custom gestures. To this end, the *XR Interaction Toolkit* (v2.6.4) was adopted to translate low-level input signals into high-level interaction events (e.g., object manipulation), while the *XR Hands* (v1.5.0) plugin was used to acquire and process hand poses and gestures, enabling controller-free interaction. Custom gesture definitions were implemented following the workflow described in Section 4.3.2. In this application, three primary gestures were introduced: a customized pinch grasp, an index-pointing gesture, and an open-hand gesture with the palm facing downward. The first gesture enables the selection and grasping of virtual objects, the second is used for interaction with user interfaces (UI), and the third is employed to activate the rotation of a turntable supporting the *Tray*, thereby facilitating assembly operations.

The customized pinch gesture (Figure 6.35) is defined according to the following configuration:

- Thumb: *Shape Type* set to *Base Curl*, with a target normalized value of 0.5.
- Index: *Shape Type* set to *Pinch*, with a target normalized value of 1.0.
- Middle, Ring, and Little fingers: *Shape Type* set to *Full Curl*, with a target normalized value of 0.2, an upper tolerance of 0.8, and a lower tolerance of 0.2, so as not to hinder gesture activation due to the position of the non-involved fingers.
- The *Hand Pose* is not explicitly defined, as the gesture must remain independent of hand orientation.

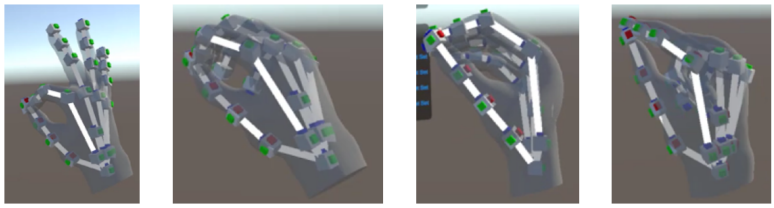


Figure 6.35: Possible hand configurations that trigger the customized pinch gesture.

The definition of the index-pointing gesture is based on the following configuration:

- Index finger: *Shape Type* set to *Full Curl*, with a target normalized value of 0 and a tolerance of 0.15.
- Other fingers: *Shape Type* set to *Full Curl*, with a target normalized value of 1 and a tolerance of 0.25.
- The *Hand Pose* is defined as follows: the hand reference axis is the *Fingers Extended Direction*, the alignment mode is *Aligns With*, and the reference direction is *Hand to Target*.

The definition of the palm-down open-hand gesture is based on the following configuration:

- All fingers: *Shape Type* set to *Full Curl*, with a target normalized value of 0 and a tolerance of 0.25.
- The *Hand Pose* is defined as follows: the hand reference axis is the *Palm Direction*, the alignment mode is *Opposite To*, and the reference direction is *Origin Up*.

All virtual objects within the simulation environment are required to obey the laws of motion implemented by the Unity physics engine. This includes gravitational effects, contact interactions, and inertial properties associated with the objects' mass distribution. In order for a *GameObject* to be governed by the physics engine, appropriate *Collider* components (Sections 4.1 and 4.2) and a *Rigidbody* component must be assigned. While colliders define the geometric representation used for collision detection, the *Rigidbody* enables the simulation of the object's dynamic behavior. Regarding mass properties, Unity automatically computes the inertia tensor based on the geometric discretization defined by the associated colliders of the *GameObject*. As a result, when the object geometry is not strictly convex or is not adequately discretized, the inertia properties estimated by the physics engine may significantly deviate from the true physical values, potentially degrading the accuracy of the dynamic simulation. To mitigate this limitation, the principal inertia tensor computed in the CAD environment is exported and subsequently transformed into the coordinate system of the corresponding *GameObject* in Unity. These values are then used to correctly configure the *Inertia Tensor Rotation* parameter of the *Rigidbody* component.

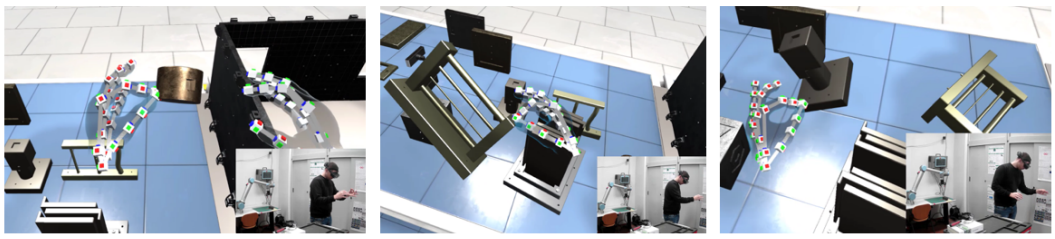


Figure 6.36: Example of Dummy manipulation.

Preserving the ordering of the principal moments of inertia and their corresponding principal directions as defined in the CAD software, the transformation matrix $\mathbf{R}$ is introduced, whose columns represent the components of the principal inertia directions expressed in the model reference frame. Subsequently, the transformation matrix $\mathbf{R}_T$ is computed by introducing the matrix $\mathbf{M}$, which is required to convert a right-handed coordinate system into a left-handed one, and the matrix $\mathbf{R}_U$, which describes the transformation between the principal inertia reference frame used in Unity (assumed to be right-handed) and that defined in the CAD environment. The latter reduces to the identity matrix $\mathbf{I}_d$ in the case where the CAD model has not been obtained through reverse engineering procedures. The matrix $\mathbf{R}_T$ is defined as follows:

$$\mathbf{R}_T = \mathbf{M}\,\mathbf{R}\,\mathbf{R}_U\,\mathbf{M}^T \tag{6.7}$$

where

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}. \tag{6.8}$$

Starting from the matrix $\mathbf{R}_T$, the Euler angles required to configure the *Inertia Tensor Rotation* parameter are obtained through the decomposition of the rotation matrix $\mathbf{R}_{213}$, defined in Equation 6.9. The Euler angles θ, φ, and ψ are then derived from the relations reported in Equations 6.10.

$$\mathbf{R}_{213} = \mathbf{R}_\varphi^y \mathbf{R}_\theta^x \mathbf{R}_\psi^z = \begin{bmatrix} c_\varphi c_\psi + s_\theta s_\varphi s_\psi & -c_\varphi s_\psi + s_\theta s_\varphi c_\psi & c_\theta s_\varphi \\ c_\theta s_\psi & c_\theta c_\psi & -s_\theta \\ -s_\varphi c_\psi + s_\theta c_\varphi s_\psi & s_\varphi s_\psi + s_\theta c_\varphi c_\psi & c_\theta c_\varphi \end{bmatrix} \tag{6.9}$$

$$\theta = -\arcsin\left(\mathbf{R}_T(2,3)\right) \tag{6.10a}$$

$$\varphi = \operatorname{arctan2}\left(\mathbf{R}_T(1,3)/\cos\theta, \mathbf{R}_T(3,3)/\cos\theta\right) \tag{6.10b}$$

$$\psi = \operatorname{arctan2}\left(\mathbf{R}_T(2,1)/\cos\theta, \mathbf{R}_T(2,2)/\cos\theta\right) \tag{6.10c}$$

The resulting angles are subsequently assigned via a C# script (Listing G.1) to the *Inertia Tensor Rotation* parameter of the *Rigidbody* component. Finally, the inertia tensor expressed in the Unity reference frame is obtained through the following transformation:

$$\mathbf{I}_i = \mathbf{R}_T\,\mathbf{I}_0\,\mathbf{R}_T^T, \tag{6.11}$$

where $\mathbf{I}_0$ denotes the principal inertia tensor and $\mathbf{I}_i$ represents the inertia tensor expressed in the Unity *GameObject* reference frame. It is observed that the resulting tensor $\mathbf{I}_i$ is equivalent to the one obtained in the CAD software, with the exception of the deviatoric components, whose sign is inverted due to the conversion between right-handed and left-handed coordinate systems. This approach allows the inertia properties of the original model to be preserved with high accuracy, ensuring a physically consistent dynamic simulation that closely reflects the real-world behavior of the object.

To enable the user to manipulate virtual objects such as *Dummies* and *Screws*, the corresponding *GameObjects* are equipped with the *XR Grab Interactable* and *XR General Grab Transformer* components, leveraging the functionality provided by the *XR Interaction Toolkit*. On top of this infrastructure, a custom framework has been developed to manage object grasping through hand tracking and gesture recognition.

The adopted software architecture follows an *event-driven* paradigm, in which the evolution of the interaction state is not determined through periodic polling of scene objects, but rather through events generated by the XR framework. In particular, the *hoverEntered* and *hoverExited* events of the *XR Grab Interactable* component respectively signal the entry and exit of the hand within an object's interaction range. Similarly, the *gesturePerformed* and *gestureEnded* events generated by the gesture recognition system indicate the activation and release of the custom grab gesture. This approach reduces computational overhead and decouples interaction logic from simulation logic, thereby improving the modularity and scalability of the framework.

The interaction process is structured into three main phases: *hover*, *grab*, and *release*.

During the *hover* phase, the system identifies virtual objects located in the immediate proximity of the hand through the colliders associated with interactive *GameObjects*. For each hand, a LIFO (*Last In, First Out*) data structure is maintained, containing all currently reachable objects. This structure allows the system to properly handle situations in which multiple objects are simultaneously selectable, by assigning priority to the most recently detected object.

The grabbing phase is triggered by the recognition of the custom pinch gesture. When the gesture is detected, the system checks the currently selected object and temporarily modifies its physical properties by disabling gravity and setting its *Rigidbody* to kinematic mode (*isKinematic = true*), i.e., excluding it from the physics simulation. This allows direct control of the object's pose without interference from the physics engine (Figure 6.36).

The objective of the grabbing phase is to rigidly constrain the pose of a virtual object to the kinematics of the user's hand, while preserving a constant relative configuration between the pinch point and the object itself. To formally describe this constraint, three reference frames are considered: $\mathcal{F}_W$, representing the global scene reference frame (*world frame*); $\mathcal{F}_H$, associated with the user's hand at the pinch point (*hand frame*); and $\mathcal{F}_O$, associated with the manipulated object (*object frame*). The fundamental idea is that, at the moment of grabbing, the relative configuration between the hand and the object is stored and subsequently used to update the object pose over time. Let t_0 denote the instant at which the grab gesture is activated. At this time, the global poses of both the hand and the object are acquired: ${}^W_H\mathbf{T}(t_0)$ and ${}^W_O\mathbf{T}(t_0)$, where each homogeneous

transformation matrix is defined as follows:

$$^{W}\mathbf{T} = \begin{bmatrix} ^{W}\mathbf{R} & ^{W}\mathbf{p} \\ \mathbf{0}^{T} & 1 \end{bmatrix}, \tag{6.12}$$

where $^{W}\mathbf{R}$ denotes the rotation matrix and $^{W}\mathbf{p}$ the position vector in the world frame. From these quantities, the relative transformation between the object and the hand is defined, expressed in the hand reference frame at the moment of grasping:

$$^{H}_{O}\mathbf{T}(t_0) = \left(^{W}_{H}\mathbf{T}(t_0)\right)^{-1} {}^{W}_{O}\mathbf{T}(t_0). \tag{6.13}$$

This relation describes the pose of the object expressed in the hand reference frame at the exact instant the grasp is performed, and therefore represents the kinematic offset that must be preserved throughout the interaction. By separating rotational and translational components, the following expressions are obtained:

$$^{H}_{O}\mathbf{p}(t_0) = \left(^{W}_{H}\mathbf{R}(t_0)\right)^{T} \left(^{W}_{O}\mathbf{p}(t_0) - {}^{W}_{H}\mathbf{p}(t_0)\right), \tag{6.14}$$

which represents the position of the object expressed in the hand's local reference frame at the moment of grasping.

$$^{H}_{O}\mathbf{R}(t_0) = \left(^{W}_{H}\mathbf{R}(t_0)\right)^{T} {}^{W}_{O}\mathbf{R}(t_0), \tag{6.15}$$

which describes the orientation of the object with respect to the hand at the moment of grasping. During the manipulation phase, the hand evolves over time with pose $^{W}\mathbf{T}_{H}(t)$ determined by tracking. The object is then kinematically constrained to the hand, maintaining the previously computed relative transformation constant:

$$^{W}_{O}\mathbf{T}(t) = {}^{W}_{H}\mathbf{T}(t)\, {}^{H}_{O}\mathbf{T}(t_0). \tag{6.16}$$

Expanding the relation, we obtain:

$$^{W}_{O}\mathbf{p}(t) = {}^{W}_{H}\mathbf{R}(t)\, {}^{H}_{O}\mathbf{p}(t_0) + {}^{W}_{H}\mathbf{p}(t), \tag{6.17a}$$

$$^{W}_{O}\mathbf{R}(t) = {}^{W}_{H}\mathbf{R}(t)\, {}^{H}_{O}\mathbf{R}(t_0). \tag{6.17b}$$

This formulation enforces that the object is not directly "attached" to the position of the hand, but rather that a constant rigid transformation between the two reference frames is preserved. In this way, the interaction is equivalent to the manipulation of a rigid body attached to the user's hand.

To improve the realism of the interaction, the system distinguishes between two different grasping conditions. If the pinch gripping point lies within the object, the relative position and rotation between the hand and the object are maintained. If the pinch gripping point lies outside the object, the closest point on the object's mesh to the gripping point is identified, and the relative position and rotation between the hand and the object are preserved accordingly (Listing G.2).

The release phase is triggered at the end of the grasp gesture. During this phase, the object's original physical properties are restored, re-enabling gravity and dynamic simulation if they were enabled prior to the grasp. At the same time, any residual linear

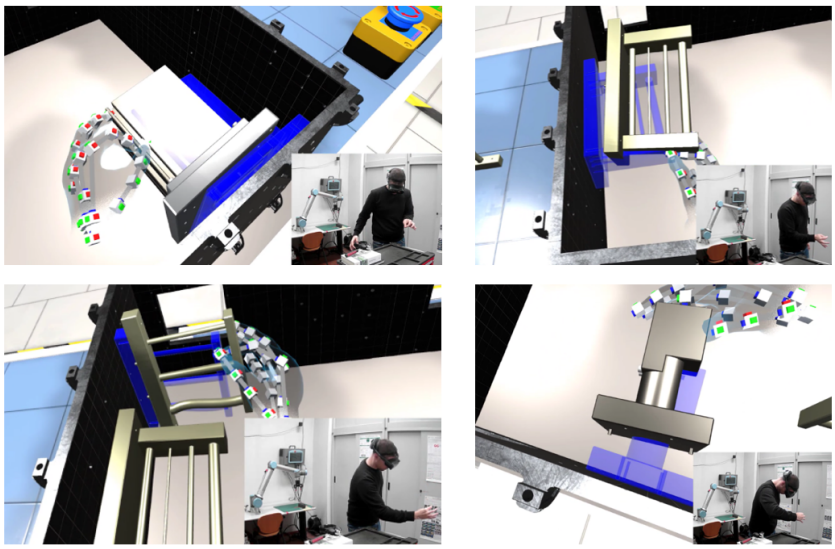


Figure 6.37: Example of alignment between Dummies and their corresponding Shadow objects.

and angular velocities are cleared to prevent unstable behavior due to the transition between kinematic control and physical simulation.

The entire system has been designed to support both hands of the user simultaneously, maintaining independent states for the right and left hand.

For the integration of real objects within the virtual environment, the HTC VIVE Wrist Tracker was used. The implementation of the tracker within a Windows-based application required the development of a dedicated software component for acquiring the device's raw data (Listing G.3). Subsequently, an *Input Map* was defined, i.e., an abstraction layer that converts the device's raw data into actions within the virtual scene. In particular, the tracker's position and rotation data are mapped respectively to the position and orientation of the corresponding virtual *GameObject*. Once the code was integrated and the tracker registered among the available inputs, a *Tracked Pose Driver* component was assigned to a Unity *GameObject*, which directly uses the tracker's position and orientation data to update the pose of the corresponding virtual object (GO tracker). A fundamental element for inducing a sense of "ownership" of the virtual tool is the precise alignment between the virtual model and the real device. To this end, the real tracker was placed at a point on the physical tool whose position relative to the tracker's reference frame is known. Subsequently, CAD models of the tools were created: for the screwdriver, a 3D scan using an ARTEC EVA was employed to obtain an accurate geometric reconstruction, while for the torque wrench a manual CAD model was developed. The CAD models were imported into Unity and made hierarchical children of the GO tracker, so that they rigidly inherit the tracker's rotational transformation. The position of the CAD model relative to the GO tracker was defined consistently with the position of the real tracker with respect to the physical tool, thus ensuring spatial registration between the real and virtual worlds.

Regarding the assembly procedure, a framework was developed to ensure correct alignment between the object to be assembled and the actual assembly pose. In the

case of the *Dummies*, for each component a corresponding virtual object called *Shadow* was defined. The *Shadow* represents the desired final configuration of the component and is geometrically identical to the real Dummy. Its pose with respect to the *Tray* therefore coincides with the pose that the component must assume once the assembly phase is correctly completed. The assembly procedure requires the user to grasp the Dummy and position it so as to align its position and orientation with those of the corresponding Shadow. The alignment is considered valid when the pose error between the Dummy and the Shadow falls below predefined tolerance thresholds. These tolerances compensate for the inevitable inaccuracies introduced by the hand tracking system and manual manipulation in the virtual environment. An example of the alignment process between Dummy and Shadow is shown in Figure 6.37.

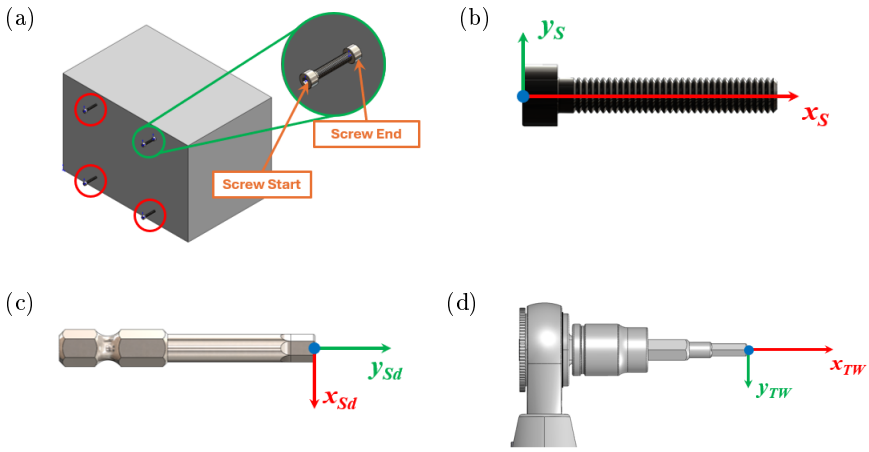


Figure 6.38: (a) Position of the *Screw Start* and *Screw End* with respect to a test surface, (b) reference frame of the screw, (c) reference frame of the screwdriver, and (d) reference frame of the torque wrench.

With regard to the *Screws*, the assembly process is more complex as it requires both the correct initial positioning of the screw and the simulation of the subsequent tightening operation. For each screw, two different Shadows have been defined. The first, named *Screw Start*, represents the configuration that the screw must assume immediately before the screwing operation. The second, named *Screw End*, represents instead the final configuration of the screw at the end of the tightening process. The two poses are shown in Figure 6.38a.

The management of alignments was developed by adapting the GAF/OAF methodology described in Section 4.3.2.1. In particular, the geometric and kinematic constraints associated with the assembly operations were translated into appropriate tolerance volumes used for alignment management. To this end, the reference frames associated with the screw, the screwdriver, and the torque wrench were defined (Figures 6.38b, 6.38c, and 6.38d).

The initial alignment between the screw and the corresponding *Screw Start* is verified through two geometric constraints (Figure 6.39a):

- a spherical tolerance centered at the origin of the Shadow reference frame, used to verify the coincidence of the origins of the two reference frames;

- a conical tolerance used to verify the alignment between the longitudinal axis of the screw and the axis of the *Screw Start*.

Only when both conditions are satisfied is the screw considered correctly positioned and it automatically aligns with the *Screw Start* configuration.

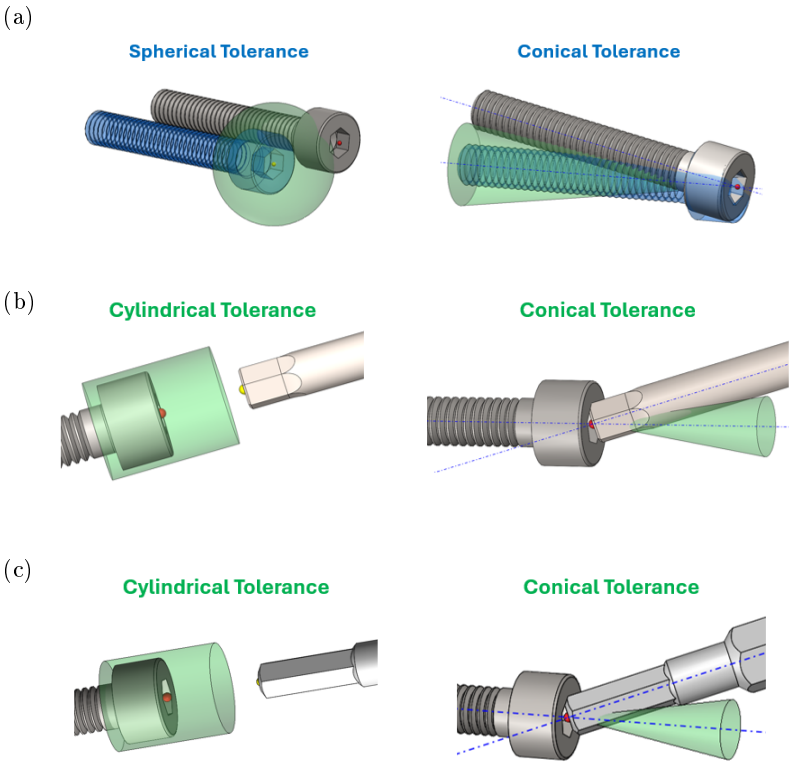


Figure 6.39: (a) Spherical tolerance (left) and conical tolerance (right) for the *Screw–Screw Start* alignment, (b) cylindrical tolerance (left) and conical tolerance (right) for the *Screwdriver–Screw* alignment, (c) cylindrical tolerance (left) and conical tolerance (right) for the *Torque Wrench–Screw* alignment.

The screwing operation requires the correct alignment between the screw and the tool being used. In both the screwdriver and torque wrench cases, the *Tool–Screw* alignment is verified using a cylindrical tolerance and a conical tolerance (Figures 6.39b and 6.39c). The cylindrical tolerance is used to verify the proximity between the origins of the tool and screw reference frames, while allowing small lateral displacements necessary during interaction. The conical tolerance instead checks the coaxiality between the tool axis and the screw axis.

The tightening procedure depends on the type of tool being used. In the case of the screwdriver, once alignment with the screw has been verified, the user can activate tool rotation via a button. The advancement of the screw is simulated by verifying that the force vector applied by the tool is aligned with the longitudinal axis of the screw. Referring to Figure 6.40b, screw advancement is simulated when the vector AB has the same direction as the screw's x-axis. The previously described cylindrical tolerance contributes to maintaining this alignment stable throughout the entire insertion phase.

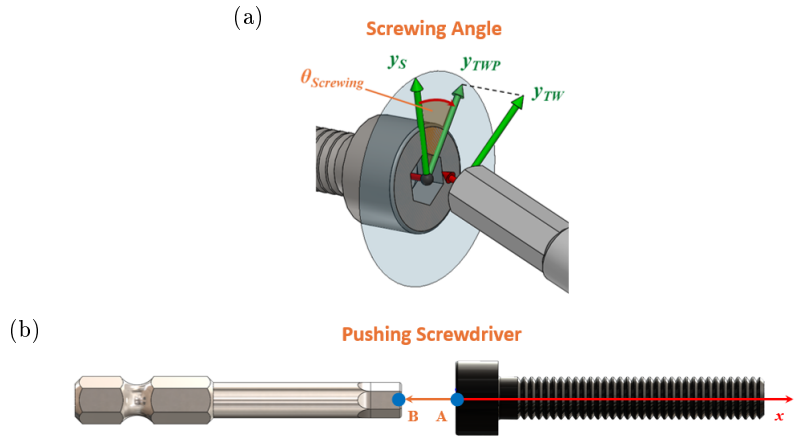


Figure 6.40: (a) Screwing conditions for tightening via torque wrench, (b) screwing conditions for tightening via screwdriver.

In the case of the torque wrench, in addition to the geometric alignment constraints, a behavior equivalent to a unidirectional bearing has been implemented. In particular, only rotations applied in the tightening direction produce screw advancement, while rotations in the opposite direction are ignored, replicating the real behavior of the tool (Figure 6.40a). The tightening angle is computed by projecting the y-axis of the torque wrench onto the yz plane of the screw reference frame and evaluating the angle formed with the y-axis of the screw itself, disregarding the initial configuration. This quantity is used to estimate the incremental rotation effectively transferred to the screw during the tightening procedure (Listing G.4).

To improve system usability, dedicated UI tools have also been developed, activated via an index pointing gesture. These tools allow the user to handle problematic situations such as geometric interpenetrations or difficulties in reaching certain virtual components. Two virtual buttons are also available, activated via a *palm down* gesture, which enable rotation of the Tray support plate in order to facilitate the assembly operations.

The assembly procedure implemented within the virtual environment is summarized below and graphically represented in the flowchart of Figure 6.41:

- The selected Dummy is grasped using the customized pinch gesture and manipulated within the virtual workspace.
- The Dummy is aligned with the corresponding Shadow by satisfying the prescribed positional and orientational tolerances. When necessary, the orientation of the Tray can be adjusted using the virtual buttons activated via the *palm-down* gesture.
- A screw is then grasped and positioned by aligning it with the corresponding *Screw Start* pose according to the defined assembly constraints.
- The user then grasps either the screwdriver or the torque wrench and aligns the selected tool with the mounted screw. The alignment constraints must be maintained throughout the screwing operation.

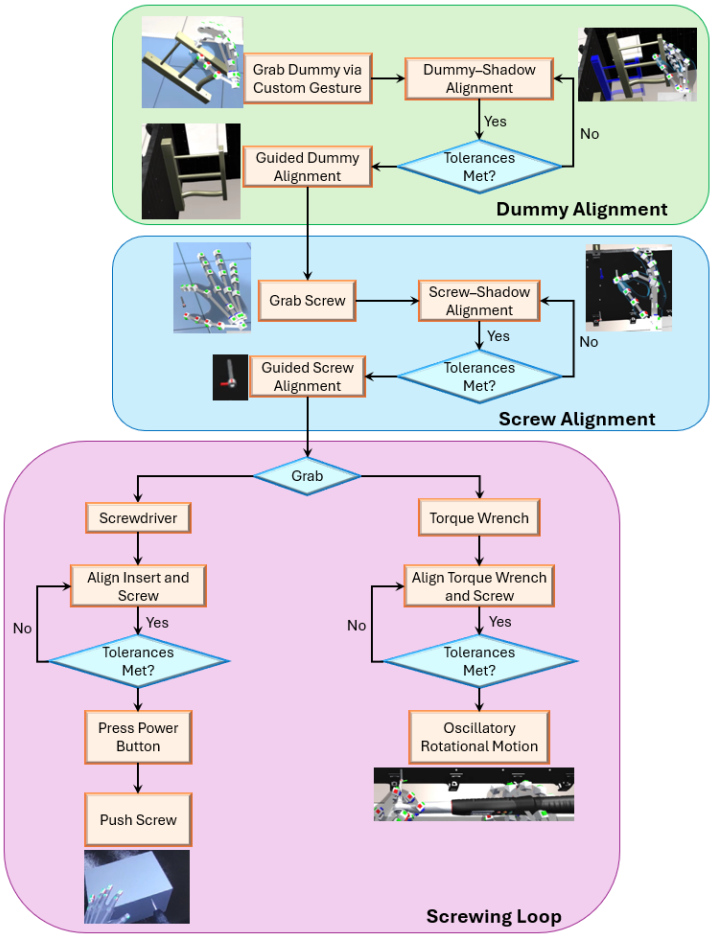


Figure 6.41: Flow diagram of the assembly procedure.

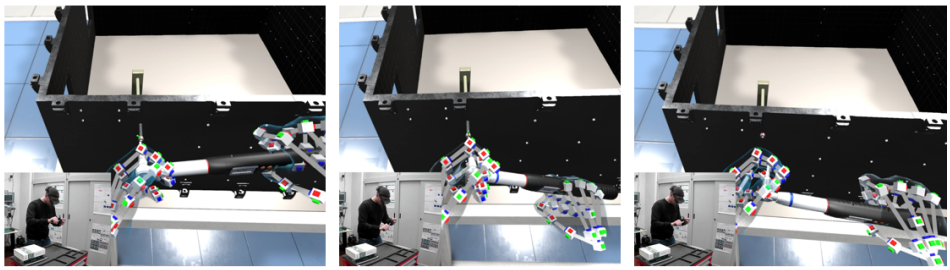


Figure 6.42: Example of a tightening procedure using the torque wrench.

- Depending on the selected tool, the screwing process is completed either by applying an axial insertion force while using the screwdriver or by performing an oscillatory rotational motion with the torque wrench (Figure 6.42).
- Once the required screwing conditions are satisfied, the screw reaches its final *Screw End* configuration, and the assembly operation is considered complete.

Chapter 7

Real-time interactive simulations

The increasing complexity of engineering systems and the growing emphasis on safety, efficiency, and sustainability requirements have profoundly changed the way industrial products and processes are designed, validated, and managed. In this context, numerical simulations are no longer used exclusively as offline analysis tools, but are integrated into digital architectures capable of evolving in parallel with physical systems. This approach has led to the emergence of the Digital Twin paradigm, understood as a dynamic and continuously updated virtual representation of the real system throughout the entire product lifecycle.

For safety-critical applications such as industrial machinery, energy systems, and aerospace components, the ability to predict system behaviour under varying operating conditions is essential. High-fidelity numerical models, such as multibody simulations, Finite Element Method (FEM) analyses, and Computational Fluid Dynamics (CFD), provide an accurate description of physical phenomena but are often incompatible with real-time constraints due to their high computational cost. This limitation becomes even more evident in multiphysics problems, simulations with large meshes, or parametric studies over wide operating condition spaces.

To enable real-time interaction, several strategies for reducing computational complexity have been developed. These include reducing the number of iterations in multibody solvers, using simplified formulations based on impulses and velocity-level constraints, and Reduced-Order Modeling (ROM) techniques. These approaches significantly reduce computational cost while preserving the essential physical characteristics of the system, making them particularly suitable for the development of Digital Twins driven by interactive simulations.

A fundamental requirement of Digital Twins in XR environments is the ability to support bidirectional communication between the physical system, the numerical model, and the user, as illustrated in Figure 7.1. In this context, three main categories of information can be distinguished.

The first category concerns data acquired from sensors installed on the physical system. This information is used to update the boundary conditions and inputs of the numerical model. Transmission occurs through standard protocols such as Ethernet, USB, Wi-Fi, or Bluetooth; however, the main challenge is not merely data acquisition, but the design of a low-latency, reliable, and scalable communication architecture ca-

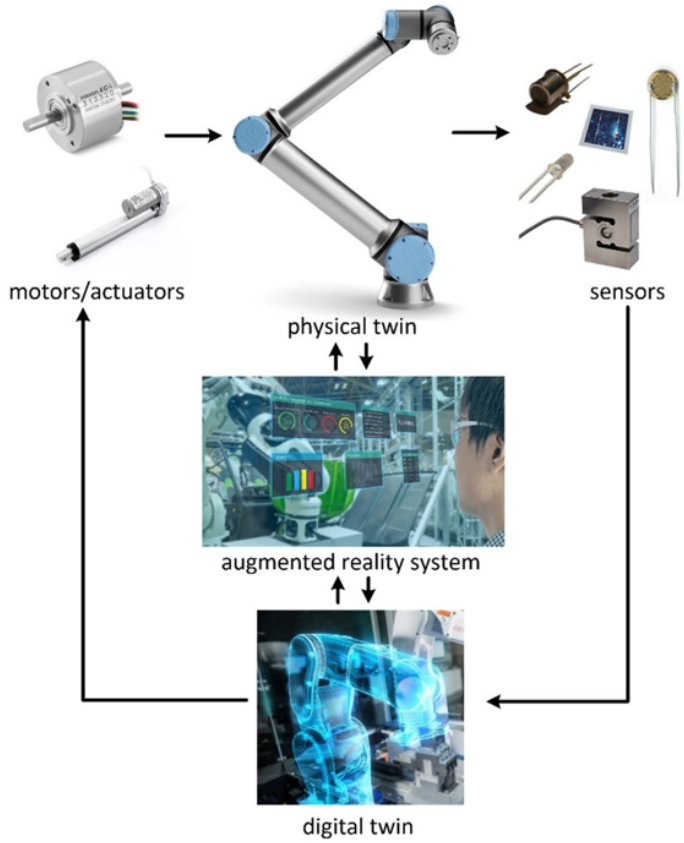


Figure 7.1: Interaction among digital twin, sensors, actuators and augmented reality interface for achieving an AR digital twin.

pable of managing continuous information exchange between the physical system, the simulation model, and immersive interfaces.

The second category includes inputs generated by the user. These can be provided through physical devices or through interactive virtual elements within the augmented environment, such as knobs, sliders, and buttons. These inputs are interpreted as boundary conditions or control parameters of the numerical model, directly influencing its dynamic evolution.

The third category concerns the results produced by the computational model, which are returned to the user in the form of physical variables and derived quantities such as positions, velocities, accelerations, reaction forces, energies, and powers. These data are overlaid in real time onto the physical system through three-dimensional or holographic graphical elements, enabling an augmented visualization of the system state.

The distinction between the second and third categories lies in the direction of the information flow: the second consists of user inputs to the model, while the third represents the model outputs to the user. In this scheme, data coming from physical sensors belong to the first category, while interactive inputs can be interpreted as coming from virtual sensors.

As an example, a physical system may be equipped with position and force sensors

that feed a multibody model, influencing its kinematic constraints and applied forces. In parallel, the user can interact with virtual interfaces to control actuators or system parameters. The Digital Twin multibody model processes this information in real time and returns the dynamic state of the system, which is then visualized in the augmented environment.

In this way, the Digital Twin does not merely replicate the behaviour of the physical system, but extends its informational capabilities, enabling access to quantities that are not directly measurable and providing a more complete and interactive understanding of the system.

7.1 Real-time dynamic analysis

The dynamic behaviour of a constrained multibody system combines the equations of motion of free rigid bodies with the forces generated by constraints [100]. Mechanical constraints define the space of admissible configurations and impose geometric relationships that restrict rigid body motion. Such constraints can be expressed in holonomic form as:

$$\mathbf{\Phi}(\mathbf{q}, t) = 0 \tag{7.1}$$

where $\mathbf{q}$ is the vector of generalized coordinates of the system, describing the position and rotation of each rigid body. From this description, it is possible to derive velocity- and acceleration-level equations. By differentiating once with respect to time, the velocity constraint condition is obtained:

$$\begin{aligned} \frac{d\mathbf{\Phi}}{dt} &= \frac{\partial \mathbf{\Phi}}{\partial \mathbf{q}}\dot{\mathbf{q}} + \frac{\partial \mathbf{\Phi}}{\partial \mathbf{t}} = 0 \\ &= \mathbf{\Phi}_q \dot{\mathbf{q}} + \mathbf{\Phi}_t = 0 \end{aligned} \tag{7.2}$$

where $\mathbf{\Phi}_q = \frac{\partial \mathbf{\Phi}}{\partial \mathbf{q}}$ is the Jacobian of the constraint equations $\mathbf{\Phi}(\mathbf{q}, t)$ with respect to the generalized coordinates $\mathbf{q}$, while $\mathbf{\Phi}_t = \frac{\partial \mathbf{\Phi}}{\partial \mathbf{t}}$ is the partial derivative of the constraint equations with respect to time. A further time differentiation leads to the acceleration-level formulation:

$$\begin{aligned} \frac{d^2\mathbf{\Phi}}{dt^2} &= \frac{d}{dt}\left(\mathbf{\Phi}_q \dot{\mathbf{q}}\right) + \frac{d}{dt}\left(\mathbf{\Phi}_t\right) = 0 \\ &= \mathbf{\Phi}_q \ddot{\mathbf{q}} + \left[\frac{\partial(\mathbf{\Phi}_q \dot{\mathbf{q}})}{\partial \mathbf{q}}\dot{\mathbf{q}} + \frac{\partial(\mathbf{\Phi}_q \dot{\mathbf{q}})}{\partial t}\right] + \left[\frac{\partial \mathbf{\Phi}_t}{\partial \mathbf{q}}\dot{\mathbf{q}} + \frac{\partial \mathbf{\Phi}_t}{\partial t}\right] = 0 \\ &= \mathbf{\Phi}_q \ddot{\mathbf{q}} + (\mathbf{\Phi}_q \dot{\mathbf{q}})_q \dot{\mathbf{q}} + 2\mathbf{\Phi}_{qt}\dot{\mathbf{q}} + \mathbf{\Phi}_{tt} = 0 \end{aligned} \tag{7.3}$$

where $(\mathbf{\Phi}_q \dot{\mathbf{q}})_q$ is the compact form of $\frac{\partial(\mathbf{\Phi}_q \dot{\mathbf{q}})}{\partial \mathbf{q}}$, $\mathbf{\Phi}_{qt}$ is the compact form of $\frac{\partial^2 \mathbf{\Phi}}{\partial q \partial t}$, and $\mathbf{\Phi}_{tt}$ is the compact form of $\frac{\partial^2 \mathbf{\Phi}}{\partial t^2}$.

These three formulations describe the same set of constraints, but at different levels of time differentiation. In classical multibody dynamics methods, these conditions are imposed simultaneously together with the equations of motion.

As introduced in Section 4.2.2, the motion of a single rigid body is described by the Newton–Euler equations [117]:

$$m\dot{\mathbf{v}} = \mathbf{f}, \tag{7.4a}$$

$$\mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\mathbf{I}\boldsymbol{\omega}) = \boldsymbol{\tau}, \tag{7.4b}$$

where m is the mass, $\mathbf{v}$ is the linear velocity, $\mathbf{I}$ is the inertia tensor, $\mathbf{f}$ are the external forces, and τ are the external torques. For a complete multibody system, these relations are assembled into a global formulation that highlights how system dynamics are influenced both by external forces and constraint reactions:

$$\mathbf{M}\ddot{\mathbf{q}} - \boldsymbol{\Phi}_q^T \boldsymbol{\lambda} = \mathbf{F}_e \tag{7.5}$$

where $\mathbf{M}$ is the mass matrix, which also includes inertia terms, $\mathbf{F}_e$ is the vector of applied generalized forces, and $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers associated with constraint forces.

The simultaneous enforcement of the equations of motion 7.5 and the constraint equations 7.1 leads to the solution of a coupled system of differential-algebraic equations (DAEs). The global system can be written in the following form:

$$\begin{bmatrix} \mathbf{M} & \boldsymbol{\Phi}_q^T \\ \boldsymbol{\Phi}_q & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_e \\ \boldsymbol{\gamma} \end{bmatrix} \tag{7.6a}$$

$$\boldsymbol{\gamma} = -(\boldsymbol{\Phi}_q \dot{\mathbf{q}})_q \dot{\mathbf{q}} - 2\boldsymbol{\Phi}_{qt}\dot{\mathbf{q}} - \boldsymbol{\Phi}_{tt} \tag{7.6b}$$

This formulation clearly highlights the saddle-point nature of the problem, whose solution requires the simultaneous handling of dynamic and constraint variables. Although this approach is rigorous and widely used in high-fidelity solvers, it is computationally expensive for real-time applications due to the size and sparsity of the matrices involved.

For real-time applications, such as digital twins in immersive environments or interactive systems, this cost can become incompatible with the temporal constraints imposed by user interaction. In these contexts, alternative formulations become relevant, which trade the globally exact solution for iterative or impulse-based sequential schemes, capable of ensuring numerical stability and performance compatible with real-time execution while still preserving an accurate representation of the essential physical behaviour.

7.1.1 Real-time solution strategy in Project Chrono

The classical formulation of constrained multibody systems leads to the definition of a differential-algebraic system, in which the equations of motion and the constraint conditions must be satisfied simultaneously. The direct solution of such a global system requires the joint handling of dynamic variables and constraint reactions, with a computational cost that is often incompatible with real-time interactive applications.

For this reason, real-time solvers adopt alternative numerical strategies, based on problem reformulations and iterative methods that prioritize numerical stability and execution time over the exact solution of the global system. Within this context lies the approach adopted by Project Chrono [219] in the *Chrono-Unity* plugin, which allows

the multibody physics engine developed in C++ to be connected with the Unity environment through an interface generated using SWIG (Simplified Wrapper and Interface Generator). This architecture enables the execution of complex physical simulations and the visualization of their results directly within the graphics engine, while maintaining the computational efficiency of the numerical core. Figure 7.2 shows some application examples of the Chrono-Unity solver.

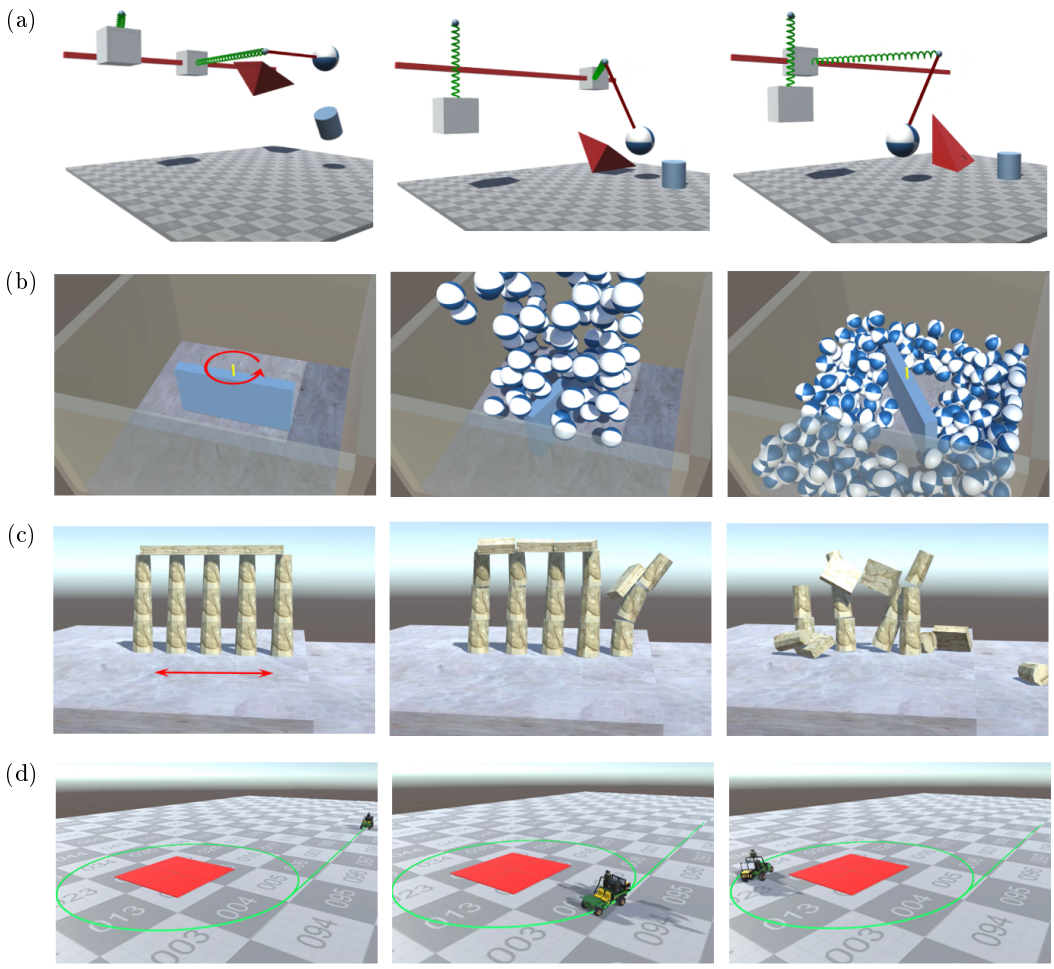


Figure 7.2: Chrono-Unity examples: sandbox, mixer, earthquake and vehicle dynamics.

Within this framework, the multibody system is not treated as a single global saddle-point problem, but is reformulated in terms of local subproblems associated with individual constraints and contacts. In particular, contact handling represents a central element of the numerical approach. Chrono-Unity distinguishes two main contact models. The first is the *Non-Smooth Contact* (NSC), which assumes perfectly rigid bodies and enforces the non-penetration constraint through complementarity conditions. In compact form, these conditions can be expressed as:

$$\mathbf{d} \geq \mathbf{0}, \quad \mathbf{F}_R \geq \mathbf{0}, \quad \mathbf{d}^T \cdot \mathbf{F}_R = 0 \tag{7.7}$$

where $\mathbf{d}$ represents the vector of distances between bodies and $\mathbf{F}_R$ the vector of normal

reaction forces. This formulation implies that, at each contact point, either the distance or the reaction force can be active, but not both simultaneously. The resulting problem is formulated as a Differential Variational Inequality (DVI), which, after time discretization, can be reduced to a Linear Complementarity Problem (LCP), suitable for the numerical solution of rigid contact constraints [220, 221].

The second model is the *Smooth Contact* (SMC), which introduces a penalty-based formulation of contact. In this case, a small interpenetration between bodies is allowed, and contact is modeled through elastic and dissipative forces proportional to the virtual deformation [222].

Once the problem is discretized in time, the solution is not obtained through direct factorization of the global system, but through iterative projected methods that progressively update contact forces and associated multipliers. The solvers implemented in Chrono-Unity are:

- Projected Successive Over-Relaxation (PSOR): an iterative method derived from Successive Over-Relaxation (SOR), in which variables are updated sequentially and then projected onto the admissible solution space. It is a simple and low-cost-per-iteration method, suitable for small-scale problems or when high accuracy is not required. However, convergence may deteriorate in poorly conditioned systems or in the presence of strong mass heterogeneities [223].

- Accelerated Projected Gradient Descent (APGD): a method based on projected gradient descent, in which the solution is updated along a descent direction and then projected onto the feasible space. The introduction of an acceleration strategy enables faster and more stable convergence compared to basic methods, making it suitable for simulations with numerous contacts and friction [224].

- Barzilai–Borwein: a variant of gradient-based methods in which the step size is dynamically adapted based on the evolution of previous iterations. This helps avoid overly conservative or unstable steps, improving convergence speed. It is often used as an accelerator for APGD, particularly in systems characterized by highly heterogeneous mechanical properties [225].

A fundamental aspect of these methods is that the solution of the constrained problem is not computed exactly at each time step, but approximated through a limited number of iterations. This allows direct control of the computational cost and its adaptation to the constraints imposed by real-time simulation.

Further performance improvements are achieved through acceleration techniques. The *Diagonal Preconditioner* modifies the system by scaling variables using the diagonal of the matrix, improving conditioning and reducing the number of iterations required for convergence [226, 227]. The *Warm Start* instead uses the solution from the previous time step as the initial guess for the current one [228]. Since dynamic simulations often evolve smoothly, this initialization significantly reduces the number of required iterations. This strategy is particularly effective when combined with linearized implicit integration schemes.

Time discretization is typically based on implicit schemes, which ensure greater stability compared to explicit approaches in the presence of contacts and rigid constraints.

In particular, variants of the implicit Euler method are used. *Euler Implicit Linearized* represents the most commonly used formulation for complementarity-constrained problems. *Euler Implicit Projected* introduces a projection step that enforces constraints during integration, while *Euler Implicit* represents the basic formulation for less complex problems [229].

Overall, the efficiency of Chrono-Unity is achieved through the use of iterative formulations that replace the direct solution of the system. The number of iterations per time step thus becomes a design parameter, allowing a balance between numerical accuracy and computational cost according to real-time constraints. This trade-off enables the use of the multibody model for digital twins of dynamic systems and interactive applications in XR environments.

7.1.2 Sequential impulse solver

An alternative, highly efficient approach for solving the system of differential-algebraic equations arising in multibody dynamics is the Sequential Impulse Solver, in which, instead of directly solving for accelerations, the system computes the impulses required to satisfy velocity-level constraints at each time step [101, 118]. This approach enables a robust and efficient real-time solution, introducing only a minimal approximation with respect to the exact solution while maintaining an accuracy level that is generally acceptable in the context of Digital Twins.

The sequential impulse method has been widely used in real-time simulations, even in the presence of direct user interaction [230]. The solution strategy for the equations of motion of a set of rigid bodies can be summarized starting from the multibody formulation based on Lagrange multipliers and constraint equations [124].

Sequential impulse solvers follow a strategy that can be divided into two main phases:

1. First, the equations of motion are solved by neglecting kinematic constraints, considering only external and internal forces acting on the bodies. In this way, the resulting solution is only approximate since constraints are not satisfied. It should be noted that contact relationships can also be treated as constraints, albeit unilateral ones.

2. Subsequently, a sequence of impulses is applied to the solution, one at a time, in an iterative and sequential manner, in order to correct the motion of the bodies and enforce the constraint equations within a specified tolerance.

In detail, in the first step the equations are solved while neglecting constraints:

$$\mathbf{M}\,\ddot{\mathbf{q}}_{\text{approx}} = \mathbf{F}_e \tag{7.8}$$

where $\ddot{\mathbf{q}}_{\text{approx}}$ is the vector of approximate accelerations, $\mathbf{M}$ is the mass matrix, and $\mathbf{F}_e$ is the vector of applied generalized forces. The approximate velocities $\dot{\mathbf{q}}_{\text{approx}}$ and positions $\mathbf{q}_{\text{approx}}$ can be computed using a linear approximation:

$$\dot{\mathbf{q}}_{\text{approx}} = h\,\ddot{\mathbf{q}}_{\text{approx}} \tag{7.9}$$

$$\mathbf{q}_{\text{approx}} = h\,\dot{\mathbf{q}}_{\text{approx}} \tag{7.10}$$

where h is the (constant) integration time step.

The corrective impulses $\mathbf{P}_{\text{constraint}}$ generate variations according to Newton's impulse law:

$$\mathbf{M}\left(\dot{\mathbf{q}}_{\text{corrected}} - \dot{\mathbf{q}}_{\text{approx}}\right) = \mathbf{P}_{\text{constraint}} \tag{7.11}$$

where $\dot{\mathbf{q}}_{\text{corrected}}$ is the corrected velocity vector, which can be computed as:

$$\dot{\mathbf{q}}_{\text{corrected}} = \dot{\mathbf{q}}_{\text{approx}} + \mathbf{M}^{-1}\mathbf{P}_{\text{constraint}} \tag{7.12}$$

Since impulses are applied to enforce constraints, it holds that:

$$\mathbf{P}_{\text{constraint}} = \mathbf{\Phi}_q^T \boldsymbol{\delta} \tag{7.13}$$

where, following the notation of Section 7.1, $\mathbf{\Phi}$ denotes the constraint equations and $\mathbf{\Phi}_q = \frac{\partial \mathbf{\Phi}}{\partial \mathbf{q}}$, while $\boldsymbol{\delta}$ is the vector of Lagrange multipliers associated with the impulses.

Considering the time derivatives of the constraint equations $\mathbf{\Phi}$, we obtain:

$$\mathbf{\Phi} = \mathbf{0} \Rightarrow \frac{d\mathbf{\Phi}}{dt} = \mathbf{\Phi}_q \dot{\mathbf{q}} + \mathbf{\Phi}_t = \mathbf{0} \tag{7.14a}$$

$$\mathbf{\Phi}_q \dot{\mathbf{q}}_{\text{corrected}} + \mathbf{\Phi}_t = \mathbf{0} \tag{7.14b}$$

Substituting the corrected velocities into the previous equations:

$$\mathbf{\Phi}_q \left(\dot{\mathbf{q}}_{\text{approx}} + \mathbf{M}^{-1}\mathbf{\Phi}_q^T \boldsymbol{\delta} + \mathbf{\Phi}_t\right) = \mathbf{0} \tag{7.15}$$

Solving for the impulse Lagrange multipliers:

$$\boldsymbol{\delta} = \left(\mathbf{\Phi}_q \mathbf{M}^{-1} \mathbf{\Phi}_q^T\right)^{-1} \left(\mathbf{\Phi}_q \dot{\mathbf{q}}_{\text{approx}} + \mathbf{\Phi}_t\right) \tag{7.16}$$

and consequently the applied impulses are:

$$\mathbf{P}_{\text{constraint}} = \mathbf{\Phi}_q^T \boldsymbol{\delta} \tag{7.17}$$

The correction of accelerations and the enforcement of constraints require several iterations. Finally, it is possible to compute the reaction forces by evaluating the accumulated impulses. First, the accumulated Lagrange multipliers $\mathbf{\Delta}$ over the iterations are computed:

$$\mathbf{\Delta} = \sum_{\text{iterations}} \boldsymbol{\delta} \tag{7.18}$$

Substituting into the previous equations, the accumulated constraint impulses are obtained:

$$\mathbf{P}^{\text{tot}}_{\text{constraint}} = \mathbf{\Phi}_q^T \mathbf{\Delta} \tag{7.19}$$

From this, the constraint reaction forces can be determined:

$$\mathbf{F}_{\text{constraint}} = \frac{\mathbf{P}^{\text{tot}}_{\text{constraint}}}{h} \tag{7.20}$$

Figure 7.3 shows the flow diagram of the sequential impulse solution strategy for the dynamic digital twin model, which takes as input the physical twin and user commands, outputs signals to actuators, and builds augmented projections (holograms).

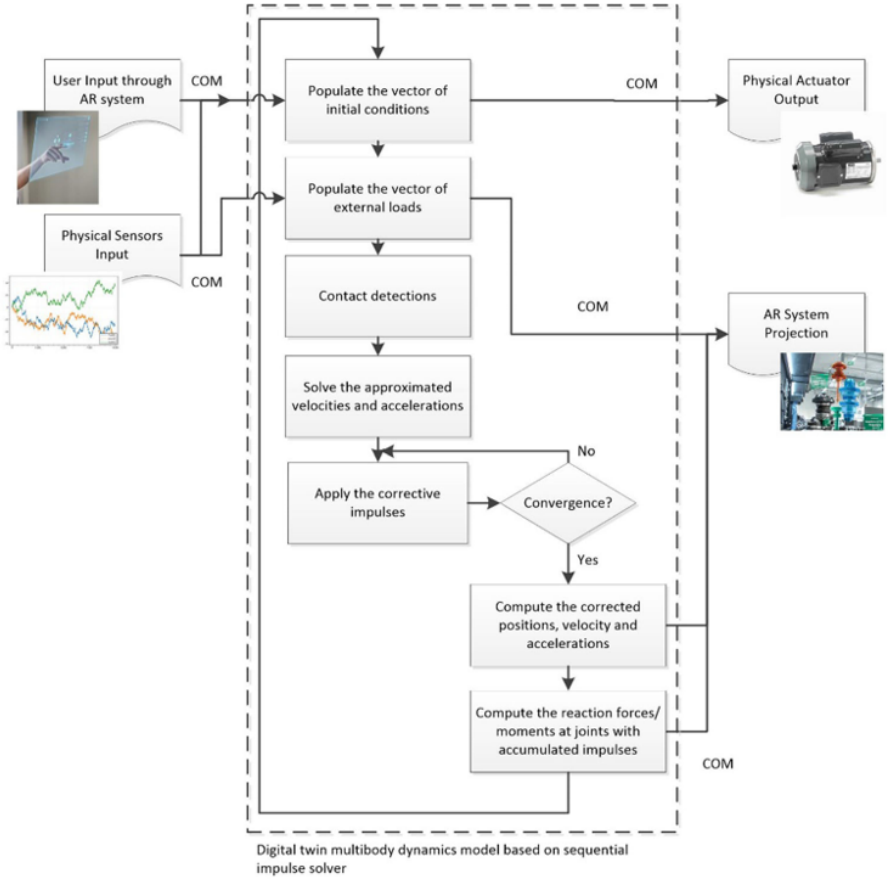


Figure 7.3: A multibody dynamic model based on sequential impulse solvers with communication ports with real actuators and sensors.

The sequential impulse formulation is also particularly suitable for managing contacts, which are necessary to implement the general forms of interaction with virtual objects in the immersive scene. In particular, the collisions may be treated as penalty constraints. The generic not-overlapping condition between two bodies, C_n can be written as:

$$C_n = (\mathbf{P}_A - \mathbf{P}_B) \cdot \mathbf{n}_A \geq 0 \tag{7.21}$$

where $\mathbf{P}_A$ and $\mathbf{P}_B$ are the closest points of two bodies that can come in contact and $\mathbf{n}_A$ is the normal unit vector of the body shape at P_A.

Equation 7.21 can be written at the velocity level as:

$$\frac{dC_n}{dt} = \dot{C}_n = \frac{d(\mathbf{P}_A - \mathbf{P}_B)}{dt} \cdot \mathbf{n}_A + (\mathbf{P}_A - \mathbf{P}_B) \cdot \frac{d\mathbf{n}_A}{dt} \geq 0 \tag{7.22}$$

The Baumgarte scheme [231] can be used to push the bodies apart when they overlap. The velocity constraint is augmented with a feedback term proportional to the penetration depth:

$$\dot{C}_n + \beta C_n = 0 \tag{7.23}$$

where β is a tunable scalar parameter which governs the speed of the penetration resolution. Equation 7.23 is then appended to the vector $\mathbf{\Phi}$ and treated as a generic constraint.

7.1.3 Case study: interactive multibody simulations

This section presents a framework for interactive multibody simulation with real-time haptic feedback integration. The objective is to enable direct interaction between a user and a constrained multibody dynamic system, allowing the grasping and manipulation of rigid or flexible bodies while maintaining kinematic consistency and returning physically coherent reaction forces (Figure 7.4a).

Haptic devices are already widely used in training, ergonomics, and operational support applications. A first category of applications concerns interaction with free objects, such as mechanical tools, scalpels, drills, or manipulation instruments, whose motion is kinematically driven by the haptic device and which interact with rigid or deformable environments [232,233]. In such scenarios, the user perceives contact forces, viscous effects, friction, and surface adhesion phenomena.

A second category includes the direct manipulation of objects or systems grasped and connected to the haptic device [234, 235]. In this case, the feedback returned to the operator is mainly associated with weight and inertial effects during object motion.

There are also applications based on the generation of artificial forces, in which the device does not reproduce physically present forces in the real system, but mathematically defined synthetic forces [236, 237]. Such forces are used to guide the operator along a correct sequence of operations, prevent errors, or restrict undesired motions through vibrations, virtual walls, and other artificial constraints.

However, the integration of a haptic device within an interactive multibody dynamic simulation is significantly more complex, since the system contains constrained bodies and has a limited number of degrees of freedom available for interaction. Directly imposing the full 6-DOF pose measured by the haptic device onto the multibody system would lead to an overconstrained formulation, resulting in kinematic inconsistencies and numerical instability, thus requiring dedicated coupling strategies between the haptic device and the constrained elements of the simulation.

The implemented system is based on the integration of the Geomagic Touch X haptic device within the virtual environment, using the OpenHaptics SDK integrated into the *HapticsDirect* plugin to provide low-level access to device data. The dynamic simulation is solved in parallel using both the Project Chrono multibody solver, via the Chrono-Unity plugin, and the sequential impulse solver. The virtual environment was developed in Unity Engine 2022.3.41, with a setup for the Meta Quest Pro. Interaction between the haptic device and the simulation occurs through a bidirectional synchronous loop:

- from the haptic device to the multibody solver: the tool tip pose (6 DOF) is acquired via low-level APIs and mapped into input conditions of the dynamic system using the command
  `hdGetDoublev(HD_CURRENT_POSITION, position);`

- from the solver to the haptic device: the reaction forces computed by the model are sent back to the device to generate haptic feedback using the command
  `hdSetFloatv(HD_CURRENT_FORCE, force)`.

The system operates in a closed-loop configuration: the haptic device provides the pose of the stylus tip, the multibody model computes the resulting dynamics, and the

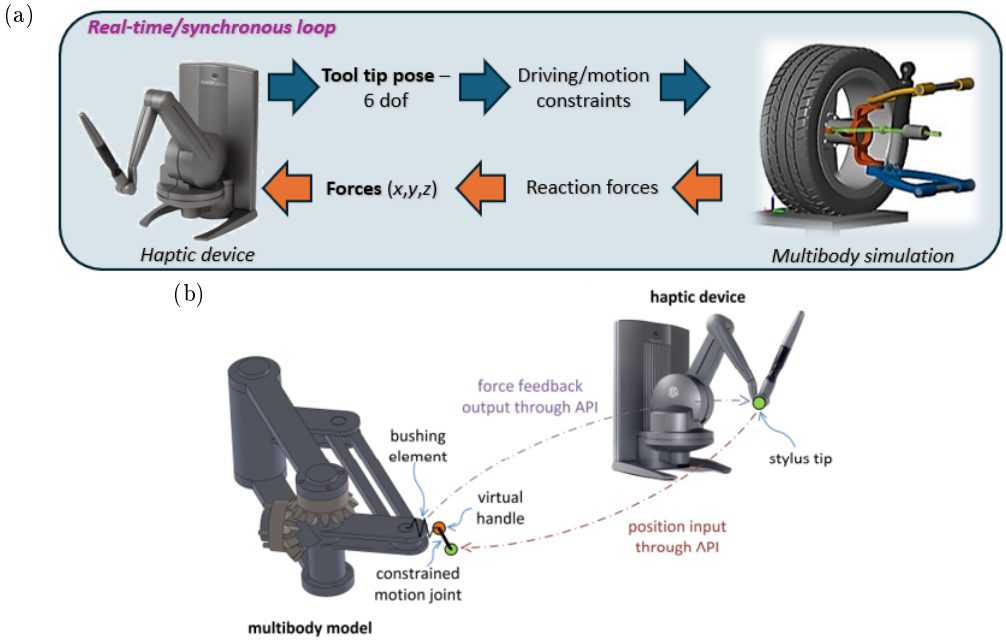


Figure 7.4: (a) Haptic feedback real-time/synchronous loop, (b) object manipulation through the virtual handle.

reaction forces are returned to the user, ensuring bidirectional coupling between the physical and virtual systems.

To avoid overconstraining the system, the interaction between the haptic device and the simulated system is not imposed as a direct kinematic constraint. Instead, a compliant coupling element, called the *virtual handle* (Figure 7.4b), is introduced. The virtual handle is modeled as a viscoelastic element connecting the haptic tip position $\mathbf{x}_h$ to an attachment point $\mathbf{x}_a$ rigidly fixed to the selected body, which influences the system without causing constraint inconsistencies. The resulting interaction force is defined as:

$$\mathbf{F}_h = k_h \left(\mathbf{x}_h - \mathbf{x}_a\right) + c_h \left(\dot{\mathbf{x}}_h - \dot{\mathbf{x}}_a\right), \tag{7.24}$$

where k_h and c_h are the stiffness and damping coefficients of the virtual coupling, respectively. The computed force is applied to the multibody system and, by Newton's third law, returned to the haptic device and thus to the user at every simulation step. The framework operation can be described through the following logical steps:

1. detection of the contact point and identification of the body;
2. creation of the virtual attachment point;
3. introduction of the virtual handle as a compliant element;
4. dynamic solution of the system under the induced perturbation;
5. return of reaction forces to the haptic device.

A robust implementation of the framework presents several fundamental challenges. First, the virtual handle stiffness must be sufficiently high to ensure responsiveness in interaction, but not so high as to significantly alter the system dynamics or introduce numerical instability. Moreover, user-generated signals are inherently affected by noise due to physiological limitations of motor control and the discrete nature of the acquisition system. Similarly, the forces returned to the device must be filtered to avoid high-frequency oscillations that would degrade the perceptual comfort of the interaction.

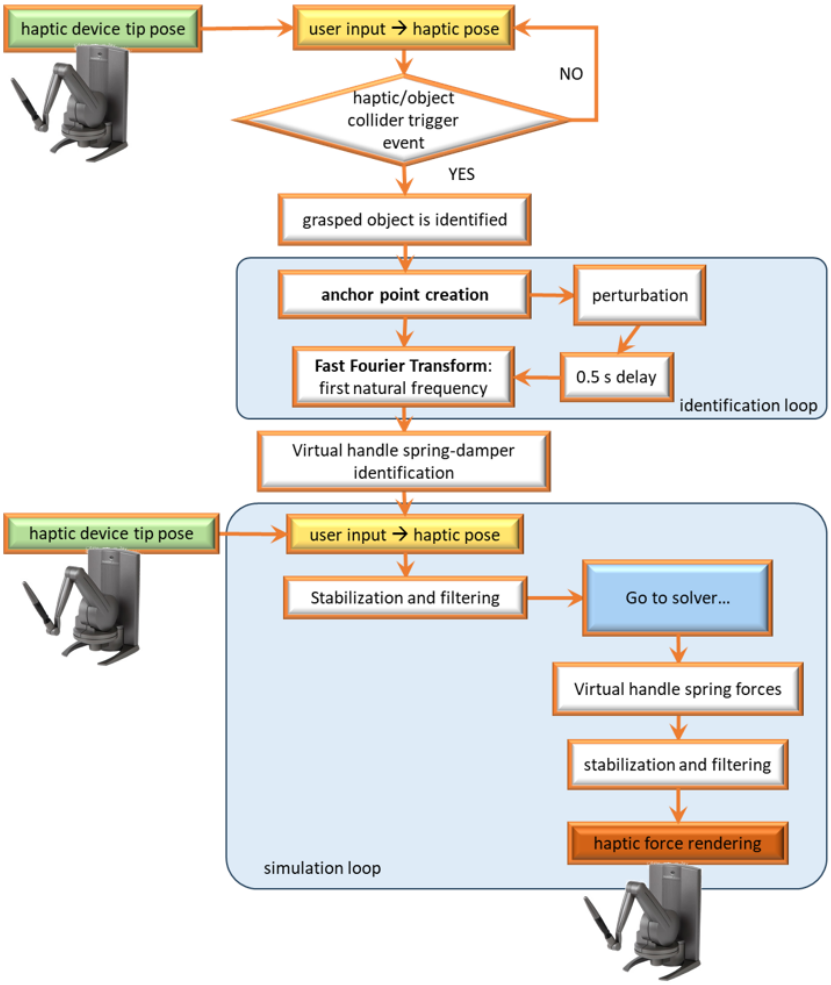


Figure 7.5: Flow diagram for computing virtual handle parameters and the haptic feedback perceived by the user.

An additional challenge of the framework lies in the selection of virtual handle parameters, which cannot be predefined a priori as they depend on the dynamic characteristics of the simulated system. To address this issue, an adaptive identification procedure is introduced. When the user establishes a grasp at a point $\mathbf{P}$ of the selected body, an initialization phase of duration 0.5 s is activated. During this interval:

1. a small perturbation of approximately $F \approx 0.06\,\mathrm{N}$ is applied at $\mathbf{P}$;

2. the system displacement response is recorded;
3. a Fast Fourier Transform (FFT) is computed to extract the dominant frequency f_{FFT}.

The virtual handle stiffness is then defined as:

$$k_h = m\,(a \cdot 2\pi f_{\mathrm{FFT}})^2\,, \tag{7.25}$$

where m is the equivalent mass of the body and $a = 10$ is an empirical scaling coefficient.

The damping coefficient is defined as a fraction of critical damping:

$$c_h = 0.2\sqrt{m k_h}. \tag{7.26}$$

After the identification phase, the parameters are fixed and the system enters the steady-state interaction phase.

This procedure enables automatic adaptation of the virtual handle behavior to the dynamic characteristics of the system, ensuring robustness and stability even in heterogeneous configurations.

The high-frequency nature of haptic interaction requires specific considerations regarding signal stability. Although the haptic device supports sampling rates up to 4 kHz, the simulation loop runs at approximately 90 Hz, introducing a natural filtering effect. However, user-induced noise and numerical discretization can still propagate through the system. In this context, the viscous term of the virtual handle plays a key role in reducing unwanted oscillations and improving the perceptual continuity of feedback.

Figure 7.5 summarizes the entire framework, from contact point definition to haptic feedback delivery.

The proposed framework has been evaluated through three representative case studies: an articulated quadrilateral, a slider-crank mechanism, and a flexible beam. For each case, the full interaction scheme has been implemented to enable manipulation of virtual objects subject to kinematic constraints.

To consistently evaluate the performance of the different simulation approaches, a comparison was also conducted between the Project Chrono multibody solver and the sequential impulse solver, considering the forces generated at the virtual handle in response to a prescribed trajectory of the attachment point.

The comparison was carried out under controlled conditions, disabling the haptic device and analyzing only the system dynamics through the virtual handle. This choice ensured identical boundary conditions for both solvers, making the resulting reaction forces directly comparable.

7.1.3.1 Four-bar mechanism

The first case study is the four-bar mechanism with two rotational springs between the cranks and the ground, as shown in Figure 7.6b. The forces computed by the multibody solvers and returned to the haptic device, and consequently to the user, are evaluated with respect to the left-handed reference frame in Unity shown in Figure 7.6a.

The integration of the *Project Chrono* multibody engine within the *Unity* environment was implemented through the *Chrono-Unity* wrapper. In this context, the solver

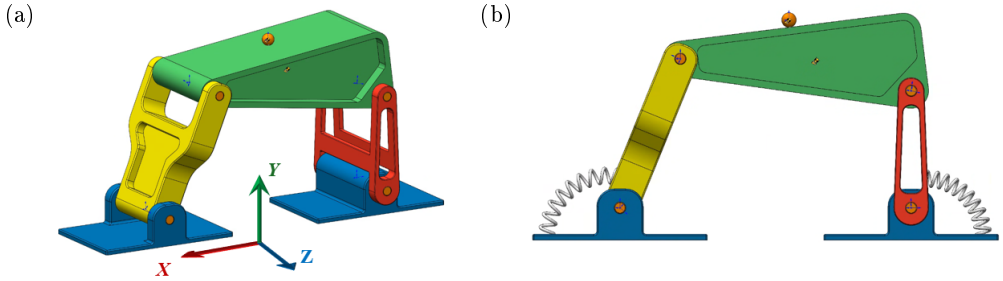


Figure 7.6: (a) Unity left-handed reference frame of the four-bar mechanism, (b) rotational springs between the cranks and the ground.

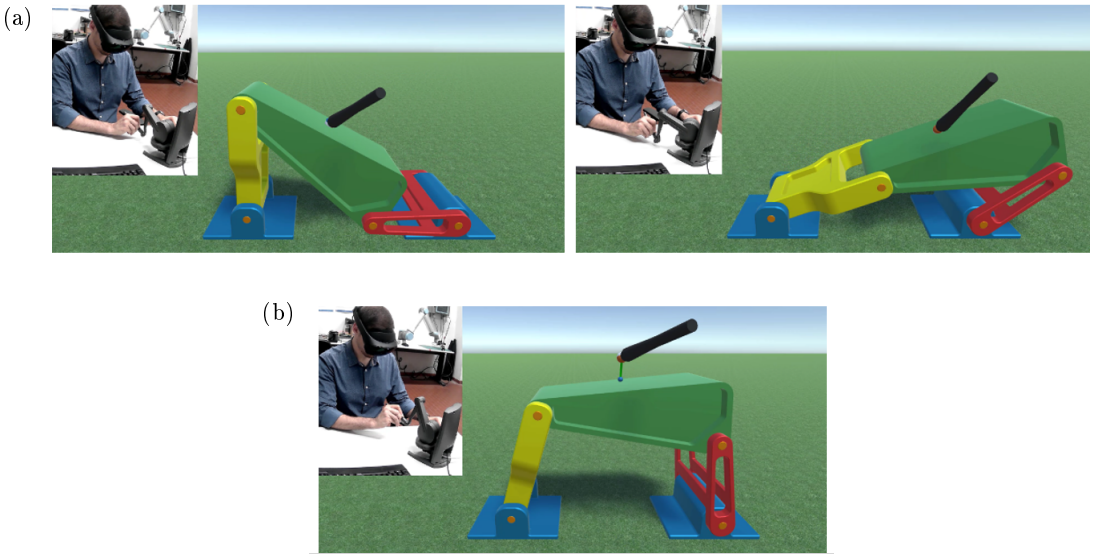


Figure 7.7: (a) Manipulation of the four-bar mechanism using a haptic device in a VR environment, (b) virtual handle for motion transmission.

configuration is managed by associating a dedicated *GameObject* with the *UCHSystem* script, in which global solver parameters are defined as described in Section 7.1.1. In all simulations presented in the case studies, a Barzilai-Borwein iterative scheme with diagonal preconditioning was adopted, limiting the maximum number of iterations to 5000. Contact modeling parameters are not explicitly reported, since the considered scenario does not include rigid body impact interactions.

The representation of physical elements within the Chrono-Unity engine requires the transformation of *GameObjects* into instances of the *ChBody* class, which constitutes the fundamental unit for rigid body modeling in the *Project Chrono* framework. This conversion is achieved through the combined assignment of two main components: *UChMaterialSurface* and *UCHConvexHull*. The former describes contact interface properties, ensuring consistency with the system-level contact method defined in the *UCHSystem* script, and includes friction coefficients and the characterization of elastic or inelastic impact behavior. The latter defines the collision geometry through a convex hull representation and also includes inertial properties such as mass and kinematic state (linear

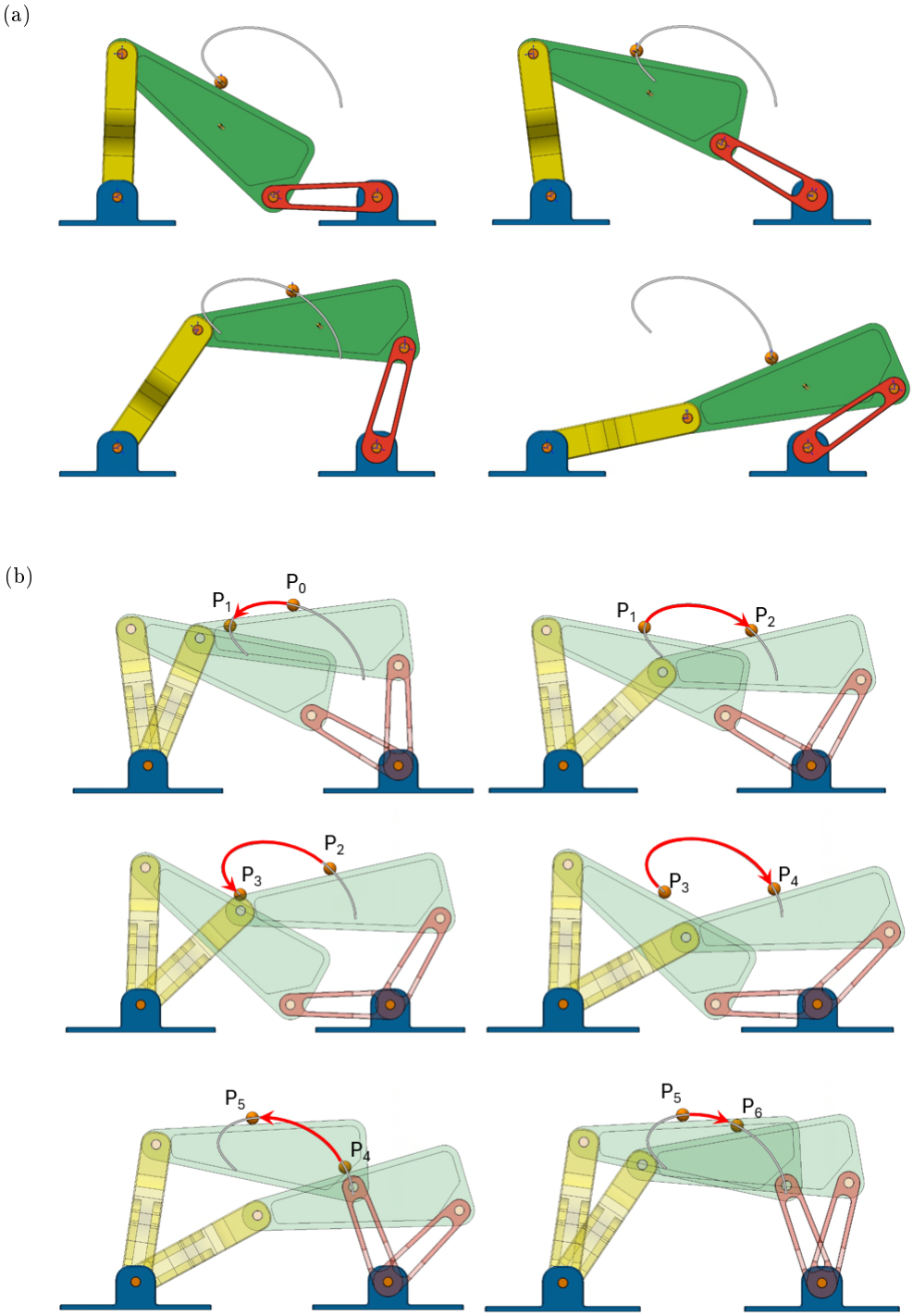


Figure 7.8: (a) Trajectory of the connecting rod point, (b) imposed motion of the crank point for force comparison between the two solvers.

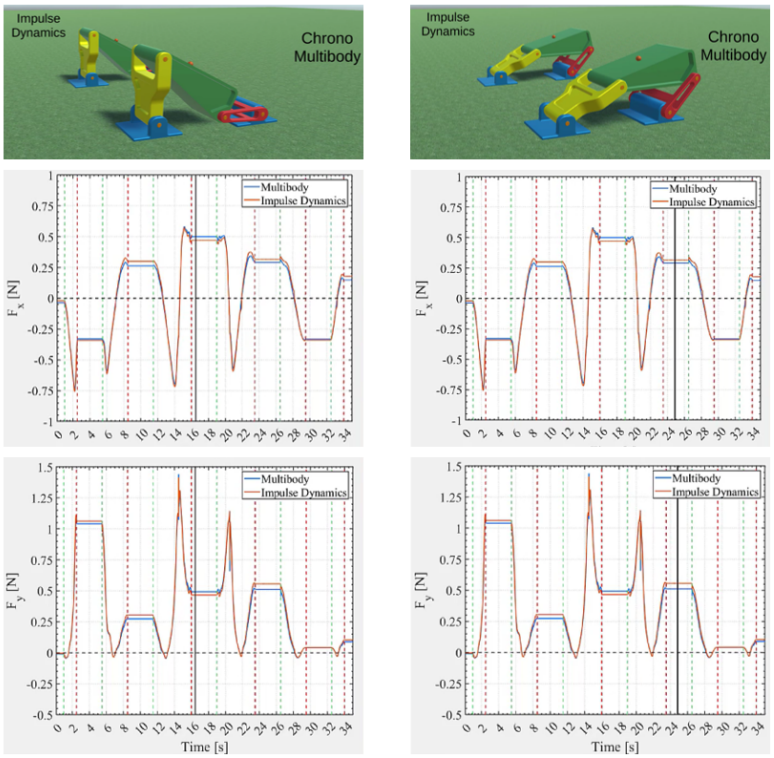


Figure 7.9: Comparison between Chrono-Unity and sequential impulse solver of the force applied by the haptic device on the user for a prescribed motion path of the connecting rod point, with results expressed in the reference frame of Figure 7.6a.

and angular velocities). In the case of the ground frame, the *UCHConvexHull* script is replaced by *UCHBodyGround*, which imposes the appropriate ground constraint conditions.

The definition of constraints between rigid bodies is implemented through *ChLink* classes, which allow the introduction of kinematic relationships between two *ChBody* objects. Each constraint is formalized through a pair of *ChMarker*, i.e., local reference frames rigidly attached to the involved bodies and used as the basis for the formulation of constraint equations. The position of a dedicated *GameObject* determines the spatial location of the constraint within the scene. Each *ChLink* is defined with respect to a master reference frame, in which constraint reactions and principal axis directions are expressed. The involved bodies are then assigned as master and slave, determining the direction of the kinematic dependency.

In the specific case of the articulated four-bar mechanism, revolute joints are implemented using the *UChJointRevolute* script, associated with a *GameObject* whose position defines the joint center and thus its axis of rotation. This joint is described by a set of five constraint equations between the two associated *ChMarker* frames: three equations enforce the coincidence of frame origins, while the remaining two constrain the alignment of two local directions, typically the z-axes of the marker frames, which are kept parallel during motion evolution (Appendix D.7). In this way, the z-axis of the *GameObject* associated with the joint explicitly defines the allowed rotation axis about which the relative

motion between the two bodies occurs.

The two torsional springs between the cranks and the ground are implemented using the *UChLinkBushing* script, associated with dedicated empty *GameObjects*. The bushing element constitutes a generalized viscoelastic coupling capable of introducing up to six independent stiffness and damping components, corresponding to the three translational and three rotational degrees of freedom. Similarly to revolute joints, the bushing also requires the definition of a master and a slave body. The elastic forces and torques generated by the element are computed as actions applied to the slave body with respect to the master reference frame. The reaction components are expressed in the local coordinate system of the associated *GameObject*; consequently, both stiffness and damping directions, as well as the sign of resulting forces and torques, depend on the orientation of this local frame.

For the implementation of the *virtual handle*, the *UChTSDA* script (Translational Spring-Damper-Actuator) was used, associated with a dedicated empty *GameObject*. This component implements a translational spring-damper-actuator coupling between two bodies, requiring stiffness, damping, and an optional external actuation force (set to zero in this case). The equilibrium position of the element can be automatically defined by enabling the *Auto Rest Length* option, which initializes the rest length based on the system configuration at the time of constraint creation; this rest length is defined in the *Rest Length* parameter. In the developed framework, the rest length was set to zero so that the elastic force depends exclusively on the relative displacement between the haptic device tip and the attachment point on the selected body. During simulation, the force generated by the coupling is used both to act on the multibody model and to generate the haptic feedback returned to the user. This quantity is directly accessible through the Chrono-Unity API via the method
`nameGameObject.GetComponent<UChTSDA>().spring.GetForce()`.

In parallel, the sequential impulse-based model was developed using the standard constraint components provided by the Unity engine.

Regarding haptic manipulation, a connecting rod point was selected as the attachment point of the virtual handle. Figure 7.7b shows the presence of the virtual handle between the haptic device and the connecting rod point as a green spring. Figure 7.7a instead shows an example of manipulation of the articulated mechanism using the haptic device.

To compare the forces returned to the haptic device computed in parallel by the two solvers under identical boundary conditions, a prescribed motion path of the attachment point was imposed. First, the trajectory of the connecting rod point used as the attachment point was identified (Figure 7.8a). Subsequently, a motion path was defined: for the four-bar mechanism, six different motion segments were imposed over time intervals ranging from 1.5 to 4.5 seconds, interleaved with 3-second rest periods (Figure 7.8b). The force results are reported in Figure 7.9; motion start instants are highlighted with green dashed lines, while stop instants are indicated with red dashed lines. The comparison between the sequential impulse solver and the Chrono reference shows deviations below 10%, with larger discrepancies occurring during abrupt direction changes due to inertial effects, where peak differences appear in the sequential impulse results.

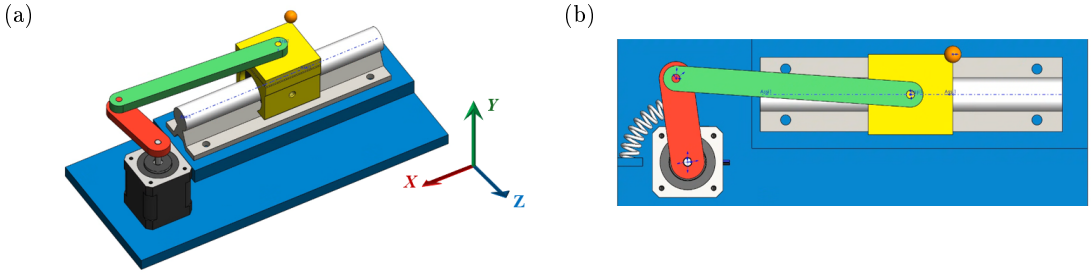


Figure 7.10: (a) Unity left-handed reference frame of the slider-crank mechanism, (b) system configuration with an external resisting torque applied to the crank with respect to the ground frame.

7.1.3.2 Slider-crank mechanism

The second case study concerns a slider-crank mechanism subjected to a resisting torque applied at the joint between the crank and the ground, as illustrated in Figure 7.10b. Analogously to the four-bar case, the forces returned by the multibody solvers and subsequently transmitted to the haptic device are expressed in the Unity left-handed reference frame shown in Figure 7.10a.

From a kinematic modeling perspective, compared to the previous case, the presence of a prismatic pair between the slider and the ground is introduced. This constraint is implemented using the *UChJointPrismatic* script, associated with a *GameObject* defining its position and orientation in space. The joint is described by five constraint equations formulated between pairs of *ChMarker* frames attached to the involved bodies, enforcing consistency of the three translational coordinates and constraining two local directions, while allowing free translation along the z-axis of the joint reference frame (Appendix D.9).

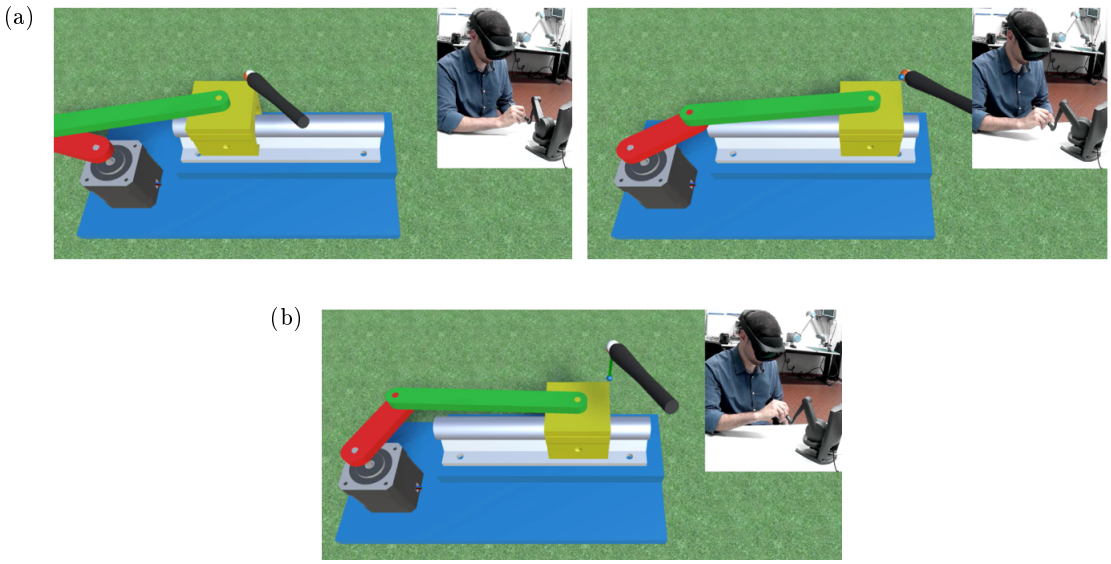


Figure 7.11: (a) Interaction with the slider-crank mechanism in a VR environment using a haptic device, (b) virtual handle attached to the slider point for motion transmission.

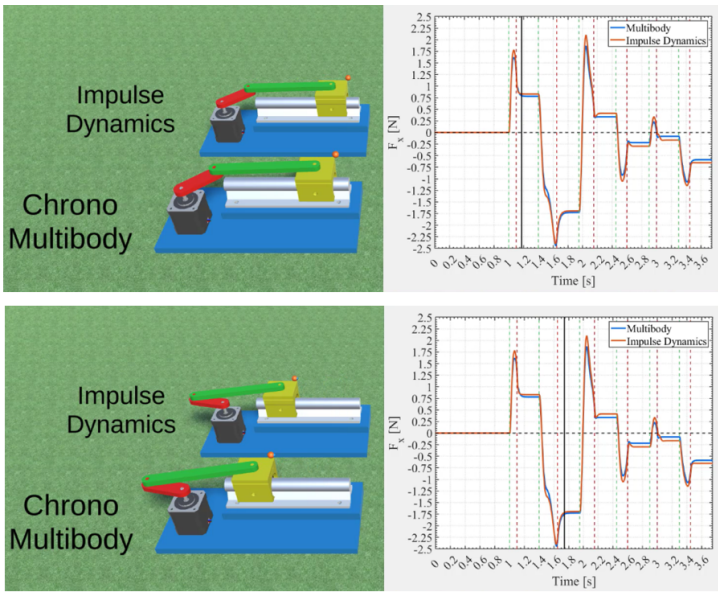


Figure 7.12: Comparison between Chrono-Unity and sequential impulse solver of the force applied by the haptic device on the user for a prescribed motion path of the slider, with results expressed in the reference frame of Figure 7.10a.

For interactive manipulation, a point belonging to the slider was selected as the attachment point of the *virtual handle*. Figure 7.11b highlights the compliant coupling between the haptic device tip and the slider, while Figure 7.11a shows an instantaneous configuration of the mechanism during interaction in the immersive environment.

To consistently compare the forces returned by the two solvers, a prescribed motion profile was imposed on the slider attachment point, while keeping identical boundary conditions across the two numerical formulations. The trajectory was defined through a sequence of six displacement segments, applied over time intervals ranging between 0.1 and 0.2 s and separated by rest phases of approximately 0.3 s, with the aim of emphasizing the system behavior in a strongly transient regime.

The results related to the axial force component are reported in Figure 7.12. The comparison between the sequential impulse solver and the Project Chrono reference shows deviations below 15%, higher than in the articulated four-bar case, particularly during rapid motion reversals, where inertial effects and the discrete nature of constraint correction produce more pronounced transient usual peaks in sequential impulse results.

7.1.3.3 Flexible beam

The third case study concerns the simulation of a flexible beam modeled using a discrete flexible approach, in which the continuous structure is approximated by a chain of 10 rigid bodies interconnected through multidirectional viscoelastic elements (Figure 7.13b). This formulation enables an efficient representation of the global deformability of the system while maintaining compatibility with a constraint-based multibody solver. The elastic formulation of the system is defined through a generalized stiffness matrix $\mathbf{K}$

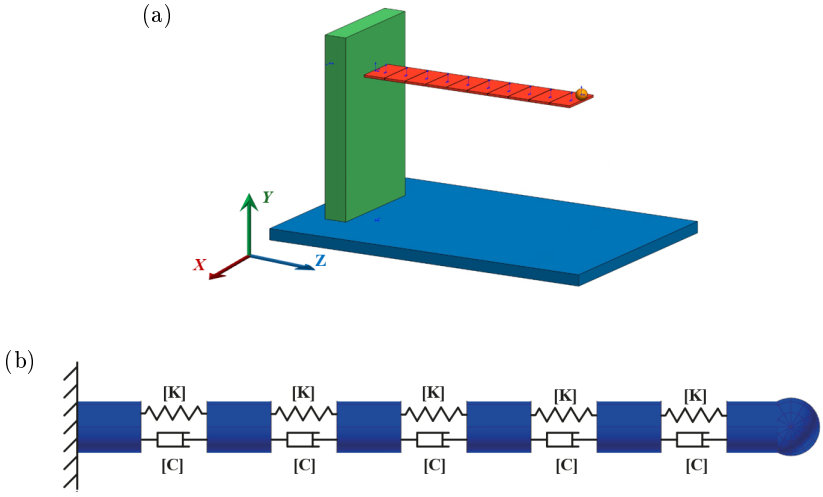


Figure 7.13: (a) Unity left-handed reference frame of the flexible beam in the Unity environment, (b) discrete flexible beam representation.

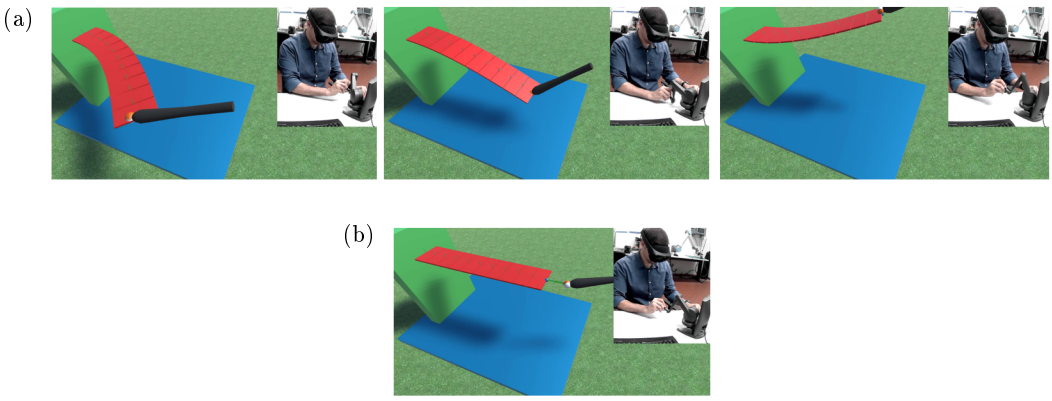


Figure 7.14: (a) Deformation of the flexible beam through haptic interaction in a VR environment, (b) virtual handle used for transmitting the interaction to the multibody system.

of the form:

$$\mathbf{K} = \begin{bmatrix} k_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & k_{22} & 0 & 0 & 0 & k_{26} \\ 0 & 0 & k_{33} & 0 & k_{35} & 0 \\ 0 & 0 & 0 & k_{44} & 0 & 0 \\ 0 & 0 & k_{53} & 0 & k_{55} & 0 \\ 0 & k_{62} & 0 & 0 & 0 & k_{66} \end{bmatrix}, \tag{7.27}$$

accompanied by a damping matrix $\mathbf{C}$ defined locally on the interconnections between the individual beam segments. In the present case, only flexural deformations were considered, enforcing high stiffness values in the directions associated with tension, compression, and shear, obtained through equivalent spherical constraints between beam segments.

The interconnections between segments are implemented through a combination of spherical joints and bushing elements. In particular, spherical joints eliminate relative

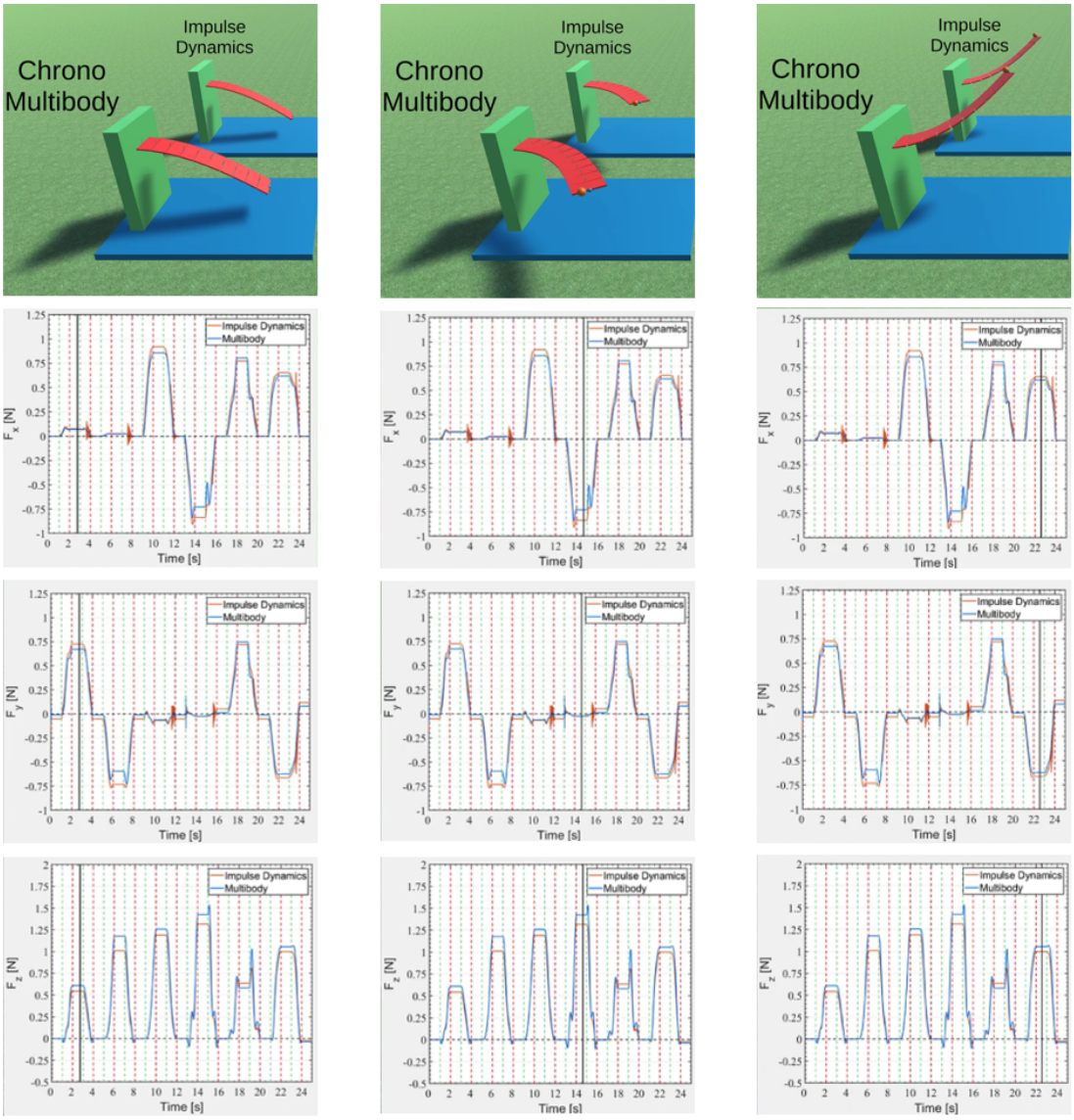


Figure 7.15: Comparison between Chrono-Unity and Sequential Impulse of the haptic feedback force applied to the user during manipulation of the beam end, expressed in the reference frame of Figure 7.13a.

translations between bodies while allowing only relative rotations (three degrees of freedom), as described in Appendix D.10. In parallel, bushing-type elements, implemented via the *UChLinkBushing* script, introduce the stiffness and damping required to model the distributed flexural behavior of the structure.

The forces returned by the multibody model and sent to the haptic device, and therefore perceived by the user, are expressed with respect to the right-handed Unity reference frame shown in Figure 7.13a.

Regarding haptic interaction, the virtual handle attachment point was placed at the free end of the beam. As illustrated in Figure 7.14b, the virtual handle behaves as a

compliant coupling element between the haptic device and the attachment point on the structural model, while Figure 7.14a shows a representative example of the deformation induced during manipulation in the immersive environment.

For the comparative evaluation of the two solvers, a prescribed motion of the attachment point was imposed. In particular, twelve displacement configurations were considered, each applied for 1-second intervals and separated by equal-length steady-state phases. The results of the three components of the haptic feedback force are reported in Figure 7.15. In this case, discrepancies between the two numerical approaches are more pronounced than in previous cases, particularly under large deformations and strong elastic gradients, where the approximate nature of the sequential impulse method introduces numerical oscillations and more significant transient peaks.

7.2 Real-time structural analysis

Recent developments in numerical simulation have enabled the use of high-fidelity multiphysics models even in interactive applications, fostering the creation of digital twins capable of supporting design, validation, monitoring, and training activities in real time. However, when the problem involves deformable structural components, thermomechanical systems, or more generally phenomena described by partial differential equations, the direct solution of finite element models becomes incompatible with the temporal requirements of immersive applications.

Unlike the real-time dynamic analysis discussed in the previous chapter, in which the problem is reduced to the integration of the dynamics of constrained multibody systems, *real-time structural analysis* deals with continuous systems whose state is described by spatially distributed variables such as displacements, strains, stresses, or temperature fields. The discretization of such problems using the finite element method generally leads to models with a very large number of degrees of freedom, whose solution requires computational times that are incompatible with real-time execution. In this context, the direct use of high-fidelity numerical solvers is not a viable solution, making it necessary to adopt simplified numerical models that preserve the essential physical behavior of the system.

The most widely adopted approach consists in the use of *Reduced-Order Models* (ROMs), which provide a compact representation of the original finite element model. The goal of model order reduction is to drastically decrease computational complexity while preserving, within a controlled error, the main dynamic and structural characteristics of the system [17]. From a mathematical standpoint, this reduction is achieved by projecting the high-fidelity model equations onto a significantly lower-dimensional subspace, appropriately constructed to represent the dominant modes of the system response [238]. The resulting reduced model can thus be solved in times compatible with interactive applications while maintaining high accuracy in predicting the structural response. Figure 7.16 shows an augmented reality digital twin of a structural component based on ROM.

ROMs are particularly effective in problems requiring repeated simulations under varying operating conditions, geometric parameters, or loading scenarios, as occurs in parametric analyses, structural optimization, control, and digital twins [239, 240]. In

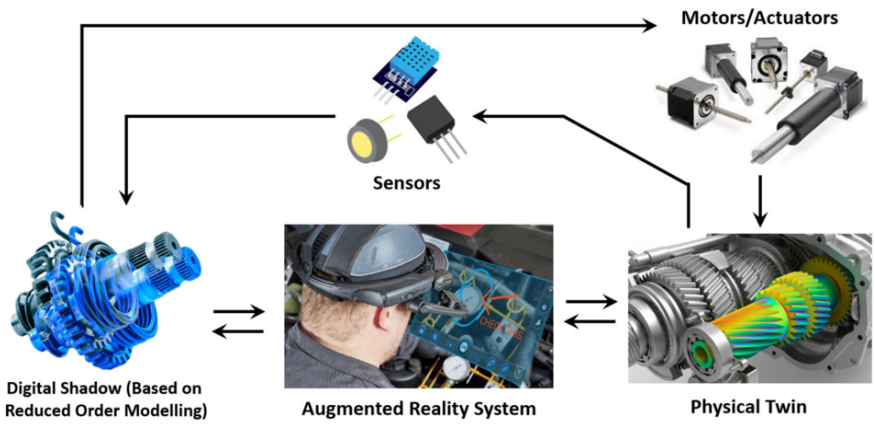


Figure 7.16: Interaction among digital shadows (based on ROM), sensors, actuators and augmented reality interface for achieving an augmented reality digital twin of a structural component.

such contexts, the reduced model acts as a *digital shadow*, enabling the reproduction of the structural behavior with a computational cost compatible with real-time applications.

The construction of a reduced model generally follows a workflow divided into an offline phase and an online phase. In the offline phase, the finite element model and the parameters of interest are defined and subsequently sampled using appropriate *Design of Experiments* (DOE) techniques [241]. For each parameter configuration, high-fidelity simulations are performed, generating a set of solutions, known as *snapshots*, which are used to construct the reduced basis through projection techniques [242]. Among these, *Proper Orthogonal Decomposition* (POD) is one of the most widely used methods due to its ability to identify an optimal orthogonal basis that captures most of the system energy using a limited number of modes [243]. Parametric and nonlinear extensions of POD also allow addressing problems characterized by large deformations, geometric variations, and multiphysics couplings [244].

Once the reduced basis has been constructed, the problem consists in efficiently determining the coefficients that describe the solution as the design parameters vary. For instance, this stage can be addressed using *Gaussian Process Regression* (GPR), a probabilistic regression technique that allows interpolation of reduced coefficients within the parameter space using a relatively small number of training simulations. In addition to providing a continuous estimate of the structural response, GPR yields an explicit measure of the uncertainty associated with the prediction, which is particularly valuable in engineering applications. Compared to other machine learning approaches such as deep neural networks, it requires fewer training samples, exhibits a lower tendency to overfitting, and maintains a fully interpretable mathematical formulation. The following illustrates a complete ROM framework.

7.2.1 Reduced order model

Reduced-order modelling reduces the computational complexity of a full-order, high-fidelity mathematical model in numerical simulations whilst preserving the expected fidelity within a satisfactory error. ROMs can thus simplify analysis and control de-

sign with applications in all areas of mathematical modelling, especially for exploring many configurations and different scenarios [245]. Many modern mathematical models of real-life systems pose challenges when utilized in numerical simulations because of excessive complexity and massive size. Model order reduction targets to decrease the computational complexity of such problems, especially for very complex models such as multi-physics, fluid-dynamics, multi-scale and structural simulations or, in general, for non-linear complex scenarios. By a reduction of the model's related dimension (or degrees of freedom), an approximation of the original system is introduced. ROMs are beneficial in applications where the complete full-order model cannot be used to perform numerical simulations due, for example, to computational resource limitations or the requirements for real-time simulations [242, 246]. To be applicable to real-world problems, ROMs often require a moderate approximation error in estimating the results, preserving the full-order model characteristics and the use of computationally efficient and robust algorithms [247, 248]. In the field of structural mechanics, which is the main topic of the paper, creating a reduced model follows a set series of steps: definition of the system geometry, creation of a set of simulation points, namely a design of experiments (DOE), execution of finite element structural analysis at each DOE point, synthesis of the mathematics underlying the ROM. A detailed workflow of the steps to perform to create a generic reduced model is shown in Figure 7.17.

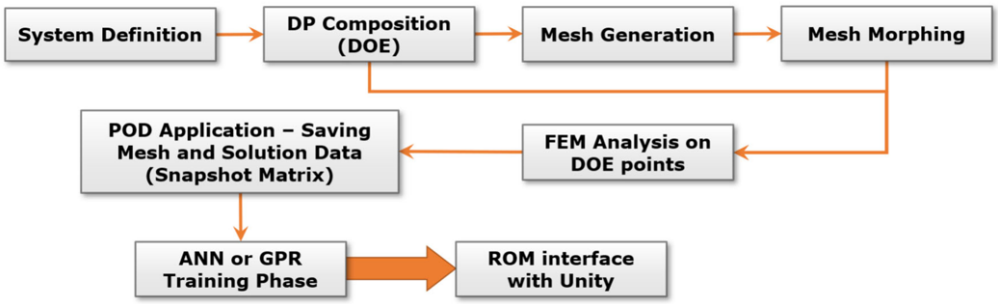


Figure 7.17: Procedural steps for generating the complete reduced-order model

The first step involves the definition of the system's geometry and the main boundary conditions that are also intrinsically defined and represent the major constraints of the problem to be analysed. The so-called even extremes, or DOE boundary conditions, are also determined in this phase [249, 250]. They usually correspond to the general operating outer limits of the component (e.g. amplitude of applied loads) and the desired design conditions (e.g. maximum and minimum dimensions). They also represent the domain limit where subsequent simulations are performed [251]. To create a reduced model, the most important thing is to note that the model itself exists as a simplified version of a more complex system; it is, therefore, crucial to build the ROM using computational time-saving logic. To do so, for example, the full factorial plan of a DOE may be reduced using a *Latin Hypercube Sampling* (LHS) [241]. LHS is a statistical method for generating a near-random sample of parameter values from a multidimensional distribution (Figure 7.18). It differs from random sampling methods because compared to the latter, where new sample points are generated randomly without considering the previously generated ones, it is formed by a square grid containing sample positions, which is a Latin square

if and only if there is only one sample in each row and each column. A Latin hypercube is therefore obtained by extending this concept to an arbitrary number of dimensions, whereby each sample is the only one in each axis-aligned hyperplane containing it [252, 253].

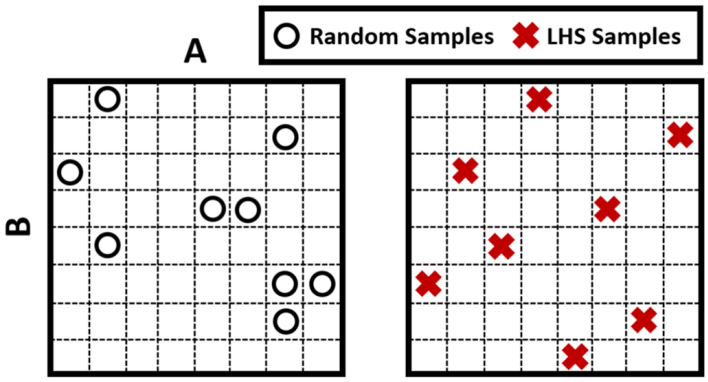


Figure 7.18: Comparison between samples produced via Random Sampling and Latin Hypercube Sampling for two generic parameters A and B, each with eight levels.

Once the points to simulate are defined, a parametric FEM model is built. It is used to compute the response of the system, varying the design parameters within the DOE domain. The results, such as stresses, strains, and displacements, are organized into matrices. To perform this step and to facilitate the subsequent postprocessing phases, it is preferable to use a finite element solver that allows easy access to the mesh data and simulation results. The proposed method, compared to a classical parametric study, differs in the changes applied to the system geometry: the first one allows to perform geometrical modifications on the mesh itself, without the need to rebuild the model or the mesh, and then the reduced model updates the simulation results automatically; the latter needs to rebuild the mesh and the model after alterations in the geometry, and subsequently must follow a new run of the simulation to obtain the results. To do this, the mesh morphing technique is adopted, which consists of applying shape functions on the nodes of the mesh to modify their position as desired without having to go through a re-mesh of the model. This feature is fully integrated into the ROM creation workflow. Mesh morphing not only simplifies the computation, avoiding the burden of remeshing, but is able to preserve the same topology and connectivity of the nodes and elements, simplifying the subsequent post-processing of results and the integration in the visualization environment. The mesh morphing phase can be achieved using radial basis functions (RBF) [254]: an interpolator $\mathbf{s}$ is provided to have an approximate smooth function in the same space of data, which is given as scattered scalar values at a sequence of points $\mathbf{x}_i$ in the space $\mathbb{R}^n$. Its value at a generic $\mathbf{x}$ position is given by:

$$\mathbf{s}(\mathbf{x}) = \sum_{i=1}^{N} \omega_i \varphi_i(\mathbf{x}) \tag{7.28}$$

where N is the number of points where data are specified, called source points (in case of mesh morphing, these points are the mesh boundary points); $\varphi_i(\mathbf{x})$ is the so-called

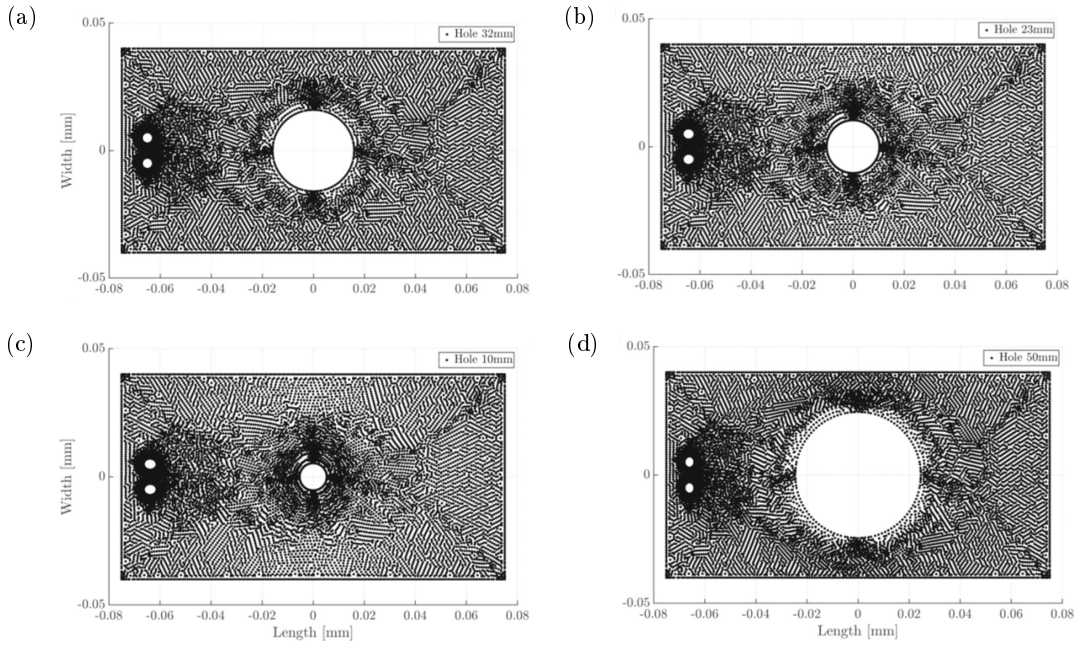


Figure 7.19: Example of mesh nodes position after mesh morphing phase for various configurations: (a) is the baseline mesh; (b) is a random hole diameter; (c) is the minimum hole diameter, and (d) the maximum one.

radial basis function referred to the i-th target point, namely a Euclidean distance scalar function between source and target points, that is $\varphi_i(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_i\|$; ω_i are weights of the radial basis for three spatial coordinates of the generic point.

To get the mesh nodes' new position following a movement (for example, a parametric modification of the geometry), the RBF system in Eq.7.28 must be solved for all the mesh nodes given a series of boundary conditions. The first step is to obtain the weights ω_i: for their computation, a linear system of equations, whose order is equal to the number of source points introduced, needs to be solved; subsequently, the resolution of the above system for all the nodes can be completed.

The new position of the generic node is then obtained as follows:

$$\mathbf{x}_{new} = \mathbf{x} + \mathbf{s}(\mathbf{x}) \tag{7.29}$$

where $\mathbf{x}_{new}$ is the new position of the i-th node (with respect to the three spatial coordinates) and $\mathbf{x}$ is the node position before the mesh modification. An example of node displacement in the mesh due to a geometric modification for a plate with a hole in the middle is shown in Figure 7.19.

At this point, the FEM model takes the new mesh coordinates, modified with RBF morphing under the DOE indications, as new simulation inputs and the generated results are stored as columns of the so-called *snapshot matrix*. This matrix is a large dimension array containing the desired results from the DOE simulations as a series of "snapshots" of the system's structural state. The process is then iterated for all the m parameters combinations in the DOE map, and the complete results matrix is stored for the sub-

sequent mathematical manipulations. The next step uses mathematical techniques to actively reduce the model complexity into a simplified structure by decreasing the number of variables that describe the system's behaviour. To do so, several methodologies have been proposed through the years, including the *reduced basis methods* [244,255,256].

The methodology used in this study adopts the *proper orthogonal decomposition* (POD) method to identify the principal modes of response and encode the system to a reduced dimension that intrinsically characterizes the whole system. In fact, this method has shown excellent efficiency and robustness in the process, as well as high reliability of the results produced [243,257]. To achieve an efficient reduction, the POD method uses the *singular value decomposition* (SVD) to decompose the results matrix and to extract the condensed modes. This methodology requires the following steps to be performed: firstly, the snapshots are composed in a matrix $\mathbf{T}$. It has a dimension of $n \times m$, where n is the number of results on each node of the mesh and m is the number of simulations performed in the DOE.

$$\mathbf{T} = [\mathbf{t}_1 \ \dots \ \mathbf{t}_m]_{n\times m} \tag{7.30}$$

where $\mathbf{t}_i$, $i = 1, \dots, m$ are the snapshot vectors containing the solution for a single DOE simulation.

Next, the matrix $\mathbf{T}$ is reduced through the SVD decomposition into three different matrices, as follows:

$$\mathbf{T}_{n\times m} = \mathbf{U}_{n\times n}\mathbf{S}_{n\times m}\mathbf{V}^T_{m\times m} \tag{7.31}$$

where $\mathbf{U}$ is an $n\times n$ orthonormal matrix containing the left singular vectors; $\mathbf{S}$ is an $n\times m$ pseudo-diagonal and semi-positive definite matrix with diagonal entries containing the singular values σ_i and $\mathbf{V}$ is an $m \times m$ orthonormal matrix containing the right singular vectors, with $\mathbf{V}^T$ its conjugate transpose.

The snapshot matrix can then be approximated by a reduced matrix $\mathbf{T}_{red}$, obtained from the SVD through a column and row truncation:

$$\mathbf{T} \cong \mathbf{T}_{red} = \mathbf{U}_r\mathbf{S}_r\mathbf{V}^T_r \tag{7.32}$$

where r is the column up to which the truncation of the matrix $\mathbf{U}$ is carried out.

The choice of where to truncate the data matrix is one of the most important and delicate decisions when using the SVD. Several factors influence the choice, including specifications on the desired rank, the noise magnitude, and the distribution of the singular values. Often, the rank r is chosen so that it captures a pre-determined amount of the variance or energy in the original data. The so-called energy truncation method is a more quantitative and objective way to find the optimal number of singular values for SVD. It involves calculating the ratio of the cumulative sum of the squared singular values to the total sum of the squared singular values. This ratio represents the proportion of the data variance that is captured by the selected singular values. The energy method requires setting a threshold for the desired level of variance, such as 90% or 95%, and choosing the smallest number of singular values that satisfy this threshold.

The cumulative energy is therefore defined as:

$$\boldsymbol{\varepsilon} = [e_1 \ e_2 \ \dots \ e_n]^T \tag{7.33}$$

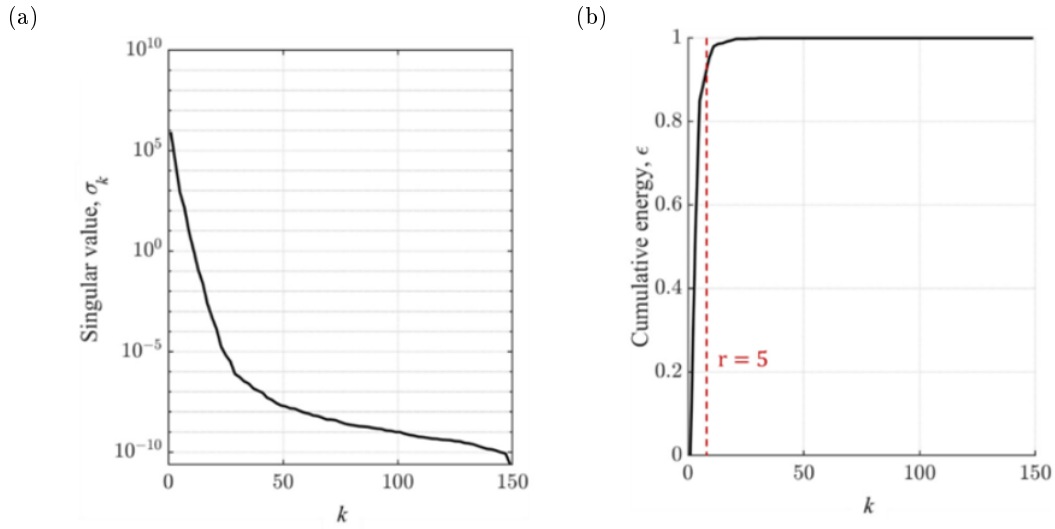


Figure 7.20: Singular values σ_i of the matrix $\mathbf{S}$ (a) and cumulative energy retained by the first k modes (b) for the cantilevered plate with a central hole subjected to transverse loading at the free end.

where the i-th term e_i is defined as:

$$e_i = \frac{\sum_{j=1}^{i} \sigma_j}{\sum_{k=1}^{n} \sigma_k} \tag{7.34}$$

where the σ_i are the different singular values on the diagonal of matrix $\mathbf{S}$, and the term $\sum_{j=1}^{i} \sigma_j$ is the cumulative sum for the σ values until the one in the i-th position.

Therefore, the number of truncation modes can be selected from the computed energy as the value which grants a determined amount of energy in the data matrix. The Eq. 7.32 describes the condensed behaviour of the system, considering that r represents the number of principal modes of $\mathbf{U}$ that are sufficient to approximate the solution within a given tolerance. It can easily be seen in Figure 7.20 how the whole set of modes defined by $\mathbf{U}$ is not equally used to compose the solution. The error committed in the approximation is contained as long as the reduced number of modes r allows the resolution of the system to be defined while maintaining the cumulative energy close to its asymptotic value (value 1 in Fig. 7.20 corresponds to the maximum energy) [243].

Defined the solution vector $\mathbf{t}_{set}$ for a fixed set of base parameters, the approximate solution can be found as the linear combination of the weighted modes:

$$\mathbf{t}_{set} = \alpha_1 \mathbf{u}_1 + \cdots + \alpha_r \mathbf{u}_r \tag{7.35}$$

where α_i, $i = 1, \ldots, r$ are the mode weights, and $\mathbf{u}_i$, $i = 1, \ldots, r$ are the snapshot vectors obtained as the columns of the reduced matrix $\mathbf{U}_r$. From the inversion of Eq. 7.35, the weights α_i can be computed, as $\mathbf{t}_{set}$ and $\mathbf{U}_r$ are both known. Regarding $\mathbf{t}_{set}$, it represents one of the simulations made in the DOE phase for a predefined combination of input parameters: iterating the weights resolution for all the m results columns in the matrix $\mathbf{T}$, the modes weights matrix can be obtained.

Afterwards, a surrogate model is created, applying interpolation or regression methods on principal modes and this procedure properly defines the ROM. Several methods are available in the literature to predict a certain output given a set of inputs, ranging from nonparametric regressions to neural networks and machine learning [258]. The method to be adopted depends on the requirements of the problem to be analyzed: a complex system with more than a dozen independent parameters and a large amount of data can benefit from the use of neural networks (although they require more time and computational efforts for the training phase). In contrast, a more straightforward system can be analyzed with regression methods that are more easily manageable.

For this last step, the most common method used to predict the results of the ROM is the *gaussian process regression* model (GPR). The *gaussian process* (GP) is a nonparametric supervised learning method based on training samples adopted to solve regression and probabilistic classification problems. A GP-based regression model allows the use of a probabilistic approach to prediction by providing the mean and standard deviation as output when predicting [259–261]. An illustration of GPR values prediction for the considered case study is reported in the following sections.

Concerning the general GPR problem, it can be stated that for a gaussian process f observed at coordinates $\mathbf{w}$, one sample from a multivariate GP distribution with a dimension equal to the number of observed coordinates n is represented by the vector of values $f(\mathbf{w})$. Therefore, under the assumption of a zero-mean distribution, the following equation can be derived for the normal probably distribution:

$$f(\mathbf{w}') \sim \mathcal{N}(\mathbf{0}, \mathbf{K}(\boldsymbol{\theta}, \mathbf{w}, \mathbf{w}')) \tag{7.36}$$

where the term $\mathbf{K}(\boldsymbol{\theta}, \mathbf{w}, \mathbf{w}')$ is the covariance matrix between all possible pairs of known coordinates $(\mathbf{w}, \mathbf{w}')$ for a given set of hyperparameters $\boldsymbol{\theta}$ [262] and $\mathcal{N}(\mathbf{0}, \mathbf{K}(\boldsymbol{\theta}, \mathbf{w}, \mathbf{w}'))$ states the normal distribution.

Having specified $\boldsymbol{\theta}$, making predictions about unobserved values $f(\mathbf{w}^*)$ at coordinates $\mathbf{w}^*$ is then only a matter of drawing samples from the predictive distribution:

$$p(f(\mathbf{w}^*) \mid \mathbf{w}^*, f(\mathbf{w}), \mathbf{w}) = \mathcal{N}(f(\mathbf{w}^*) \mid \mathbf{A}, \mathbf{B}) \tag{7.37}$$

where the posterior mean estimate $\mathbf{A}$ is defined as:

$$\mathbf{A} = \mathbf{K}(\boldsymbol{\theta}, \mathbf{w}^*, \mathbf{w})\mathbf{K}(\boldsymbol{\theta}, \mathbf{w}, \mathbf{w})^{-1} f(\mathbf{w}) \tag{7.38}$$

and the posterior variance estimate $\mathbf{B}$ is defined as:

$$\mathbf{B} = \mathbf{K}(\boldsymbol{\theta}, \mathbf{w}^*, \mathbf{w}^*) + \mathbf{K}(\boldsymbol{\theta}, \mathbf{w}^*, \mathbf{w})\mathbf{K}(\boldsymbol{\theta}, \mathbf{w}, \mathbf{w}')^{-1}\mathbf{K}(\boldsymbol{\theta}, \mathbf{w}^*, \mathbf{w})^T \tag{7.39}$$

where $\mathbf{K}(\boldsymbol{\theta}, \mathbf{w}^*, \mathbf{w})$ is the covariance between the new coordinate of estimation $\mathbf{w}^*$ and all the other observed coordinates $\mathbf{w}$ for a given hyperparameter vector $\boldsymbol{\theta}$, $\mathbf{K}(\boldsymbol{\theta}, \mathbf{w}, \mathbf{w}')$ and $f(\mathbf{w})$ are defined as before and $\mathbf{K}(\boldsymbol{\theta}, \mathbf{w}^*, \mathbf{w}^*)$ is the variance at point $\mathbf{w}^*$ as dictated by $\boldsymbol{\theta}$. The covariance matrix $\mathbf{K}$ can thus be expressed as:

$$\mathbf{K}(\boldsymbol{\theta}, \mathbf{w}, \mathbf{w}) = \begin{bmatrix} k(\boldsymbol{\theta}, \mathbf{w}_1, \mathbf{w}_1) & \dots & k(\boldsymbol{\theta}, \mathbf{w}_1, \mathbf{w}_n) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{\theta}, \mathbf{w}_n, \mathbf{w}_1) & \dots & k(\boldsymbol{\theta}, \mathbf{w}_n, \mathbf{w}_n) \end{bmatrix} \tag{7.40}$$

where the terms $k(\boldsymbol{\theta}, \mathbf{w}_i, \mathbf{w}_j)$ are called kernel function and defines the GP regression logic. For each mode weight α_i, $i = 1, \ldots, r$ a covariance matrix $\mathbf{K}$ is needed.

As seen in the above Eqs. 7.36, 7.37, 7.40, defining the covariance function for GPR completely defines the process' behaviour. Covariance functions are crucial elements which determine the shape of the prior and posterior of the GP. They encode the assumptions on the function being learned by defining the "similarity" of two data points combined with the assumption that similar data points should have similar target values. Two categories of kernels can be distinguished: stationary kernels depend only on the distance of two datapoints and not on their absolute values and are thus invariant to translations in the input space, while non-stationary kernels also depend on the specific values of the datapoints.

The most used function for unseen results prediction is the *automatic relevance determination* (ARD) squared exponential, which has shown a robust and reliable prediction of results in generic applications of modest complexity [263]. It is defined as:

$$k(\boldsymbol{\theta}, \mathbf{w}_i, \mathbf{w}_j) = \sigma_f^2 e^{\left[-\frac{1}{2}\sum_{m=1}^{r} \frac{(w_{im}-w_{jm})^2}{\sigma_m^2}\right]} \tag{7.41}$$

where σ_f is the signal standard deviation, σ_m is the characteristic length scale, which define how far apart the input values $\mathbf{w}_i$ can be for the response values to become uncorrelated. In this formulation, a separate length scale σ_m is used for each predictor m, whose number is equal to the number of mode weights selected.

With respect to the POD formulation, the GPR method can be used to predict the α_i mode weights (GP predictors) for an unseen combination of DOE parameters (input values $\mathbf{w}_i$). The values to be predicted (α_{pred}) can then be obtained from Eq. 7.34 as:

$$\boldsymbol{\alpha}_{pred} = \mathbf{K}(\mathbf{D}_{new}, \mathbf{D}_{train})\mathbf{K}(\mathbf{D}_{train}, \mathbf{D}_{train})^{-1}\boldsymbol{\alpha} \tag{7.42}$$

where $\mathbf{D}_{train}$ is the DOE matrix parameters used to compute the simulations in $\mathbf{T}$, $\mathbf{D}_{new}$ is the matrix containing the DOE parameters of the new unseen case and $\boldsymbol{\alpha}$ is the weights vector for all the simulations.

The new solution $\mathbf{T}_{new}$ for all the mesh nodal parameters is composed by the vectors $\mathbf{t}_{new}$ for each node; them can thus be obtained as:

$$\mathbf{t}_{new} = \alpha_1\mathbf{u}_1 + \cdots + \alpha_r\mathbf{u}_r \tag{7.43}$$

Chapter 8

Digital twin case studies

In this chapter, several digital twin case studies are presented, developed by leveraging the methodologies introduced and integrated in the preceding chapters. First, a digital twin for VR-based training in the use of a sensorized upper-limb prosthesis is analysed. Subsequently, a dynamic digital twin enabling bidirectional data exchange between physical sensors and an AR headset is presented. Finally, two structural digital twin applications are reported: the first is based on the framework introduced in Section 7.2, while the second consists of an AR-based digital twin developed for monitoring the displacement field of a vehicle chassis.

8.1 Digital twin for VR prosthesis training

The human hand represents a highly sophisticated system from both a functional and motor-control perspective. For this reason, research in the field of prosthetics has long pursued the goal of restoring, at least in part, the capabilities lost following upper-limb and/or forearm amputations, through increasingly advanced devices equipped with sensors [264–266]. Even partial recovery of the original functionalities can significantly improve the quality of life of amputees. In this context, current upper-limb prostheses, particularly those including articulated hand components, are often controlled through the interpretation of myoelectric signals acquired from the residual limb [267–270]. This type of interface enables users to perform different grasping and manipulation tasks, bringing prosthetic behavior closer to that of a natural hand. However, in general, as device functionality increases, so does the time and effort required for users to learn how to operate it effectively [271, 272]. In particular, three main critical issues can be identified: the need for the user to adapt to a "new limb" that may feel unnatural even from a psychological perspective; the long and often demanding training process required; and the difficulty in ensuring accurate recognition of the user's motor intentions for proper prosthetic hand control.

In this scenario, a particularly relevant role can be played by preliminary training pathways conducted in safe and controlled environments, such as virtual simulations [273–276]. These environments not only facilitate learning of prosthesis control but also allow exploration and definition of possible functional customizations before the physical device is fabricated and fitted to the residual limb. Within this scientific framework, the

use of virtual reality and the construction of digital twins can represent very useful tools to improve training, even for the use of complex and advanced prostheses.

For this reason, a new interactive training methodology has been developed specifically to allow amputees to become familiar with a sensorized upper-limb prosthesis, based on the advantages offered by virtual reality environments and numerical simulation models, combining computer graphics, multibody simulation, computer vision, sensors, and virtual reality.

The main objective is to integrate different computational methodologies to build a complete digital twin to be embedded in a virtual reality environment. This environment includes a set of standardized exercises aimed at reducing the training time required for the use of a sensorized prosthesis. Although the procedure was specifically developed and tested for the iHannes prosthesis, developed by the INAIL Prosthetic Center and the Italian Institute of Technology (IIT) [277,278], the proposed algorithms are generally applicable to any type of sensorized upper-limb prosthesis.

This work can be divided into three main parts: the sensorized upper-limb prosthesis with myoelectric input; the digital replica of the complex system (i.e., the digital twin) of the prosthesis, with particular attention to algorithms, numerical models, and implementation procedures; and training tests and usability considerations. The specific context and state of the art of each integrated methodology are reported in Section 3 and divided according to each digital twin component [121].

8.1.1 The sensorized prosthesis

In general terms, an upper-limb prosthesis, equipped with an interface (socket) that connects the amputated anatomical segment to an anthropomorphic structure, aims to replicate as many natural functions of the missing limb as possible [279,280]. Sensorized prostheses may include sensors (epidermal or implanted) to acquire myoelectric signals from the residual limb, combined with actuators capable of controlling the different configurations of the anthropomorphic hand [281].

The acquired myoelectric signals are processed using classification algorithms to identify which posture the user's neuromuscular system is expressing. Different strategies can be adopted to train the classification system, including machine learning techniques [282], in order to recognize intended gestures such as pinch grip, tripod grasp, and fist.

Downstream of the classifier, the resulting signal is used to actuate the prosthesis accordingly, as already highlighted in previous works [283].

8.1.2 Digital twin and physical twin integration

The core of the proposed methodology is based on the development of a digital twin of the sensorized upper-limb prosthesis.

In this context, the goal is to improve the dexterity of sensorized upper-limb prostheses, evaluating their long-term adoptability, while also enabling customization and optimization of classifiers, electronic components, and mechanical parts of the system before actual device use.

The virtual reality approach immerses the user in an environment where the prosthesis is highlighted, allows verification that the interfaced electronics correctly interpret user

intent, and enables a series of manipulation and dexterity exercises to be performed an arbitrary number of times.

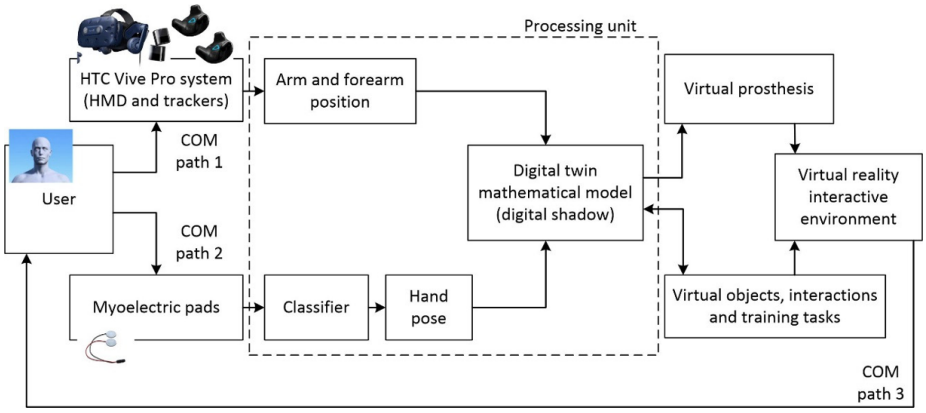


Figure 8.1: Communication paths between the physical and the digital twins in the training environment. The user communicates with the digital twin through the virtual reality trackers (COM Path 1) and the myoelectric sensors (COM Path 2). The digital twin, through the mathematical model, returns an updated virtual environment by means of the head-mounted display (COM Path 3).

Figure 8.1 shows the scheme of the three communication channels (COM paths) between DT and PT. The first channel transmits real-time information related to the position of the arm in space with respect to the user's head, in order to correctly align the virtual prosthesis with the upper limb. The second channel concerns communication from the myoelectric sensors for the evaluation of hand grasp postures. Both of these channels flow from the physical twin to the digital twin, while the third channel operates in the opposite direction to provide visual feedback to the user. A data processing unit is located between the three channels with the aim of synchronizing information, ensuring real-time rendering and scene realism, and consequently guaranteeing successful training.

From a hardware perspective, the HTC VIVE Pro 1 and HTC VIVE Tracker 2.0 were adopted. From a software perspective, the digital twin and virtual scene were developed on the Unity platform.

During training, the user wears the headset and two trackers attached via elastic straps (Figure 8.2). One tracker is firmly attached to the arm and the other to the residual forearm. In addition, the user wears myoelectric sensors (electrodes) and connects the entire system to the processing workstation. The DT acquires information from the PT sensors and updates the immersive scene through a specific computational procedure and algorithm (described in the next subsection). In the virtual environment, the virtual arm representing the sensorized prosthesis is positioned according to the physical sensors. The virtual scenario is complemented by objects classified as interactable entities, capable of contacting and interacting with the virtual hand. Instructional cues are also present in the scene to guide the user during training tasks.

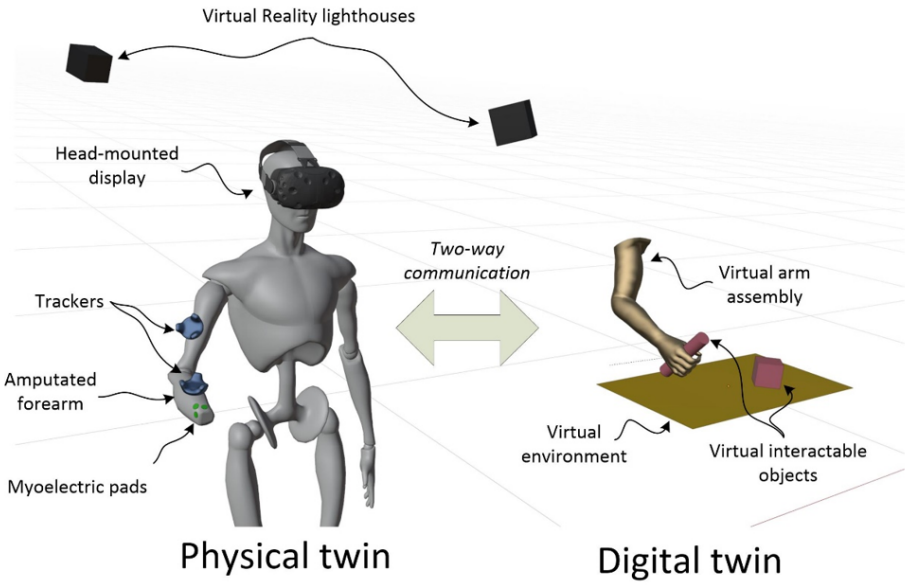


Figure 8.2: Physical and digital twins of the VR training environment. The physical twin includes the kinematic sensors (trackers), the myoelectric sensors, and the head-mounted display. The digital twin is driven by the multibody dynamic model able to simulate real-time physics including collisions.

8.1.2.1 Virtual prosthesis dynamic model and trackers interface

The dynamic model aims to accurately reproduce the behavior of the virtual prosthesis based on data acquired from trackers and epidermal myoelectric sensors. For clarity, it can be divided into two functional subsystems. The first subsystem reproduces the dynamics of the arm and forearm to which the prosthesis is attached, using an inverse dynamics model based on tracker information. The second subsystem is dedicated to simulating the motion of the prosthetic fingers (Figure 8.3).

The first subsystem consists of four rigid bodies. Two of them represent the main anatomical segments, i.e., the arm and forearm, while the other two are dummy bodies used to model the presence of trackers. The arm and forearm are connected via a spherical joint located at the elbow. Although the elbow joint exhibits a more complex behavior from an anatomical standpoint than a simple spherical joint, characterized by articulated elasto-kinematic properties [284], this simplification reduces the complexity of constraint equations without compromising model reliability. Moreover, this choice is easily adaptable to the tracker configuration used to reconstruct relative motion between body segments. The two dummy bodies are not connected to the system via kinematic pairs, but only through elastic bushing elements. This solution effectively transfers tracker information while keeping the model stable and computationally efficient. The use of bushing elements represents a compromise in modeling overconstrained systems, as shown in [285].

Indeed, directly transferring tracker information to the corresponding arm and forearm bodies (e.g., by enforcing equality of position and orientation constraints) leads to an overconstrained system, i.e., with more constraint equations than available degrees

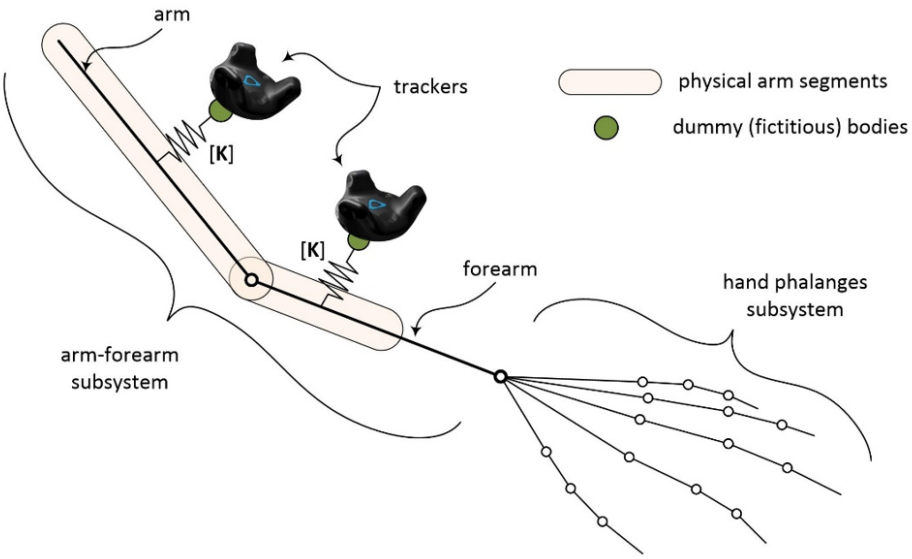


Figure 8.3: Schematic overview of the dynamic model of the arm. The locations of the anchor points of the springs connecting the trackers and the body segment are computed during initial calibration.

of freedom. This complicates numerical integration and may make the dynamic solution unstable or extremely slow. Considering that a rigid body in 3D space has 6 degrees of freedom, a system composed of two rigid bodies connected by a spherical joint has 9 total degrees of freedom. However, the constraints imposed by the trackers amount to $2 \times 6 = 12$, resulting in an overconstrained system ($12 > 9$). The use of bushing elements instead of kinematic constraints does not change the system's degrees of freedom, which remain 9. Mathematically, a bushing element between two reference frames belonging to different bodies can be expressed as:

$$\mathbf{F} = \mathbf{K}\boldsymbol{\Delta} \tag{8.1}$$

where $\mathbf{F}$ is the generalized force vector, $\mathbf{K}$ is a 6×6 stiffness matrix, and $\boldsymbol{\Delta}$ is the generalized displacement vector (linear and angular) between the two reference frames. In other words, the bushing behaves as a multidimensional elastic force. By assigning sufficiently large values to the entries of $\mathbf{K}$, the bushing behaves similarly to a rigid constraint, ensuring proper coupling between bodies. However, the penalty-based approach allows small measurement errors, tracker slippage on anatomical segments, skin/clothing artifacts, and minor calibration errors without significantly affecting system stability.

The second subsystem is composed of 20 rigid bodies representing the finger phalanges (four bodies per finger). In this case, the segments are connected via spherical joints. This subsystem represents a rigid structure with anatomical configuration but not physically present due to limb amputation. Consequently, it is not included in the inverse dynamics problem and is used solely to simulate prosthetic finger motion and interactions during grasping and manipulation tasks. The motion of these rigid bodies is not controlled by trackers or physical devices but is driven by the interpretation of myoelectric signals through a neural network-based classifier. For each recognizable gesture, a set of time

laws for joint angles between adjacent phalanges is defined and enforced via driving constraints when the classifier identifies a posture with sufficient confidence. Motion speed is calibrated based on the performance of the real prosthesis to increase realism and introduce consistent delay. In the absence of classifier updates, the subsystem behaves as a single rigid body, since imposed constraints enforce zero relative velocities. The two subsystems are finally connected via a rigid constraint between the distal end of the forearm and the proximal end of the palm. Overall, the virtual limb dynamic system consists of:

- 24 rigid bodies (4 in the first subsystem and 20 in the second);
- 17 kinematic constraints with 54 scalar equations (1 spherical joint between arm and forearm, 1 rigid constraint between forearm and palm, and 15 spherical joints between adjacent phalanges);
- 57 motion constraints (6 scalar equations for each tracker and 3 for each spherical joint between adjacent phalanges);
- 2 bushing elements (one for each tracker).

The dynamic model is constructed and solved using a rigid multibody dynamics approach, not relying on a global formulation but adapting a sequential impulse formulation. Therefore, the entire procedure—from signal acquisition to digital twin model solving and integration with the virtual reality environment—can be summarized in the diagram shown in Figure 8.4.

In summary, the proposed procedure starts with data acquisition from trackers and myoelectric sensors. From the trackers, position and orientation are extracted, while from the latter electrical potential differences are obtained and then sent to the classifier. The classifier interprets patterns and returns the most likely grasp posture. Based on this posture, joint actuation constraints are updated to move the phalanges. Tracker position and orientation data and phalange motion constraints are then sent to the sequential impulse solver, which integrates the system equations. Once the system is solved, updated positions and orientations of all rigid bodies are obtained. The arm and hand mesh are then consistently deformed using a skeleton-based skinning approach, a common technique in computer graphics [286, 287]. The updated model and all other virtual objects in the scene are then rendered in the virtual reality environment. Knowing mesh polygon positions allows computation of interpenetrations between the virtual arm and scene objects, which are then fed back into the solver for the next integration step. The entire algorithm must be completed within a single display frame to ensure synchronization between the real scene and the virtual model, guaranteeing correct alignment between digital twin and physical twin.

8.1.2.2 Arm and hand mesh skinning

Mesh skinning is a technique used in computer graphics to animate and deform a three-dimensional model [288]. The basic idea is to use a skeleton, i.e., a hierarchical set of interconnected bones, to represent the underlying structure of a character. The actual 3D

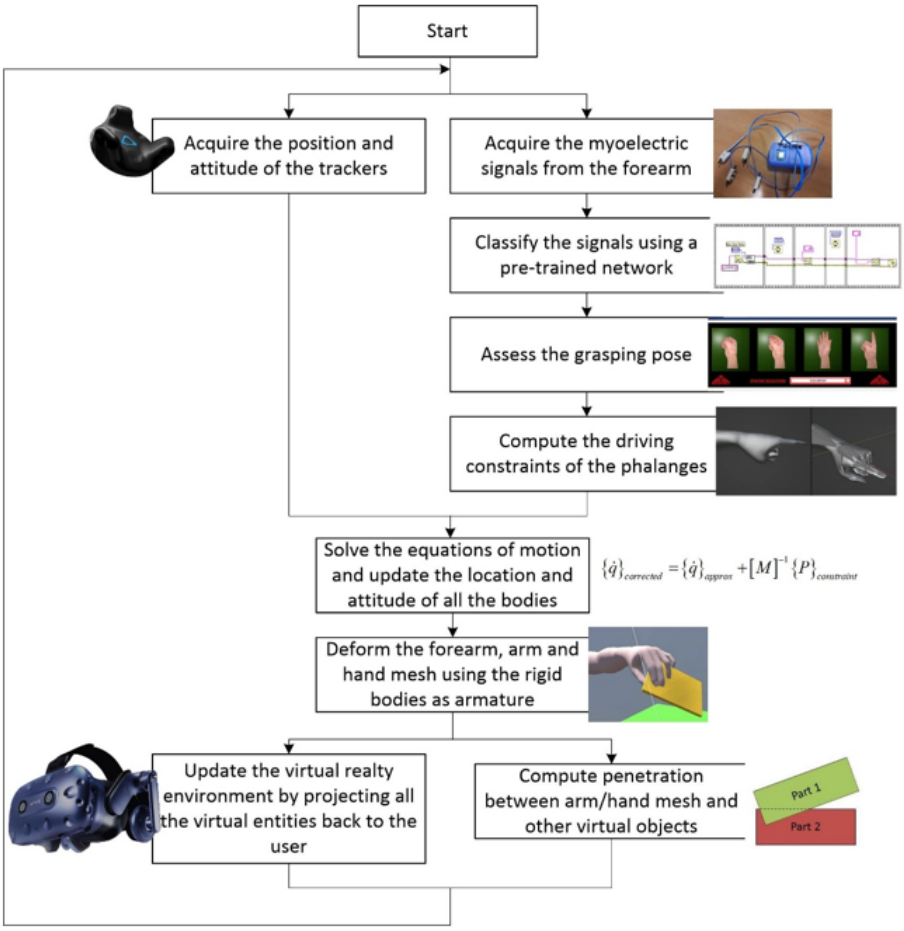


Figure 8.4: The proposed procedure to manage the dynamic behaviour of the prosthetic arm digital twin. Both communication paths from the physical sensors go to the multibody model that updates the position and attitude of all the objects in the virtual environment, according to an accurate physics simulation.

model (mesh) is then deformed according to the motion of these bones, enabling realistic and dynamic animations. When a bone moves, the mesh vertices are influenced to a greater or lesser extent depending on the assigned weights. If a bone has a weight of 1 on a mesh vertex, the vertex moves rigidly with the bone, sharing its spatial transformation. If instead a bone has a weight of 0 on a vertex, that vertex is not influenced by the motion of that bone. A weight between 0 and 1 indicates that the resulting motion of the vertex is a combination (blending) of the motion of two or more bones.

Through mesh skinning, the deformation of the model becomes smooth and highly realistic. In the digital twin, the multibody model of the arm, forearm, and hand acts as the skeleton, while the anatomical shape defines the deformable mesh (Figure 8.5). The entire deformable virtual limb is represented by a mesh composed of 4446 faces and 2506 vertices. This choice represents a good trade-off between accuracy and computational cost. Each bone is associated with a set of vertex weights; some examples of these weights are shown in Figure 8.6. In general, vertices located between two joints exhibit a weight

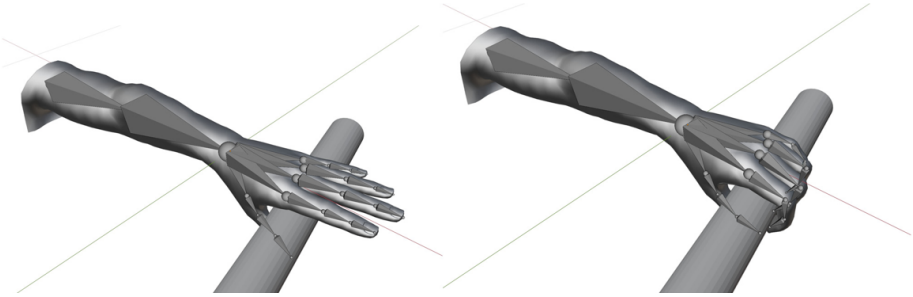


Figure 8.5: An example of mesh skinning applied to a rigid-body rig for a cylindrical grasping pose. The pyramidal bodies represent the bones of the armature (rigid bodies of the simulation model), while the anatomical mesh represents the deformable virtual character.

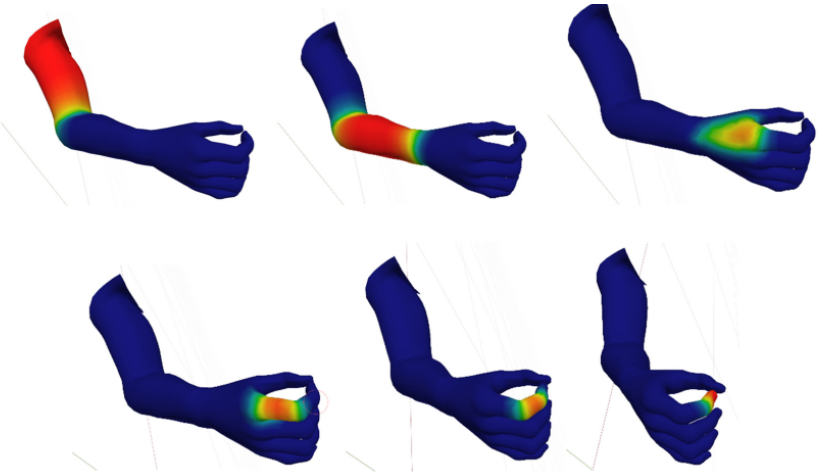


Figure 8.6: Mesh vertex weights for realistic skinning of the virtual arm. Red corresponds to a weight of 1 and blue corresponds to a weight of 0. Higher weights indicate stronger coupling between mesh motion and bone motion.

of 1 with respect to the dominant bone, while this value decreases in regions close to the joints, where the main deformations occur.

8.1.2.3 Implementation of stable grasping actions

Although contact interaction among all virtual objects in the environment is always active to enhance realism, object grasping must be made more stable through a dedicated coupling strategy. Indeed, rigid bodies whose kinematics are driven by external sources (such as the arm and forearm) may generate excessive contact impulses, making it difficult to achieve a stable grasp due to rapid motion variations. For this reason, the grasping condition is enhanced using the GAF/OAF methodology (Section 4.3.2.1). The main steps of the method are:

- when the user expresses an intention to grasp via a specific posture, the corresponding GAF is activated;
- the relative position and orientation between the activated GAF and all OAFs of the objects in the scene are evaluated;
- if the check is successful (GAF and OAF are sufficiently close and aligned), the grasp is confirmed, a constraint is imposed between the two reference frames, and the vector $\mathbf{\Phi}$ is updated accordingly;
- if the check is unsuccessful (GAF and OAF are too distant and/or misaligned), the grasp is rejected and the simulation proceeds without modifying the system equations.

8.1.3 Training tasks

Prosthesis training is carried out through:

- updating the skinned virtual arm driven by the user's movements, according to the output of the myoelectric signal classifier, similarly to what occurs in the real prosthesis;
- implementing a set of grasping and manipulation exercises to improve dexterity, based on the *Southampton Hand Assessment Procedure* (SHAP) [289], with a virtual version named VR-SHAP.

The training tasks are based on the recognition of four main hand postures: power grip, three-finger grip, pointing finger, and flat hand, as shown in Figure 8.7.

In particular, SHAP is a tool used to evaluate both impaired natural hands and prostheses. It consists of a timed assessment including 12 tasks with abstract objects and 14 activities of daily living. The functional profiles derived from execution times provide metrics to evaluate the level of (dis)ability of the subject, offering an objective assessment of function and enabling quantification of impairment within a single test rather than relative improvement measures. Moreover, SHAP allows evaluation of the effectiveness of a prosthetic terminal device and its control system, focusing on user performance and enabling monitoring of rehabilitation progress. This approach highlights functional

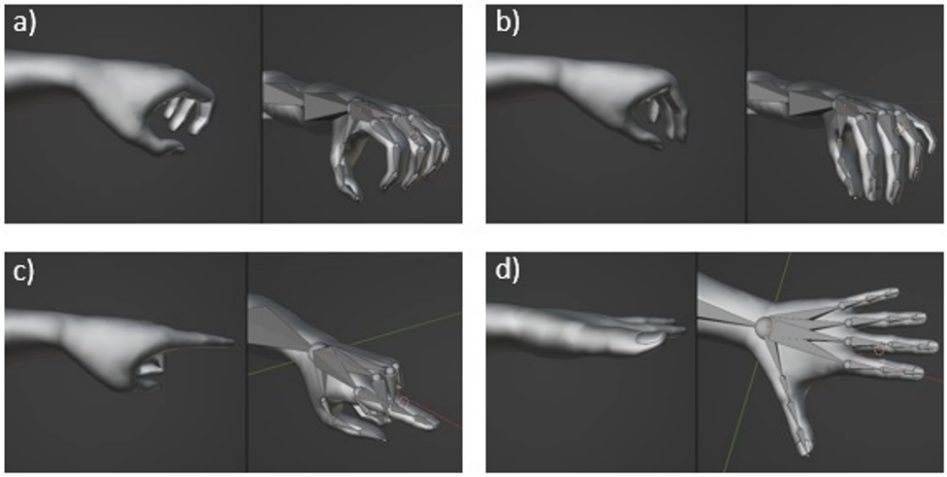


Figure 8.7: The four hand poses and the corresponding skeleton and skinned mesh: (a) power grip, (b) three-finger grip, (c) pointing finger, and (d) flat hand.

differences between different prosthetic devices and control strategies while providing a contextualized measure of hand function.

Based on the four main postures described above (recognized by the classifier), six compatible tasks were implemented, defining the procedure as the *Virtual Reality Southampton Hand Assessment Procedure* (VR-SHAP). The VR-SHAP procedures (Figure 8.8) are divided into three difficulty levels.

At the first level, the user must grasp an object (e.g., a bottle, a ball, or a plate) and place it on a target base while overcoming an obstacle (a barrier). The bottle and ball are grasped using a power grip (Figure 8.9), while the plate is grasped using a three-finger grip.

The second level involves rotating a handle and sliding a cylindrical cursor along a guide until a final target is reached. The task involves elbow rotation and the power grip posture. For the handle (Figure 8.10), the rotation is considered completed when the angle between the handle's rotation axis and a reference direction is less than 5 degrees.

The third level consists of inserting small objects (buttons) into a box (Figure 8.11). This task is classified as more difficult due to the reduced object size and higher precision requirements.

The start and end of all tasks are controlled via a start/stop button using the pointing finger posture; execution time is used as the performance metric.

Hand–object interaction is implemented using the GAF/OAF methodology to ensure stable grasping, while simple interactions rely on 3D contact detection via mesh colliders, using the corresponding equations of the sequential impulse solver.

The VR-SHAP evaluation methodology is quantified according to the Index of Functionality (IOF) algorithm [289]. This approach considers the quadratic sum of scores obtained across all tasks, defining a range from 0 (poor performance) to 100 (excellent

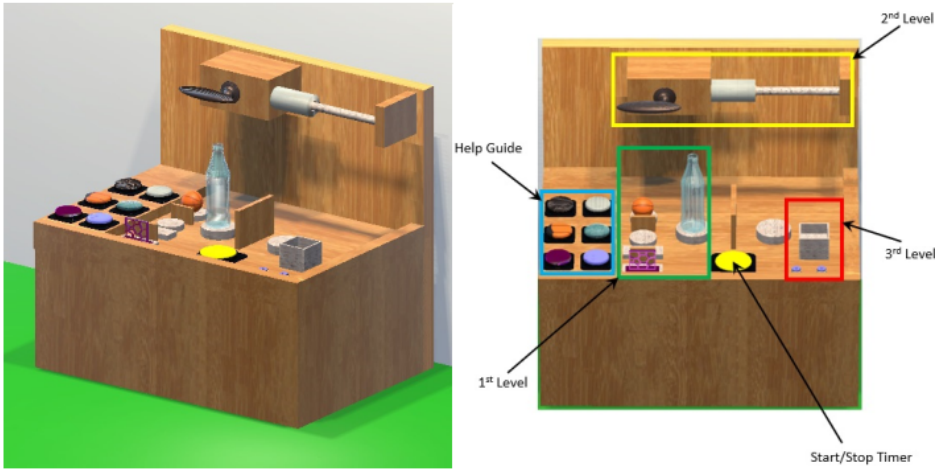


Figure 8.8: VR-SHAP test implementation in the virtual environment. Tasks are divided into three groups according to difficulty level. A help guide, using virtual buttons, can be activated by the user during task execution.

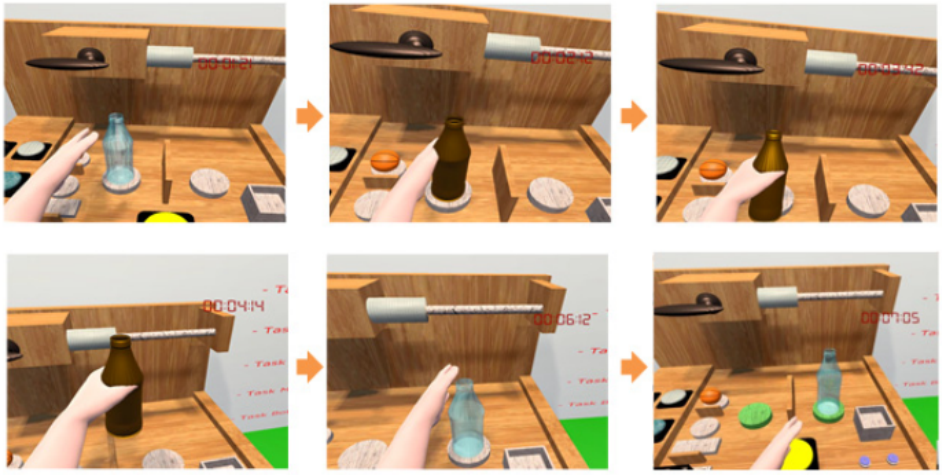


Figure 8.9: An example of the first level of the VR-SHAP test: grasping and manipulation of a bottle. The user locates the bottle, grasps it, and places it stably on a cylindrical base.

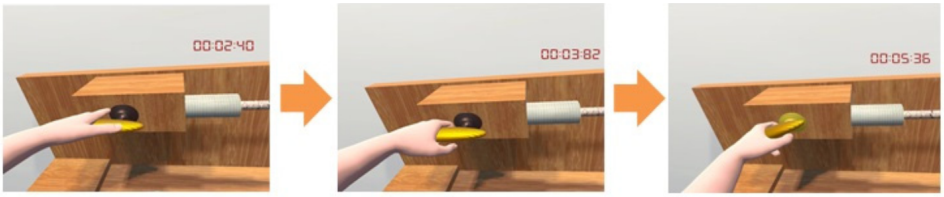


Figure 8.10: An example of the second level of the VR-SHAP test: rotation of a handle. The user locates the handle, grasps it, and rotates it by 45°.

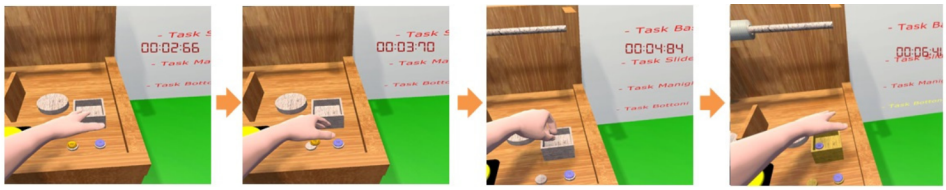


Figure 8.11: An example of the third level of the VR-SHAP test: manipulation of buttons. The user locates the buttons, grasps them one at a time, and releases them inside a box.

performance). According to this formalism, the overall score d is computed as:

$$d = \sqrt{\sum_{i=1}^{k} z_i^2} \tag{8.2}$$

where z_i is the score associated with the i-th task and k is the total number of tasks. The score for each task is computed as:

$$z_i = \frac{x_i - \tilde{x}_i}{s_i} \tag{8.3}$$

where x_i is the completion time of the i-th task, $\tilde{x}_i$ is the mean completion time for the same task in the reference population, and s_i is the standard deviation of the completion time for the i-th task in the reference group.

The mean values and standard deviations used in Equations 8.2 and 8.3 are reported in the next section, together with usability results shown in Table 8.1.

Task	**Average Time [s]**	**Standard Deviation [s]**	**Difficulty Level**
Bottle	8.4	2.4	Level 1
Ball	7.2	3.2	Level 1
Plate	6.8	2.4	Level 1
Handle	6.0	4.0	Level 2
Slider	19.2	4.0	Level 2
Buttons	17.6	4.0	Level 3

Table 8.1: Mean time and standard deviation for the six VR-SHAP tasks.

In addition to the VR-SHAP test, used for user performance evaluation, additional environments dedicated exclusively to training were developed. In the same VR-SHAP scenario, an area was implemented where the user must interact with four targets using a raycasting system activated by adopting the pointing finger posture. The first two targets are static and are used to verify the user's ability to maintain the posture for 3 seconds. The remaining two follow mono- and bidimensional sinusoidal trajectories (Figure 8.12a), aiming to train the user to maintain the posture while tracking a moving target. In the same environment, an additional area is dedicated to free manipulation of simple objects such as a cylinder, a cuboid, and a sphere. This training mode allows the user to become familiar with the grasping and releasing system implemented in the virtual prosthesis (Figure 8.12b).

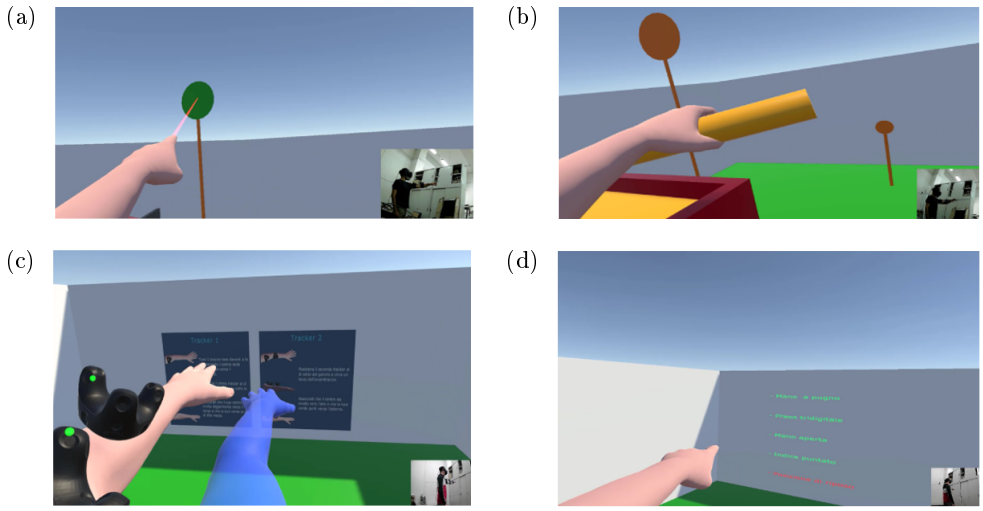


Figure 8.12: (a) Training for target interaction using the pointing finger posture; (b) free object manipulation environment; (c) Calibration Room used for tracker calibration and arm model alignment; (d) Hand-Move Room dedicated to familiarization with the classifier and the virtual environment.

Before accessing the training environments and the VR-SHAP test, the user must complete a preliminary phase consisting of two dedicated environments:

- Calibration Room: environment designed for correct tracker placement and arm model calibration. The user receives visual feedback on tracker axial orientation and, through the Shadow-Arm virtual model, can verify transversal alignment and relative sensor distance (Figure 8.12c).
- Hand-Move Room: introductory environment designed to allow the user to become familiar with the myoelectric classifier and the prosthesis control system. In this phase, the user can practice different grasps, understand classifier responses, and develop proper motor and spatial awareness in the virtual environment (Figure 8.12d).

8.1.4 Usability assessment results

At the beginning of the test, users were trained on the overall procedure (donning trackers, myoelectric sensors, and the head-mounted display, and interacting with the VR environment) through both verbal instructions and graphical guidelines. Users were then asked to repeat the procedure five times to gain familiarity with the system.

A preliminary usability analysis of the developed virtual environment was conducted using a questionnaire assessing perceived difficulty, clarity of instructions, setup procedure, and scene interaction (excluding classification performance). The study involved thirty-six users, evenly distributed by gender, aged 20–50 years (mean 30 years, standard deviation 8.5 years). Participants rated items on a scale from 0 to 10 (very poor/difficult to very good/easy).

According to the results reported in Figure 8.13, the usability scores indicate a generally high level of acceptance, confirming the adequacy of the implemented DT simulation model and communication interfaces.

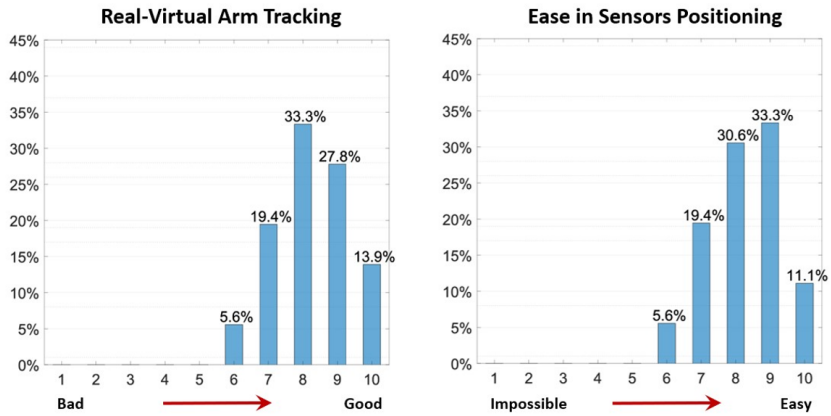


Figure 8.13: Usability analysis results on arm tracking (left) and ease of sensor placement (right). Both tracking fidelity and ease of use received high scores, confirming good overall usability.

Moreover, users performed the VR-SHAP tasks as described previously. The results in terms of average time and standard deviation are reported in Table 8.1. These values were then used to evaluate performance according to Equations 8.2 and 8.3.

At the end of the tests, users were asked to rate task difficulty and provide qualitative feedback.

As expected (Figure 8.14), first-level tasks were rated as easier due to fewer constraints in the OAF implementation. Despite higher difficulty in third-level tasks (due to smaller object sizes), perceived ease was higher than for second-level tasks. This may be attributed to the reduced degrees of freedom of the virtual prosthetic wrist, which makes large rotations more difficult.

Regarding clarity of instructions and overall VR experience (including immersion effects), users rated discomfort and cybersickness. Results (Figure 8.15) show that instructions were considered clear, the system was largely independent of operator intervention, and cybersickness levels were generally low, with a mild correlation with user age. Overall, the immersive experience was well tolerated.

Regarding setup, higher difficulty was reported for positioning the second tracker, which may interfere with myoelectric sensor placement. Additionally, some users initially experienced difficulties with handle rotation tasks; therefore, real-time self-alignment of the handle with the virtual hand was improved by reducing the torsional stiffness of the handle spring. Several interactions were further optimized to reduce the gap between the digital twin and the physical twin. Finally, initial discomfort at the end of the experience was reduced by softening the color palette of the virtual environment.

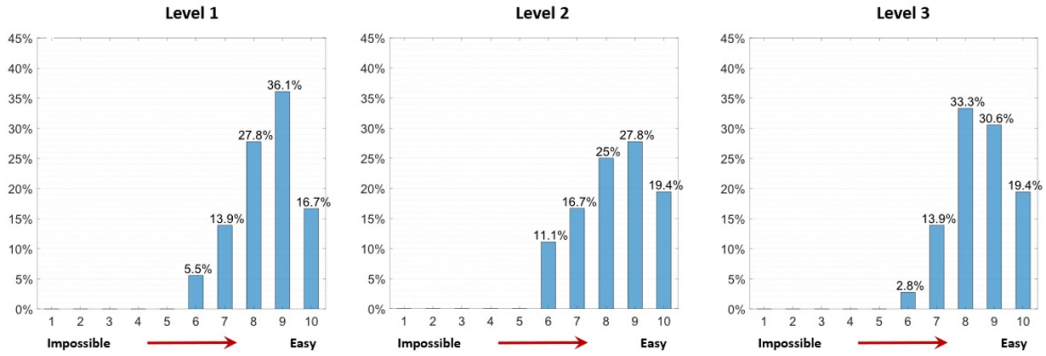


Figure 8.14: Usability results for VR-SHAP tasks at different difficulty levels. First-level tasks were rated easiest due to simplicity, while second-level tasks were perceived as more difficult due to limited wrist degrees of freedom.

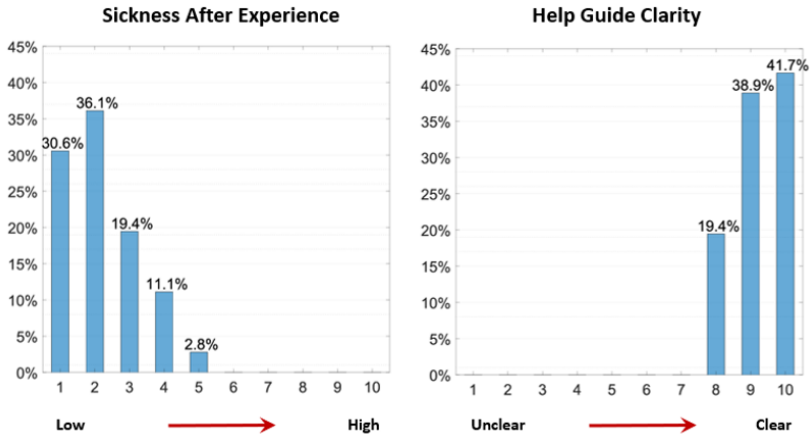


Figure 8.15: Usability results on cybersickness (left) and help guide clarity (right). Reported sickness levels were low, and the graphical guidance system was considered highly useful.

8.2 AR multibody digital twin exploration and interrogation

In this chapter, a dynamic digital twin of a slider-crank mechanism operating in an augmented reality environment has been developed [16]. The real-time multibody simulation is performed using the sequential impulse solver. The interaction between the digital twin and the user is bidirectional, allowing both the visualization of simulation data and user interaction with the virtual model. A *Microsoft HoloLens 2* headset, which is an *optical see-through* display [290], has been employed to project holographic information generated by the digital twin directly onto the corresponding components of the physical twin, enabling an intuitive and immersive visualization of the system dynamics.

8.2.1 Interaction in AR and digital twin integration

The Microsoft HoloLens 2 device is an advanced mixed-reality system that provides an immersive and interactive computing experience. The device integrates sophisticated hand and eye tracking capabilities. From a hardware perspective, it is equipped with a dedicated *Holographic Processing Unit* (HPU) and a Qualcomm Snapdragon processor. These components enable real-time rendering of complex holographic content and the seamless integration of virtual objects into the surrounding environment through spatial mapping and spatial awareness functionalities.

The functionalities of Microsoft HoloLens 2 were managed using the *Microsoft Mixed Reality Toolkit* (MRTK), which was adopted in the present study. At the time of implementation, version 3.0 (MRTK3) was used. Regarding hand tracking and posture recognition, MRTK integrates the detection of 25 joints, including fingertip endpoints.

MRTK is an open-source Software Development Kit (SDK) developed by Microsoft in 2016 for building augmented reality and mixed reality applications. It includes a collection of components and features designed to enhance both developer and user experiences. Its development philosophy is based on the use of predefined functional building blocks, which are appropriately configured and combined to create even highly complex applications. The latest version, MRTK3, represents the third generation of Microsoft's Unity toolkit and is built on top of Unity's XR Management system and XR Interaction Toolkit. The toolkit is organized into nine fundamental building blocks: button, bounds control, object manipulation, hand menu, near menu, slider, solver, dialog, and slate. In this work, only the button and object manipulation components were used.

The alignment of the augmented scene was achieved using the *PTC Vuforia Engine*. In this work, version 10.15 of Vuforia Engine was used, fully compatible with Unity and the Microsoft Mixed Reality Toolkit. To correctly align virtual information with physical objects, the Image Target shown in Figure 3.1b (Section 3.1.1) was used, in order to achieve global collimation of the fixed reference frame. These targets represent an evolution of classical fiducial markers and can be detected even in the presence of partial occlusions or when only part of the image is visible [291]. The position of all other movable bodies is subsequently derived directly from the same virtual model used in the simulation. The simulation model, based on sequential impulses, constitutes an invisible *digital shadow* that faithfully replicates the behavior of the Physical Twin, sharing the same reference frame. Since the dynamic model of the digital twin and the physical system evolve equivalently, the position of all real bodies can be determined with high accuracy directly from the simulation results.

MRTK, Vuforia Engine, and the efficient multibody dynamics model were integrated within the Unity development environment. In this work, version 2023.1 of Unity was used. Figure 8.16 summarizes the software architecture and the main libraries used to implement the proposed methodology. The integration of the four tools illustrated in Figure 8.16 is achieved using Unity as the central coordination environment. The Unity project is configured for the Universal Windows Platform (UWP), while MRTK and PTC Vuforia Engine are imported directly as packages into the active scene.

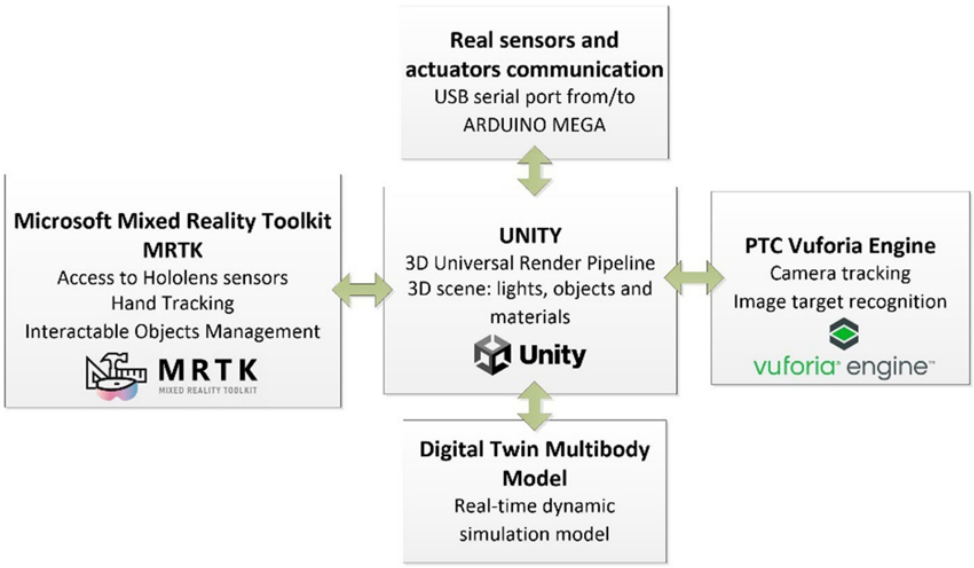


Figure 8.16: Development framework and software libraries supporting the construction of the interactive augmented reality digital twin application.

8.2.2 Experimental test bench

The proposed methodology is validated through an experimental test bench based on a slider-crank mechanism, selected for its nonlinear dynamic behavior and its ability to generate operating regimes characterized by inertia forces, impacts, and elastic interactions. The system consists of a crank driven by a NEMA 17 stepper motor, connected to a connecting rod and a slider that translates along a low-friction lubricated linear guide (Figure 8.17). Despite the simplicity of its geometric configuration, this mechanism exhibits complex dynamic responses, making it an ideal candidate for validating a real-time dynamic digital twin.

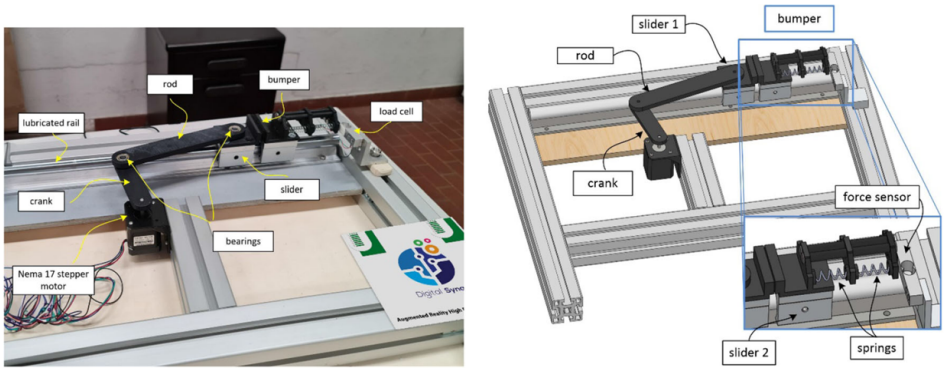


Figure 8.17: The real system (left) and its corresponding 3D virtual digital twin model (right).

During motion, the slider may impact a dedicated subsystem composed of three rigid bodies and four springs, designed to act as a nonlinear bumper. This subsystem absorbs part of the impact energy, generates a structured elastic response, and transmits

the resulting force to a load cell positioned along the slider motion direction. This configuration enables a direct comparison between experimentally measured forces and the corresponding dynamic quantities predicted by the digital twin model.

From a mechatronic perspective, the stepper motor is driven by a 4 A TB6600 driver, while the signal from the load cell is acquired and conditioned through an HX711 analog-to-digital converter. Both devices are connected to two dedicated Arduino MEGA 2560 microcontrollers, each assigned a specific function: the first is responsible for generating motor control commands, while the second manages data acquisition and filtering from the load cell (Figure 8.18).

The use of two separate controllers enables processing input and output signals at different update frequencies, avoiding delays and ensuring true real-time execution. A local personal computer acts as a communication server and executes the digital twin model, while the Microsoft HoloLens 2 head-mounted display communicates wirelessly via TCP/IP protocol, closing the interaction loop between the physical system, the virtual model, and the augmented reality environment.

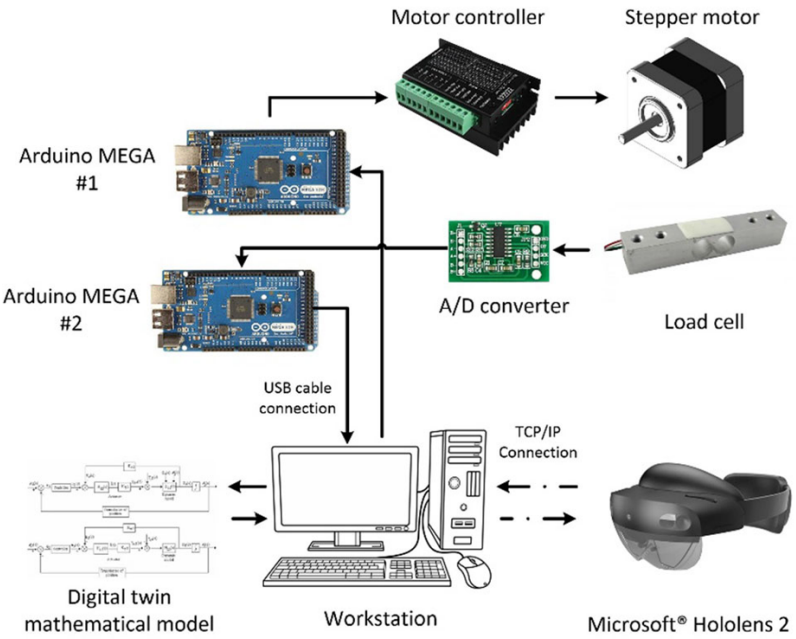


Figure 8.18: Communication scheme and supporting tools linking the physical twin, the digital twin, and the head-mounted display.

8.2.3 Real-time coupling between digital twin model and physical system

Within the digital twin, the mechanical behavior of the slider-crank system is replicated using a multibody model comprising five dynamic bodies: the crank, the connecting rod, the slider, and the bumper assembly, which consists of the shuffle and two intermediate blocks. These elements are interconnected through three revolute joints (frame–crank, crank–connecting rod, connecting rod–slider) and four prismatic joints

(slider–frame, shuffle–frame, and the two block–frame joints). The elastic subsystem of the bumper is modeled using four springs, reproducing the same topology as the real physical system.

The digital twin operates under two main boundary conditions:

1. the driving constraint, which relates the commanded motor angle to the user-controlled virtual knob;
2. the external force measured in real time by the load cell and applied to the second intermediate block.

User interaction modifies the system through a position-level kinematic constraint:

$$\psi_{\text{driving}} = \vartheta_{\text{motor}} - \vartheta_{\text{knob}} = 0 \tag{8.4}$$

which synchronizes the motor angle in the physical system with the command provided by the user in augmented reality. Any variation of the virtual knob generates an immediate command to the physical motor and, simultaneously, updates the digital twin solver, ensuring a continuous bidirectional loop between user, model, and hardware.

Interaction also occurs through contact-based querying: each physical component has a corresponding invisible collision shape within the digital twin. When the user touches a real object, the intersection between the fingertip and its virtual counterpart triggers the display of an information panel containing velocities, accelerations, reaction forces, and constraints. These holographic overlays are anchored to the object of interest and follow the user's perspective, transforming the mechanism itself into a three-dimensional interactive interface.

8.2.4 AR interaction framework and holographic data visualization

In the augmented environment, the user interacts naturally with the digital twin through the Microsoft HoloLens 2 headset, leveraging hand tracking and gesture-based interaction (Figure 8.19, left). The virtual knob displayed above the physical mechanism represents the main control interface: when the user grasps and rotates this holographic element, the system interprets the resulting roll angle as the new reference for the motor position. This interaction is smooth and intuitive, allowing the user to influence the behavior of the physical system exclusively through the augmented reality interface. The Image Target is placed within the physical scene to align the graphical overlay of the digital twin with the real components. Through a natural interface, the operator can interact via hand gestures with a holographic control panel, accessing in real time data related to velocities, accelerations, and reaction forces of the various mechanism components (Figure 8.19, right).

For system inspection, each physical body is associated with a corresponding virtual collision geometry, continuously updated by the multibody simulation but not directly rendered. When the user's fingertip enters one of these geometries, the system triggers a contextual panel. These panels appear as holographic pop-ups displaying component specific information, such as instantaneous velocity, acceleration, reaction forces, or spring deformation (Figure 8.20, right).

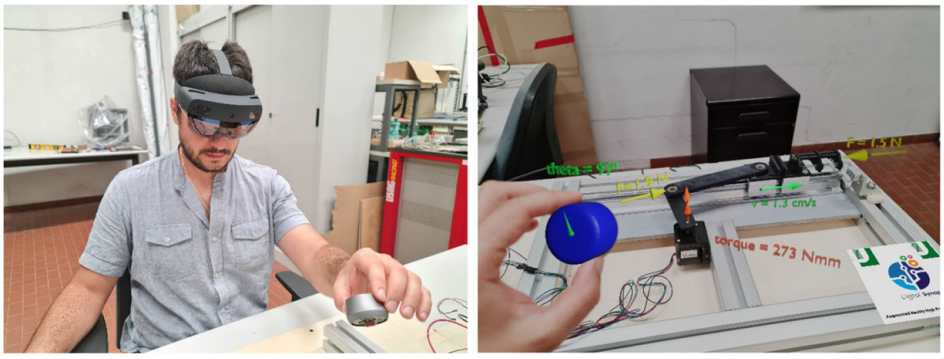


Figure 8.19: The operator wearing a HoloLens 2 headset interacting with the physical twin (left), and a sample view of the augmented scene (right).

Graphical elements such as vectors, reference frames, and color-coded highlights are dynamically anchored to the bodies and adapt to the user's viewpoint. This creates the perception that mechanical quantities exist within the same physical space as the mechanism itself, resulting in a highly intuitive and spatially coherent diagnostic environment (Figure 8.20, left).

As observed in the validation images, the simulated bodies are almost perfectly aligned with their real counterparts when full graphical overlays are enabled. Although complete overlays may be used for debugging or validation purposes, during normal operation the visualization is restricted to selected elements only, in order to preserve scene clarity.

Figure 8.20: Comparison between physical and digital twins through graphical overlays (left), and gesture-based interactive inspection of components (right): when the user points to the rod, a contextual panel appears displaying its system parameters (center of mass position, velocity, and acceleration).

8.2.5 Communication architecture and synchronization strategy

The entire ecosystem comprising augmented reality, digital twin, and physical twin is implemented using the Unity 2023.1 platform. The *Microsoft Mixed Reality Toolkit* (MRTK) is used for device input management, spatial awareness, hand tracking, and user interface components. Marker recognition, feature extraction, and target tracking are instead handled by the Vuforia Engine plugin. The Microsoft HoloLens 2 headset

transmits to the computer, via TCP/IP protocol, data related to device pose and user interactions, while the PC exchanges commands and sensor data with the Arduino MEGA 2560 boards through serial COM communication.

Communication between Unity and Arduino is based on TCP/IP. In Unity, a `TcpClient` object opens a connection to the IP address and port of the Arduino board, while a `NetworkStream` object is used for reading and writing sensor data. A `try-catch` block ensures connection error handling, which is captured and logged via `Debug.LogError`. On the Arduino side, an `EthernetServer` object listens for incoming TCP connections, while the function `Ethernet.begin(mac,ip)` initializes the network interface using a unique MAC address and a static IP address. The `Serial.print()` and `Serial.println()` instructions provide debugging information regarding server configuration and the assigned IP address.

The simulation is based on a Sequential Impulse dynamic solver, executed with a fixed time step of $\Delta t = \frac{1}{60}$ s, ensuring synchronization between the augmented reality visualization and the mechanical evolution of the system.

Synchronization between the digital twin and the physical twin is achieved by directly enforcing a position constraint on the stepper motor angle, rather than relying on velocity or acceleration information. This strategy eliminates numerical drift, enabling stable simulations even over long time horizons. Furthermore, the relatively slow dynamics of the actuator contribute to maintaining high numerical stability and accurate alignment between the virtual model and the physical system. Overall, this approach ensures continuous synchronization between DT and PT, providing a stable, accurate, and immersive augmented reality interaction experience.

8.2.6 Evaluation of the real-time simulation accuracy

To verify the accuracy of the dynamic model of the digital twin based on the Sequential Impulse solver, the input from the virtual knob and the load cell are processed offline as inputs to a standard multibody model constructed using the well-established Lagrangian formulation [124], resulting in a fully differential-algebraic system in which constraint equations are expressed as nonlinear algebraic equations and solved globally. This model is considered the gold standard for comparison and will be referred to as the "standard multibody model" or simply the "multibody model" in the following sections and figures. Table 8.2 reports the main numerical parameters related to the geometry and elastic properties of the components.

The comparison between the proposed model and the reference multibody model focuses on three main aspects: kinematics, dynamics, and constraint reactions. From a dynamic perspective, the motor angular displacement and the velocities of the two sliders are analyzed. The results show that the Sequential Impulse solver is able to reproduce both the motor rotation and the slider trajectories with high accuracy. Deviations with respect to the reference multibody model are generally negligible (below 2%) and mainly occur during impact events, which are characterized by very short durations in which instantaneous dynamics produce rapid velocity variations. These transient peaks correspond physically to the sudden compression of the bumper subsystem springs and the resulting rebound of the sliders.

Figure 8.21a reports the angular displacement of the motor, driven by the user

Parameter	Value	Unit
Crank length	80	mm
Connecting rod length	170	mm
Distance between motor axis and linear guide	118	mm
Distance between motor axis and load cell	298	mm
Crank mass	25	g
Connecting rod mass (including bearings)	76	g
Slider 1 mass	80	g
Slider 2 mass	85	g
Mass of each intermediate bumper block	6	g
Bumper spring stiffness	0.05	N/mm
Bumper spring damping	0.0003	N/mm s
Stepper motor maximum speed	50	deg/s

Table 8.2: Main input parameters for the multibody models

through the virtual knob. The plot shows a smooth and continuous evolution of the motor angle, which reflects in real time the user input. The comparison with the reference multibody results shows excellent agreement between the two curves, confirming that the Sequential Impulse solver accurately reproduces the rotational motion of the motor. Small discrepancies are observed only during rapid reversals of the rotation direction, where time-stepped impulse integration introduces short transient oscillations. However, these deviations remain below 2% of the overall angular displacement and do not compromise the global dynamic fidelity of the digital twin.

Figure 8.21c shows the velocity profiles of the two sliders: slider 1, directly connected to the crank, and slider 2, belonging to the bumper assembly. The velocity profiles obtained with the Sequential Impulse solver and those provided by the reference multibody model are almost indistinguishable during continuous motion phases, demonstrating the solver's ability to reproduce system dynamics with high accuracy. In particular, during impacts with the bumper subsystem, the solver captures the sharp velocity changes associated with collisions and spring compression. These peaks represent the rapid energy transfer between the sliders and the elastic elements of the system. Although the reference multibody model produces a slightly smoother response due to the simultaneous solution of the full constraint system, the Sequential Impulse solver correctly reproduces both the timing and magnitude of the velocity variations. This confirms the suitability of the proposed approach for real-time simulation of mechanical systems involving impacts and strong dynamic interactions.

From a dynamic standpoint, Figures 8.21b and 8.21d compare the motor torque and the reaction forces at the most stressed joints, such as the pin connecting the rod and slider 1. Figure 8.21b shows that the sequential impulse solver closely follows the torque profile computed by the reference multibody model. During impact events, short peaks are observed due to the accumulation of dynamic impulses within discrete integration steps. These peaks physically correspond to the sudden increase in resistance acting on the motor after collisions and demonstrate the solver's ability to correctly model such phenomena in real time.

Similarly, Figure 8.21d shows that the reaction forces at the Slider 1 pin are in excellent agreement between the proposed model and the reference multibody formulation. The Sequential Impulse solver correctly reproduces both the temporal evolution and magnitude of the reaction forces, exhibiting only brief spikes during impact phases. This confirms the algorithm's ability to handle transient events characterized by high accelerations without compromising overall solution accuracy.

Overall, these results demonstrate that the Sequential Impulse solver not only maintains high dynamic fidelity but is also capable of accurately reproducing the mechanism response in terms of driving torque, reaction forces, and impact events, which are essential for ensuring the reliability of a real-time digital twin system. The solver reconstructs the torque profile and force magnitudes with high precision; the only differences with respect to the reference model are limited to short peaks occurring during collisions. These peaks can be interpreted as the numerical representation of impulsive energy exchange between the slider and the bumper subsystem. Importantly, these discrepancies do not alter the global dynamic behavior of the system and remain well within engineering tolerances.

The analysis of constraint reactions further confirms the reliability of the proposed method. Reaction forces in revolute and prismatic joints show excellent agreement with those obtained from the conventional multibody model. Deviations are localized exclusively during impact phases characterized by high accelerations and are consistent with the discrete-time impulse integration performed by the solver. This behavior highlights the algorithm's ability to effectively handle highly transient phenomena without introducing numerical drift accumulation.

The quantitative evaluation of differences is performed by computing the relative percentage error for all analyzed quantities:

$$\%\text{error} = \frac{P_{\text{impulse}} - P_{\text{multibody}}}{P_{\text{multibody}}} \times 100 \tag{8.5}$$

where P_{impulse} and $P_{\text{multibody}}$ denote the amplitudes of the quantities computed by the Sequential Impulse solver and the reference multibody model, respectively. All signals are filtered using a low-pass filter with a cutoff frequency of 50 Hz in order to remove isolated impulsive spikes and focus the analysis on meaningful dynamic trends. Table 8.3 reports the maximum observed deviations, which are systematically below 2% for all considered quantities except for brief transient impact events.

Overall, the obtained results demonstrate that the Sequential Impulse solver provides a high-fidelity approximation of the dynamics computed by the classical multibody model, reproducing in real time both kinematic trajectories and system dynamic responses. The ability to accurately represent physical interactions, including collisions and spring compression, confirms the suitability of the proposed approach for integration into a real-time digital twin framework. This ensures that the augmented reality interface can provide the user with accurate, up-to-date, and physically consistent information, enabling effective human-in-the-loop interaction and intuitive control of the real system. The observed discrepancies are localized, predictable, and do not affect the global system behavior, confirming the robustness and practical validity of the proposed real-time simulation approach.

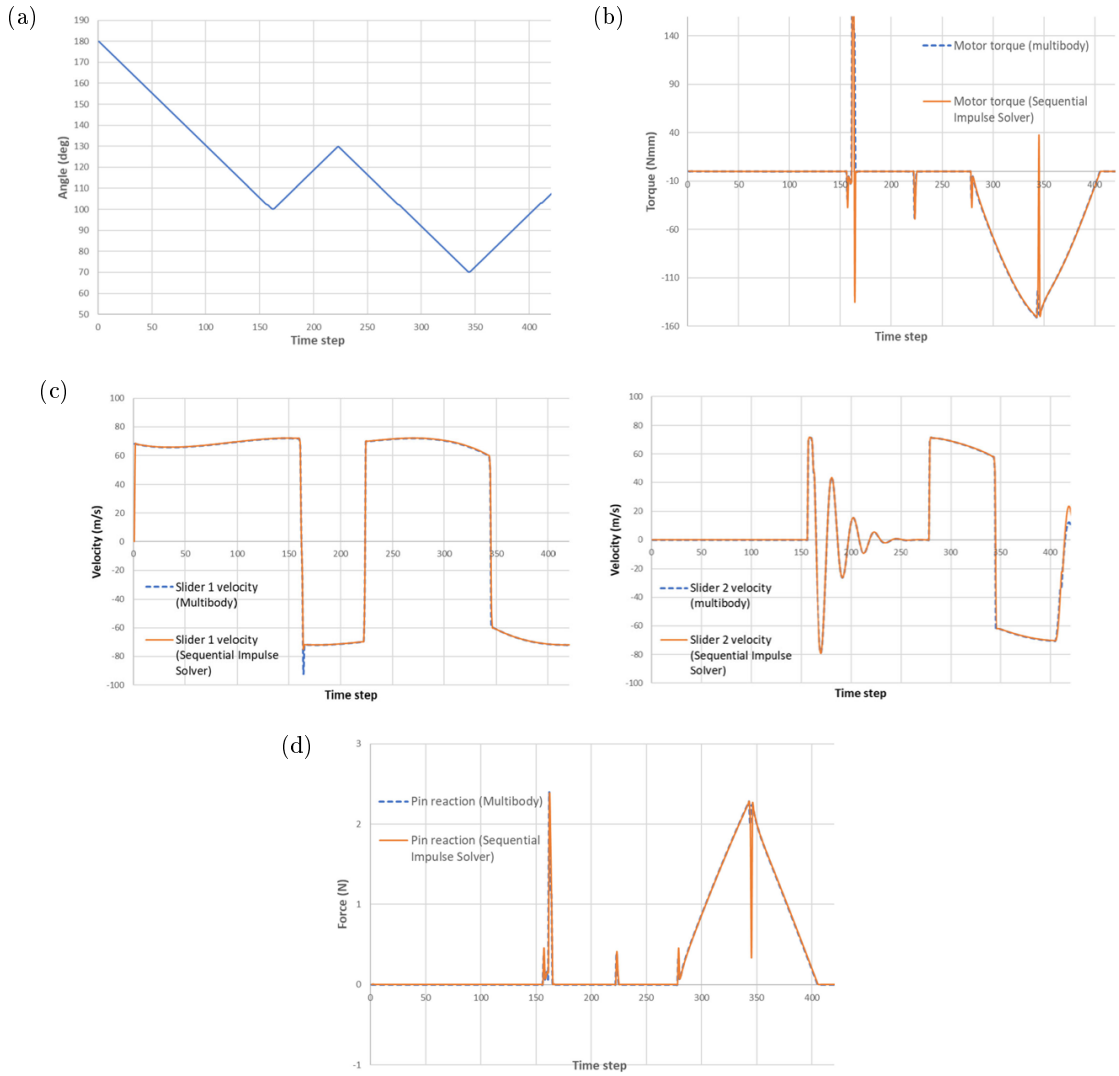


Figure 8.21: (a) Motor-axis angular displacement driven by user input on the virtual knob; (b) comparison of driving torque computed via the Lagrangian method and the Sequential Impulse solver (time step: 1/60 s); (c) comparison between slider velocities computed using the standard multibody approach and the Sequential Impulse solver (time step: 0.01 s); (d) comparison of reaction force magnitude at the pin connecting the rod to Slider 1, computed using the standard multibody approach and the Sequential Impulse solver (time step: 0.01 s).

Computed entity	Max % difference with respect to standard multibody solver
Driving Torque	5.5%
Slider 1 velocity	7.2%
Slider 2 velocity	6.5%
Rod/Slider 1 pin force	4.2%

Table 8.3: Peak percentage deviation between the Sequential Impulse solver and the conventional multibody solver.

8.3 ROM-based AR structural digital twin exploration and interrogation

In this chapter, a structural digital twin of a cantilever plate with a central hole, clamped at one end and subjected to a transverse load, has been developed [14]. The case study was selected because it is sufficiently simple to allow a comprehensive description of all the stages involved in the construction and integration of the proposed methodology, while at the same time being sufficiently complex to address challenges related to user interaction, the development of the simulation model, the presence of geometric nonlinearities, and morphological variations of the component.

8.3.1 Experimental test bench

The plate was manufactured in ABS using a Zortrax M200 3D printer and subsequently assembled within the physical system. Table 8.4 summarizes the main geometric and mechanical properties of the component.

Parameter	Value	Unit
Width	80	mm
Length	150	mm
Thickness	3	mm
Diameter of the hole	$d_i = 32$ $(10 \leq d \leq 50)$	mm
Distance of the centre of the hole from the fixed edge	75	mm
Young's Modulus	1.47	GPa
Poisson's Ratio	0.35	—

Table 8.4: Main geometrical and material features of the plate.

The physical twin is mounted on a modular frame made of 50×50 mm extruded aluminum profiles (Figure 8.22a). The plate is bolted to a 5 kg load cell, which is in turn attached to a dedicated support rigidly connected to the frame. The load cell measures the magnitude of the vertical load applied to the plate. Signal acquisition is performed by an HX711 analog-to-digital converter connected to an Arduino MEGA 2560 board, which communicates with the computer through a serial COM interface (Figure 8.23).

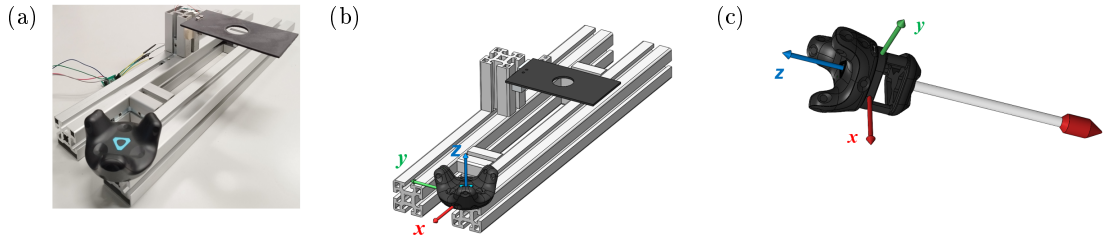


Figure 8.22: The physical system (a) and its 3D digital-twin model (b) for the considered use case; (c) the load application pointer used in the case study.

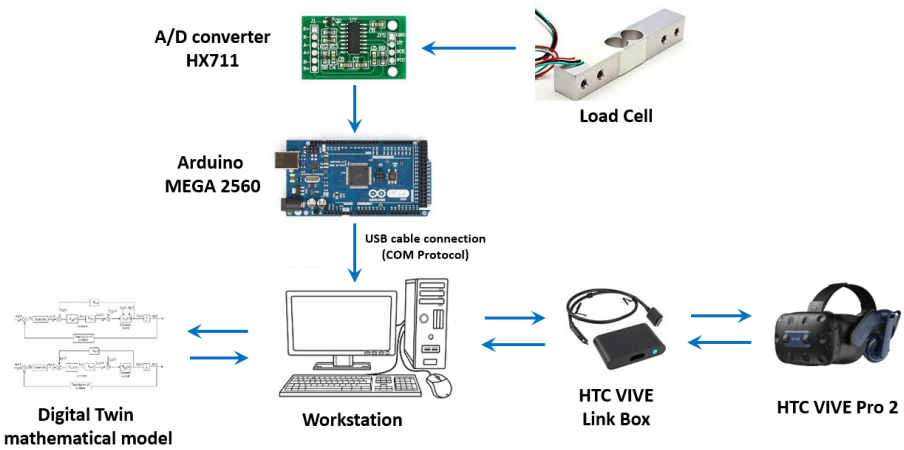


Figure 8.23: Communication architecture and enabling tools connecting the physical twin, the digital twin, and the head-mounted display.

The augmented reality environment was developed using Unity 2022.3.10f1. The HTC VIVE Pro 2 headset was selected because, unlike several other commercially available devices, it does not impose restrictions on access to its front-facing cameras. The headset is connected to the computer via the HTC VIVE Link Box. To access the HTC VIVE Pro 2 cameras and integrate the virtual and real worlds, the *VIVE SRWorks SDK* toolkit was employed. Through the headset's stereo cameras, SRWorks acquires stereoscopic images and depth information of the surrounding environment, processing these data to reconstruct a three-dimensional representation of the real scene.

The software architecture of the system is summarized in Figure 8.24. The integration between Unity and the tracking system is managed by SteamVR, which ensures data synchronization and communication between the hardware and the virtual environment, while the OpenVR plugin is responsible for the localization of the HTC VIVE Trackers.

The reduced-order model (ROM), based on POD and LHS, was developed in MATLAB and subsequently imported into Unity in binary format. The Gaussian regression model was implemented in C# and enables the real-time estimation of the modal coefficients from the data provided by the sensors installed on the physical twin.

The correct alignment between the virtual model and the physical system is achieved using an HTC VIVE Tracker 2.0 rigidly attached to the physical twin (Figure 8.22a).

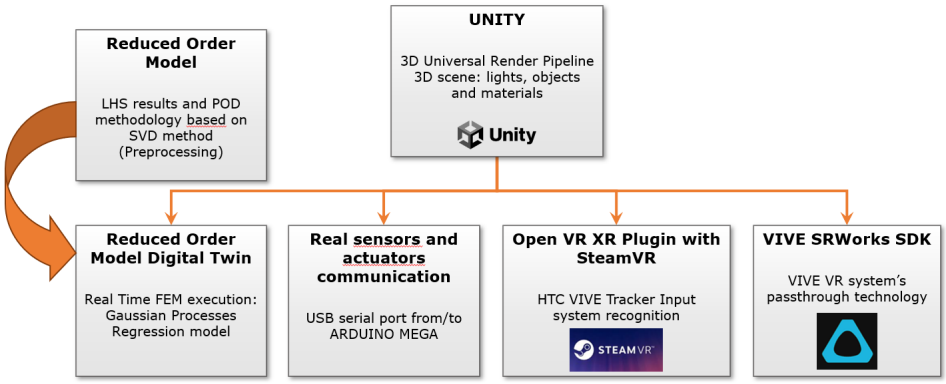


Figure 8.24: Integration of development environments and software libraries for implementing an interactive augmented-reality digital twin experience.

The tracker's xy plane coincides with the upper surface of the aluminum profile; the y-axis is orthogonal to the longer side of the profile, while the z-axis coincides with its plane of symmetry (all reference frames are left-handed).

To enable user interaction, a custom handheld probe was designed and manufactured, allowing the user to interact simultaneously with both the physical component and its digital twin. An HTC VIVE Tracker 3.0 is mounted on the upper part of the probe to determine the position of the probe tip in real time. Figure 8.22c shows the virtual model of the probe. The tracker's z-axis is aligned with the probe's axis of symmetry. Owing to its smaller size and lower weight compared with the Tracker 2.0, the HTC VIVE Tracker 3.0 is particularly well suited for handheld devices. In terms of tracking accuracy, no significant differences exist between the two versions, and their operation is handled transparently by the OpenVR plugin.

The reduced-order model-based digital twin was developed to operate in two different modes:

- Mode 1: nominal geometry with load application;
- Mode 2: interactive modification of the plate geometry followed by load application.

Thanks to the model training phase, the same ROM is capable of simultaneously handling both geometric variations and changes in loading conditions.

In Mode 1, the ROM input parameters consist of the load magnitude and the coordinates of the load application point. The load magnitude is indirectly determined from the reaction force measured by the load cell and transmitted to the *digital shadow*. The coordinates of the load application point are computed in real time from the position of the probe tip, which is determined using data provided by the HTC VIVE Trackers.

The digital twin uses these inputs to estimate the modal coefficients of the ROM in real time and reconstruct the corresponding finite element solution. The FEM model of the plate assumes a clamped boundary condition on the surface connected to the load cell, while the external load is modeled as a vertical force applied to the mesh nodes defined during the DOE process. Figure 8.25 illustrates an example of the plate deformation under combined bending and torsion obtained in Mode 1. Since the ROM was developed

considering large displacements, it is also capable of accurately capturing the effects of geometric nonlinearities.

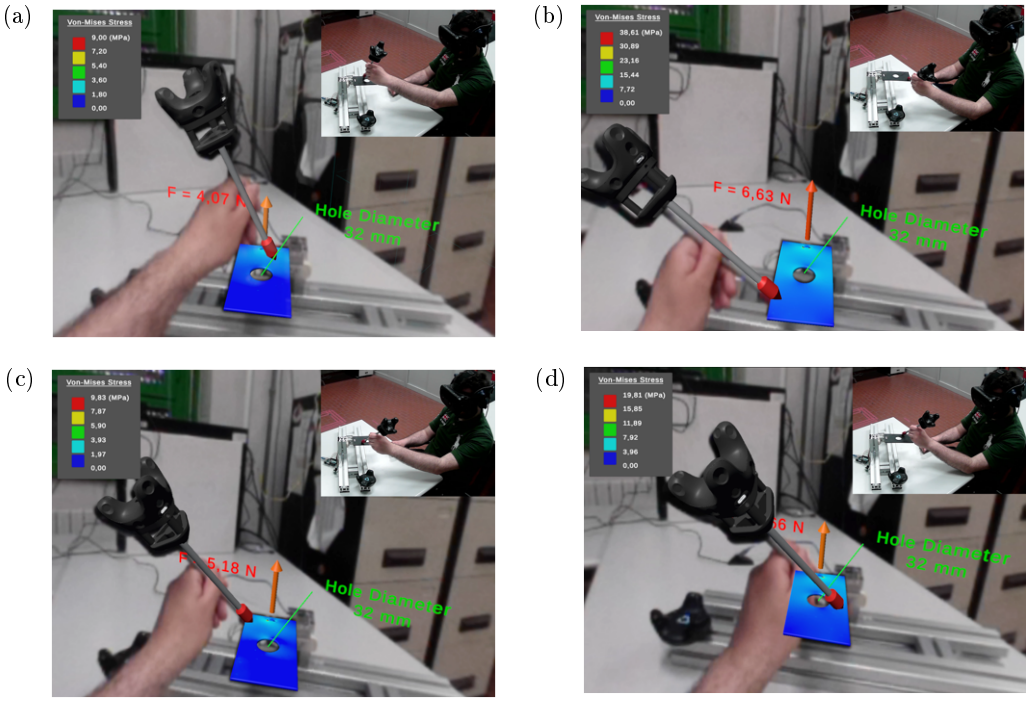


Figure 8.25: Real-time simulation in mode 1 under four different load magnitudes and application points: applied force of (a) 4.07 N, (b) 6.63 N, (c) 5.18 N, and (d) 4.66 N, with a hole diameter of 32 mm.

In Mode 2, the user can interactively modify the diameter of the central hole directly on the virtual model before applying the load and visualizing the structural response. As illustrated in Figure 8.26, during the geometry modification stage, a point located on the circumference of the hole is highlighted. This point can be selected and moved with the tip of the probe along the longitudinal direction of the plate. Throughout this operation, the hole diameter is updated dynamically in real time, providing the user with immediate visual feedback on the applied geometric modifications.

In the proposed case study, the hole diameter can vary between a minimum value of 10 mm (Figure 8.26e) and a maximum value of 50 mm (Figure 8.26a), while Figure 8.26c shows an intermediate configuration. Finally, Figures 8.26b, 8.26d, and 8.26f present the corresponding stress distributions obtained after the geometric modification for the three configurations considered.

8.3.2 From simulative model to graphic model

The most critical aspect in the development of the digital twin is the graphical representation of the results. In fact, the reduced-order model only serves as a starting point for visualization: a three-dimensional matrix describes a point cloud, while a vector contains the values of the physical quantity of interest associated with each point.

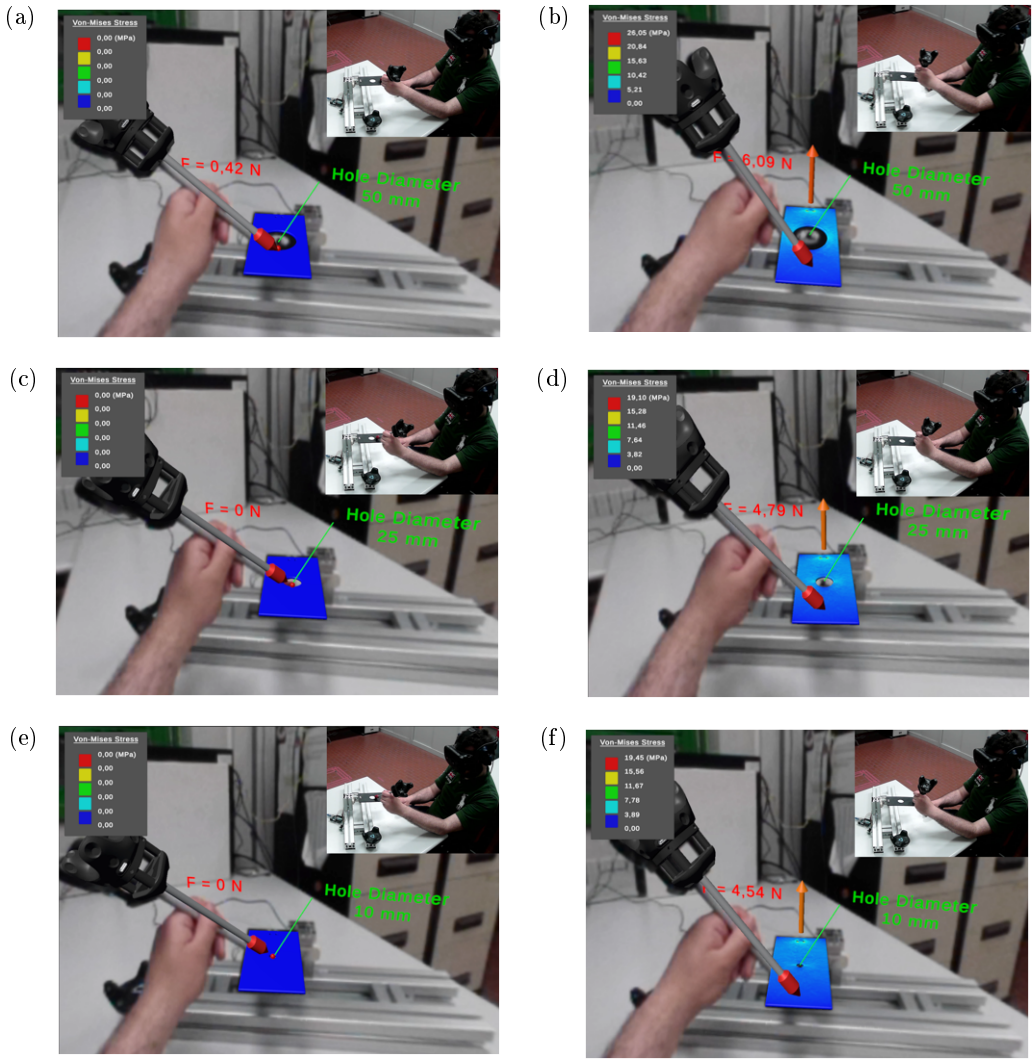


Figure 8.26: (a), (c), and (e) Real-time simulation in mode 2 with interactive variation of the central hole diameter (50, 25, and 10 mm, respectively); (b), (d), and (f) corresponding superimposed stress contour maps of the plate configurations after load application.

These data must be properly processed in order to be represented as three-dimensional graphical entities.

The first step consists of extracting the data related to the component mesh and the corresponding results from the finite element model. In particular, three different data sets are extracted:

- an $n \times 3$ matrix containing the spatial coordinates (x, y, z) of the n mesh nodes;
- an $l \times 4$ matrix representing the connectivity matrix, which describes how nodes are connected to form the l tetrahedral elements of the mesh;
- a vector of n elements containing, for each node, the value of the physical quantity

of interest, such as stress or displacement.

Subsequently, the three-dimensional geometry is reconstructed from this information. The matrix containing the nodal coordinates is directly used to define the vertices of the graphical mesh. As for the mesh elements, an additional processing step is required, since rendering engines display only the external surfaces of three-dimensional objects. To overcome this limitation, the triangular elements of the mesh are classified by distinguishing those belonging to the external surface from those located internally. The surface triangles are sent to the rendering pipeline for visualization, while the internal ones are kept in memory exclusively for model query operations.

This strategy significantly reduces the geometric complexity of the scene, improving rendering performance without losing the ability to query results at any point in the domain. When the user selects a point of interest, the system identifies the requested coordinates and computes the value of the physical quantity through interpolation of the results associated with the nearest nodes, immediately returning the estimated value. This procedure is also valid for points located inside the volume, since the nodal matrix retains all information from the original mesh, including data not directly visible.

The system also allows visualization of results inside the component through virtual sectioning. When the user defines a cutting plane, the mesh elements are reprocessed to obtain a consistent graphical representation. A dedicated script automatically removes from the connectivity matrix all elements located outside the cutting plane and reconstructs the generated surface by creating new triangles that connect the boundaries of the truncated mesh (Figure 8.27).

The final step consists of mapping the numerical results onto the graphical mesh. Within the rendering engine, a colormap based on a color gradient is defined and associated with the mesh vertices, using the same convention commonly adopted in finite element analysis software. In particular, blue represents the minimum values of the analyzed quantity, while red represents the maximum values.

The corresponding color is then assigned to each vertex of the mesh by directly preserving the nodal information coming from the *digital shadow*. In this way, each computed value is associated with the corresponding node, while the graphics engine shader automatically performs interpolation between vertex colors, producing a continuous distribution of results within the triangular elements of the mesh (Figure 8.28).

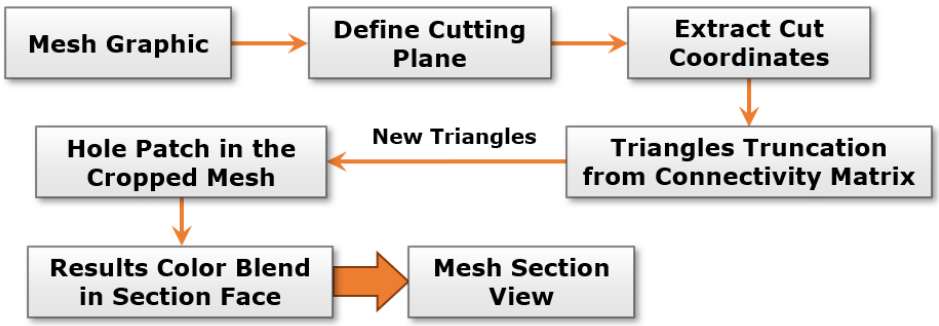


Figure 8.27: Mesh section view workflow.

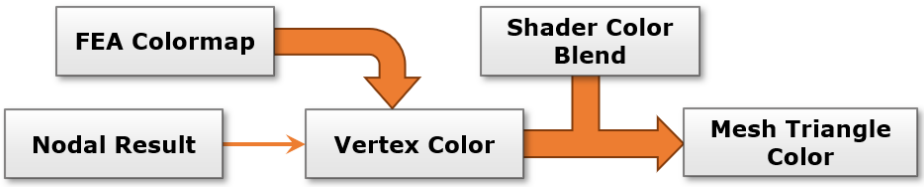


Figure 8.28: Workflow from FEA nodal outputs to the mapped graphical mesh.

8.3.3 Real-time accuracy assessment and performance evaluation of the experimental digital twin

The reduced-order model of the plate was trained using 200 simulation cases and subsequently validated using 20 verification scenarios. The generation of the simulation models, training sets, and validation sets was automated through a MATLAB script that sequentially manages the stages of geometric morphing, model solving, and regression model training.

The simulations were performed on a workstation equipped with an INTEL i9-12900KF CPU, 128 GB of RAM, and an NVIDIA RTX 3070 8 GB graphics card. The entire ROM training and construction process was completed in approximately 30 minutes. Each model included in the design of experiments was generated from an initial mesh, which was subsequently deformed using a radial basis function approach in order to reduce the computational cost associated with repeated remeshing.

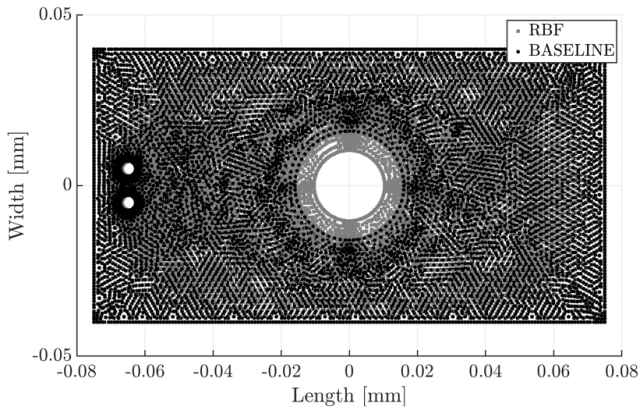


Figure 8.29: List of points for stress verification.

The error introduced by mesh deformation was evaluated by comparing stress results at six key locations (Figure 8.29) with those obtained using a structured, non-deformed mesh. The error, defined as the difference between the results obtained with the deformed mesh and those obtained with the structured mesh, was computed for three geometric configurations of the central hole (diameters of 10 mm, 30 mm, and 50 mm) and is reported in Table 8.5.

The level of mesh distortion was quantified using the element aspect ratio (mean and maximum AR), where a value equal to 1 represents an ideal element. The configuration

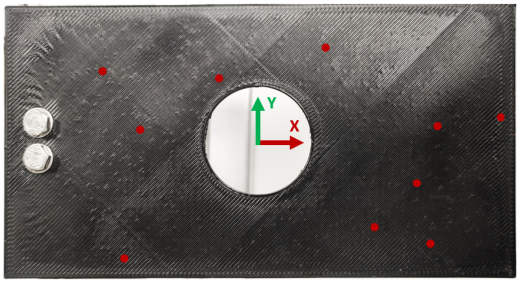
with the largest hole diameter (50 mm) exhibits a higher degree of element distortion, associated with an increase in prediction error compared to the structured mesh case. However, the average aspect ratio remains limited (approximately equal to 3), making the overall mean error acceptable across all nodes.

Similar considerations apply to the configuration with the minimum diameter (10 mm), while the intermediate configuration (30 mm) shows the closest agreement with the performance of the structured mesh, resulting in a reduced error. Even in the worst-case scenario, the maximum error recorded at point A (corresponding to the region of maximum stress) remains within engineering-acceptable limits.

The main characteristic of the ROM is its ability to significantly reduce the complexity of the dataset by obtaining a compact representation through a limited number of modes, while still maintaining a satisfactory level of accuracy. As shown in Figure 7.20, the results can be condensed using five principal modes, ensuring a cumulative energy greater than 0.9. This reduction leads to a significant advantage in terms of computational cost, although it introduces an inherent approximation of the solution.

To assess the overall accuracy of the Digital Twin, a system-level validation was performed in interactive mode. In particular, 10 interactive simulations were extracted from the augmented reality environment and sent to a full finite element solver, without using the ROM simplification. The positions of the load application points were determined through pixel-based interpolation of high-resolution images acquired with a telecentric lens. This procedure implicitly includes the verification of the probe tip tracking, ensuring a complete validation of the entire DT system.

Figure 8.30 shows the load application points and the corresponding values measured by the load cell (hole diameter equal to 32 mm).



Case	X Force [mm]	Y Force [mm]	Load [N]
1	22.4	27.5	8.83
2	37.1	-25.7	2.52
3	-34.0	4.0	0.79
4	72.4	9.3	3.83
5	-13.4	17.3	8.26
6	52.3	-11.3	9.23
7	60.3	-34.5	2.95
8	67.7	5.3	4.30
9	-44.2	-38.6	1.52
10	-31.5	22.2	2.12

Figure 8.30: Accuracy-assessment test cases: load application locations and load magnitudes.

Figure 8.31 shows a comparison between the stresses predicted by the ROM and those computed using FEM for the first case reported in Table 8.30. A nearly perfect overlap between the two sets of results can be observed, highlighting the high accuracy of the reduced model. The same behavior is also observed for the vertical displacements of the plate. Similar results are obtained for the remaining nine test cases.

In particular, Figure 8.32 compares the displacements computed via ROM and FEM for the same set of tests, confirming that the differences between the two approaches are negligible also in the case of deformations.

Figure 8.33a shows the percentage error of the ROM with respect to the FEM results,

	Error in stress estimation (%)							**Element distortion**			
	Verification points							Morphed mesh		Structured mesh	
Hole Diam. (mm)	A	B	C	D	E	F	Mean	Max AR	Mean AR	Max AR	Mean AR
50	5.9	5.2	4.7	4.5	4.4	4.6	3.5	6.7	3.4	3.3	1.8
30	2.7	1.6	2.3	1.5	2.1	2.2	1.8	3.8	1.5	3.5	1.3
10	4.8	3.6	3.7	3.0	3.3	3.4	2.7	4.8	2.3	3.2	1.5

Table 8.5: Stress estimation errors at six critical points, along with mean and maximum element distortion for three plate configurations.

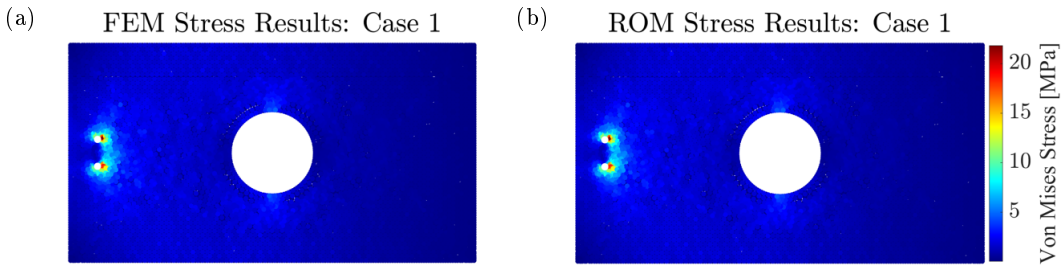


Figure 8.31: Example comparison of nodal stress results for case 1 obtained from the full FEM (a) and the ROM (b), both visualized using an identical colormap.

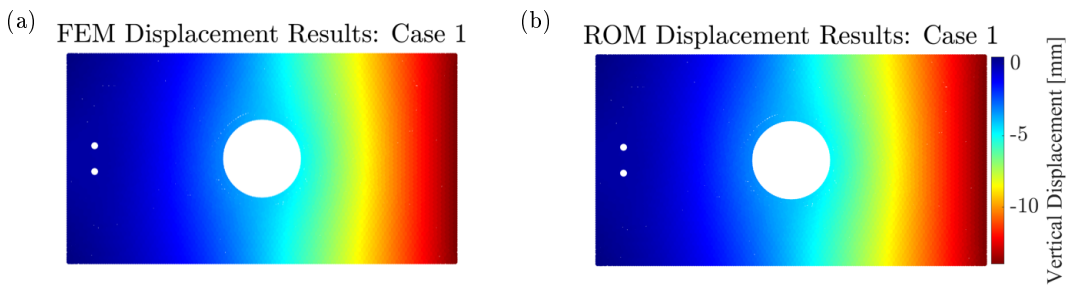


Figure 8.32: Example comparison of nodal displacement results for case 1 obtained fromthe full FEM (a) and the ROM (b), both visualized using an identical colormap.

computed according to the following expression:

$$\varepsilon_i = \frac{u_i^{\mathrm{ROM}} - u_i^{\mathrm{FEM}}}{u_i^{\mathrm{FEM}}} \times 100 \tag{8.6}$$

where u_i^{ROM} and u_i^{FEM} denote the reduced-order and full-order solutions at node i, respectively.

The observed differences are generally below 2% over the entire simulation domain, except for some localized regions where stress concentrations occur, in which the error can reach values below 5%. In these regions, the ROM tends to slightly overestimate stresses due to the limitations of the GPR regression model in accurately capturing local nonlinearities.

Figures 8.33c and 8.33d report the percentage error in stresses and displacements, respectively, for all test cases considered in Figure 8.30. The error peaks remain within 5% for all cases, further confirming the robustness of the proposed method.

For a more comprehensive assessment of the effectiveness and accuracy of the GPR model, the coefficient of determination R^2 [292] was also computed, defined as:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \tag{8.7}$$

where $y = (y_1, \ldots, y_N)$ represents the vector of FEM results, $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_N)$ the ROM predictions, $\bar{y}$ the mean of the observed values, and N the number of samples.

Figure 8.33b shows the R^2 value for the stress results of Case 1, while similar results are obtained for the other cases. The GPR model exhibits very high performance, confirming strong predictive capability.

For completeness, Table 8.6 reports the maximum deviations observed between ROM and FEM for all configurations analyzed in Figure 8.30, together with the corresponding R^2 values.

Test Case	Max stress % difference with FEM solver	Max disp. % difference with FEM solver	R^2 stress	R^2 disp.
1	2.8%	3.5%	0.9998	0.9996
2	4.1%	2.7%	0.9997	0.9998
3	2.2%	3.2%	0.9999	0.9997
4	2.9%	4.1%	0.9998	0.9995
5	3.3%	2.8%	0.9998	0.9997
6	3.7%	3.7%	0.9997	0.9996
7	4.4%	2.3%	0.9996	0.9998
8	2.5%	2.6%	0.9998	0.9998
9	2.6%	3.4%	0.9999	0.9997
10	4.7%	3.9%	0.9995	0.9995

Table 8.6: Maximum percentage deviation between the ROM and the full FEM solver.

8.4 ROM-driven AR digital twin for structural vehicle chassis monitoring

The automotive domain has seen a growing interest in the development of digital twins, i.e. virtual counterparts of physical vehicles capable of integrating sensor measurements, mathematical models, and real-time updates. The main purpose is the continuous, real-time monitoring of the main quantities describing the motion of the vehicle, commonly referred to as vehicle states. Tracking these quantities not only enables dynamic characterisation of the vehicle's behaviour, but also opens the door to structural analysis of onboard components, including monitoring stress concentrations or estimating deformations [293,294].

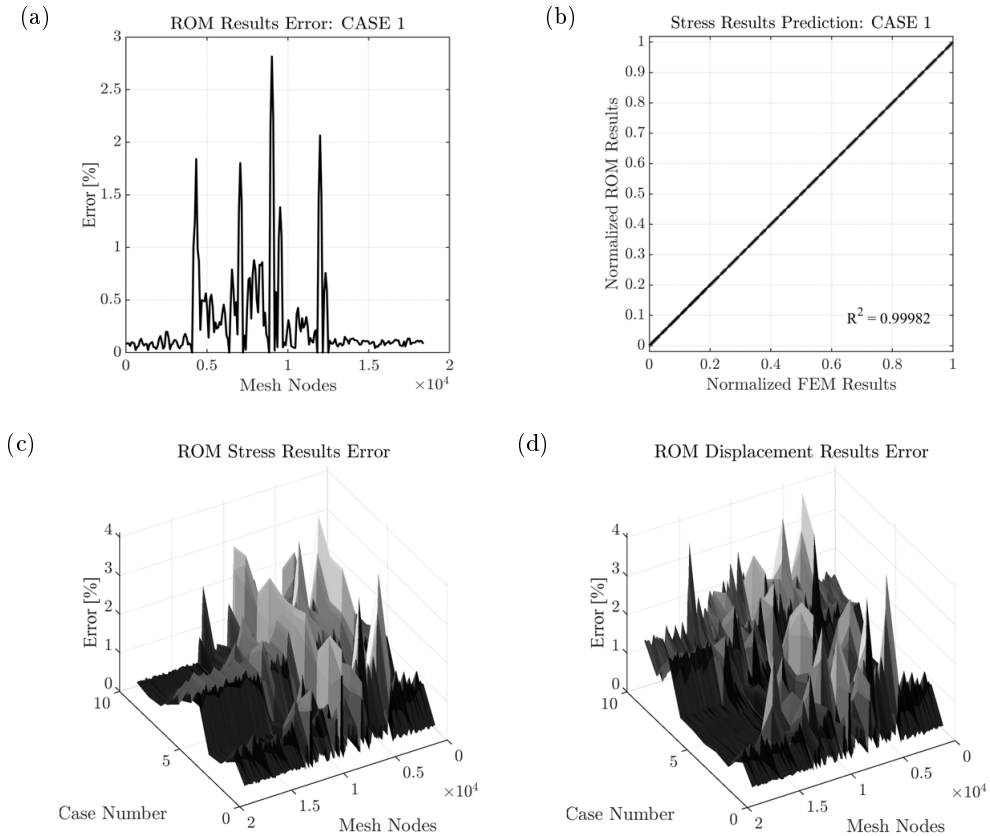


Figure 8.33: Example of percentage error in the stress-result comparison for case 1 (a);normalized stress values obtained from FEA versus normalized predicted stress values (b); percentage error distribution across all ten test cases for von Mises stress (c) and vertical displacement (d).

Based on this idea, in this chapter a structural digital twin of a vehicle chassis and, more generally, of flexible components operating in the linear regime, aimed at enhancing real-time structural state estimation, such as chassis displacements, to ultimately improve passenger safety, has been developed [22].

In addition, in linear structural regimes, the reduced model can be further simplified by exploiting the principle of superposition, which allows the system response to be expressed as a linear combination of independent basis contributions. Although these projection-based models are highly flexible and general, for purely linear problems they may introduce an unnecessarily complex formulation compared to direct analytical modal superposition approaches.

8.4.1 Architecture of a sensor-driven vehicle digital twin

The hardware architecture relies on a Raspberry Pi 4 for high-frequency sensor data acquisition. Data flow is organized according to a client–server architecture. The Raspberry Pi 4 operates as the client that is responsible for collecting and transmitting sensor measurements. A dedicated laptop functions as the central server, where data are stored,

processed, and used to update the computational model. The augmented reality headset connects to the same server as a receiving client, retrieving the processed data to execute the reduced order model and render the structural response in real time.

All collected data are transmitted via TCP/IP to the server, ensuring reliable communication between the sensing layer and the computational environment. This architecture enables synchronized data processing and immersive visualization of the chassis structural response. Figure 8.34 illustrates the overall Digital Twin architecture and the corresponding data flow from the onboard sensors to the AR device. Once acquired, the sensor measurements are processed to compute equivalent external forces applied at the suspension mounting points and used as input for the FEA and the ROM.

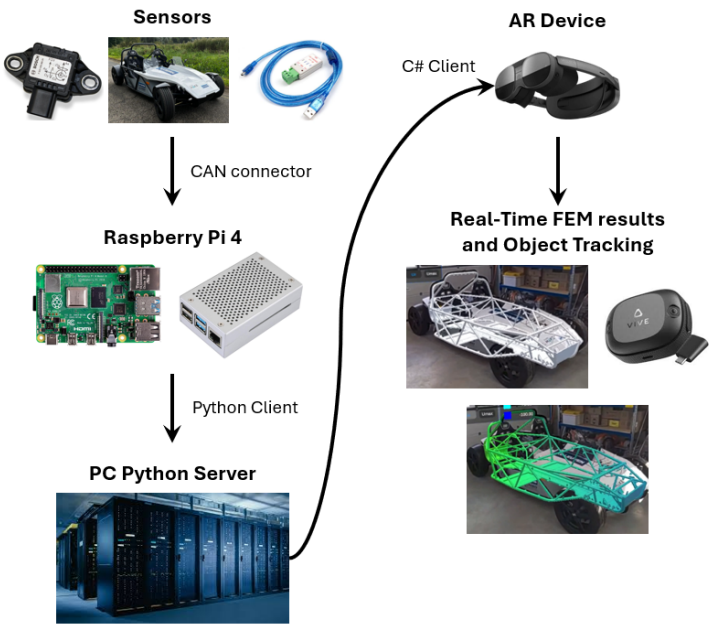


Figure 8.34: Digital Twin architecture and communication diagram.

8.4.1.1 Sensor integration and real-time data acquisition

By definition, a digital twin requires real-time information exchange between the physical system and its virtual counterpart. In this specific case, the information flow goes from the physical twin to the digital twin, as the transmitted data originates from the sensors installed on the vehicle. The vehicle is equipped with an inertial measurement unit (IMU) of the type Acceleration Sensor MM7.10, whose reference frame origin coincides with the vehicle's center of gravity (COG) and whose coordinate system follows the SAE J670 convention [295].

This unit integrates signal conditioning electronics and provides digitally processed outputs via high-speed CAN communication (up to 1 Mbaud), enabling real-time monitoring of vehicle dynamics.

The dynamic data from the IMU are transmitted through three main CAN messages with predefined identifiers, specifically:

- CAN ID 0x174 (ID 01): Yaw rate and Y-axis acceleration;
- CAN ID 0x178 (ID 02): Roll rate and X-axis acceleration;
- CAN ID 0x17C (ID 03): Pitch rate and Z-axis acceleration.

The sensor provides an absolute physical resolution of $0.1°/s$ for angular rates and $0.01\,g$ for linear accelerations.

The physical connection to the vehicle's high-speed CAN bus and the Raspberry unit is established through a USB-to-CAN analyzer adapter. This interface is responsible for the conversion of differential CAN signals into a serial data stream accessible via the USB protocol. The adapter supports the CAN 2.0A and 2.0B standards and is configured to match the 1 Mbaud communication speed of the IMU sensor.

The data acquisition infrastructure is centered on the Raspberry unit, configured as a client node and remotely managed by the server laptop via the virtual network computing (VNC) protocol. Real-time data exchange with the same server is achieved using the open platform communications unified architecture (OPC UA) [296], implemented over TCP/IP protocol [297].

Through the USB-to-CAN Analyzer Adapter and a Seeed Studio CAN-BUS interface, the Raspberry Pi directly acquires CAN frames from the vehicle network, with each IMU transmitting measurements at a 200 Hz update rate. The software running on the Raspberry Pi, developed in Python, integrates both acquisition and client functionalities. The acquisition software uses the `python-can` library for low-level communication with the CAN interface and `cantools` for semantic decoding of messages. Signal decoding is performed through the loading of a Database CAN (DBC) file, enabling the deterministic mapping of raw CAN frames into physical quantities. The client component manages the real-time acquisition, preprocessing, and transmission of IMU data to the server.

To mitigate measurement noise and optimize bandwidth usage toward the server, a two-stage preprocessing pipeline is implemented. The first stage consists of an exponentially weighted moving average (EWMA) filter [298], selected due to its capability to attenuate high-frequency noise while maintaining a low computational complexity. The filtered signal is defined as:

$$y_i = \alpha x_i + (1 - \alpha) y_{i-1} \tag{8.8}$$

where x_i is the current sample, y_{i-1} the previous filtered value, and α the smoothing factor (set to 0.1 in this application). The EWMA filter ensures that the filtering can be performed in real time on high-frequency IMU signals.

The second stage is a threshold-based deadband filter, which limits unnecessary data transmissions by ensuring that only signals exhibiting variations above a defined threshold are sent to the server, thereby reducing communication overhead during periods of minimal sensor change and preventing sensor noise from affecting the resulting signal.

The transmission of preprocessed IMU data to the server is handled via the `asyncua` library. The client employs asynchronous programming to transmit data in a non-blocking manner. To this end, a fire-and-forget strategy is used, whereby each write request is initiated without awaiting server acknowledgment, thereby minimizing communication latency and maintaining a continuous real-time data stream.

On the server side, the IMU data are organized in a structured set of nodes, each representing a specific measurement such as linear acceleration or angular velocity. The server is implemented as an OPC UA application using the `asyncua` library and operates according to an event-driven mechanism, whose subscription/notification architecture ensures real-time responsiveness by minimizing unnecessary computations.

The IMU data are then used to define a representative loading condition for the chassis. Specifically, linear accelerations and angular velocities, processed to derive angular accelerations via finite-difference calculations, are employed to conceptually derive twelve oriented forces acting on the suspension housings. Each force trio corresponds to a specific suspension, providing a preliminary framework to relate sensor measurements to the mechanical response of the system. The force computation is performed at a sampling frequency of 50 Hz.

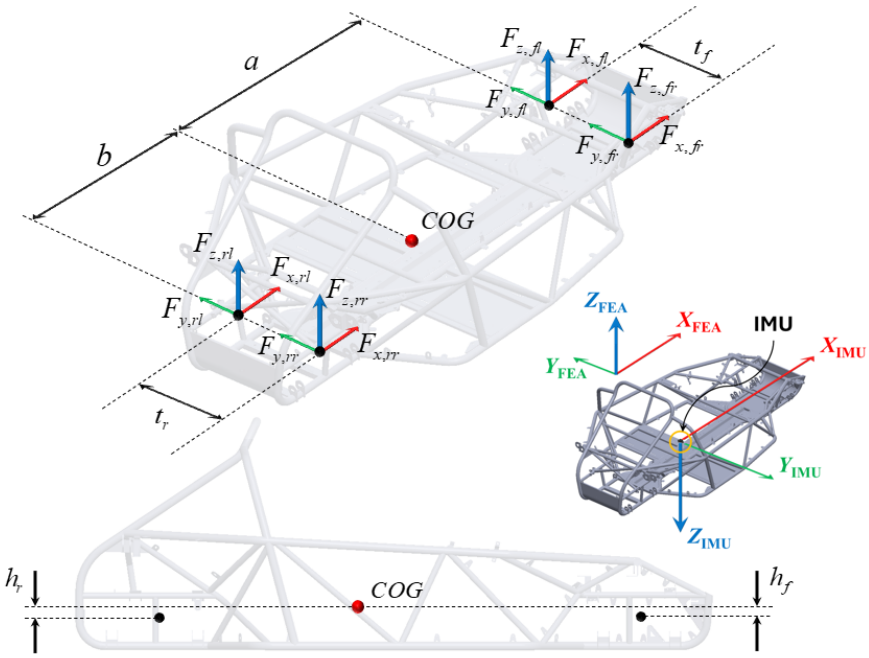


Figure 8.35: Nomenclature used for the external forces formulation, IMU reference system and finite element analysis reference system.

To convert the IMU measurements into the twelve suspension forces and obtain a determined system, twelve independent equations are required. Six of them are derived from the rigid-body equations of motion of the vehicle. The force application points, used to evaluate the corresponding moment arms, are identified as the geometric centroids of the joint housings for each suspension quadrant (Figure 8.35). Let m denote the vehicle mass; I_{xx}, I_{yy}, and I_{zz} the mass moments of inertia about the x, y, and z axes, respectively; a and b the longitudinal distances between the front and rear load application axes and the COG; h_f and h_r the vertical distances from the front and rear load application axes to the COG; t_f and t_r the front and rear track widths (distance between left and right force application points), assuming the vehicle's center of gravity is

symmetric with respect to the longitudinal mid-plane; a_x, a_y, a_z and $\dot{\omega}_x, \dot{\omega}_y, \dot{\omega}_z$ the linear and angular accelerations about the x, y, and z axes, respectively; and g the gravitational acceleration. The force components are denoted as $F_{i,jk}$, where the subscript $i \in \{x, y, z\}$ indicates the spatial direction, while the subscripts $j \in \{f, r\}$ (front, rear) and $k \in \{l, r\}$ (left, right) identify the suspension quadrant.

For instance, $F_{x,fr}$ represents the longitudinal force component acting on the front-right suspension. Neglecting gyroscopic coupling terms, the equations of motion can be written as follows:

$$ma_x = \sum F_{x,jk} \tag{8.9}$$

$$-ma_y = \sum F_{y,jk} \tag{8.10}$$

$$-m(a_z - g) = \sum F_{z,jk} \tag{8.11}$$

$$I_{xx}\dot{\omega}_x = h_r(F_{y,rr} + F_{y,rl}) + h_f(F_{y,fr} + F_{y,fl}) \tag{8.12}$$

$$+ \frac{t_r}{2}(F_{z,rl} - F_{z,rr}) + \frac{t_f}{2}(F_{z,fl} - F_{z,fr}) \tag{8.13}$$

$$-I_{yy}\dot{\omega}_y = b(F_{z,rl} + F_{z,rr}) - a(F_{z,fl} + F_{z,fr}) \tag{8.14}$$

$$- h_r(F_{x,rr} + F_{x,rl}) - h_f(F_{x,fr} + F_{x,fl}) \tag{8.15}$$

$$-I_{zz}\dot{\omega}_z = a(F_{y,rr} + F_{y,rl}) - b(F_{y,fr} + F_{y,fl}) \tag{8.16}$$

$$+ \frac{t_r}{2}(F_{x,rr} - F_{x,rl}) + \frac{t_f}{2}(F_{x,fr} - F_{x,fl}) \tag{8.17}$$

The sign convention for the linear and angular accelerations is defined according to the reference frame adopted in the mathematical model. The front and rear track widths are assumed to be equal, $t_f = t_r = t$. Furthermore, the vertical distances h_f and h_r are small compared to the other characteristic dimensions and are therefore neglected in the formulation.

Two equations are derived using the single-track vehicle model [299]:

$$-ma_y = 2(F_{yf} + F_{yr}) \tag{8.18}$$

$$-I_{zz}\dot{\omega}_z = 2F_{yr} \cdot a - 2F_{yf} \cdot b \tag{8.19}$$

Three more equations are derived respectively as translational equilibrium on the x and z axis and a rotational equilibrium about the y axis:

$$ma_x = 2(F_{xf} + F_{xr}) \tag{8.20}$$

$$-m(a_z - g) = 2(F_{zf} + F_{zr}) \tag{8.21}$$

$$-I_{yy}\dot{\omega}_y = 2F_{zr} \cdot b - 2F_{zf} \cdot a \tag{8.22}$$

A sixth linearly independent equation is introduced by assuming a unitary static friction coefficient between tire and ground [300]. Under this simplifying hypothesis, the longitudinal force at the front axle is assumed to be limited by the available vertical load, leading to:

$$F_{xf} = F_{zf} \tag{8.23}$$

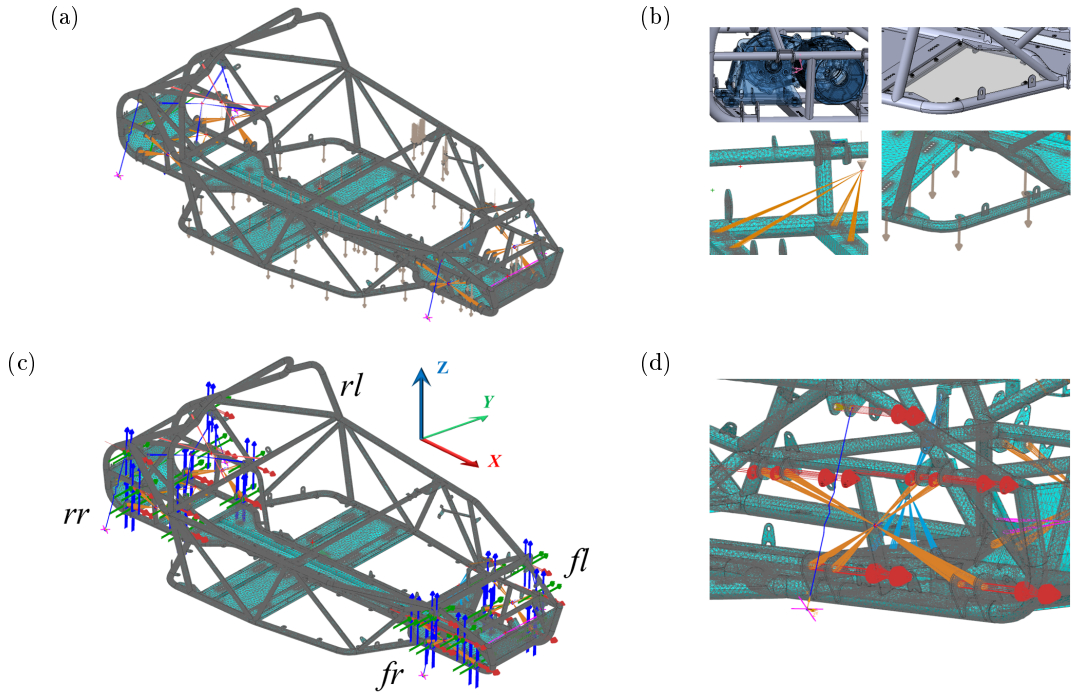


Figure 8.36: (a) Static masses mounted on the chassis, (b) cantilevered and distributed static masses, (c) IMU converted external forces, (d) external forces distributed on the suspension housings.

8.4.1.2 Real-time reduced-order finite element model

The structural response of the vehicle chassis is assumed to operate within the linear elastic regime; consequently, the principle of superposition is leveraged to construct a computationally efficient reduced-order model. Under the assumption of linearity, the response can be discretized into a set of m fundamental load cases. Specifically, for each discrete load case, the external excitation is normalized to a unit magnitude, thereby ensuring that the snapshot basis functions are independent of the specific loading intensity.

Let $\mathbf{S} \in \mathbb{R}^{n \times m}$ denote the snapshot matrix, whose m-th column represents the nodal response associated with the corresponding m-th unit load condition, evaluated via FEA. Let s represent the quantity of interest at the n-th node of the discretized domain under the m-th unit load condition. Let $\mathbf{c}$ be the vector of time-varying scaling factors associated with the actual loading state. In particular, q denotes the current value of the m-th loading component, as obtained from sensor readings. The resulting nodal state vector $\mathbf{r}$ is synthesized as:

$$\mathbf{r}_n = \mathbf{S}_{n \times m} \mathbf{c}_m = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1m} \\ s_{21} & s_{22} & \cdots & s_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n1} & s_{n2} & \cdots & s_{nm} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_m \end{bmatrix} \tag{8.24}$$

In the present study, the loading conditions are categorized into two primary contributions based on their physical nature. The first set includes the static masses mounted

on the chassis, such as the powertrain and energy storage systems (Figure 8.36a). These are modeled either as concentrated or distributed loads to accurately capture local bending effects (Figure 8.36b). The second set comprises dynamic loads derived from IMU data (Figure 8.36c). These inputs are transformed into equivalent external force trios (F_x, F_y, F_z) distributed among the respective joint housings of each suspension (Figure 8.36d).

Let $\mathbf{d}$ represent the nodal displacement within the discretized mesh, and let $\mathbf{w}$ denote the vector of displacements $\mathbf{d}^w$ induced by the static load distribution. Consequently, Equation 8.25 is reformulated as follows:

$$\mathbf{r}_n = \mathbf{w}_n + \mathbf{S}_{n\times m}\mathbf{c}_m = \begin{bmatrix} d_1^w \\ d_2^w \\ \vdots \\ d_n^w \end{bmatrix} + \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1m} \\ d_{21} & d_{22} & \cdots & d_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & \cdots & d_{nm} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_m \end{bmatrix} \tag{8.25}$$

where m represents the twelve external load cases.

The numerical investigation was conducted through a linear static analysis, performed using the *Simcenter 3D Nastran* solver (SOL 101). The transition from the CAD assembly to the chassis structural abstraction involved a geometric simplification process. This de-featuring phase consisted of removing non-structural components, such as brackets, and suppressing local geometric singularities, including fillets, chamfers, and minor holes, to enhance mesh quality and computational efficiency.

The domain was discretized using a linear tetrahedral mesh, comprising 786,418 nodes and 2,353,713 elements.

In the literature, standard static analyses of automotive chassis [301, 302] often employ simplified boundary conditions that do not fully capture the actual constraint state of the vehicle under operational conditions. To overcome these limitations and develop a high-fidelity numerical model, the suspension-to-chassis interfaces were modeled using specialized element formulations. Specifically, the shock absorber connections were represented through CBUSH elements, configured to match the physical orientation and the equivalent stiffness of the actual damping units (Figure 8.37a). This approach ensures a consistent linearization of the suspension compliance within the ROM framework.

To replicate the kinematic joint between the damper and the chassis, a reference point was defined at the geometric midpoint of each damper joint housing. This point was coupled to the internal surfaces of the housing via RBE2 (Rigid Body Element type 2) elements. This multi-point constraint approach ensures a uniform load distribution across the interface, avoiding local stress singularities and accurately transmitting reaction forces to the frame.

Furthermore, to ensure numerical stability while replicating the physical constraints of a stationary vehicle, a specific set of boundary conditions was implemented. The longitudinal translation is constrained by simulating the braking torque on the rear axle, while the lateral degrees of freedom are governed by the structural stiffness of the suspension linkages. The application points P_{jk}, where $j \in \{f, r\}$ (front, rear) and $k \in \{l, r\}$ (left, right), are identified as the geometric centroids of the joint housings for each suspension quadrant.

The resulting constraint scheme is summarized as follows:

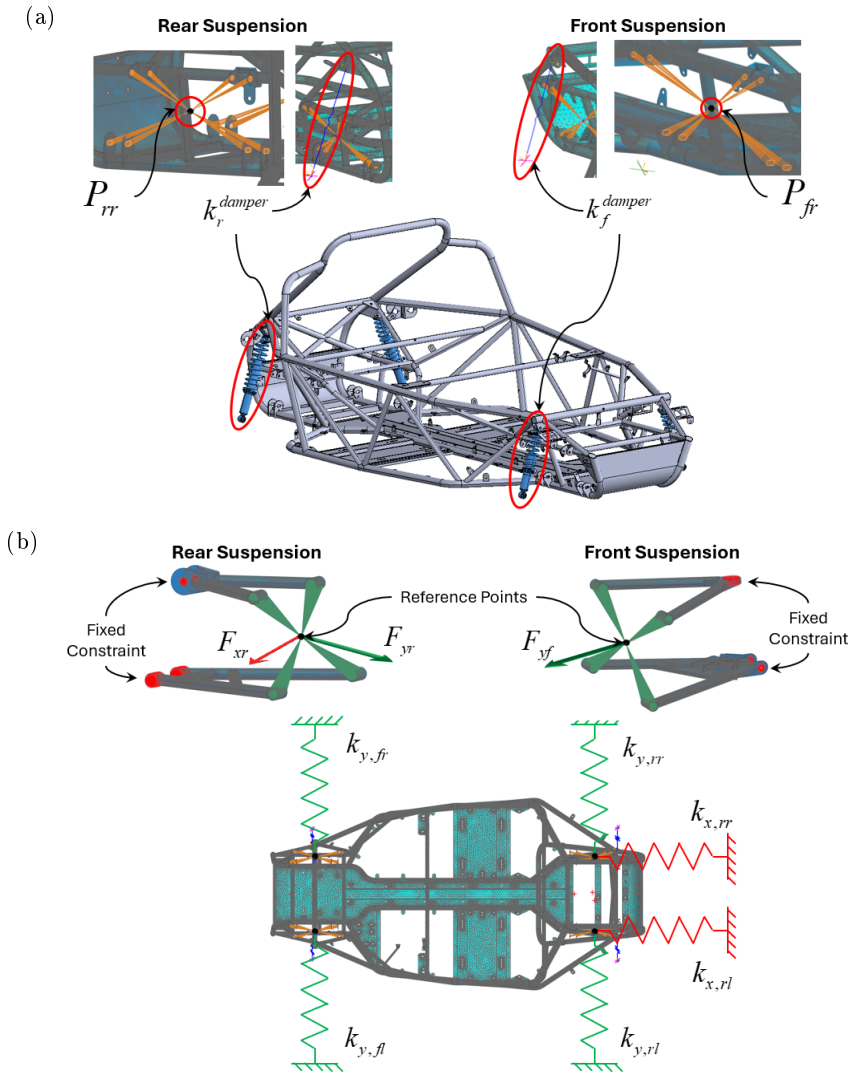


Figure 8.37: (a) CBUSH elements for damping units, (b) CBUSH elements for longitudinal and transversal constraints.

- Longitudinal constraints: Two CBUSH elements were applied exclusively to the rear suspension points (P_{rl} and P_{rr}), effectively modeling the longitudinal locking provided by the handbrake system and the tire-road friction interface.
- Transversal constraints: Four CBUSH elements were implemented (one at each P_{jk}) to account for the lateral reactions provided by the suspension control arms and the tire-road friction interface.

The equivalent stiffness values were obtained through a sub-modeling procedure, assuming the suspension linkages to be constrained at the wheel-side interfaces and defining a reference point (RP) at the geometric centroid of the chassis-side mounts (Figure 8.37b). By applying a unit force ($F = 1$ N) along the direction of interest and measuring the RP resulting displacement δ, the stiffness was computed as $k = F/\delta$. This allowed the

evaluation of the longitudinal stiffness at the rear axle and the lateral stiffness for each axle.

Upon completion of the analyses, the nodal displacement components (U_x, U_y, U_z) were extracted for each load case.

8.4.1.3 Augmented reality interface and digital twin integration

In advanced digital twin frameworks, data are often exchanged bidirectionally between the physical system and its virtual counterpart. To manage large volumes of data, both twins can be interrogated simultaneously using a head-mounted device (HMD) with passthrough visualization. The AR headset used in this application is the HTC VIVE XR Elite, a standalone device featuring inside-out 6DoF tracking.

The AR scene is developed in Unity 6000.2.7f2. Building an APK for the HTC VIVE XR Elite requires setting the Android build target. Input from the XR Elite is handled using the OpenXR Plugin, a cross-platform standard that provides unified access to device poses and input data across different devices.

Common interaction patterns, such as object manipulation, are implemented via the XR Interaction Toolkit, which abstracts low-level controller inputs into high-level interaction events. Additionally, XR Hands is employed to capture and process the user's hand poses and gestures, enabling controller-free hand interactions within the virtual environment. For rendering passthrough and mixed reality elements, the VIVE OpenXR Plugin was employed. The plugin utilizes the concept of a composition layer, which merges multiple visual sources, including the front-facing camera feed and the virtual scene, into a single stereoscopic output. This layer enables virtual content to be overlaid on the real world in real time, ensuring accurate depth perception and correct spatial alignment relative to the user.

Figure 8.38 summarizes all the software packages and plugins used to implement the digital twin.

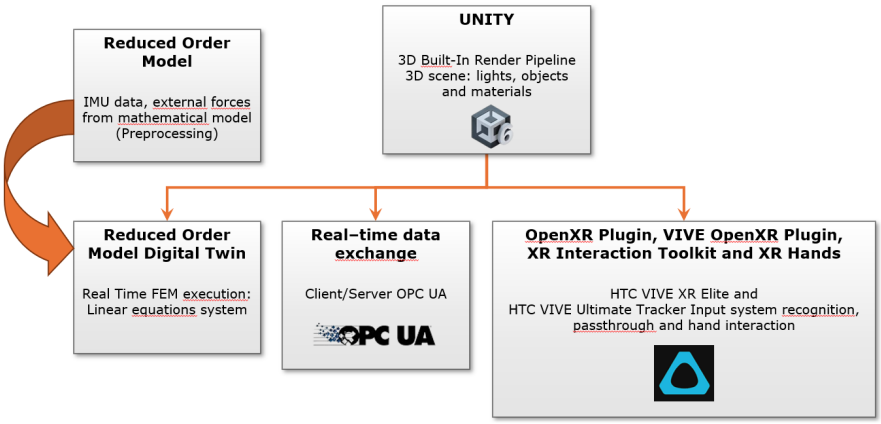


Figure 8.38: Integration of development environments and software libraries implemented in the case study.

To determine the position and orientation of virtual objects within the real environment and to superimpose the virtual objects onto real-world counterparts, the HTC

VIVE Ultimate Tracker was utilized.

The nodes and connectivity of the finite element mesh were extrapolated and processed in MATLAB to satisfy the left-handed clockwise winding convention, as shown in Figure 2.13b, which is required by Unity. The communication between the HTC VIVE XR Elite and the server is implemented through an OPC UA client developed in C#. Within the AR application, the user is required to enter the server IP address, while the communication port is predefined. When the user triggers the corresponding button, a session is asynchronously created by the AR client and the connection is established (Figure 8.39a).

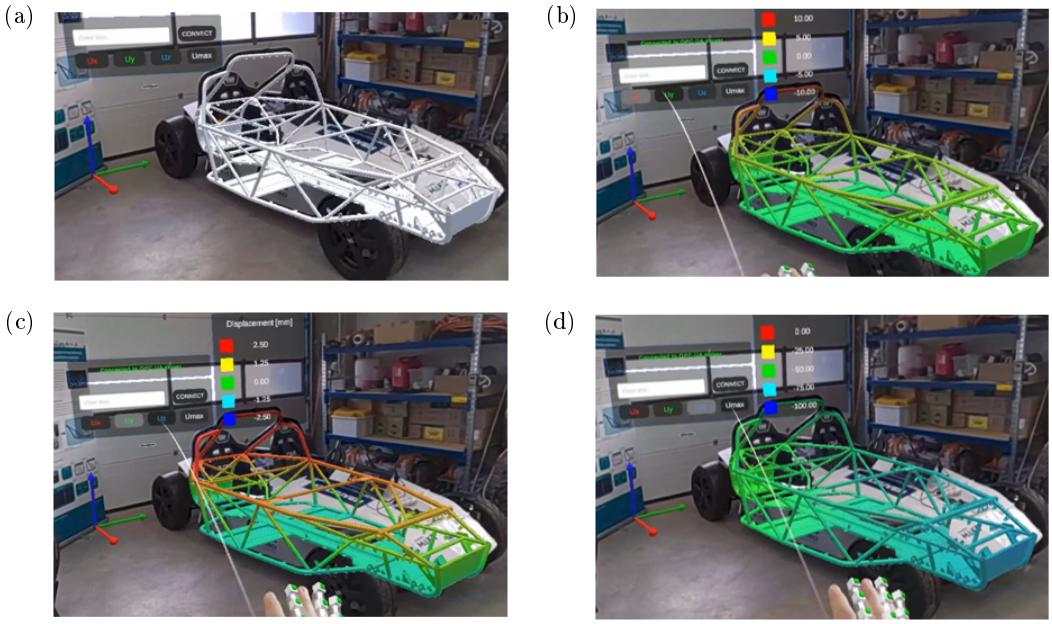


Figure 8.39: (a) Starting connection to the server; (b), (c), and (d) displacements along the x, y, and z axes, respectively,caused by static masses mounted on the chassis.

Once the connection is active, a subscription-based mechanism is employed on the C# client to receive real-time updates from a dedicated server node containing the twelve suspension force components. This event-driven architecture eliminates the need for continuous polling and significantly reduces communication latency, thereby ensuring efficient and responsive synchronization between the physical system and its digital counterpart.

During runtime, the displacement field is updated by scaling each precomputed unit-load response by the corresponding force component and summing all contributions. For each mesh node n, the displacement components along the x, y, and z axes U_x, U_y, and U_z, indexed by i, are evaluated as:

$$U_i^{(n)} = U_i^{(n,w)} + \sum_{j=1}^{12} F_j \, U_i^{(n,j)} \tag{8.26}$$

where $U_i^{(n,w)}$ are the displacement components due to static loads along the i-th direction and F_j the j-th external force component. The displacement magnitude $U_{\max}(n)$ at node

n is then computed as:

$$U_{\max}(n) = \sqrt{\left(U_x^{(n)}\right)^2 + \left(U_y^{(n)}\right)^2 + \left(U_z^{(n)}\right)^2} \tag{8.27}$$

Figures 8.39b, 8.39c, and 8.39d show the node displacements along the x, y, and z axes, respectively, due to the static masses. Finally, the computed displacements are mapped onto the mesh for visualization. A vertex-based colormap is generated within the rendering environment, employing a gradient similar to those commonly used in FEA. In this scheme, lower displacement values are represented in blue, while higher values are indicated in red. For each displacement direction component, a fixed legend is defined. Each vertex of the mesh receives a colour corresponding to its nodal displacement, preserving the spatial distribution of the digital twin data. The rendering engine shader then interpolates colours across the triangular elements, producing a smooth representation of the deformation field.

Figures 8.40a and 8.40b illustrate the chassis nodal displacements along the y and z axes, respectively, as determined from IMU measurements in response to user-applied external forces.

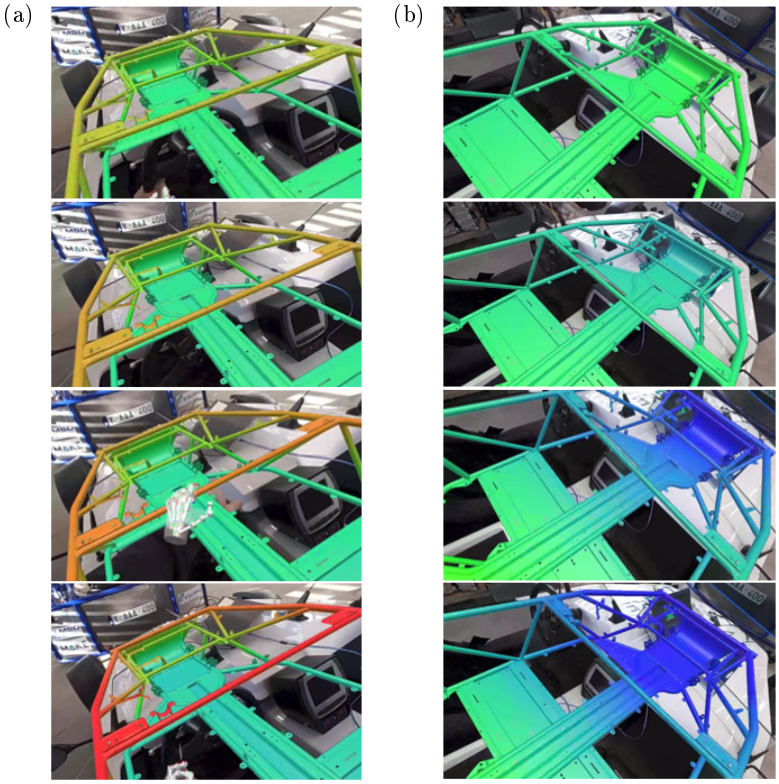


Figure 8.40: Chassis displacements under external forces,measured via IMU, along (a) the y-axis and (b) the z-axis.

Chapter 9

Conclusions and future perspectives

This doctoral research has addressed the development of methodologies for the design and implementation of interactive Digital Twins based on Virtual and Augmented Reality technologies. The work proposes a comprehensive methodological framework that integrates numerical simulation, real-time data acquisition, and immersive interaction within a unified computational environment.

The research was motivated by the observation that, although the Digital Twin paradigm has reached a high level of maturity across many industrial domains, most existing implementations remain primarily focused on monitoring, visualization, and supervision of physical systems. While these capabilities represent fundamental components of a Digital Twin, the integration of immersive interaction, real-time physics simulation, and active human participation in analysis and decision-making processes is still relatively limited. Within this context, the main objective of this research has been to contribute to the evolution of the Digital Twin paradigm by developing methodologies that seamlessly combine numerical models, data acquired from the physical system, and Extended Reality technologies into a single interactive computational framework.

A first contribution of this thesis concerns the definition of a methodological pipeline for spatial registration between the physical and virtual domains. Different tracking approaches, including both computer vision-based techniques and commercial tracking systems, were investigated and formalized in order to identify common procedures capable of ensuring consistent coordinate system alignment, stable registration, and robust tracking throughout user interaction. Furthermore, several interaction techniques for manipulating virtual objects were analyzed and compared. Among the available solutions, handheld controllers, hand tracking, and haptic devices emerged as the most effective and intuitive interfaces for immersive interaction.

The research also investigated the role of Extended Reality technologies in advanced geometric modeling. A comprehensive review of the current state of immersive geometric modeling techniques was first conducted, providing an overview of the methodologies already available in XR environments. Building upon this analysis, a framework for direct geometric manipulation in immersive environments was developed using subdivision surfaces as the underlying representation, enabling intuitive editing while preserving geometric smoothness.

Another significant aspect of this work concerns the application of immersive tech-

nologies to industrial scenarios and their impact on human-machine interaction. In particular, the effectiveness of XR systems was evaluated in virtual assembly applications through a series of human-in-the-loop usability studies. These experiments provided a quantitative assessment of operator performance and cognitive workload, highlighting the critical influence of perceptual fidelity and system latency on the overall effectiveness of immersive industrial applications.

To enable the real-time execution of computationally demanding structural and multibody simulations, several dynamic formulations were implemented, including sequential impulse methods, iterative multibody solvers, and Model Order Reduction techniques. Both the real-time multibody solvers and the Reduced Order Models (ROMs) demonstrated an effective balance between physical accuracy and the computational efficiency required for high-frequency interactive simulations. These approaches made it possible to reproduce the behavior of complex mechanical systems with high accuracy while significantly reducing computational cost compared with conventional high-fidelity models. As a result, one of the major obstacles preventing the integration of physics-based simulation into immersive Digital Twins was effectively addressed, enabling real-time updates of the simulated system response.

The proposed methodologies were validated through several case studies, each targeting specific challenges associated with interactive Digital Twins. These applications included immersive environments for prosthetic training, augmented visualization and exploration of multibody mechanisms, and structural Digital Twins based on reduced-order models for the real-time evaluation of stress fields in both individual mechanical components and a vehicle chassis.

Although the experimental results demonstrate the effectiveness of the proposed methodologies, the research also highlights several limitations that define the current scope of the developed framework. The stability and accuracy of real-time simulations still depend to a considerable extent on the geometric and physical simplifications adopted during model reduction. Likewise, vision-based tracking systems remain susceptible to jitter and temporary target loss under challenging illumination conditions or partial occlusions. Moreover, coupling high-fidelity numerical models with interactive constraints inevitably introduces a trade-off between physical realism and responsiveness.

These limitations naturally suggest several directions for future research. One promising avenue is the extension of the proposed framework toward distributed and collaborative Digital Twin architectures, where multiple users can simultaneously interact with the same virtual representation while sharing simulations, sensor data, and analysis results in real time. In addition, the integration of multiphysics models together with advanced data fusion techniques could further extend the applicability of the proposed methodologies to increasingly complex industrial scenarios.

In conclusion, this work contributes to the advancement of Digital Twin technology by promoting its evolution from a predominantly descriptive tool to an interactive and predictive computational environment, in which mathematical models and human operators are tightly coupled through immersive technologies. This integration provides a foundation for a new generation of Digital Twins capable not only of representing physical systems, but also of supporting real-time simulation, intuitive interaction, and informed decision-making in advanced engineering applications.

Scientific contributions

- A. Cellupica, M. Tristano, M. Cirelli, P. P. Valentini, and F. Naets, "A Reduced-order model–driven augmented reality digital twin for real-time vehicle chassis displacement monitoring," *in Proceedings of the VEHITS Thematic Conference, Benidorm, Spain*, 2026.
- A. Cellupica, M. Cirelli and P. P. Valentini, "Reliable and smooth haptic force feedback in interactive multibody simulations," in *Proceedings of the ECCOMAS Thematic Conference on Multibody Dynamics, Innsbruck, Austria*, 2025.
- A. Cellupica, M. Cirelli, O. Giannini and P. P. Valentini, "Interactive modelling in augmented reality with subdivision surfaces and advanced user gesture recognition," *Applied Sciences*, vol. 14, p. 11873, 2024.
- M. Cirelli, P. Canonico, A. Cellupica and P. P. Valentini, "Reduced-order models and augmented reality for real-time interactive structural digital twin exploration and interrogation," *," International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 19, no. 10, pp. 7263-7281, 2025
- A. Cellupica, L. D'Angelo, M. Cirelli, M. Mazur and P. P. Valentini, "Flexible multibody approaches for simulating the deflection and contact mechanics of toothbrush bristles: modelling and testing," *Multibody System Dynamics*, vol. 63, no. 1, pp. 159-181, 2025.
- A. Cellupica, L. D'Angelo, M. Cirelli, M. Mazur and P. P. Valentini, "Flexible multibody approaches for simulating the deflection and contact mechanics of toothbrush bristles: modelling and testing," *Multibody System Dynamics*, vol. 63, no. 1, pp. 159-181, 2025.
- A. Cellupica, M. Cirelli, G. Saggio, E. Gruppioni and P. P. Valentini, "An interactive digital-twin model for virtual reality environments to train in the use of a sensorized upper-limb prosthesis," *Algorithms*, vol. 17, no. 1, pp. 35, 2024.
- M. Cirelli, A. Cellupica, P. Canonico and P. P. Valentinni, "Impulse dynamics and augmented reality for real-time interactive digital twin exploration and interrogation," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 18, no. 2, pp. 929–941, 2024.
- A. Cellupica, L. D'Angelo, M. Cirelli, M. Mazur and P. P. Valentini, "Multibody approach to model toothbrush bristles elasto-kinematics," in *Proceedings of the ECCOMAS Thematic Conference on Multibody Dynamics, Lisbon, Portugal*, pp. 24–28, 2023
- L. D'Angelo, M. Cirelli, O. Giannini, A. Cellupica and P. P. Valentini, "Uncertainty analysis of a contact-based multibody model of meshing spur gears," in *Proceedings of the ECCOMAS Thematic Conference on Multibody Dynamics, Lisbon, Portugal* 2023

- M. Cirelli, A. Cellupica, L. D'Angelo, M. Mazur and P. P. Valentini, "Numerical and machine-aided experimental models for simulating the 3d compliance of a toothbrush," *Machines*, vol. 11, no. 8, p. 783, 2023.
- M. Cirelli, A. Cellupica, M. Cera, O. Giannini, P. P. Valentini and E. Pennestrí, "Dynamic analysis of lightweight gears through multibody models with movable teeth," *Materials Research Proceedings*, vol. 26, pp. 659–664, 2023.

Appendices

Appendix A

Rigid body transformations in space

A.1 Properties of the rotation matrix

The rotation matrix

$$\mathbf{R} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \tag{A.1}$$

has several remarkable properties that are worth highlighting [303].

If the distance between two points, for example P and S, must remain unchanged before and after the displacement, then it must hold that

$$\|\mathbf{p}_1 - \mathbf{s}_1\| = \|\mathbf{p}_0 - \mathbf{s}_0\|. \tag{A.2}$$

where $\mathbf{p}_0$ and $\mathbf{s}_0$ are the initial positions of the respective points, while $\mathbf{p}_1$ and $\mathbf{s}_1$ are the final positions. Substituting the equation

$$\mathbf{r}_1 = {}^1_0\mathbf{R} \cdot \mathbf{r}_0 = \mathbf{R} \cdot \mathbf{r}_0 \tag{A.3}$$

into A.2, it follows that

$$\begin{aligned} \|\mathbf{r}_1 - \mathbf{s}_1\| &= \|{}^1_0\mathbf{R}\,\mathbf{r}_0 - {}^1_0\mathbf{R}\,\mathbf{s}_0\| \\ &= \|{}^1_0\mathbf{R}\,(\mathbf{r}_0 - \mathbf{s}_0)\| \\ &= \sqrt{(\mathbf{r}_0 - \mathbf{s}_0)^T({}^1_0\mathbf{R})^T({}^1_0\mathbf{R})(\mathbf{r}_0 - \mathbf{s}_0)}. \end{aligned} \tag{A.4}$$

from which the orthogonality condition is obtained:

$$\mathbf{R}^T\mathbf{R} = \mathbf{I}. \tag{A.5}$$

The columns of matrix $\mathbf{R}$ can be interpreted as the unit vectors of the Cartesian axes $\mathbf{i}_1$, $\mathbf{j}_1$, and $\mathbf{k}_1$, whose components are evaluated in the reference frame $O - x_0y_0z_0$. Therefore, in addition to A.5, the following also holds:

$$\mathbf{R}\mathbf{R}^T = \mathbf{I}. \tag{A.6}$$

Hence,

$$\mathbf{R}^{-1} = \mathbf{R}^T \tag{A.7}$$

For the rigid motion to represent a rotation, the determinant of matrix $\mathbf{R}$ must be equal to +1:

$$\det(\mathbf{R}) = 1 \tag{A.8}$$

A.2 Displacement of a rigid body

Position of a point, from the book [303]. Let $\mathcal{S}$ be a Cartesian coordinate system whose position remains fixed in space, and let P be a generic point. The position of P in $\mathcal{S}$ is identified by the three coordinates x_P, y_P, and z_P (Figure A.1).

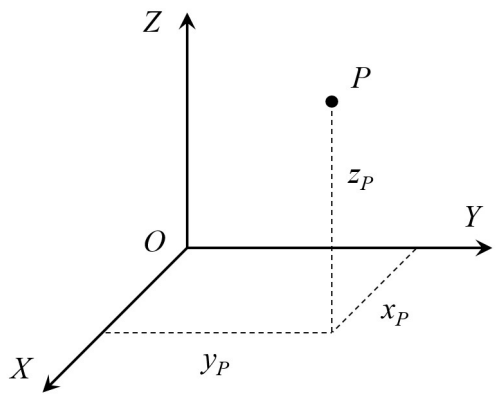


Figure A.1: Position of a point P.

We define a *rigid body* as a set of material points whose mutual distances remain unchanged. Although this concept is abstract, since all bodies in nature are deformable, it is widely used to study the motion of bodies in three-dimensional space.

To determine the position of an unconstrained body, the coordinates of at least three non-collinear points are required. Among these coordinates, however, three relations can be established that express the rigidity condition. Therefore, six independent coordinates are needed to define the position of a body in the Cartesian space $O - x_0y_0z_0$. For this purpose, it is customary to associate with the rigid body a Cartesian reference frame $O_1 - x_1y_1z_1$ fixed to the body itself. Knowing the position of this Cartesian triad is equivalent to knowing the position of the body. The relation

$$\mathbf{r}_1 = \mathbf{R}\,\mathbf{r}_0 + \mathbf{d} \tag{A.9}$$

represents the initial position $\mathbf{r}_0$ and the final position $\mathbf{r}_1$, with components expressed with respect to the absolute reference system $O - x_0y_0z_0$, of a point belonging to the rigid body subjected to a displacement. The matrix $\mathbf{R}$ represents the rotation of the rigid body with respect to the absolute reference frame, while $\mathbf{d}$ is the vector containing the coordinates of the origin of the body-fixed reference frame with respect to the origin of the absolute reference frame. All vectors appearing in Eq. A.9 are expressed in the same Cartesian reference frame.

Relation A.9 is formally identical to

$$^{1}\mathbf{r} = {}^{1}_{0}\mathbf{R}\ {}^{0}\mathbf{r} + {}^{1}\mathbf{d} \tag{A.10}$$

which transforms the coordinates $^{0}\mathbf{r} = \{^{0}r_x\ ^{0}r_y\ ^{0}r_z\}^T$ of a generic point P, measured in the reference frame $O - x_0y_0z_0$, into the coordinates $^{1}\mathbf{r} = \{^{1}r_x\ ^{1}r_y\ ^{1}r_z\}^T$ of the same point in the reference frame $O_1 - x_1y_1z_1$. The matrix ${}^{1}_{0}\mathbf{R}$ represents the rotation matrix for passing from the reference frame $O - x_0y_0z_0$ to the reference frame $O_1 - x_1y_1z_1$, while the vector $^{1}\mathbf{d}$ contains the coordinates of the origin of the system $O_1 - x_1y_1z_1$ with respect to the origin of the system $O - x_0y_0z_0$.

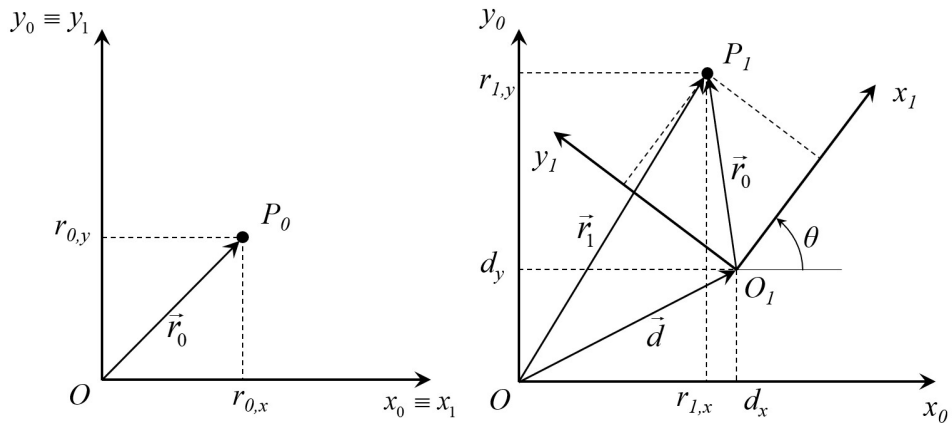


Figure A.2: Planar displacement of a rigid body: homologous positions of a point P.

In order to better understand the differences between Eq. A.9 and Eq. A.10, reference is made to the planar case. Considering a particular case of Eq. A.9, Figure A.2 represents the homologous positions, initial P_0 and final P_1, of a generic point P of the rigid body, such that the relation linking the coordinates of these positions is

$$\begin{Bmatrix} r_{1x} \\ r_{1y} \end{Bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{Bmatrix} r_{0x} \\ r_{0y} \end{Bmatrix} + \begin{Bmatrix} d_x \\ d_y \end{Bmatrix} \tag{A.11}$$

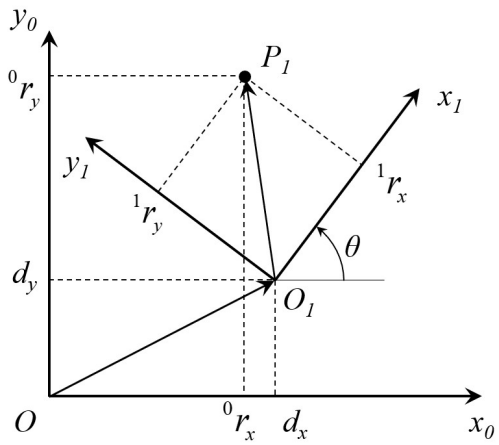


Figure A.3: Coordinate transformation.

The geometry of Figure A.3, on the other hand, illustrates the situation defined by Eq. A.10, which in this case specializes to the following

$$\begin{Bmatrix} {}^1r_x \\ {}^1r_y \end{Bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{Bmatrix} {}^0r_x \\ {}^0r_y \end{Bmatrix} + \begin{Bmatrix} {}^1d_{0x} \\ {}^1d_{0y} \end{Bmatrix} \tag{A.12}$$

with

$$\begin{Bmatrix} {}^1d_{0x} \\ {}^1d_{0y} \end{Bmatrix} = -\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{Bmatrix} d_x \\ d_y \end{Bmatrix} \tag{A.13}$$

Using the properties of transformation matrices (Appendix A.1), it is necessary to point out that $\mathbf{R} = {}^1_0\mathbf{R}^T = {}^0_1\mathbf{R}$. Therefore, the matrix relations concerning coordinate transformations between two distinct Cartesian reference frames also characterize rigid displacements that lead to the superposition of the homologous axes of the aforementioned reference systems.

A.3 Orientation of a rigid body

The relative orientation between two bodies c_0 and c_1 is defined by the matrix ${}^1_0\mathbf{R}$, which transforms the vector ${}_0\mathbf{r}$, whose components are expressed in the Cartesian reference frame attached to c_0, into the same vector ${}_1\mathbf{r}$, but with components measured in the coordinate system attached to c_1 [303].

This transformation is described by the matrix equation, which represents a spherical motion (${}^1\mathbf{d} = 0$ in equation A.10, to obtain the equation A.14)

$${}^1\mathbf{r} = {}^1_0\mathbf{R}\ {}^0\mathbf{r} \tag{A.14}$$

As is well known, the components of a generic vector $\mathbf{r} \equiv \{r_x, r_y, r_z\}^T$ depend on the vector basis adopted. In particular, if the reference frame $O_0 - x_0y_0z_0$ is chosen, these

components are obtained from the scalar products

$$ {}^0r_x = \mathbf{r} \cdot \mathbf{i}_0, \tag{A.15a} $$
$$ {}^0r_y = \mathbf{r} \cdot \mathbf{j}_0, \tag{A.15b} $$
$$ {}^0r_z = \mathbf{r} \cdot \mathbf{k}_0, \tag{A.15c} $$

and the vector, in the basis formed by the unit vectors $\mathbf{i}_0, \mathbf{j}_0$, and $\mathbf{k}_0$, is expressed by the relation

$$ \mathbf{r} = {}^0r_x\, \mathbf{i}_0 + {}^0r_y\, \mathbf{j}_0 + {}^0r_z\, \mathbf{k}_0. \tag{A.16} $$

Similarly, by adopting as basis the unit vectors $\mathbf{i}_1, \mathbf{j}_1$, and $\mathbf{k}_1$, we have

$$ \mathbf{r} = {}^1r_x\, \mathbf{i}_1 + {}^1r_y\, \mathbf{j}_1 + {}^1r_z\, \mathbf{k}_1, \tag{A.17} $$

with

$$ {}^1r_x = \mathbf{r} \cdot \mathbf{i}_1, \tag{A.18a} $$
$$ {}^1r_y = \mathbf{r} \cdot \mathbf{j}_1, \tag{A.18b} $$
$$ {}^1r_z = \mathbf{r} \cdot \mathbf{k}_1, \tag{A.18c} $$

By comparing Eqs. A.14, A.16, and A.17, we obtain

$$ a_{11}\,(\mathbf{r} \cdot \mathbf{i}_0) + a_{12}\,(\mathbf{r} \cdot \mathbf{j}_0) + a_{13}\,(\mathbf{r} \cdot \mathbf{k}_0) = (\mathbf{r} \cdot \mathbf{i}_1)\,, \tag{A.19a} $$
$$ a_{21}\,(\mathbf{r} \cdot \mathbf{i}_0) + a_{22}\,(\mathbf{r} \cdot \mathbf{j}_0) + a_{23}\,(\mathbf{r} \cdot \mathbf{k}_0) = (\mathbf{r} \cdot \mathbf{j}_1)\,, \tag{A.19b} $$
$$ a_{31}\,(\mathbf{r} \cdot \mathbf{i}_0) + a_{32}\,(\mathbf{r} \cdot \mathbf{j}_0) + a_{33}\,(\mathbf{r} \cdot \mathbf{k}_0) = (\mathbf{r} \cdot \mathbf{k}_1)\,. \tag{A.19c} $$

Therefore, in defining a transformation between the components $\{{}^0r_x, {}^0r_y, {}^0r_z\}^T$ and $\{{}^1r_x, {}^1r_y, {}^1r_z\}^T$, we must first identify the coefficients a_{ij} $(i = 1, 2, 3;\ j = 1, 2, 3)$ in the relations

$$ \mathbf{i}_1 = a_{11}\, \mathbf{i}_0 + a_{12}\, \mathbf{j}_0 + a_{13}\, \mathbf{k}_0 \tag{A.20a} $$
$$ \mathbf{j}_1 = a_{21}\, \mathbf{i}_0 + a_{22}\, \mathbf{j}_0 + a_{23}\, \mathbf{k}_0 \tag{A.20b} $$
$$ \mathbf{k}_1 = a_{31}\, \mathbf{i}_0 + a_{32}\, \mathbf{j}_0 + a_{33}\, \mathbf{k}_0. \tag{A.20c} $$

Taking into account the general expressions in A.15, Eqs. A.20 imply the equalities

$$ a_{11} = \mathbf{i}_0 \cdot \mathbf{i}_1, \quad a_{12} = \mathbf{j}_0 \cdot \mathbf{i}_1, \quad a_{13} = \mathbf{k}_0 \cdot \mathbf{i}_1, \tag{A.21a} $$
$$ a_{21} = \mathbf{i}_0 \cdot \mathbf{j}_1, \quad a_{22} = \mathbf{j}_0 \cdot \mathbf{j}_1, \quad a_{23} = \mathbf{k}_0 \cdot \mathbf{j}_1, \tag{A.21b} $$
$$ a_{31} = \mathbf{i}_0 \cdot \mathbf{k}_1, \quad a_{32} = \mathbf{j}_0 \cdot \mathbf{k}_1, \quad a_{33} = \mathbf{k}_0 \cdot \mathbf{k}_1, \tag{A.21c} $$

which allow us to interpret the meaning of the elements of the transformation matrix.

Indeed, it can be observed that the elements of the columns of such matrix coincide with the cosines of the angles formed between the unit vectors of the two reference systems. In particular, the elements of the first column are, respectively, the cosines of the angles formed by the axis x_0 with the axes x_1, y_1, and z_1. For this reason, $[{}^1_0\mathbf{R}]$ is also called the *direction cosine matrix.*

If we now substitute Eqs. A.20 into Eq. A.17, we obtain

$$\begin{aligned}\mathbf{r} = &\left(a_{11}\ {}^1r_x + a_{21}\ {}^1r_y + a_{31}\ {}^1r_z\right)\mathbf{i}_0 \\ &+ \left(a_{12}\ {}^1r_x + a_{22}\ {}^1r_y + a_{32}\ {}^1r_z\right)\mathbf{j}_0 \\ &+ \left(a_{13}\ {}^1r_x + a_{23}\ {}^1r_y + a_{33}\ {}^1r_z\right)\mathbf{k}_0\end{aligned} \tag{A.22}$$

and, by comparison with Eq. A.16, the equalities follow

$${}^0r_x = a_{11}\ {}^1r_x + a_{21}\ {}^1r_y + a_{31}\ {}^1r_z \tag{A.23a}$$

$${}^0r_y = a_{12}\ {}^1r_x + a_{22}\ {}^1r_y + a_{32}\ {}^1r_z \tag{A.23b}$$

$${}^0r_z = a_{13}\ {}^1r_x + a_{23}\ {}^1r_y + a_{33}\ {}^1r_z, \tag{A.23c}$$

or, equivalently, the matrix relations

$$\begin{Bmatrix} {}^0r_x \\ {}^0r_y \\ {}^0r_z \end{Bmatrix} = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \begin{Bmatrix} {}^1r_x \\ {}^1r_y \\ {}^1r_z \end{Bmatrix}, \tag{A.24}$$

$$\begin{Bmatrix} {}^1r_x \\ {}^1r_y \\ {}^1r_z \end{Bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{Bmatrix} {}^0r_x \\ {}^0r_y \\ {}^0r_z \end{Bmatrix}. \tag{A.25}$$

The obtained results confirm the properties of the transformation matrix ${}^1_0\mathbf{R}$. Since the basis formed by the unit vectors $\mathbf{i}_1$, $\mathbf{j}_1$, and $\mathbf{k}_1$ is orthogonal, the orthogonality condition A.5 follows directly.

It is important to observe that the following conditions hold among the nine elements of the transformation matrix:

$$a_{1m}^2 + a_{2m}^2 + a_{3m}^2 = 1, \quad (m = 1, 2, 3) \tag{A.26}$$

as a consequence of the fact that each unit vector has unit norm, and

$$a_{11}a_{12} + a_{12}a_{22} + a_{31}a_{32} = 0 \tag{A.27a}$$

$$a_{11}a_{13} + a_{12}a_{23} + a_{31}a_{33} = 0 \tag{A.27b}$$

$$a_{12}a_{13} + a_{22}a_{23} + a_{32}a_{33} = 0 \tag{A.27c}$$

due to the orthogonality of the aforementioned basis unit vectors.

Ultimately, since relation A.14 can represent both a spherical motion and a transformation between coordinate systems with coincident origins, we conclude that such motion is characterized by three independent parameters. Several methods for describing the orientation of a rigid body can be found in the literature, such as Euler angles, Rodrigues parameters, Euler parameters, and quaternions.

A.3.1 Composition of rotations about the coordinate axes

Consider two Cartesian coordinate systems. The first one is inertial, $O - XYZ$, and the second one is $O - xyz$, body-fixed to the rigid body, whose origin is always

superimposed on that of the first system. Let the body undergo a rotation of an angle θ_1 about the x-axis. As a result of this rotation, the system $O - xyz$ is brought to $O - x_1y_1z_1$, with the unit vectors of the new axes defined by

$$^1\mathbf{r} = {}^1_0\mathbf{R} \cdot {}^0\mathbf{r} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & \sin\theta_1 \\ 0 & -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \cdot {}^0\mathbf{r} \tag{A.28}$$

Subsequently, consider a rotation of angle θ_2 about the y_1-axis. The new position of the body-fixed reference frame is defined by the transformation

$$^2\mathbf{r} = {}^2_1\mathbf{R} \cdot {}^1\mathbf{r} = \begin{bmatrix} \cos\theta_2 & 0 & -\sin\theta_2 \\ 0 & 1 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 \end{bmatrix} \cdot {}^1\mathbf{r} \tag{A.29}$$

Finally, after a rotation by an angle equal to θ_3 about the z_2-axis, the position occupied by the body-fixed reference frame is similarly defined by the transformation

$$^3\mathbf{r} = {}^3_2\mathbf{R} \cdot {}^2\mathbf{r} = \begin{bmatrix} \cos\theta_3 & \sin\theta_3 & 0 \\ -\sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot {}^2\mathbf{r} \tag{A.30}$$

Combining equations A.28, A.29 and A.30 one obtains

$$^3\mathbf{r} = {}^3_2\mathbf{R}\,{}^2_1\mathbf{R}\,{}^1_0\mathbf{R}\,{}^0\mathbf{r} = {}^3_0\mathbf{R}\,{}^0\mathbf{r} \tag{A.31}$$

It is straightforward to verify that, since matrix products cannot in general be commuted (i.e. ${}^2_1\mathbf{R}\,{}^1_0\mathbf{R} \neq {}^1_0\mathbf{R}\,{}^2_1\mathbf{R}$), finite angular displacements cannot be represented through vectors whose sum instead satisfies the commutative property.

A.3.2 Euler angles

The orientation between two right-handed Cartesian reference frames with a common origin is considered, where the first frame, $O - XYZ$, is attached to a fixed body, while the second frame, $o - xyz$, is attached to a moving rigid body c_1.

Euler angles describe three successive finite rotations applied to body c_1 in order to bring the reference frame $o - xyz$ from its initial configuration aligned with $O - XYZ$ to the final configuration $o - x_3y_3z_3$, corresponding to the final orientation of the body.

Although the order according to which the rotations are applied is arbitrary [117], the 313 sequence (Figure A.4) is among the most commonly adopted conventions (e.g. [304])[1]:

1. Starting from the initial configuration (Figure A.4a), body c_1 undergoes a rotation about axis z through an angle ψ. The corresponding transformation matrix is

$$^1_0\mathbf{R} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{A.32}$$

[1]Since the sequence of finite rotations assigned to the body is arbitrary, three initial possibilities exist. As the only requirement imposed on successive rotations is that they must not occur about the same axis consecutively, it follows that, for right-handed Cartesian systems, Euler angles may be defined in twelve different ways.

After this displacement, axis x is mapped into x_1 and axis y into y_1, while $Z \equiv z_1$ (Figure A.4b).

2. Body c_1 then undergoes a rotation about axis x_1 through an angle θ. The corresponding transformation matrix is

$$ {}^2_1\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}. \tag{A.33} $$

Following this rotation, axis y_1 is mapped into y_2 and axis z_1 into z_2, while axis x_1 remains unchanged, so that $x_1 \equiv x_2$ (Figure A.4c).

3. Finally, body c_1 undergoes a rotation about axis z_2 through an angle φ. The corresponding transformation matrix is

$$ {}^3_2\mathbf{R} = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{A.34} $$

After this rotation, axis x_2 is mapped into x_3, axis y_2 into y_3, while $z_3 \equiv z_2$ (Figure A.4d).

By introducing the notation $c_\varphi = \cos\varphi$, $s_\varphi = \sin\varphi$, etc., the matrix transforming the components of vector $\{r\}$ expressed in frame $O - XYZ$ into the components of the same vector expressed in frame $o - x_3y_3z_3$ after rotation is obtained from the matrix product

$$ {}^3_0\mathbf{R} = {}^3_2\mathbf{R}\,{}^2_1\mathbf{R}\,{}^1_0\mathbf{R} = \begin{bmatrix} c_\varphi c_\psi - s_\varphi c_\theta s_\psi & c_\varphi s_\psi + s_\varphi c_\theta c_\psi & s_\varphi s_\theta \\ -s_\varphi c_\psi - c_\varphi c_\theta s_\psi & -s_\varphi s_\psi + c_\varphi c_\theta c_\psi & c_\varphi s_\theta \\ s_\theta s_\psi & -s_\theta c_\psi & c_\theta \end{bmatrix}. \tag{A.35} $$

The convention described above for defining Euler angles and the corresponding matrix $\mathbf{R}$ is referred to as the *x-convention*, since the intermediate rotation occurs about the position reached by axis x after the first rotation.

Euler angles can be computed from the elements of matrix $\mathbf{R}$ using the following relations:

$$ \cos\theta = a_{33}, \tag{A.36a} $$

$$ \sin\theta = \pm\sqrt{1 - \cos^2\theta}, \tag{A.36b} $$

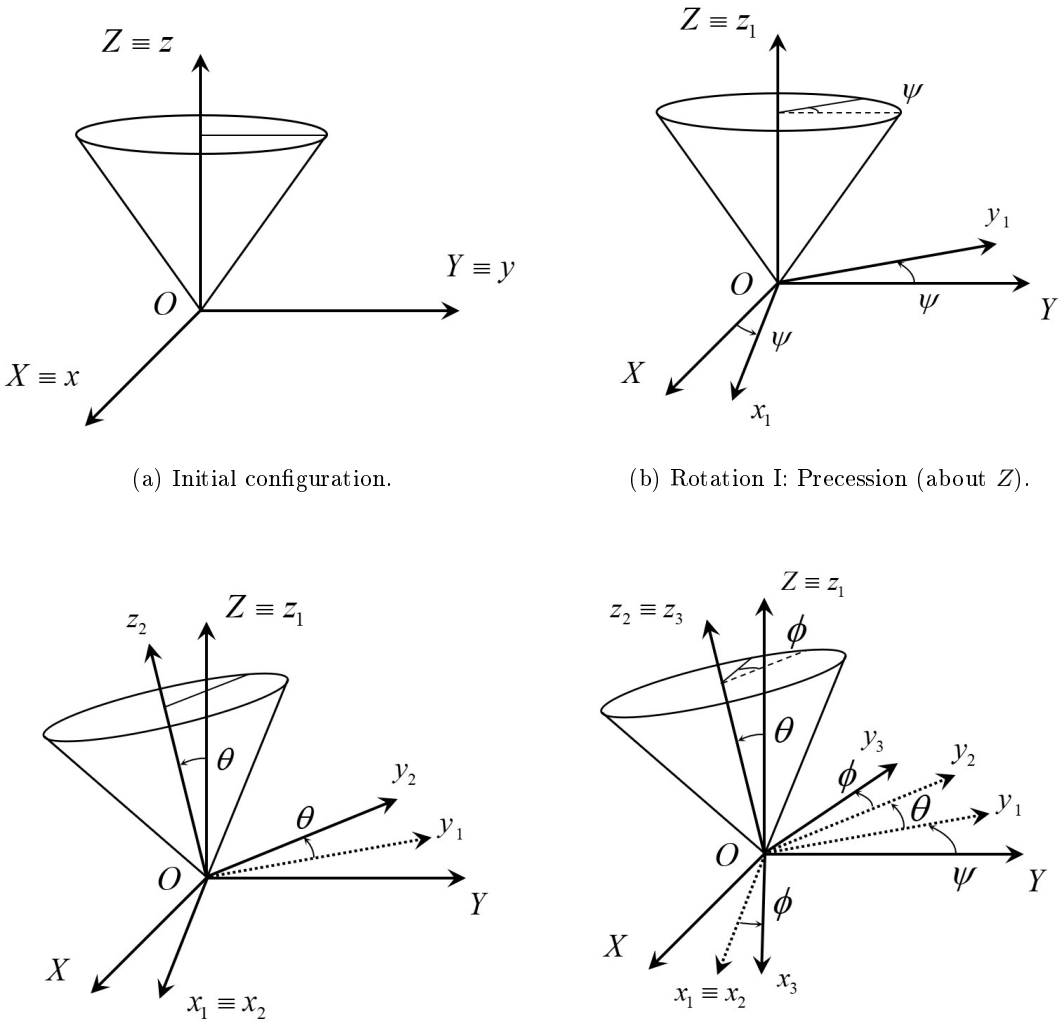
$$ \cos\psi = -\frac{a_{32}}{\sin\theta}, \tag{A.36c} $$

$$ \sin\psi = \frac{a_{31}}{\sin\theta}, \tag{A.36d} $$

$$ \cos\varphi = \frac{a_{23}}{\sin\theta}, \tag{A.36e} $$

$$ \sin\varphi = \frac{a_{13}}{\sin\theta}. \tag{A.36f} $$

It is evident that these expressions lead to numerical singularities when $\theta = \pm n\pi$ $(n = 0, 1, \dots)$. This drawback can be overcome by introducing the *Euler parameters*, which can be derived from the Cayley formula and Rodrigues parameters.



(a) Initial configuration.

(b) Rotation I: Precession (about Z).

(c) Rotation II: Nutation (about x_1).

(d) Rotation III: Spin (about z_2).

Figure A.4: Definition of Euler angles according to the 313 convention.

A.3.3 Rodrigues' formula

The following formulation makes it possible to determine the position of a vector attached to a rigid body undergoing a rigid rotation about a fixed axis. With reference to Figure A.5, let:

- $\mathbf{u}$ denote the unit vector defining the axis of rotation;
- θ denote the rotation angle of the body about $\mathbf{u}$;
- $\mathbf{r}_1$ denote the vector identifying the new position of vector $\mathbf{r}_0$ after the rigid rotation about $\mathbf{u}$;
- $\mathbf{q}_0$ and $\mathbf{q}_1$ denote the projections of $\mathbf{r}_0$ and $\mathbf{r}_1$, respectively, onto plane Π, orthogonal to the rotation axis and passing through the endpoints S_0 and S_1 of the corresponding vectors;
- O denote a point on the rotation axis, P the intersection between the axis and plane Π, and Q the foot of the perpendicular drawn from S_1 to vector $\mathbf{q}_0$.

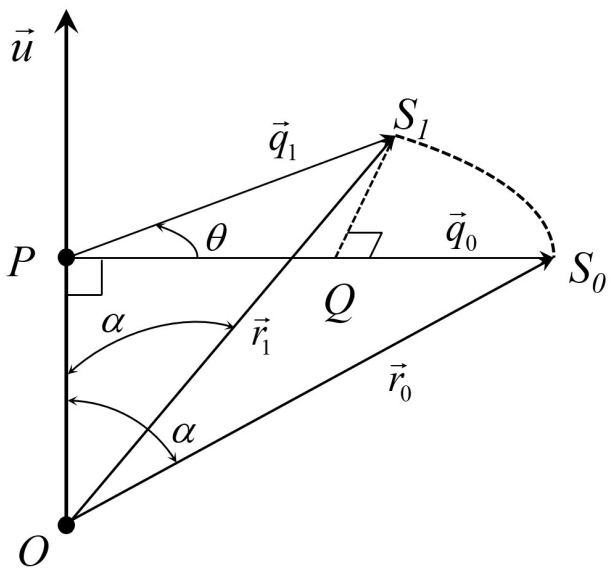


Figure A.5: Notation adopted for Rodrigues parameters.

Rodrigues' formula can be derived through a purely geometric treatment; however, such an approach would not naturally lead to the introduction of Rodrigues parameters and is therefore omitted in the present discussion. An alternative derivation can be obtained through the application of the Cayley formula. This approach enables the introduction of three independent parameters, alternative to Euler angles, for describing the rotation of a rigid body.

A.3.3.1 Cayley's formula

In spherical motion, vector $\mathbf{r}$ preserves its magnitude, so that its initial and final positions define a rhombus whose diagonals are orthogonal.

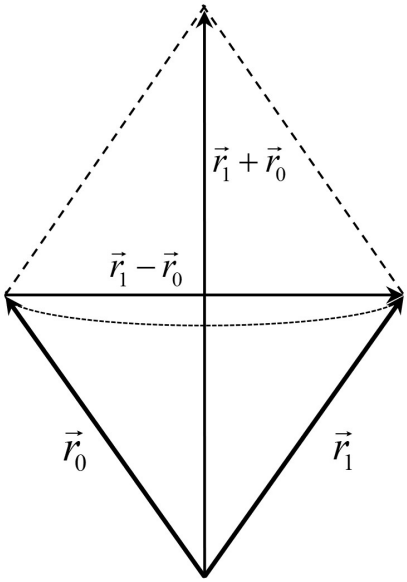


Figure A.6: Cayley formula.

With reference to the geometry shown in Figure A.6, the orthogonality condition can be established as

$$(\mathbf{r}_1 - \mathbf{r}_0)^T (\mathbf{r}_1 + \mathbf{r}_0) = 0. \tag{A.37}$$

Since

$$\mathbf{r}_1 = \mathbf{R}\,\mathbf{r}_0, \tag{A.38}$$

it follows that

$$(\mathbf{r}_1 - \mathbf{r}_0) = [\mathbf{R} - \mathbf{I}]\,\mathbf{r}_0, \tag{A.39a}$$

$$(\mathbf{r}_1 + \mathbf{r}_0) = [\mathbf{R} + \mathbf{I}]\,\mathbf{r}_0, \tag{A.39b}$$

and by defining

$$\mathbf{B} = [\mathbf{R} - \mathbf{I}]\,[\mathbf{R} + \mathbf{I}]^{-1} \tag{A.40}$$

one obtains

$$(\mathbf{r}_1 - \mathbf{r}_0) = \mathbf{B}\,(\mathbf{r}_1 + \mathbf{r}_0). \tag{A.41}$$

Therefore, when matrix $\mathbf{B}$ operates on vector $(\mathbf{r}_1 + \mathbf{r}_0)$, it generates vector $(\mathbf{r}_1 - \mathbf{r}_0)$, which is orthogonal to the former. In order for this to be possible, matrix $\mathbf{B}$ must be skew-symmetric[2], namely,

$$\mathbf{B} = \begin{bmatrix} 0 & -b_3 & b_2 \\ b_3 & 0 & -b_1 \\ -b_2 & b_1 & 0 \end{bmatrix}, \tag{A.42}$$

[2]Such matrices satisfy the property $\mathbf{B} = -\mathbf{B}^T$. The eigenvalues of skew-symmetric matrices are either zero or purely imaginary.

which depends on only three independent parameters.

Rearranging Eq. A.40 with respect to matrix $\mathbf{R}$ yields

$$\mathbf{B}\,[\mathbf{R}+\mathbf{I}] = [\mathbf{R}-\mathbf{I}]\,, \tag{A.43}$$

or equivalently the *Cayley formula*

$$\mathbf{R} = [\mathbf{I}-\mathbf{B}]^{-1}\,[\mathbf{I}+\mathbf{B}]. \tag{A.44}$$

It is worth noting that Eq. A.44 can alternatively be written as[3]

$$\mathbf{R} = [\mathbf{I}+\mathbf{B}]\,[\mathbf{I}-\mathbf{B}]^{-1}. \tag{A.45}$$

Hence, the transformation matrix $\mathbf{R}$ can be expressed as a function of a 3×3 skew-symmetric matrix $\mathbf{B}$, whose definition requires only three independent parameters.

A.3.3.2 Rodrigues parameters

By applying the Cayley formula, Rodrigues' formula can be obtained through the introduction of Rodrigues parameters.

The Cayley formula makes it possible to compute matrix $\mathbf{R}$ from the skew-symmetric matrix

$$\mathbf{B} = \begin{bmatrix} 0 & -b_3 & b_2 \\ b_3 & 0 & -b_1 \\ -b_2 & b_1 & 0 \end{bmatrix} \tag{A.46}$$

which depends only on vector $\mathbf{b} = \{b_x \quad b_y \quad b_z\}^T$.

The objective is to derive an algebraic expression for the elements of vector $\mathbf{b}$ as functions of unit vector $\mathbf{u}$ defining the rotation axis and of rotation angle θ.

First, it is noted that the matrix product $\mathbf{B}\cdot\mathbf{y}$ is equivalent to the vector product $\mathbf{b}\times\mathbf{y}$, where $\mathbf{y}$ is a three-dimensional vector. Furthermore, from the Cayley formula,

$$[\mathbf{R}-\mathbf{I}]\mathbf{p}_1 = [\mathbf{I}-\mathbf{B}]^{-1}[\mathbf{I}+\mathbf{B}]\mathbf{p}_1 - \mathbf{p}_1, \tag{A.47}$$

where $\mathbf{p}_1$ is the unit vector along the rotation axis.

Premultiplying by $[\mathbf{I}-\mathbf{B}]$ gives

$$2\mathbf{B}\,\mathbf{p}_1 = \mathbf{0}. \tag{A.48}$$

This implies that $\mathbf{b}$ is parallel to $\mathbf{p}_1$, namely,

$$\mathbf{b} = c\cdot\mathbf{p}_1, \tag{A.49}$$

where c is an unknown scalar.

From Eq. A.41,

$$(\mathbf{r}_1-\mathbf{r}_0) = \mathbf{b}\times(\mathbf{r}_1+\mathbf{r}_0), \tag{A.50}$$

[3]Letting $\mathbf{R}_1 = [\mathbf{I}-\mathbf{B}]^{-1}\,[\mathbf{I}+\mathbf{B}]$ and $\mathbf{R}_2 = [\mathbf{I}+\mathbf{B}][\mathbf{I}-\mathbf{B}]^{-1}$ denote the two expressions whose equivalence is to be demonstrated, the skew-symmetry property of $\mathbf{B}$ implies that $\mathbf{R}_1^T\,\mathbf{R}_2 = \mathbf{I}$ and $\mathbf{R}_2^T\,\mathbf{R}_1 = \mathbf{I}$. These relations hold simultaneously only if $\mathbf{R}_1 = \mathbf{R}_2$.

and, according to the notation of Figure A.5,

$$(\mathbf{q}_1 - \mathbf{q}_0) = \mathbf{b} \times (\mathbf{q}_1 + \mathbf{q}_0).$$

Thus,

$$\|\mathbf{b}\| = \frac{\|\mathbf{q}_1 - \mathbf{q}_0\|}{\|\mathbf{q}_1 + \mathbf{q}_0\|} \tag{A.51}$$

From the geometry of Figure A.5, one obtains

$$\tan\frac{\theta}{2} = \frac{\|\mathbf{q}_1 - \mathbf{q}_0\|/2}{\|\mathbf{q}_1 + \mathbf{q}_0\|/2}, \tag{A.52}$$

and comparison of Eqs. A.51 and A.52 gives

$$\|\mathbf{b}\| = \tan\frac{\theta}{2}. \tag{A.53}$$

Since $\mathbf{p}_1 = \mathbf{u}$, it follows that

$$\mathbf{b} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix} = \tan\frac{\theta}{2}\mathbf{u} \tag{A.54}$$

defines the *Rodrigues vector*, whose components are referred to as the *Rodrigues parameters*. In order to obtain an algebraic expression for matrix $\mathbf{R}$ as a function of these parameters, the following identities are considered:

$$[\mathbf{I} - \mathbf{B}] = \begin{bmatrix} 1 & b_3 & -b_2 \\ -b_3 & 1 & b_1 \\ b_2 & -b_1 & 1 \end{bmatrix}, \tag{A.55}$$

$$\det[\mathbf{I} - \mathbf{B}] = \Delta = b_1^2 + b_2^2 + b_3^2 + 1, \tag{A.56}$$

$$[\mathbf{I} - \mathbf{B}]^{-1} = \frac{1}{\Delta}\begin{bmatrix} 1 + b_1^2 & b_1b_2 - b_3 & b_1b_3 + b_2 \\ b_1b_2 + b_3 & 1 + b_2^2 & b_2b_3 - b_1 \\ b_1b_3 - b_2 & b_2b_3 + b_1 & 1 + b_3^2 \end{bmatrix}, \tag{A.57}$$

$$[\mathbf{I} + \mathbf{B}] = \begin{bmatrix} 1 & -b_3 & b_2 \\ b_3 & 1 & -b_1 \\ -b_2 & b_1 & 1 \end{bmatrix}, \tag{A.58}$$

therefore,

$$\begin{aligned} \mathbf{R} &= [\mathbf{I} - \mathbf{B}]^{-1}[\mathbf{I} + \mathbf{B}] \\ &= \frac{1}{\Delta}\begin{bmatrix} 1 + b_1^2 - b_2^2 - b_3^2 & 2(b_1b_2 - b_3) & 2(b_1b_3 + b_2) \\ 2(b_1b_2 + b_3) & 1 - b_1^2 + b_2^2 - b_3^2 & 2(b_2b_3 - b_1) \\ 2(b_1b_3 - b_2) & 2(b_2b_3 + b_1) & 1 - b_1^2 - b_2^2 + b_3^2 \end{bmatrix}. \end{aligned} \tag{A.59}$$

Substituting the components of $\mathbf{b}$, as defined in Eq. A.54, into Eq. A.46 and applying the Cayley formula yields[4]

$$\begin{aligned}\mathbf{R} &= [\mathbf{I}-\mathbf{B}]^{-1}[\mathbf{I}+\mathbf{B}] \\ &= \left[\mathbf{I}-\tan\frac{\theta}{2}\tilde{\mathbf{u}}\right]^{-1}\left[\mathbf{I}+\tan\frac{\theta}{2}\tilde{\mathbf{u}}\right] \\ &= \mathbf{I}+\sin\theta\tilde{\mathbf{u}}+(1-\cos\theta)\tilde{\mathbf{u}}\,\tilde{\mathbf{u}}. \end{aligned} \tag{A.60}$$

By considering the relation

$$\tilde{\mathbf{u}}^2 = \mathbf{u}\mathbf{u}^T - \mathbf{I} \tag{A.61}$$

Eq. A.60 may be rewritten in the form

$$\mathbf{R} = \mathbf{I}\cos\theta + \sin\theta\tilde{\mathbf{u}} + (1-\cos\theta)\mathbf{u}\mathbf{u}^T]. \tag{A.62}$$

Expanding the previous expression for $\mathbf{u} = \{u_x \quad u_y \quad u_z\}^T$ gives

$$\mathbf{R} = \begin{bmatrix} u_x^2V_\theta + c_\theta & u_xu_yV_\theta - u_zs_\theta & u_xu_zV_\theta + u_ys_\theta \\ u_xu_yV_\theta + u_zs_\theta & u_y^2V_\theta + c_\theta & u_yu_zV_\theta - u_xs_\theta \\ u_xu_zV_\theta - u_ys_\theta & u_yu_zV_\theta + u_xs_\theta & u_z^2V_\theta + c_\theta \end{bmatrix}, \tag{A.63}$$

where $V_\theta = 1-\cos\theta$, $s_\theta = \sin\theta$, and $c_\theta = \cos\theta$.

A.3.4 Euler parameters

Eq. A.60 can also be rewritten in the form

$$\mathbf{R} = \left[\cos\frac{\theta}{2}\mathbf{I} - \sin\frac{\theta}{2}\tilde{\mathbf{u}}\right]^{-1}\left[\cos\frac{\theta}{2}\mathbf{I} + \sin\frac{\theta}{2}\tilde{\mathbf{u}}\right]. \tag{A.64}$$

This suggests the introduction of four parameters e_0, e_1, e_2, and e_3, referred to as the *Euler parameters*, defined as follows:

$$e_0 = \cos\frac{\theta}{2}, \tag{A.65a}$$

$$e_1 = u_1\sin\frac{\theta}{2}, \tag{A.65b}$$

$$e_2 = u_2\sin\frac{\theta}{2}, \tag{A.65c}$$

$$e_3 = u_3\sin\frac{\theta}{2}, \tag{A.65d}$$

[4]The notation $\tilde{\mathbf{u}}$ denotes the *skew-symmetric matrix*

$$\tilde{\mathbf{u}} = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix},$$

associated with vector $\mathbf{u} = \{u_x, u_y, u_z\}^T$. In general, a square matrix is said to be skew-symmetric when, for any row i and column j, the condition $a_{ij} = -a_{ji}$ is satisfied.

By defining

$$\mathbf{e} = \mathbf{u} \sin \frac{\theta}{2}, \tag{A.66}$$

and recalling the trigonometric identities

$$1 - \cos\theta = 2\sin^2 \frac{\theta}{2}, \tag{A.67a}$$

$$\sin\theta = 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}, \tag{A.67b}$$

$$\cos\theta = 2\cos^2\frac{\theta}{2} - 1, \tag{A.67c}$$

Eq. A.62 provides Rodrigues' formula expressed in terms of Euler parameters:

$$\mathbf{r}_1 = \left[(2e_0^2 - 1)\mathbf{I} + 2\mathbf{e}\,\mathbf{e}^T + 2e_0\tilde{\mathbf{e}}\right]\mathbf{r}_0. \tag{A.68}$$

Since vector $\mathbf{r}_1$ represents the position of $\mathbf{r}_0$ after a rigid rotation about axis $\mathbf{u}$, it follows that the rotation matrix associated with such motion is

$$\mathbf{R} = \left[(2e_0^2 - 1)\mathbf{I} + 2\mathbf{e}\,\mathbf{e}^T + 2e_0\tilde{\mathbf{e}}\right], \tag{A.69}$$

or, in expanded form,

$$\mathbf{R} = 2\begin{bmatrix} e_0^2 + e_1^2 - \frac{1}{2} & e_1e_2 - e_0e_3 & e_1e_3 + e_0e_2 \\ e_1e_2 + e_0e_3 & e_0^2 + e_2^2 - \frac{1}{2} & e_2e_3 - e_0e_1 \\ e_1e_3 - e_0e_2 & e_2e_3 + e_0e_1 & e_0^2 + e_3^2 - \frac{1}{2} \end{bmatrix}. \tag{A.70}$$

For subsequent developments, it is useful to highlight that matrix $\mathbf{R}$ admits a factorized form. Introducing the matrices

$$\begin{aligned} \mathbf{E} &= \begin{bmatrix} -e_1 & e_0 & -e_3 & e_2 \\ -e_2 & e_3 & e_0 & -e_1 \\ -e_3 & -e_2 & e_1 & e_0 \end{bmatrix} \\ &= [-\mathbf{e}\,|\,\tilde{\mathbf{e}} + e_0\mathbf{I}], \end{aligned} \tag{A.71}$$

$$\begin{aligned} \mathbf{G} &= \begin{bmatrix} -e_1 & e_0 & e_3 & -e_2 \\ -e_2 & -e_3 & e_0 & e_1 \\ -e_3 & e_2 & -e_1 & e_0 \end{bmatrix} \\ &= [-\mathbf{e}\,|\,-\tilde{\mathbf{e}} + e_0\mathbf{I}], \end{aligned} \tag{A.72}$$

it can be readily verified that

$$\mathbf{R} = \mathbf{E}\,\mathbf{G}^T, \tag{A.73}$$

which indicates that the transformation described by matrix $\mathbf{R}$ may be interpreted as the result of two successive linear transformations.

The Euler parameters satisfy the constraint[5]

$$e_0^2 + e_1^2 + e_2^2 + e_3^2 = 1, \tag{A.74}$$

which directly follows from Eqs. A.65.

As in the case of Euler angles, practical computations require formulas allowing Euler parameters to be determined from the elements of rotation matrix $\mathbf{R}$. To this end, observe that

$$a_{11} + a_{22} + a_{33} = 2(3e_0^2 + e_1^2 + e_2^2 + e_3^2) - 3 = 4e_0^2 - 1, \tag{A.75}$$

from which

$$e_0^2 = \frac{a_{11} + a_{22} + a_{33} + 1}{4}. \tag{A.76}$$

Moreover, since

$$a_{11} = 2(e_0^2 + e_1^2) - 1, \tag{A.77}$$

and taking Eq. A.76 into account,

$$e_1^2 = \frac{1 + a_{11} - (a_{22} + a_{33})}{4}. \tag{A.78}$$

Similarly,

$$e_2^2 = \frac{1 + a_{22} - (a_{11} + a_{33})}{4}, \tag{A.79}$$

$$e_3^2 = \frac{1 + a_{33} - (a_{11} + a_{22})}{4}. \tag{A.80}$$

However, these relations do not determine the algebraic sign of the Euler parameters. Therefore, the following computational formulas are generally preferred:

$$e_1 = \frac{a_{32} - a_{23}}{4e_0}, \tag{A.81a}$$

$$e_2 = \frac{a_{13} - a_{31}}{4e_0}, \tag{A.81b}$$

$$e_3 = \frac{a_{21} - a_{12}}{4e_0}, \tag{A.81c}$$

to be used when $e_0 \neq 0$.

When $e_0 \approx 0$, it is recommended to employ, among the following expressions, those corresponding to the largest divisor e_i in absolute value ($i = 0, 1, 2, 3$):

- Case 1

$$e_1 = \pm\frac{\sqrt{1 + a_{11} - a_{22} - a_{33}}}{2}, \tag{A.82a}$$

$$e_2 = \frac{a_{12} + a_{21}}{4e_1}, \tag{A.82b}$$

$$e_3 = \frac{a_{13} + a_{31}}{4e_1}, \tag{A.82c}$$

$$e_0 = \frac{a_{32} - a_{23}}{4e_1}. \tag{A.82d}$$

[5]This condition is typically employed to reduce approximation and truncation errors arising during the numerical integration of the equations of motion. In practice, Euler parameters e_i ($i = 0, 1, 2, 3$), obtained through integration or other procedures, are commonly normalized according to $e_i^* = \frac{e_i}{\sqrt{\sum_{j=0}^{3} e_j^2}}$.

- Case 2

$$e_2 = \pm\frac{\sqrt{1 - a_{11} + a_{22} - a_{33}}}{2}, \tag{A.83a}$$

$$e_3 = \frac{a_{23} + a_{32}}{4e_2}, \tag{A.83b}$$

$$e_0 = \frac{a_{13} - a_{31}}{4e_2}, \tag{A.83c}$$

$$e_1 = \frac{a_{12} + a_{21}}{4e_2}. \tag{A.83d}$$

- Case 3

$$e_3 = \pm\frac{\sqrt{1 - a_{11} - a_{22} + a_{33}}}{2}, \tag{A.84a}$$

$$e_2 = \frac{a_{32} + a_{23}}{4e_3}, \tag{A.84b}$$

$$e_1 = \frac{a_{13} + a_{31}}{4e_3}, \tag{A.84c}$$

$$e_0 = \frac{a_{21} - a_{12}}{4e_3}. \tag{A.84d}$$

- Case 4

$$e_0 = \pm\frac{\sqrt{1 + a_{11} + a_{22} + a_{33}}}{2}, \tag{A.85a}$$

$$e_1 = \frac{a_{32} - a_{23}}{4e_0}, \tag{A.85b}$$

$$e_2 = \frac{a_{13} - a_{31}}{4e_0}, \tag{A.85c}$$

$$e_3 = \frac{a_{21} - a_{12}}{4e_0}. \tag{A.85d}$$

If $e_0 = 0$, the corresponding rotation is equal to 180°. Conversely, if $e_0 = 1$, then necessarily $e_1 = e_2 = e_3 = 0$, and matrix $\mathbf{R}$ reduces to the identity matrix $\mathbf{I}$.

A.3.5 Quaternions

Quaternions are algebraic entities introduced by William Rowan Hamilton in 1843, which are of considerable importance in computational mechanics, as they provide an effective means of representing the orientation of moving reference frames [303].

Within the framework of analytical mechanics, the use of quaternions makes it possible to avoid singularity issues in transformations to and from rotation matrices, unlike three-dimensional rotation parametrization methods. Moreover, the distinctive algebraic properties of quaternions, when properly exploited, allow rotations of points, angular velocities, and angular accelerations to be represented with significant advantages in terms of compactness and simplicity in symbolic computation.

These aspects represent only some of the main reasons motivating the widespread adoption of quaternions for describing rigid-body rotations in a variety of applications,

ranging from flight simulators to multibody codes, from CAD software to real-time aircraft control systems.

This chapter presents the fundamental properties of quaternion algebra and highlights its most relevant applications in mechanics.

A.3.5.1 Elements of quaternion algebra

Quaternion algebra, denoted as $\mathbb{H}(\cdot, +)$, constitutes a *non-commutative field*. As such, it possesses identity elements and satisfies both distributive and associative properties with respect to multiplication and addition. Unlike the algebraic structures of real and complex numbers, however, commutativity holds only for addition.

Since quaternions were originally conceived as an extension of complex numbers, they are also referred to as *hypercomplex numbers* or, with reference to their four-dimensional nature, *four-dimensional complex numbers*.

A quaternion is conventionally expressed as the sum of one real component and three imaginary components. Thus, given $\mathbf{q} \in \mathbb{H}$ and $q_0, q_1, q_2, q_3 \in \mathbb{R}$:

$$\mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k \tag{A.86}$$

where the imaginary units i, j, and k satisfy the following properties:

$$i^2 = -1 \tag{A.87a}$$
$$j^2 = -1 \tag{A.87b}$$
$$k^2 = -1 \tag{A.87c}$$
$$ijk = -1 \tag{A.87d}$$
$$ij = -ji = k \tag{A.87e}$$
$$jk = -kj = i \tag{A.87f}$$
$$ki = -ik = j \tag{A.87g}$$

The properties in Eq. A.87 may be summarized by the multiplication table reported in Table A.1, showing that the set $\mathcal{Q}_8 = \{1, -1, i, -i, j, -j, k, -k\}$ forms a particular eighth-order non-Abelian group [305].

·	1	i	j	k
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

Table A.1: Multiplication rules for the set $\{1, -1, i, -i, j, -j, k, -k\}$.

Based on these rules, quaternion addition and multiplication may be defined. In particular, the sum or difference of two quaternions $\mathbf{a}, \mathbf{b} \in \mathbb{H}$ produces a quaternion

$\mathbf{c} \in \mathbb{H}$ according to

$$\begin{aligned}\mathbf{c} &= \mathbf{a} \pm \mathbf{b} \\ &= (a_0 + a_1 i + a_2 j + a_3 k) \pm (b_0 + b_1 i + b_2 j + b_3 k) \\ &= (a_0 \pm b_0) + (a_1 \pm b_1)i + (a_2 \pm b_2)j + (a_3 \pm b_3)k \end{aligned} \tag{A.88}$$

Quaternion addition and subtraction are both associative and commutative:

$$\mathbf{a} \pm (\mathbf{b} \pm \mathbf{c}) = (\mathbf{a} \pm \mathbf{b}) \pm \mathbf{c} \tag{A.89a}$$

$$\mathbf{a} \pm \mathbf{b} = \mathbf{b} \pm \mathbf{a} \tag{A.89b}$$

The product of two quaternions $\mathbf{a}, \mathbf{b} \in \mathbb{H}$ generates a quaternion $\mathbf{c} \in \mathbb{H}$ according to

$$\begin{aligned}\mathbf{c} &= \mathbf{ab} \\ &= (a_0 + a_1 i + a_2 j + a_3 k)(b_0 + b_1 i + b_2 j + b_3 k) \\ &= (a_0 b_0 - a_1 b_1 - a_2 b_2 - a_3 b_3) \\ &\quad + (a_0 b_1 + a_1 b_0 + a_2 b_3 - a_3 b_2)i \\ &\quad + (a_0 b_2 - a_1 b_3 + a_2 b_0 + a_3 b_1)j \\ &\quad + (a_0 b_3 + a_1 b_2 - a_2 b_1 + a_3 b_0)k \end{aligned} \tag{A.90}$$

Quaternion multiplication is associative and distributive, but generally non-commutative:

$$\mathbf{a}(\mathbf{bc}) = (\mathbf{ab})\mathbf{c} \tag{A.91a}$$

$$\mathbf{ab} \neq \mathbf{ba} \tag{A.91b}$$

Scalar multiplication may also be introduced as a particular case of quaternion multiplication, where one quaternion possesses only a real component. For instance, the product between $\mathbf{q} \in \mathbb{H}$ and a scalar $s \in \mathbb{R}$ yields

$$\begin{aligned}\mathbf{r} &= \mathbf{q}s \\ &= \mathbf{q}(s + 0i + 0j + 0k) \\ &= sq_0 + sq_1 i + sq_2 j + sq_3 k \end{aligned} \tag{A.92}$$

In this case, scalar multiplication satisfies the commutative property:

$$\mathbf{a}s = s\mathbf{a} \tag{A.93a}$$

$$\mathbf{a}1 = 1\mathbf{a} \tag{A.93b}$$

$$sz\mathbf{a} = zs\mathbf{a} \tag{A.93c}$$

$$\mathbf{a}(s\mathbf{b} + z\mathbf{c}) = s\mathbf{ab} + z\mathbf{ac} \tag{A.93d}$$

Similarly to complex numbers, the conjugate of a quaternion is obtained by changing the sign of its imaginary part:

$$\begin{aligned}\mathbf{q} &= q_0 + q_1 i + q_2 j + q_3 k \\ \mathbf{q}^* &= q_0 - q_1 i - q_2 j - q_3 k \end{aligned} \tag{A.94}$$

The following identities hold:

$$(\mathbf{a}^*)^* = \mathbf{a} \tag{A.95a}$$
$$(\mathbf{ab})^* = \mathbf{b}^*\mathbf{a}^* \tag{A.95b}$$
$$(\mathbf{a}+\mathbf{b})^* = \mathbf{a}^* + \mathbf{b}^* \tag{A.95c}$$

A particularly important property concerns the product of a quaternion and its conjugate, which always yields a purely real quaternion:

$$\mathbf{qq}^* = q_0^2 + q_1^2 + q_2^2 + q_3^2 \tag{A.96a}$$
$$\mathbf{qq}^* = \mathbf{q}^*\mathbf{q} \in \mathbb{R} \tag{A.96b}$$

Equation A.96 naturally leads to the definition of the quaternion norm, analogous to the Euclidean norm in $\mathbb{R}^4$:

$$|\mathbf{q}| = \sqrt{\mathbf{qq}^*} \tag{A.97a}$$
$$|\mathbf{q}| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \tag{A.97b}$$

With the introduction of the norm, quaternion algebra $\mathbb{H}(\cdot, +, |\cdot|)$ becomes a Banach algebra [306], with $\mathbb{H}$ constituting a metric space. The following properties hold:

$$|\mathbf{q}| \geq 0 \tag{A.98a}$$
$$|\mathbf{q}| = 0 \iff \mathbf{q} = 0 \tag{A.98b}$$
$$|s\mathbf{q}| = s|\mathbf{q}| \tag{A.98c}$$
$$|\mathbf{q}+\mathbf{r}| \leq |\mathbf{q}| + |\mathbf{r}| \tag{A.98d}$$
$$|\mathbf{qr}| = |\mathbf{q}||\mathbf{r}| \tag{A.98e}$$

Quaternion algebra is also a division algebra, since for every $\mathbf{q} \neq 0$ there exists an inverse $\mathbf{q}^{-1}$ such that

$$\mathbf{q}^{-1}\mathbf{q} = 1 \tag{A.99}$$

From Eqs. A.96 and A.97, the inverse may be written as

$$\mathbf{q}^{-1} = \frac{\mathbf{q}^*}{|\mathbf{q}|^2} \tag{A.100}$$

For computational efficiency, it is worth noting that the inverse of a unit quaternion coincides with its conjugate:

$$|\mathbf{q}| = 1 \implies \mathbf{q}^{-1} = \mathbf{q}^* \tag{A.101}$$

A.3.5.2 Alternative notations

It is worth noting that, in addition to Eq. A.86, alternative notations exist for representing quaternions. In particular, Hamilton's original notation highlights the real scalar part $s \in \mathbb{R}$ and the imaginary vector part $\mathbf{v} \in \operatorname{Im}\mathbb{H}$, where $\mathbb{H} = \mathbb{R} \oplus \operatorname{Im}\mathbb{H}$:

$$\mathbf{q} = (s, \mathbf{v}) \tag{A.102}$$

This notation[6] allows a more compact formulation of addition and subtraction operations, as already introduced in Eqs. A.88 and A.90:

$$\mathbf{a} \pm \mathbf{b} = (s_a \pm s_b, \mathbf{v}_a \pm \mathbf{v}_b) \tag{A.103a}$$

$$\mathbf{ab} = (s_a s_b - \mathbf{v}_a \cdot \mathbf{v}_b,\ s_a \mathbf{v}_b + s_b \mathbf{v}_a + \mathbf{v}_a \times \mathbf{v}_b) \tag{A.103b}$$

where the dot product $(\cdot)$ and cross product $(\times)$ are used with their standard meaning in vector algebra in $\mathbb{R}^3$.

A quaternion $\mathbf{q} \in \operatorname{Im}\mathbb{H}$ with zero scalar part, i.e., $\mathbf{q} = (0, \mathbf{v})$, is referred to as a *pure* or *imaginary quaternion*, whereas a quaternion $\mathbf{q} \in \mathbb{R}$ with zero vector part is referred to as a *real quaternion*.

An alternative representation of quaternions is given by four-dimensional column vectors:

$$\mathbf{q} = \{q_0, q_1, q_2, q_3\}^T \tag{A.104}$$

This representation is particularly convenient when quaternion addition and multiplication are expressed using linear algebra tools. In this context, quaternion multiplication can be represented as a product between a 4×4 matrix and a 4×1 vector, according to the following relation derived from Eq. A.90:

$$\mathbf{ab} = \mathbf{c} \tag{A.105}$$

$$\begin{bmatrix} +a_0 & -a_1 & -a_2 & -a_3 \\ +a_1 & +a_0 & -a_3 & +a_2 \\ +a_2 & +a_3 & +a_0 & -a_1 \\ +a_3 & -a_2 & +a_1 & +a_0 \end{bmatrix} \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix} = \begin{Bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{Bmatrix} \tag{A.106}$$

For compactness, the matrices $\mathbf{M}_\oplus$ and $\mathbf{M}_\ominus$ are introduced:

$$\mathbf{ab} = \mathbf{c} \iff \mathbf{M}(\mathbf{a})_\oplus \cdot \mathbf{b} = \mathbf{M}(\mathbf{b})_\ominus \cdot \mathbf{a} \tag{A.107a}$$

$$\mathbf{M}(\mathbf{a})_\oplus = \begin{bmatrix} +a_0 & -a_1 & -a_2 & -a_3 \\ +a_1 & +a_0 & -a_3 & +a_2 \\ +a_2 & +a_3 & +a_0 & -a_1 \\ +a_3 & -a_2 & +a_1 & +a_0 \end{bmatrix} \tag{A.107b}$$

$$\mathbf{M}(\mathbf{b})_\ominus = \begin{bmatrix} +b_0 & -b_1 & -b_2 & -b_3 \\ +b_1 & +b_0 & +b_3 & -b_2 \\ +b_2 & -b_3 & +b_0 & +b_1 \\ +b_3 & +b_2 & -b_1 & +b_0 \end{bmatrix} \tag{A.107c}$$

[6] The notation $\mathbf{q} = (s, \mathbf{v})$ is commonly used in computer graphics and computational geometry.

From a computational complexity perspective, it is observed that the number of additions (12) and scalar multiplications (16) required to compute the product of two quaternions is identical in all three formulations: the representation in Eq. A.90, the vector form in Eq. A.103b, and the linear-algebra-based formulation in Eq. A.107. In particular, formulation A.107 may appear advantageous in computational implementations, due to the widespread availability of numerical libraries for matrix–vector operations. However, when generic matrix multiplication routines are used, row–column indexing operations may introduce additional computational overhead. Furthermore, when it is known a priori that a quaternion has zero scalar part (as will be the case in the representation of rotations), unnecessary operations involving structurally zero terms are still performed. Consequently, when one of the quaternions has zero scalar component, a simplified implementation of Eq. A.103b allows a saving of four scalar multiplications and four scalar additions.

A.3.5.3 Rotations

The following endomorphism $\mathbf{p} \in \mathbb{H} \rightarrow \mathbf{p}' \in \mathbb{H}$ is considered, which maps the quaternion $\mathbf{p}$ into a quaternion $\mathbf{p}'$:

$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^* \tag{A.108}$$

From property A.98, it follows that:

$$|\mathbf{p}'| = |\mathbf{q}||\mathbf{p}||\mathbf{q}^*| \tag{A.109}$$

If only unit-norm quaternions $\mathbf{q}$ (i.e., unit or unimodular quaternions) are considered, the norm of $\mathbf{p}$ is preserved under the transformation, since $|\mathbf{p}'| = |\mathbf{q}||\mathbf{p}||\mathbf{q}^*| = 1|\mathbf{p}|1$. Therefore, when $|\mathbf{q}| = 1$, the endomorphism A.108 defines a *rotation* of the quaternion $\mathbf{p}$. A particularly relevant case arises when $\mathbf{p} \in \mathrm{Im}\mathbb{H}$, since in this case, for $|\mathbf{q}| = 1$, the transformed quaternion also contains only an imaginary part, i.e., $\mathbf{p}' \in \mathrm{Im}\mathbb{H}$. This result can be shown by expanding the multiplications in A.108 with $\mathbf{p} = (0, \mathbf{v})$ and using the unimodularity condition $q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$.

On this basis, notation A.108 can be used to represent the rotation of a vector $\mathbf{v}$ (position vector in three-dimensional space) into a rotated vector $\mathbf{v}'$, by simply setting $\mathbf{p} = (0, \mathbf{v})$:

$$\begin{aligned} \mathbf{p}' &= \mathbf{q}\mathbf{p}\mathbf{q}^* \\ (0, \mathbf{v}') &= \mathbf{q}(0, \mathbf{v})\mathbf{q}^* \end{aligned} \tag{A.110}$$

It is evident that this notation allows sequences of rotations to be represented in a straightforward manner. For instance, applying a rotation of $\mathbf{p}$ into $\mathbf{p}'$ via quaternion $\mathbf{q}_a$, and subsequently rotating $\mathbf{p}'$ into $\mathbf{p}''$ via $\mathbf{q}_b$, yields:

$$\mathbf{p}' = \mathbf{q}_a \mathbf{p} \mathbf{q}_a^* \tag{A.111a}$$

$$\mathbf{p}'' = \mathbf{q}_b \mathbf{p}' \mathbf{q}_b^* \tag{A.111b}$$

$$\mathbf{p}'' = (\mathbf{q}_b \mathbf{q}_a)\mathbf{p}(\mathbf{q}_a^* \mathbf{q}_b^*) \tag{A.111c}$$

More generally, recalling A.95b, given a sequence of n rotations represented by quaternions $\mathbf{q}_1, \ldots, \mathbf{q}_n$, the following expression can be written:

$$\mathbf{p}' = \mathbf{q}_s \mathbf{p} \mathbf{q}_s^* \tag{A.112}$$

where $\mathbf{q}_s = (\mathbf{q}_n, \ldots, \mathbf{q}_2, \mathbf{q}_1)$. Since quaternion multiplication is not commutative, the order of multiplication of $\mathbf{q}_1, \ldots, \mathbf{q}_n$ cannot be arbitrarily changed. This is consistent with the fact that, in spatial rigid-body rotations, different results are obtained when the order of elementary rotations is modified.

The inverse transformation is obtained by premultiplying both sides of A.108 by $\mathbf{q}^*$ and postmultiplying by $\mathbf{q}$, yielding $\mathbf{q}^*\mathbf{p}'\mathbf{q} = \mathbf{q}^*\mathbf{q}\mathbf{p}\mathbf{q}^*\mathbf{q}$. Since $\mathbf{q}^*\mathbf{q} = \mathbf{q}\mathbf{q}^* = 1$, it follows that:

$$\begin{aligned} \mathbf{p}' &= \mathbf{q}\mathbf{p}\mathbf{q}^* \\ \mathbf{p} &= \mathbf{q}^*\mathbf{p}'\mathbf{q} \end{aligned} \tag{A.113}$$

Once it is established that the endomorphism A.108 represents a rotation via two quaternion multiplications, it is useful to derive an equivalence between this formulation and more conventional representations. For instance, given a rotation matrix $\mathbf{R}$ such that $\mathbf{v}' = \mathbf{R}\mathbf{v}$, it is often necessary to determine the quaternion $\mathbf{q}$ that produces the same rotation in $(0, \mathbf{v}') = \mathbf{q}(0, \mathbf{v})\mathbf{q}^*$, or conversely.

The product A.108 can be expressed in matrix form using A.107, yielding:

$$\begin{aligned} \mathbf{p}' &= \mathbf{M}(\mathbf{q})_\oplus \cdot (\mathbf{p}\mathbf{q}^*) \\ &= \mathbf{M}(\mathbf{q})_\oplus \cdot \mathbf{M}(\mathbf{q}^*)_\ominus \cdot \mathbf{p} \\ &= \begin{bmatrix} +q_0 & -q_1 & -q_2 & -q_3 \\ +q_1 & +q_0 & -q_3 & +q_2 \\ +q_2 & +q_3 & +q_0 & -q_1 \\ +q_3 & -q_2 & +q_1 & +q_0 \end{bmatrix} \begin{bmatrix} +q_0 & +q_1 & +q_2 & +q_3 \\ -q_1 & +q_0 & -q_3 & +q_2 \\ -q_2 & +q_3 & +q_0 & -q_1 \\ -q_3 & -q_2 & +q_1 & +q_0 \end{bmatrix} \mathbf{p} && \text{(A.114a)} \\ &= \begin{bmatrix} q_0^2 + q_1^2 + q_2^2 + q_3^2 & 0 & 0 & 0 \\ 0 & q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_3q_0) & 2(q_1q_3 + q_2q_0) \\ 0 & 2(q_1q_2 + q_3q_0) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(-q_1q_0 + q_2q_3) \\ 0 & 2(q_1q_3 - q_2q_0) & 2(q_1q_0 + q_2q_3) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \mathbf{p} && \text{(A.114b)} \end{aligned}$$

Since only the imaginary part of $\mathbf{p}' = (0, \mathbf{v}')$ is of interest, and since the real part of $\mathbf{p} = (0, \mathbf{v})$ is zero by assumption, the first row and first column of the linear system A.114b can be removed, yielding the following 3×3 rotation matrix:

$$\mathbf{v}' = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_3q_0) & 2(q_1q_3 + q_2q_0) \\ 2(q_1q_2 + q_3q_0) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(-q_1q_0 + q_2q_3) \\ 2(q_1q_3 - q_2q_0) & 2(q_1q_0 + q_2q_3) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \mathbf{v} \tag{A.115a}$$

$$\mathbf{v}' = \mathbf{R}(\mathbf{q}) \cdot \mathbf{v} \tag{A.115b}$$

It is noted that the matrix $\mathbf{R}(\mathbf{q})$ can also be expressed as a product of matrices $\mathbf{F}(\mathbf{q})_\oplus$ and $\mathbf{F}(\mathbf{q})_\ominus$, obtained by removing unused rows and columns from $\mathbf{M}(\mathbf{q})_\oplus$ and $\mathbf{F}(\mathbf{q}^*)_\ominus$ in A.114a:

$$\mathbf{v}' = \begin{bmatrix} +q_1 & +q_0 & -q_3 & +q_2 \\ +q_2 & +q_3 & +q_0 & -q_1 \\ +q_3 & -q_2 & +q_1 & +q_0 \end{bmatrix} \begin{bmatrix} +q_1 & +q_2 & +q_3 \\ +q_0 & -q_3 & +q_2 \\ +q_3 & +q_0 & -q_1 \\ -q_2 & +q_1 & +q_0 \end{bmatrix} \mathbf{v} \tag{A.116a}$$

$$\mathbf{v}' = \mathbf{F}(\mathbf{q})_\oplus \cdot \mathbf{F}(\mathbf{q})_\ominus^T \cdot \mathbf{v} \tag{A.116b}$$

where:

$$\mathbf{R}(\mathbf{q}) = \mathbf{F}(\mathbf{q})_{\oplus} \cdot \mathbf{F}(\mathbf{q})_{\ominus}^{T} \tag{A.117a}$$

$$\mathbf{F}(\mathbf{q})_{\oplus} = \begin{bmatrix} +q_1 & +q_0 & -q_3 & +q_2 \\ +q_2 & +q_3 & +q_0 & -q_1 \\ +q_3 & -q_2 & +q_1 & +q_0 \end{bmatrix} \tag{A.117b}$$

$$\mathbf{F}(\mathbf{q})_{\ominus} = \begin{bmatrix} +q_1 & +q_0 & +q_3 & -q_2 \\ +q_2 & -q_3 & +q_0 & +q_1 \\ +q_3 & +q_2 & -q_1 & +q_0 \end{bmatrix} \tag{A.117c}$$

From a practical standpoint, given a vector $\mathbf{v}$ and a unit quaternion $\mathbf{q}$ representing a rotation, the transformed vector $\mathbf{v}'$ can be computed either via the double quaternion product A.110, by explicitly constructing the rotation matrix $\mathbf{R}(\mathbf{q})$ and applying transformation A.115b, or by using the double linear transformation A.116a. In the first case, 24 additions and 32 multiplications are required; in the second case, 15 additions and 16 multiplications are needed to compute $\mathbf{R}(\mathbf{q})$, plus 6 additions and 9 multiplications for the matrix–vector product, for a total of 21 additions and 25 multiplications. However, by avoiding operations involving structural zeros, the quaternion-based formulation A.110 requires only 17 additions and 24 multiplications. A comparable efficiency is obtained with the third formulation A.116a, which also requires 17 additions and 24 multiplications.

Finally, it is noted that, while a unique rotation matrix $\mathbf{R}(\mathbf{q})$ corresponds to a given quaternion $\mathbf{q}$, two quaternions correspond to the same rotation matrix $\mathbf{R}$, differing only by sign. In fact, from A.115a, it can be verified that:

$$\mathbf{R}(\mathbf{q}) = \mathbf{R}(-\mathbf{q}) \tag{A.118}$$

A.3.5.4 Geometric interpretation of quaternions

Once it has been shown that unit quaternions can represent rotations in space, and given the possibility of transforming between quaternions and rotation matrices $\mathbf{R}$, the aim is to express quaternions in terms of other rotation parameters. In particular, by observing the formulation of the matrix $\mathbf{R}$ in terms of the *Euler parameters*, obtained in formula A.70, a perfect correspondence can be identified between these parameters and the four components of the quaternion. Specifically, given a rotation angle φ about an axis represented by a unit vector $\mathbf{u}$, one has:

$$q_0 = \cos\left(\frac{\varphi}{2}\right) \tag{A.119a}$$

$$q_1 = u_x \sin\left(\frac{\varphi}{2}\right) \tag{A.119b}$$

$$q_2 = u_y \sin\left(\frac{\varphi}{2}\right) \tag{A.119c}$$

$$q_3 = u_z \sin\left(\frac{\varphi}{2}\right) \tag{A.119d}$$

It is noted that the norm of the resulting quaternion is unitary for any choice of φ and $\mathbf{u}$, due to the properties $\sin^2\alpha + \cos^2\alpha = 1$ and $|\mathbf{u}| = 1$, since

$$|\mathbf{q}|^2 = \cos^2(\varphi/2) + (u_x^2 + u_y^2 + u_z^2)\sin^2(\varphi/2) = 1. \tag{A.120}$$

Among the various demonstrations showing that selecting the Euler parameters as

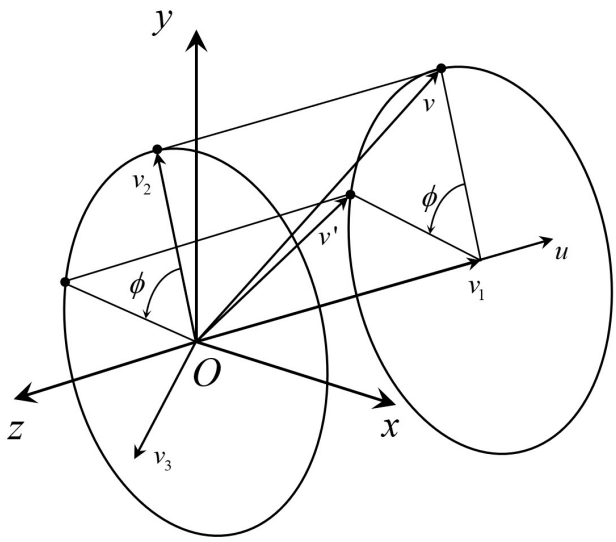


Figure A.7: Rotation of $\mathbf{v}$ about an axis $\mathbf{u}$.

quaternion components results in a rotation of angle φ about the axis $\mathbf{u}$, the following proof is reported, explicitly developing the quaternion product of formula A.110. With reference to Figure A.7 and recalling the trigonometric properties $\cos(\varphi) = \cos^2(\varphi/2) - \sin^2(\varphi/2)$, $\sin(\varphi) = 2\sin(\varphi/2)\cos(\varphi/2)$, $\cos(\varphi) = 1 - 2\sin^2(\varphi/2)$, one obtains:

$$
\begin{aligned}
\mathbf{p}' &= (0, \mathbf{v}') = \mathbf{q}\mathbf{p}\mathbf{q}^* = \\
&= \left(\cos\frac{\varphi}{2}, \mathbf{u}\sin\frac{\varphi}{2}\right)(0, \mathbf{v})\left(\cos\frac{\varphi}{2}, -\mathbf{u}\sin\frac{\varphi}{2}\right) = \\
&= \left(-\sin\frac{\varphi}{2}(\mathbf{u}\cdot\mathbf{v}), \cos\frac{\varphi}{2}\mathbf{v} + \sin\frac{\varphi}{2}(\mathbf{u}\times\mathbf{v})\right)\left(\cos\frac{\varphi}{2}, -\mathbf{u}\sin\frac{\varphi}{2}\right) = \\
&= \Big(-\sin\frac{\varphi}{2}\cos\frac{\varphi}{2}(\mathbf{u}\cdot\mathbf{v}) + \sin\frac{\varphi}{2}\cos\frac{\varphi}{2}(\mathbf{v}\cdot\mathbf{u}) - \sin^2\frac{\varphi}{2}(\mathbf{u}\times\mathbf{v})\cdot\mathbf{u}, \\
&\quad \sin^2\frac{\varphi}{2}(\mathbf{u}\cdot\mathbf{v})\mathbf{u} + \cos^2\frac{\varphi}{2}\mathbf{v} \\
&\quad + \sin\frac{\varphi}{2}\cos\frac{\varphi}{2}(\mathbf{u}\times\mathbf{v}) - \sin\frac{\varphi}{2}\cos\frac{\varphi}{2}(\mathbf{v}\times\mathbf{u}) - \sin^2\frac{\varphi}{2}((\mathbf{u}\times\mathbf{v})\times\mathbf{u})\Big) = \\
&= \left(0, \sin^2\frac{\varphi}{2}(\mathbf{u}\cdot\mathbf{v})\mathbf{u} + \cos^2\frac{\varphi}{2}\mathbf{v} + 2\sin\frac{\varphi}{2}\cos\frac{\varphi}{2}(\mathbf{u}\times\mathbf{v}) - \sin^2\frac{\varphi}{2}(\mathbf{v} - (\mathbf{u}\cdot\mathbf{v})\mathbf{u})\right) = \\
&= \left(0, 2\sin^2\frac{\varphi}{2}(\mathbf{u}\cdot\mathbf{v})\mathbf{u} + \cos^2\frac{\varphi}{2}\mathbf{v} - \sin^2\frac{\varphi}{2}\mathbf{v} + 2\sin\frac{\varphi}{2}\cos\frac{\varphi}{2}(\mathbf{u}\times\mathbf{v})\right) = \\
&= (0, (1-\cos\varphi)(\mathbf{u}\cdot\mathbf{v})\mathbf{u} + \cos\varphi\mathbf{v} + \sin\varphi(\mathbf{u}\times\mathbf{v})) = \\
&= (0, (\mathbf{u}\cdot\mathbf{v})\mathbf{u} + \cos\varphi(\mathbf{v} - (\mathbf{u}\cdot\mathbf{v})\mathbf{u}) + \sin\varphi(\mathbf{u}\times\mathbf{v}))
\end{aligned}
$$

The result, $\mathbf{v}' = (\mathbf{u}\cdot\mathbf{v})\mathbf{u} + \cos\varphi(\mathbf{v} - (\mathbf{u}\cdot\mathbf{v})\mathbf{u}) + \sin\varphi(\mathbf{u}\times\mathbf{v})$, coincides with the expression that would be obtained directly from Figure A.7 using elementary vector operations (in the figure, $\mathbf{v}_1 = (\mathbf{u}\cdot\mathbf{v})\mathbf{u}$, $\mathbf{v}_2 = \mathbf{v} - \mathbf{v}_1$ and $\mathbf{v}_3 = \mathbf{u}\times\mathbf{v}_2 = \mathbf{u}\times\mathbf{v}$). The inverse procedure is also possible, allowing the extraction of the rotation angle and axis from a

given quaternion, although in this case singularities occur for $\varphi = 0 \pm n2\pi, n \in \mathbb{N}$:

$$\varphi = 2\cos^{-1}(q_0) \tag{A.121a}$$

$$\mathbf{u} = \begin{Bmatrix} q_1 \\ q_2 \\ q_3 \end{Bmatrix} \frac{1}{\sin\frac{\varphi}{2}} \tag{A.121b}$$

Property A.118, according to which changing the sign of a quaternion yields the same rotation, i.e. $\mathbf{R}(\mathbf{q}) = \mathbf{R}(-\mathbf{q})$, can now be interpreted more intuitively, since a rotation of angle φ about axis $\mathbf{u}$ produces the same result as a rotation of angle $-\varphi$ about axis $-\mathbf{u}$. These two cases correspond to opposite signs in the components A.119.

Appendix B

Operational foundations of the Unity Engine

Unity is a cross-platform graphics engine developed by Unity Technologies and widely used for the creation of interactive three-dimensional applications, video games, simulators, and virtual and augmented reality systems. Thanks to its modular architecture and the level of abstraction provided to developers, Unity represents one of the most widely adopted platforms for the development of immersive applications.

From a functional point of view, Unity can be considered a *real-time 3D engine*, namely a platform capable of managing the real-time generation and updating of interactive three-dimensional environments. The engine integrates numerous subsystems required for the creation of immersive applications, including graphical rendering, physics management, animation systems, audio control, input management, and interaction with external hardware devices.

A distinctive feature of Unity is the presence of an integrated development environment (*Integrated Development Environment*, IDE) consisting of the Unity Editor. Through this environment, it is possible to design three-dimensional scenes, import and organize digital resources, configure the components of virtual objects, and control application behavior through scripts developed in the C# language.

Unity is based on a hierarchical structure in which the final application is composed of a collection of elements organized across multiple levels of abstraction. At the highest level is the Unity project (*Unity Project*), which represents the main container of all resources used by the application. A project includes files related to scenes, three-dimensional models, textures, materials, scripts, configurations, and external packages.

Within a project, the virtual environment is defined through one or more *Scenes*. A scene represents a configuration of the virtual space containing all elements required for visualization and interaction with the user. During application execution, scenes can be loaded, unloaded, or combined dynamically, allowing the creation of complex environments composed of multiple interconnected levels or areas.

Each scene consists of a set of *GameObjects*, namely generic entities to which specific functionalities are associated through independent *Components*. A GameObject does not directly define its graphical appearance or behavior, but represents a logical container to which different functional modules can be added.

The conceptual structure of Unity constitutes the foundation of the Unity development workflow and will be further discussed in the following sections of the appendix.

B.1 The Unity development environment

The Unity development environment is composed of the *Unity Editor*, through which scenes are designed, application objects are configured, and all project resources are managed. The interface is organized into several windows, each dedicated to a specific activity. Although the layout is fully customizable, the default configuration is the most commonly used and includes the windows shown in Figure B.1.

The *Hierarchy* window displays all the *GameObjects* present in the current scene according to a hierarchical structure. Objects can be organized through parent-child relationships, allowing the creation of complex structures and the automatic propagation of transformations and movements to the corresponding child elements.

The *Scene View* represents the main workspace of the editor and allows the construction and modification of the environment. From this window, it is possible to navigate through the scene, select objects, and modify their position, rotation, and scale through transformation tools.

The *Game View* instead displays the content rendered by the application camera and allows the final result to be verified during execution. Unlike the Scene View, this window reproduces the user's viewpoint and is mainly used during testing phases.

The properties of the selected GameObject are displayed in the *Inspector* window, from which it is possible to modify the parameters of the associated components, add new ones, or remove them. The Inspector represents the main configuration tool for objects present in the scene.

The *Project* window allows the exploration of all resources contained within the project folders. The structure of a Unity project is composed of some main directories. The `Assets` folder contains all resources developed or imported by the user, such as three-dimensional models, textures, materials, scripts, animations, and audio files. The `Packages` folder, instead, contains the packages installed and used within the project. The installation and management of packages are performed through the *Package Manager*, which allows software modules to be added, updated, or removed without manually modifying the project structure. The global application settings are finally collected in the *Project Settings* window, from which it is possible to configure parameters related to rendering, physics, the input system, plugins, and target platforms. Many of the configurations described in the following sections of the appendix are performed through this window. The `Library`, `Temp`, and `Logs` directories are instead automatically generated by the editor for cache and temporary file management and normally do not require developer intervention.

During development, compilation errors, debug messages, and warnings are reported in the *Console* window, which represents an essential tool for identifying and resolving errors both during development and during application execution.

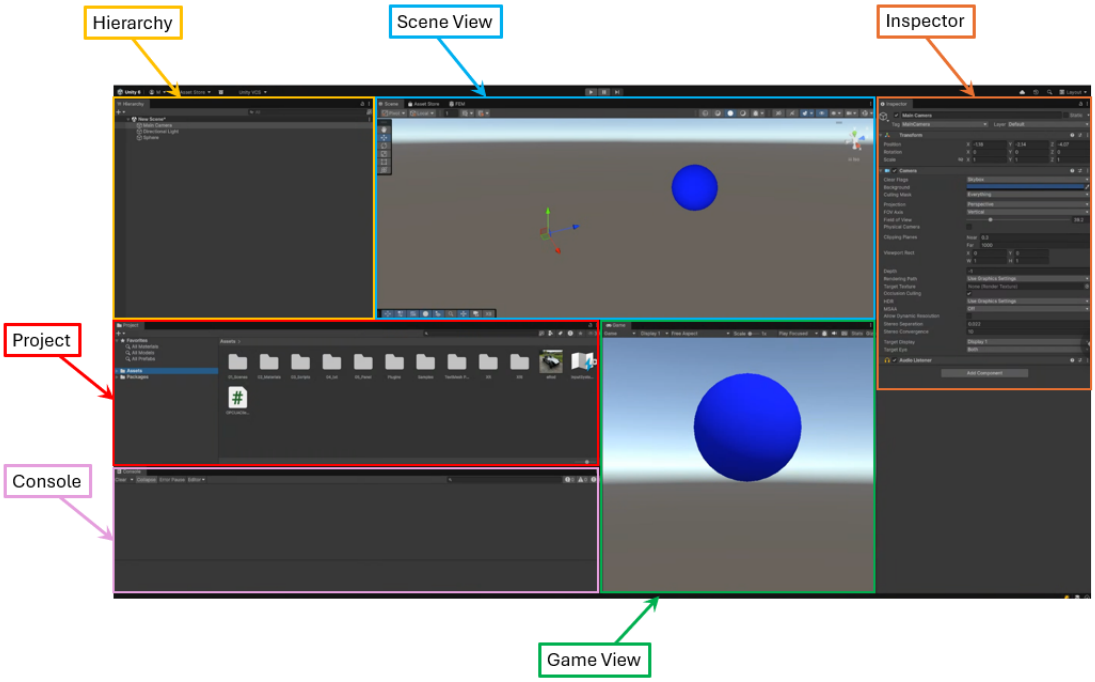


Figure B.1: Unity Editor interface.

B.2 Unity scene architecture

As described in the previous section, a Unity scene consists of a set of *GameObjects* organized hierarchically. The functionalities associated with each object are not directly defined by the GameObject itself, but by the components attached to it. This model, based on a modular (*component-based architecture*), represents one of the fundamental principles of Unity and allows the construction of complex objects by combining independent and easily reusable components. The components associated with a GameObject are visible in the *Inspector* window.

The only component present by default in every GameObject is the *Transform*, which defines the pose of the object in three-dimensional space through position, rotation, and scale. The transformations associated with a GameObject can be expressed with respect to the global reference system (*World Space*) or with respect to the local reference system defined by the parent object (*Local Space*). The relationship between these two reference systems is automatically managed by Unity through the composition of transformation matrices, avoiding the need for the developer to manually perform the mathematical operations required for coordinate conversion. The possibility of defining parent-child relationships between GameObjects allows the elements of the scene to be organized according to a hierarchical structure. In this configuration, the transformation of the child object is defined with respect to the parent's reference system, and any modification applied to the latter is automatically propagated to descendant objects, while maintaining their relative pose. This mechanism is particularly useful for managing systems composed of multiple interconnected elements, such as XR devices, where cameras, controllers, and other tracking components must maintain a coherent spatial relationship with respect to

a common reference.

When the same set of GameObjects must be reused in different parts of the project, Unity provides *Prefabs*. A Prefab represents a reusable model containing one or more GameObjects complete with all their related components and configurations. Modifications applied to the original Prefab can be automatically propagated to all instances present in different scenes.

Unity also provides the *Tags* and *Layers* systems, used respectively to logically identify GameObjects and organize them into different functional levels. *Tags* allow specific objects or object categories to be quickly identified through scripts, while *Layers* allow GameObjects to be classified and their behavior controlled in specific situations. One of the main uses of *Layers* concerns collision management: through the *Layer Collision Matrix*, accessible from the path *Project Settings* > *Physics* > *Layer Collision Matrix*, it is possible to define which Layers can interact with each other during the physics simulation, avoiding collision calculations between objects that should not influence one another. *Layers* can also be used to control the elements displayed by a specific camera through the *Culling Mask* parameter. This functionality allows specifying which categories of objects must be rendered by the camera and which must be excluded, proving particularly useful for creating multiple views or environments where different elements must be selectively displayed.

In order for a *GameObject* to represent a visible geometric element within the scene, it is necessary to associate the *Mesh Filter* and *Mesh Renderer* components. These two components perform distinct but complementary functions: the first defines the three-dimensional geometry of the object, while the second manages its visual representation during the rendering process. The *Mesh Filter* contains the reference to the *Mesh* associated with the GameObject, namely the geometric structure mainly composed of vertices and triangular faces. Vertices define the coordinates of points in three-dimensional space, while triangles determine how these points are connected to form the object's surface. A Mesh can also contain additional information associated with vertices, such as normals used for lighting calculations, texture coordinates (*UV mapping*), vertex colors, and other data required during the rendering phase. The *Mesh Renderer* is instead responsible for displaying the geometry within the scene. This component uses the Mesh provided by the *Mesh Filter* and applies one or more *Materials*, which define the visual properties of the object, such as color, texture, transparency, and lighting mode. Each Material contains the reference to the *Shader* used for rendering and its configuration parameters, which are responsible for calculating lighting and the final graphical output. During the rendering process, the *Mesh Renderer* sends the geometric information and associated materials to Unity's graphics pipeline, which are subsequently processed to generate the final image displayed to the user.

The physics simulation in Unity is mainly based on the interaction between the *Collider* and *Rigidbody* components, which represent the link between the geometric representation of objects and the dynamic model used by the physics engine.

As described in Section 4.1, collision detection requires a simplified representation of object geometry through appropriate *bounding volumes*. In Unity, these volumes are defined through *Collider* components, which describe the region of space used by the physics engine to identify possible intersections between scene objects. Colliders do not

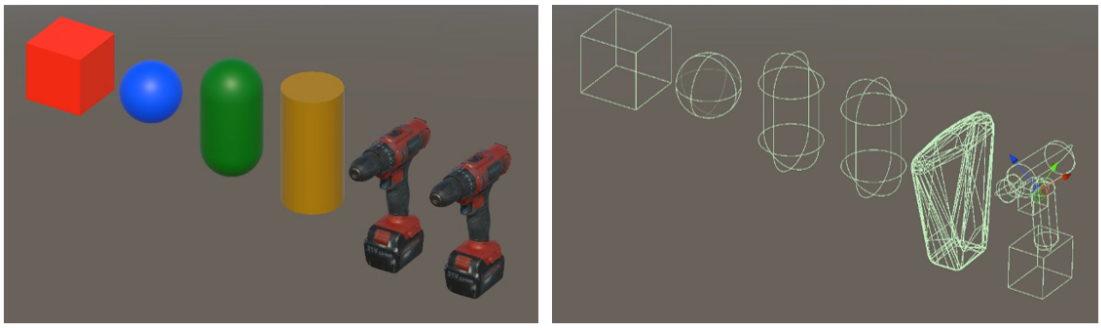


Figure B.2: Collider used in Unity.

necessarily coincide with the visible geometry of the object, but constitute a dedicated geometric representation for the simulation of physical interactions. Unity provides different types of Colliders, including *Box Collider*, *Sphere Collider*, *Capsule Collider*, and *Mesh Collider* (Figure B.2). Primitive Colliders are generally preferred when a geometric approximation of the object is sufficient, while the *Mesh Collider* allows the direct use of the three-dimensional mesh geometry, making it useful for objects with irregular convex shapes. For concave geometries, however, the Mesh Collider does not directly use the internal surface of the mesh, but generates a convex representation of the geometry. This approximation can introduce errors in collision detection; for this reason, whenever possible, complex geometry should be decomposed through a combination of primitive Colliders, obtaining a compromise between accuracy and computational cost. Each Collider has a set of parameters defining its behavior during simulation. The *Is Trigger* parameter modifies the interaction mode of the Collider: when disabled, the Collider participates in the physics simulation by generating a dynamic response to collisions; when enabled, the Collider is used exclusively for detecting intersections between objects, without generating forces or modifying the motion of involved bodies. This mode is frequently used to create activation zones, sensors, or interactions based on object presence detection. Colliders can also be associated with a *Physic Material*, which defines the properties of surface interaction, such as static friction coefficient, dynamic friction coefficient, and restitution coefficient. These parameters influence the resistance to relative motion between surfaces and the elastic behavior during impacts, allowing the modeling of materials with different characteristics.

The *Rigidbody* component instead allows a GameObject to be integrated into the Unity physics engine's dynamic system. Associating a Rigidbody makes the object a rigid body subject to equations of motion, enabling the application of forces, torques, gravity, and constraints generated by interactions with other scene elements. The Rigidbody contains all the properties required to define the dynamic behavior of the object. Among the main Rigidbody parameters is *Mass*, which defines the body mass and influences the response of the object to applied forces. The *Use Gravity* parameter enables or disables the effect of gravity, while *Is Kinematic* allows the body to be excluded from dynamic simulation while remaining within the collision system. A kinematic Rigidbody is not moved by physical forces, but can be directly controlled through transformations imposed by the developer. The Rigidbody also allows the configuration of the inertial

properties of the body. Normally, Unity automatically calculates the inertia tensor based on the geometry of the Collider associated with the GameObject. By disabling the *Automatic Tensor* option, it is possible to manually define the inertia matrix of the GameObject through the *Tensor of Inertia*, which contains the principal moments of inertia, and the *Tensor of Inertia Rotation*, which contains the orientation of the principal inertia axes with respect to the parent reference system through Euler angles. Another important parameter is represented by the *Center of Mass*, which identifies the point of application of the equivalent mass of the body. Although Unity can automatically calculate it through the associated Colliders, its position can be manually modified to correctly represent systems where the center of mass does not coincide with the geometric center of the object. Finally, the Rigidbody allows translations and rotations along individual Cartesian axes to be selectively blocked through the *Constraints* parameters. This functionality introduces kinematic constraints directly on the rigid body, preventing undesired motions during simulation.

During execution, the physics engine uses the information provided by the Colliders and the dynamic properties defined by the Rigidbody to detect collisions and calculate the corresponding response. As discussed in Sections 4.2 and 7.1, Unity uses an approach based on an iterative sequential impulse solver, in which collision-generated impulses are progressively applied to the involved bodies in order to satisfy the constraints of the multibody system.

The visualization of the scene is managed by the *Camera* component, which defines the application's observation point and determines the content sent to the rendering stage. The Camera specifies the reference system from which the scene is observed and controls several parameters related to image projection, including the field of view (*Field of View*), aspect ratio (*Aspect Ratio*), near and far clipping planes (*Near Clip Plane* and *Far Clip Plane*), and the type of projection used, generally perspective or orthographic. Through the *Culling Mask* parameter, it is also possible to define which scene Layers must be rendered by the Camera, allowing different views of the same virtual environment to be obtained. In virtual and augmented reality applications, the Camera plays a central role because it represents the user's viewpoint and is updated according to the tracking data provided by the XR device. In these systems, the position and orientation of the Camera are generally managed through the *XR Origin*, which defines the relationship between the virtual reference system and that of the real device.

The environment lighting is instead created through the *Light* component, which simulates the presence of light sources within the scene. Unity provides different types of lights, including *Directional Light*, *Point Light*, and *Spot Light*, characterized by different emission models. Directional light represents a source ideally placed at infinite distance, such as sunlight, and uniformly illuminates the entire scene according to a specific direction. Point light emits radiation in all directions from a defined position, while spot light illuminates a conical region of space defined by a direction and opening angle. Each light source can be configured through parameters such as intensity, color, influence distance, and illumination calculation mode. The combination of Camera, Light, Material, and Shader therefore determines the final result of the rendering process and the visual perception of the three-dimensional environment.

Unity supports different *Render Pipelines*, each characterized by a different com-

promise between graphical quality and performance. The *Built-in Render Pipeline* represents the historically integrated solution of the engine, while the *Universal Render Pipeline* (URP) introduces a modern rendering system optimized for multiplatform applications, including XR applications. The *High Definition Render Pipeline* (HDRP) is instead designed for applications requiring high graphical quality and high-performance hardware.

B.3 Scripting in Unity

The behavior of objects present within a Unity scene is defined through scripts developed in the C# language. Unity scripts are components that can be associated with GameObjects in the same way as the other components described in the previous section, allowing their functionalities to be extended and their behavior to be controlled during application execution.

In Unity, C# scripts generally derive from the *MonoBehaviour* class, which represents the base class provided by the engine for managing components associated with GameObjects. Inheritance from MonoBehaviour allows the script to interact with Unity's internal system and to use functions automatically called by the engine during the application lifecycle.

The lifecycle of a Unity script is organized according to a sequence of predefined functions that are automatically called at specific moments during execution:

- *Awake()*: is executed when the component is initialized and is generally used to perform preliminary operations, such as assigning references to other components or initializing internal variables.

- *Start()*: is called before the first frame update and is commonly used to initialize parameters that require all scene objects to already be loaded. Unlike *Awake()*, it is executed only if the component is enabled.

- *Update()*: is executed once for each rendered frame and is mainly used to manage logic dependent on rendering time, such as user input control or the update of graphical elements.

- *FixedUpdate()*: is called at fixed time intervals and synchronized with the physics engine cycle. The time interval between two consecutive executions of the function is defined by the *Fixed Timestep* parameter, which can be modified through the project settings (*Project Settings* > *Time*) or directly through a script using the variable *Time.fixedDeltaTime*, for example:

  ```
  Time.fixedDeltaTime = 0.01f;
  ```

  This function is suitable for all operations that must maintain behavior independent of the frame rate, such as applying forces, modifying the properties of a *Rigidbody*, or managing elements involved in the physics simulation.

The variables defined within scripts can be used to store parameters required for the operation of the application. Variables declared as *public* are automatically displayed in

the Inspector and can be modified directly from the editor, while *private* variables are accessible only within the script. Unity also provides the *[SerializeField]* attribute, which allows a private variable to be displayed in the Inspector while maintaining access control through code. This mechanism is frequently used to configure application parameters without directly modifying the script.

One of the fundamental aspects of scripting in Unity concerns access to the components associated with *GameObjects*. Through the function *GetComponent<T>()*, it is possible, during execution, to obtain a reference to a specific component present on the object and modify its properties directly through code. The parameter *T* indicates the type of component that must be retrieved and is replaced with the name of the class associated with the Unity component of interest. For example, to obtain a reference to the *Transform* component associated with the current *GameObject*, it is possible to use:

```
Transform objTransform = GetComponent<Transform>();
```

Subsequently, the obtained reference can be used to modify the position, rotation, and scale of the object in three-dimensional space. The position can be modified through the *position* property, which defines the coordinates of the GameObject with respect to the global reference system, for example:

```
objTransform.position = new Vector3(0,1,0);
```

The rotation can instead be set through the *rotation* property, generally expressed using a quaternion, or by using the function *Euler()* to directly specify Euler angles with respect to the Cartesian axes:

```
objTransform.rotation = Quaternion.Euler(0,90,0);
```

Finally, the size of the object can be modified through the *localScale* property, which defines the scale factor applied with respect to the local reference system of the GameObject:

```
objTransform.localScale = new Vector3(2,2,2);
```

These properties therefore allow the direct modification of the pose and size of objects during application execution, while maintaining consistency with the hierarchical structure based on parent-child relationships defined within the scene.

Similarly, it is possible to access components involved in physics simulation. In the case of a *Collider*, the reference is obtained by specifying the type of collider associated with the GameObject, for example:

```
BoxCollider objCollider = GetComponent<BoxCollider>();
```

Through this reference, it is possible to modify the properties of the component, such as enabling interaction detection through triggers:

```
objCollider.isTrigger = true;
```

In the case of a *Rigidbody*, the component can be retrieved through:

```
Rigidbody objRigidbody = GetComponent<Rigidbody>();
```

The reference allows the modification of the dynamic properties of the rigid body or the direct application of physical actions on the object, for example by setting the mass:

```
objRigidbody.mass = 5.0f;
```

or by applying a force during execution:

```
objRigidbody.AddForce(Vector3.up * 10.0f);
```

This mechanism represents one of the main elements of programming in Unity, since it allows the separation between the structure of the object and the logic controlling its behavior. The same GameObject can therefore dynamically modify its behavior by accessing the different components that compose it, without the need to define new classes for each type of object present in the scene.

Unity programming frequently uses an approach based on *events* (*event-driven programming*), in which the execution of specific functions is linked to the occurrence of particular conditions during application execution. An event represents a notification generated by a component when a certain action occurs, while a *listener* is a function registered to receive such notification and execute a specific behavior. This mechanism allows the component generating the event to be separated from the one managing the response, making scripts more modular and reducing direct dependencies between scene objects. In Unity, events can be defined using the C# event system, through the keyword *event* associated with a delegate. For example, a component can define a simple event generated when a specific action is completed:

```
public event Action ActionCompleted;
```

When the condition associated with the event occurs, the component invokes the event, notifying all scripts registered as listeners:

```
ActionCompleted?.Invoke();
```

Another script can then associate a function with the event using the += operator. The indicated function will be executed automatically every time the event is generated. The registration of a listener can be removed using the -= operator, interrupting the reception of notifications generated by the event.

```
objectScript.ActionCompleted += OnActionCompleted;
objectScript.ActionCompleted -= OnActionCompleted;
```

where the method associated with the listener can be defined, for example, as:

```
void OnActionCompleted() { ...  }
```

This system represents the mechanism behind numerous interactions in Unity, including collision management, user input handling, and events generated by XR frameworks. Within Unity, many engine functionalities use this approach. Events can be divided into different categories depending on the system that generates them.

- *Physics events*: are generated by the physics engine when interactions occur between objects equipped with *Collider* components and, in the case of dynamic bodies, *Rigidbody* components. The functions *OnCollisionEnter()*, *OnCollisionStay()*, and *OnCollisionExit()* are called respectively at the beginning, during the maintenance, and at the end of a physical collision. When the Collider is configured

as a *Trigger*, the physical response is replaced by geometric intersection detection. In this case Unity generates the events *OnTriggerEnter()*, *OnTriggerStay()*, and *OnTriggerExit()*, used to identify respectively the entrance, persistence, and exit of an object from a specific region of space.

- *Input events*: are generated by Unity's input system in response to user actions. Through the new *Input System*, an action can be associated with an event through an appropriate callback function. Examples of this type are the events *performed*, generated when an action is completed, *started*, generated when the action is initialized, and *canceled*, generated when the action is interrupted. For example, an XR controller button can be associated with an *Input Action* and generate an event when pressed:

  ```
  inputAction.performed += OnButtonPressed;
  ```

  where the function *OnButtonPressed()* is executed every time the action is detected.

- *XR plugin events*: many frameworks dedicated to XR application development provide specific events for managing interactions between the user and the virtual environment. These events are generated by *Interactors*, which represent the elements capable of performing an interaction, and by *Interactables*, which are the objects that can be manipulated. Among the main events are those related to object pointing and selection. The event *HoverEntered()* is generated when an interaction device begins or ends pointing at an object. The opposite event *HoverExited()* is called when the interactor leaves the object. During an actual interaction, the selection events *SelectEntered()* and *SelectExited()* are used to manage respectively the grabbing or activation of an object and its release. For example, pressing the grip button of a controller can generate a *SelectEntered()* event on a grabbable object, while releasing the button determines a *SelectExited()* event. The system also provides events related to the activation of specific actions. The events *Activated()* and *Deactivated()* allow, for example, the management of a virtual button press or the activation of a device in the scene. Additional events can be associated with hand tracking systems, allowing the detection of hand tracking acquisition or loss, joint pose updates, or recognition of custom gestures. Examples of events used in this context are *HandTrackingAcquired()* and *HandTrackingLost()*, generated respectively when the system recognizes a hand and when tracking is lost. Additional events can be defined for custom gesture management, such as *GesturePerformed()* and *GestureEnded()*, which allow the identification of the beginning and end of a specific hand configuration. The XR plugins in which these events are defined will be discussed in Section B.5.

- *Custom events*: in addition to events generated directly by the engine or external frameworks, Unity allows developers to create custom events through the tools provided by C# and the *UnityEvent* class.

Another tool frequently used in three-dimensional applications is represented by the *Raycast*, which allows the identification of the intersection between a ray and objects

present in the scene. Through a Raycast, it is possible to determine which GameObject is hit, the position of the intersection point, and the normal of the affected surface. This system is widely used to implement interactions based on gaze direction, XR controller pointing, object selection, and teleportation systems.

Finally, Unity allows the execution of operations distributed over time through *Coroutines*. Unlike normal C# methods, a Coroutine can be suspended and resumed later, allowing the management of actions such as timed animations, delays, gradual transitions, or procedures that must not block the main application loop.

B.4 Input management

The input management in Unity is handled by the input system, which allows the acquisition of information coming from different hardware devices and their translation into actions usable by the application. In recent versions of the engine, the traditional system based on direct access to devices through the *Input* class has been progressively replaced by the *Input System*, a more flexible framework designed to support multiplatform applications and different devices, including XR controllers, hand tracking systems, and customized input devices.

The principle behind the new system consists of the separation between the physical device used by the user and the logical action required by the application. An input is therefore not directly associated with a specific button or device, but an abstract action representing the desired behavior is defined first. In this way, the same application function can be controlled through different devices without modifying the code that implements the interaction logic.

For example, an action for selecting a virtual object can be associated with the pressure of the *trigger* button of an XR controller, a keyboard command, or a hand gesture. The interaction script does not need to know which device generated the input, but only needs to react to the associated action. To verify that a device is correctly integrated with the application and that the raw data are correctly read, the *Input Debugger* is used (Figure B.3).

The input configuration is performed through the *Input Actions* window, accessible from the menu *Assets* > *Create* > *Input Actions*. Creating this element generates a file containing the definition of the available actions in the application, the supported devices, and their corresponding associations. Within the Input Actions editor, actions are organized through the *Action Maps*. An Action Map represents a set of inputs belonging to the same application context, such as a specific type of device. Each Action Map allows the definition of a series of *Actions*, which specify the type of variable that must be read and the source from which this variable must be acquired, for example the position (vector) or rotation (quaternion) from a tracker. For each *Action*, the *Action Type* parameter must therefore be defined, which corresponds to the type of data produced by the action. The main categories are:

- *Button*: used for discrete events characterized by a pressed/ released state (therefore represented by a boolean true/false value), such as pressing a controller button;

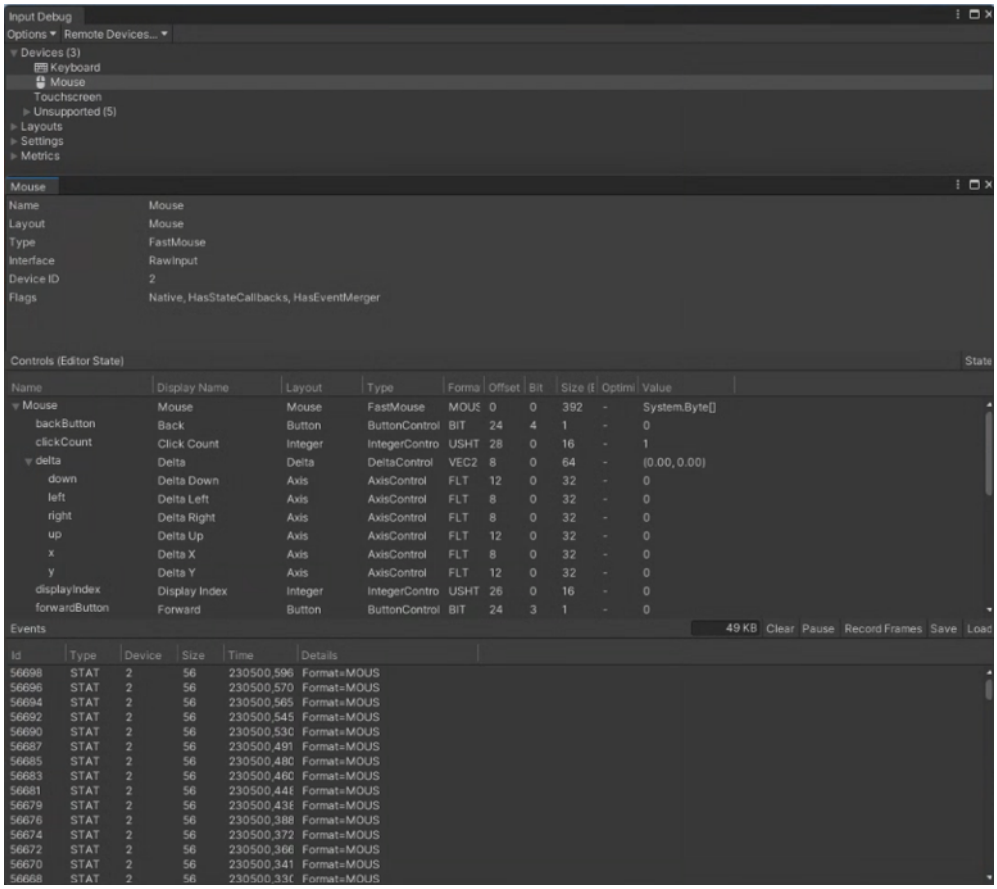


Figure B.3: Input Debugger window to check whether a device is being tracked and whether its raw data are being sent to the engine.

- *Value*: used for continuous values, such as the position and rotation of a tracker or controller;
- *Pass Through*: used when the input value must be transferred directly without applying an interaction logic based on the action state.

Subsequently, the *Control Type* parameter must be defined, which corresponds to the subclass of parameter type read from the input devices for the corresponding *Action Type*. While the Action Type describes the way Unity manages the action (button, continuous value, or pass-through), the Control Type defines the format of the acquired data. The most common *Control Type* categories are:

- *Vector3*: represents a three-dimensional position through the coordinates (x, y, z). This type is frequently used to acquire the position of controllers, trackers, or other tracking devices.
- *Quaternion*: represents the three-dimensional orientation of an object through a quaternion. It is the format normally used for the rotation of controllers, headsets, and trackers, avoiding the singularities associated with Euler angles.

- *Integer*: represents an integer value and is used in particular situations in which the device returns a discrete numerical state.
- *Any*: allows any compatible control type to be associated with the action, leaving Unity to automatically manage the type of received data.

Some examples are reported in Figures B.4 and B.5, where the *Actions* for the HTC VIVE Wrist Tracker have been defined.

Once an action has been defined, it is necessary to associate one or more devices through the *Bindings*. A Binding represents the connection between a logical action and a specific input source. For example, the action *Select*, defined as *Button*, can be connected to the trigger button of an XR controller, while the action *Position*, defined as a *Value-Vector3*, can be connected to the position of a controller, a hand, or a tracker. To use the data coming from the *Input Map* and the corresponding *Actions*, these inputs can be accessed through code.

B.5 XR project configuration and development framework

The development of applications in Unity requires a specific project configuration that allows the engine to correctly interface with immersive devices and manage tracking, input, and interaction systems. Unlike a traditional three-dimensional application, in which the Camera and inputs are directly controlled by the application, an XR system must integrate information coming from external devices characterized by sensors, tracking systems, and different interaction modalities.

B.5.1 XR project configuration

The initial configuration of an XR project in Unity concerns the definition of the target platform, the application parameters, and the services required for interaction with the immersive device. The first operation consists of selecting the *Build Target*, which determines the platform on which the application will run and influences the settings available during development. Through the *Build Settings* window, it is possible to select the build platform, manage the scenes included in the final application, and start the process of generating the executable project.

In the XR context, the choice of the *Build Target* is particularly important since different devices require different plugins and distribution pipelines. Unity allows the development of XR applications for several platforms. The main configurations supported by Unity for immersive applications are:

- *Windows Standalone*: mainly used for XR applications executed through an external computer connected to the headset, as in the case of PC-based systems. In this configuration, rendering is performed on the Windows machine and the XR device is used as a visualization and tracking interface. This mode is commonly adopted with headsets connected through SteamVR or through compatible OpenXR runtimes.

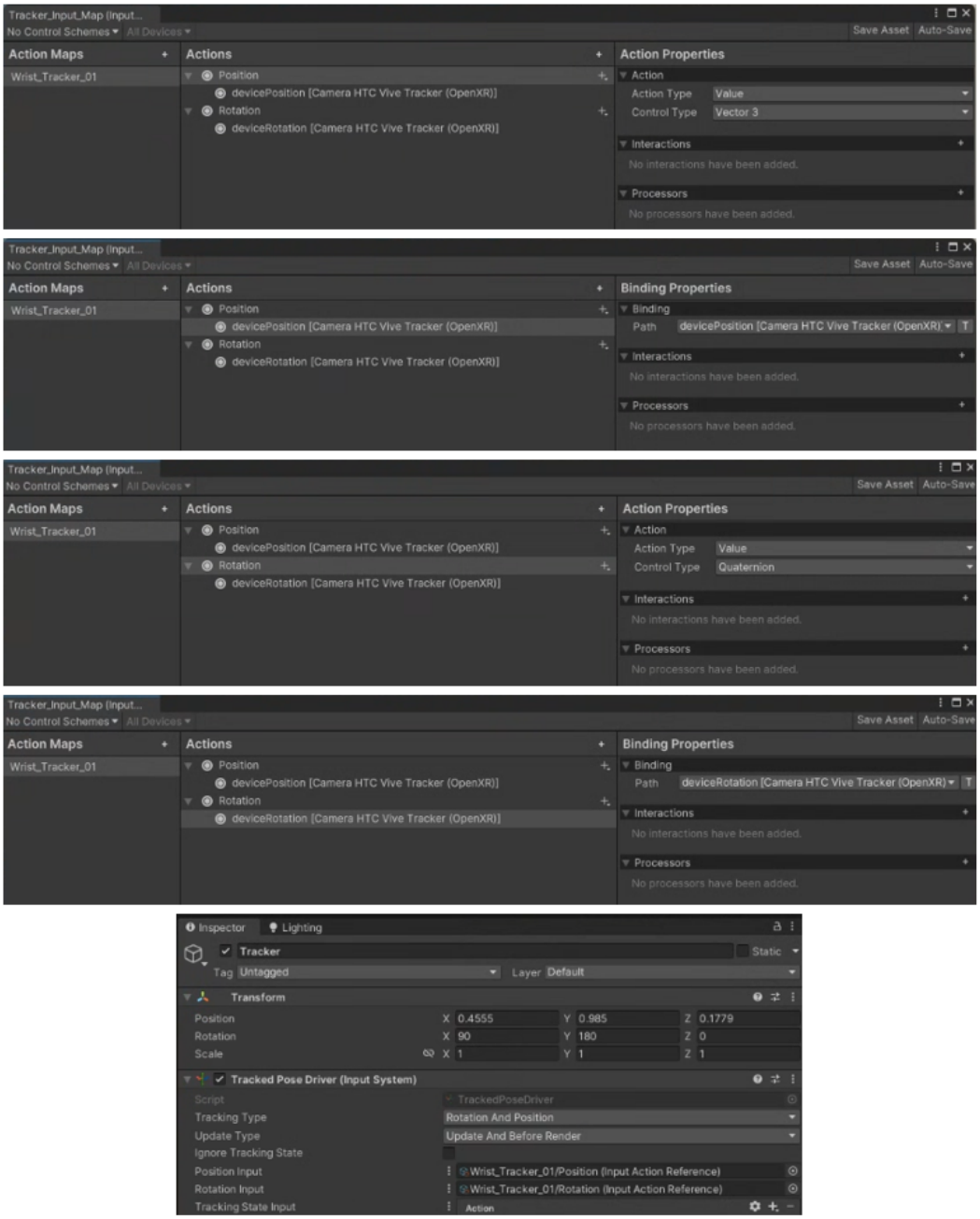


Figure B.4: Input Map of the HTC VIVE Wrist Tracker.

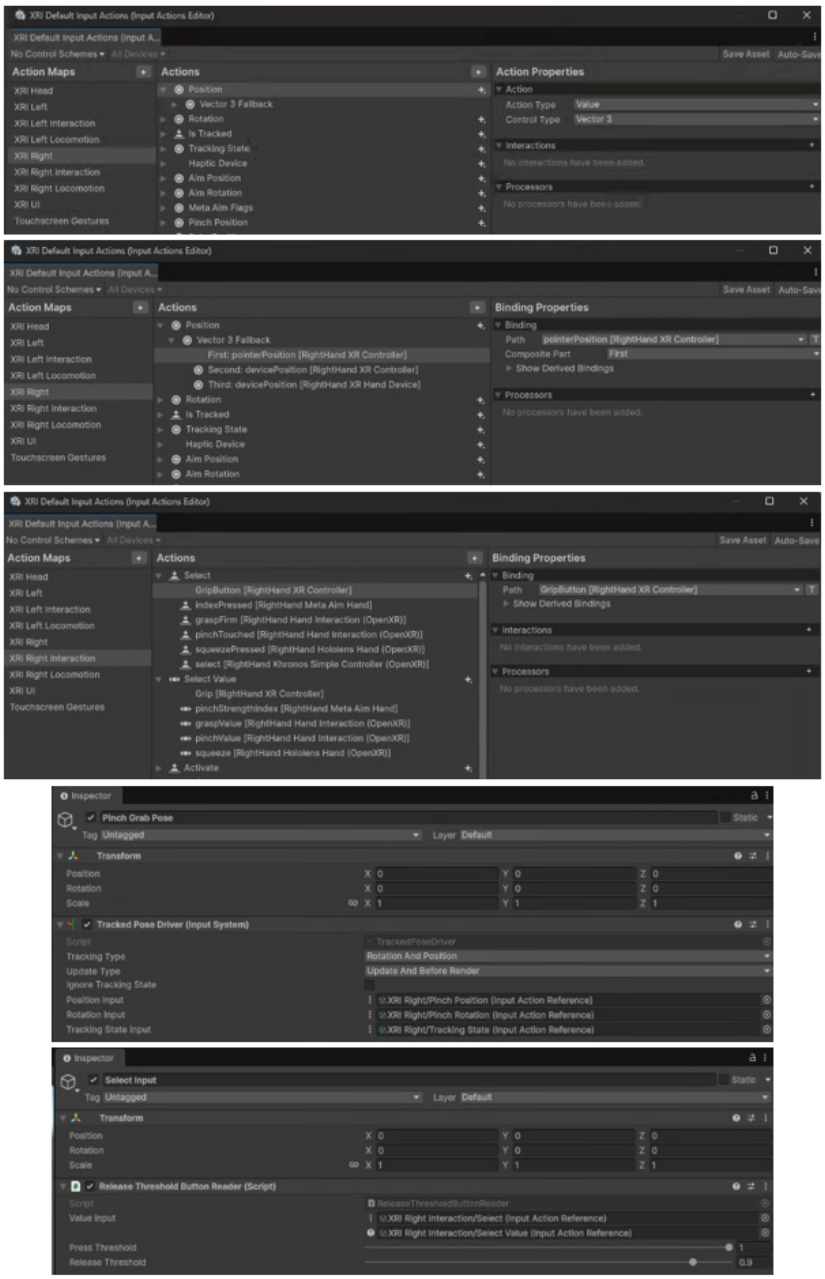


Figure B.5: Input Map for XR devices.

- *Android Build*: used for standalone XR devices based on the Android operating system. In this configuration, the project is compiled as an *APK* package that can be installed directly on the device. Compared to the desktop configuration, this mode requires greater attention to the optimization of computational resources, since rendering, tracking, and interaction management are performed directly by the hardware integrated into the headset.
- *Universal Windows Platform (UWP)*: used for augmented reality and mixed reality applications intended for the Microsoft ecosystem, such as *HoloLens* devices. This configuration enables integration with platform-specific Windows services and allows the distribution of applications developed for mixed reality environments.

Once the target platform has been selected, the project configuration is completed through the *Project Settings* and *Player Settings* windows, which allow the definition of the global application parameters and the characteristics of the final build.

The *Project Settings* window collects all the settings that influence the general behavior of the project during development and execution. These include configurations related to the physics engine, such as the simulation system used, the simulation time step, the number of iterations of the physics solver, and the parameters related to collision management. Within the same window, several engine subsystems are also configured, including the input system, audio settings, rendering and lighting parameters, graphical quality management (*Quality Settings*), and options related to memory and performance. The section dedicated to *XR Plug-in Management* allows the activation and configuration of the software components required for communication between Unity and the XR device being used. Through this configuration, the plugins and runtimes responsible for managing immersive hardware are defined.

The XR runtime represents the intermediate layer between the Unity application and the physical device, managing the data coming from the headset sensors, pose tracking, and inputs generated by the controllers. Unity therefore does not directly access the hardware, but instead uses the interfaces provided by the selected runtime. In a Windows environment, one of the most commonly used runtimes is *SteamVR*, developed by Valve for managing virtual reality devices connected to a computer. SteamVR provides Unity with information regarding the position and orientation of the headset, the controllers, and the capabilities of the connected device. Once installed on the host computer, it is launched during application execution and manages the communication between the operating system, the XR device, and the graphics engine. Through integration with OpenXR or dedicated plugins, SteamVR therefore allows the use of different compatible devices without modifying the logic of the application developed in Unity.

Among the interfaces supported by Unity, *OpenXR* currently represents one of the main standards for the development of cross-platform XR applications. Unlike a runtime specific to a particular device, OpenXR defines a common set of APIs that allows the application to communicate with different compatible runtimes. The configuration of OpenXR within Unity allows the selection of supported controller profiles and the activation of any required extensions for devices with specific functionalities.

The *Player Settings* window instead contains the settings associated with the generation of the executable application for a specific platform. Through this section, pa-

rameters such as the application name and identifier, the rendering mode, the window format, the resolution, the graphics level used, the resource compression settings, and the platform-specific options of the target environment are defined.

B.5.2 XR application deployment

Once the project configuration has been completed and the correct operation of the application has been verified within the Unity environment, it is necessary to proceed with the generation of the executable version intended for the XR device. The deployment process depends on the hardware platform used and on the characteristics of the supported runtime.

For devices connected to a computer through a desktop runtime, such as systems based on SteamVR or OpenXR Runtime for Windows, it is possible to generate a *Windows Standalone* version. This mode produces an executable application compatible with the Windows operating system and is generally used for XR applications that rely on hardware directly connected to the PC.

In the case of standalone XR devices, such as headsets based on the Android operating system, Unity instead allows the generation of an *Android APK* package. The produced file contains the application and all the resources required for its execution on the device and can be installed through dedicated development tools.

For devices belonging to the Microsoft Mixed Reality ecosystem, it is possible to use the *Universal Windows Platform (UWP)* target, which allows the creation of applications compatible with Windows devices dedicated to augmented and mixed reality.

In the case of development for Windows and UWP platforms, Unity can also automatically generate a solution compatible with Visual Studio. This solution allows additional compilation, debugging, and configuration operations to be performed before the final deployment on the device.

The final phase of the process consists of deploying the application on the target XR device and verifying the correct operation of the tracking, input, and interaction systems. During this phase, debugging tools can be used to analyze any issues related to the communication between the application, the XR runtime, and the hardware.

B.6 XR packages and platform extensions

The development of XR applications in Unity often requires the integration of additional packages beyond the standard engine components. These tools extend the available functionality by introducing dedicated systems for interaction management, hand tracking, device-specific features, and access to proprietary hardware capabilities. The following sections describe the main components introduced by each package and their role within the XR architecture.

B.6.1 XR Interaction Toolkit

The *XR Interaction Toolkit* is the main framework used in Unity for implementing interactions within immersive environments. The package provides a set of preconfigured

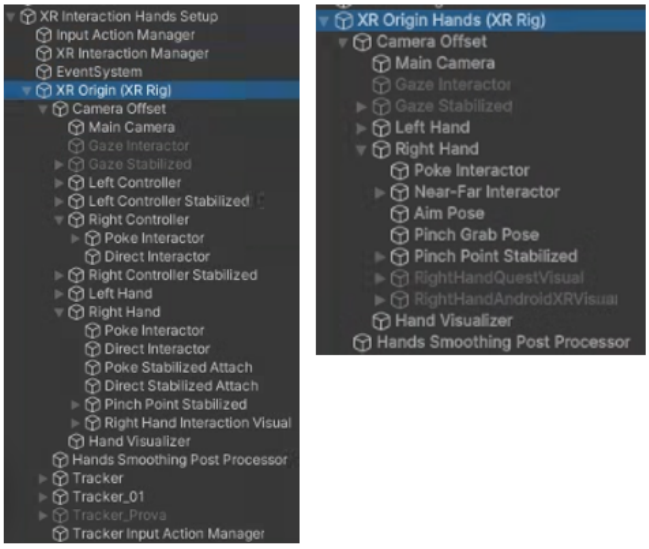


Figure B.6: 2 different configurations of the XR Origin.

components and prefabs that abstract the management of input devices and enable the development of interactions compatible with different XR platforms.

The *XR Origin* prefab provided by the *XR Interaction Toolkit* represents the basic structure used to integrate the tracking system and interaction elements into an XR scene. It replaces the manual Camera management typically required in traditional 3D applications, as the user's position is automatically updated by the XR runtime through the connected tracking devices. Therefore, transformations applied to the user must be modified by acting on the *XR Origin* or its child objects, rather than directly modifying the Camera. The default prefab configuration includes a hierarchy of GameObjects that can be adapted according to the device being used and the required interaction methods (Figure B.6). The main elements are:

- *XR Origin*: is the root object of the hierarchy and contains the associated component that defines the reference system used within the application. Through the *Requested Tracking Origin Mode* parameter, it is possible to select the spatial reference provided by the XR device, typically either *Device* or *Floor*. The former uses the initial device position as the origin, while the latter aligns the virtual origin with the real-world floor detected by the tracking system.

  The *Camera Floor Offset Object* parameter allows specifying the hierarchy object to which any vertical corrections relative to the device reference frame should be applied. Changing this parameter or the associated transform makes it possible, for example, to correct the user's height or adapt the reference system to environments with different configurations.

- *Camera Offset*: represents the intermediate layer between the user origin and the Camera. This GameObject is used to apply additional transformations without directly modifying the pose provided by the XR device. For example, changing the

z or y coordinate of its *Transform* allows modifying the initial observation point or introducing an offset relative to the physical headset position.

- *Main Camera*: is the Camera used to render the user's view. Inside this object, the *Tracked Pose Driver* component is present, which is responsible for updating the Camera position and orientation using the data provided by the XR runtime. During execution, this component automatically overwrites the Camera *Transform* with the pose detected by the headset. For this reason, manual modifications to the Camera position should be avoided and instead applied to the *XR Origin* or to the *Camera Offset*.

- *Left Controller* and *Right Controller*: are the objects associated with the physical controllers of the XR device. Each controller contains the *XR Controller (Action-based)* component, which connects the hardware device to the *Input System* through the *Input Actions* defined within the project.

  The component acquires data from the controller, such as position, orientation, button presses, and analog values, and makes them available to the interaction systems of the *XR Interaction Toolkit*. Therefore, changing the associated *Input Actions* allows modifying the behavior of physical buttons without changing the application scripts.

  Controllers generally contain several *Interactors*, which are components responsible for detecting and selecting interactive objects present in the scene.

  - *XR Ray Interactor*: uses a virtual ray generated along the controller direction to detect objects at a distance. The component performs a raycast within the scene and checks for objects compatible with the corresponding *Interaction Layer*. The parameters *Max Raycast Distance* and *Interaction Layer Mask* control, respectively, the maximum reachable distance and the categories of objects with which the interactor can interact.

    The Ray Interactor is mainly used for selecting distant objects, interacting with graphical user interfaces, or implementing teleportation systems.

  - *XR Direct Interactor*: in previous versions of the *XR Interaction Toolkit*, this component was used for short-range interactions. The interactor defines a volume around the controller and detects objects entering this region through their associated Colliders.

    When a compatible object is detected, the Direct Interactor generates the standard interaction events of the *XR Interaction Toolkit*, such as *Hover Entered*, *Select Entered*, and *Select Exited*. This component was mainly used to simulate direct grabbing of virtual objects by bringing the controller close to the object itself.

  - *XR Near-Far Interactor*: in more recent versions of the *XR Interaction Toolkit*, the Direct Interactor has been replaced by the Near-Far Interactor, which combines both near and far interaction capabilities into a single component.

    The Near-Far Interactor combines the detection of objects close to the controller with a ray-based selection system. In this way, the same controller

can be used both to directly grab nearby objects and to select more distant elements without changing the interaction mode.

Its behavior can be modified through parameters related to detection volumes, operating distances, and *Interaction Layers*, allowing the system to be adapted without modifying the application scripts.

- *XR Interactor Line Visual*: is used together with the Ray Interactor to provide a graphical representation of the ray generated by the controller. Through this component, it is possible to modify the visual appearance of the line, its length, the type of curve used, and the visualization behavior during the different interaction phases.

- *Left Hand* and *Right Hand*: these GameObjects represent the hierarchy nodes dedicated to the management of the virtual hands. They are included in the *XR Origin* prefab and are used only when the project is configured to use the hand tracking system provided by the *XR Hands* package.

  The structure of the two objects is identical: each hand contains the elements required for virtual hand visualization, pose management, and interaction with objects in the scene. Child GameObjects dedicated to the graphical representation of the hand, such as *Hand Visual*, receive the pose updates from the tracking system and allow a 3D model to be associated with either the left or right hand.

  Within the hand hierarchy, the same interaction systems used for controllers are also present, such as the *Interactors*. Their function is analogous to the one described for the *Left Controller* and *Right Controller*: they detect interactive objects present in the scene and generate the interaction events defined by the *XR Interaction Toolkit*. The main difference concerns the source of the pose information, which in this case is provided by the tracked hand rather than by the physical controller device.

  The tracking of hand joints, the generation of the hand skeleton, and the configuration of custom gestures are described in the section dedicated to the *XR Hands* package.

- *Hand Visualizer*: this component is responsible for the graphical representation of tracked hands within the virtual environment. It uses the data generated by the *XR Hands* system to update finger positions and apply the visual model associated with the left or right hand.

  The visualization can be modified by replacing the hand mesh prefab, adjusting the material used, or disabling the graphical rendering when only the kinematic information of the joints is required to implement custom interactions.

- *Locomotion System*: this component manages all user movement systems within the virtual environment. It does not directly modify the Camera transform, but instead applies the requested transformations to the *XR Origin*, maintaining consistency between the real-world tracking system and the virtual environment.

  The system relies on several *Locomotion Providers*, which are components responsible for generating movement requests. The most commonly used providers include

the *Continuous Move Provider*, which enables continuous movement through analog controller inputs, and the *Snap Turn Provider*, which provides discrete user rotation to reduce comfort issues during headset usage.

The provider parameters allow developers to modify features such as movement speed, movement direction, rotation angle, and the input type associated through the *Input Actions* system.

- *Teleportation System*: the teleportation system of the *XR Interaction Toolkit* uses a combination of interactors and providers to move the user to a new position within the virtual environment.

  The *Teleportation Provider* receives the request generated by the *Teleport Interactor* associated with the controller and applies the movement to the *XR Origin*. In this way, the position change affects the entire user tracking system while maintaining the correct alignment between the Camera, controllers, and tracking devices.

  The reachable surfaces or destinations are defined through the *Teleportation Area* and *Teleportation Anchor* components. The former allows the user to select any position belonging to a valid surface, while the latter defines a specific destination with predefined position and orientation.

  The teleportation behavior can be customized through parameters such as *Interaction Layers*, maximum reachable distance, and the type of interaction associated with the controller.

B.6.2 XR Hands

The *XR Hands* package extends Unity's XR framework by introducing support for hand tracking, allowing hand joint positions to be used as an input source. Unlike traditional controllers, where user interaction is typically associated with physical button presses, hand tracking infers user intent by analyzing the geometric configuration of the hand.

To use this system, the *XR Hands* package must be installed through Unity's *Package Manager*, and the corresponding provider must be enabled within the project's XR configuration. Once installed, the package provides the *XR Hand Subsystem*, which is the software layer responsible for receiving left and right hand tracking data from the XR runtime. The subsystem does not automatically generate gestures that can be directly used by the application; instead, it provides only the skeletal hand data. Each hand is represented as a collection of *joints*, corresponding to the major anatomical articulations of the hand. For each joint, the subsystem provides its position, orientation, and tracking validity status. These data serve as the input for the gesture recognition algorithms implemented by the application.

To visualize the virtual hands within the scene, separate GameObjects must be created for the left and right hands, or the corresponding objects already included in the *XR Origin* prefab can be used. Each hand object is assigned the *XR Hand Skeleton Driver* component, which connects the data provided by the *XR Hand Subsystem* to the hierarchy of the virtual hand skeleton. The component automatically updates the

Transform of each hand joint according to the tracking data received from the XR device. The graphical representation of the hand is managed through the *XR Hand Mesh Controller* component. This component associates a 3D hand model with the tracked skeleton and continuously updates the mesh according to the current hand pose. The default hand prefab can be replaced with a custom model, the hand material can be modified, a custom mesh can be used, or the application can rely solely on joint tracking without rendering any visible hand model.

B.6.2.1 Custom gestures

The *XR Hands* package allows custom gestures to be defined without directly analyzing the individual hand joints. Gesture recognition is performed by comparing the hand tracking data with a predefined configuration described using the following elements:

- *Hand Shape*: describes the overall configuration of the hand by considering parameters such as the degree of finger extension or flexion, finger spread, and their relative arrangement with respect to the palm. This representation makes it possible to identify general hand configurations, such as an open hand, a closed fist, or a pose characterized by the extension of a single finger.
- *Hand Pose*: defines a specific hand configuration through the spatial arrangement of the hand joints. Unlike the *Hand Shape*, which represents a general characteristic of the hand, the *Hand Pose* describes a precise finger configuration and compares it with the pose detected by the hand tracking system during application execution.

Further details on how custom gestures are defined, together with the corresponding parameters and their configuration, are provided in Section 4.3.2.

B.6.3 VIVE OpenXR plugin

The *VIVE OpenXR Plugin* extends the functionality of the *OpenXR* standard integrated into Unity by adding support for HTC VIVE-specific extensions. The plugin enables the use of advanced features available on headsets from the VIVE product family, including the *VIVE Pro*, *VIVE Pro 2*, and *VIVE XR Elite*, while maintaining a common interface based on the OpenXR standard.

Installing the plugin does not automatically enable all of its available features. The project must first be properly configured through Unity's XR settings. Within *Project Settings* > *XR Plug-in Management*, the *OpenXR* provider must be enabled for the target platform. Afterwards, the appropriate controller *Interaction Profiles* must be activated in the OpenXR settings, together with the extensions provided by the *VIVE OpenXR Plugin*. These extensions allow the OpenXR runtime to expose additional functionality to Unity that is not included in the standard OpenXR specification, such as support for external trackers and mixed reality features.

One of the key features provided by the *VIVE OpenXR Plugin* is support for *passthrough* on compatible headsets such as the *VIVE XR Elite*. This functionality allows the video stream captured by the headset cameras to be used as the background layer of the virtual scene, enabling mixed reality applications in which virtual objects are superimposed

onto the real environment. To use the passthrough functionality, the dedicated HTC OpenXR extensions must first be enabled within the project settings. Specifically, under *Project Settings* > *XR Plug-in Management* > *OpenXR Feature Groups*, both the *VIVE XR Composition Layer* and the *VIVE XR Composition Layer (Passthrough)* extensions must be activated. Composition layers are a mechanism defined by the OpenXR specification that allows content to be rendered directly by the runtime compositor, separately from Unity's standard rendering pipeline.

After enabling the OpenXR extensions required by the *VIVE XR Elite*, passthrough can be integrated into the scene using the example scripts included with the *VIVE OpenXR Plugin*. In particular, Script B.1 implements the creation and management of the passthrough layer by interfacing directly with the device's OpenXR runtime. To use this functionality, the script only needs to be attached to a GameObject in the scene, while the main camera must be configured with an RGBA background color of $(0, 0, 0, 0)$. This configuration makes Unity's background fully transparent, allowing the passthrough layer provided by the runtime to display the camera feed as the background of the virtual scene.

The script automatically creates the corresponding *composition layer* and communicates the required information to the OpenXR runtime, eliminating the need to manually render the camera stream as a texture inside Unity. As a result, virtual objects are rendered on top of the real-world image captured by the headset, producing a mixed reality configuration. This functionality is currently available only for Android builds.

The *VIVE OpenXR Plugin* also provides support for additional HTC tracking devices, such as the *VIVE Wrist Tracker* and the *VIVE Ultimate Tracker*. These devices enable the tracking of physical objects or additional body parts beyond the standard headset controllers. This feature is primarily available for *Android* applications running on standalone devices such as the *VIVE XR Elite*; it is not supported in Windows desktop builds based on the same plugin. To use a tracker within a Unity scene, the device must first be paired with the headset using HTC's tracker management system. The tracker can then be represented in the scene as a standard XR GameObject by attaching the *Tracked Pose Driver (Input System)* component. This component connects the physical device to Unity's input system through *Input Actions*, in the same way that standard controllers are handled. During execution, the component automatically updates the associated GameObject's *Transform* using the position and orientation received from the physical tracker.

The *Input Actions* configuration allows developers to specify which tracker controls the GameObject and which data should be acquired, such as position, rotation, or any additional inputs supported by the device. In this way, any virtual object in the scene can be bound to a physical tracker, enabling applications such as real-world object tracking, simulated tool tracking, or full-body tracking.

Listing B.1: C# script used to enable passthrough.

```
1
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
5
using VIVE.OpenXR.CompositionLayer;
using VIVE.OpenXR.CompositionLayer.Passthrough;
```

```
8
9 using VIVE.OpenXR.Samples;
10
11 namespace VIVE.OpenXR.CompositionLayer.Samples.Passthrough
12 {
13     public class PassthroughSample_Planar : MonoBehaviour
14     {
15         private int activePassthroughID = 0;
16         private LayerType currentActiveLayerType = LayerType.Underlay;
17
18         private void Update()
19         {
20             if (activePassthroughID == 0)
21             {
22                 StartPassthrough();
23             }
24
25             if (VRSInputManager.instance.GetButtonDown(VRSButtonReference.B)
                   ) //Set Passthrough as Overlay
26             {
27                 SetPassthroughToOverlay();
28             }
29             if (VRSInputManager.instance.GetButtonDown(VRSButtonReference.A)
                   ) //Set Passthrough as Underlay
30             {
31                 SetPassthroughToUnderlay();
32             }
33         }
34
35         public void SetPassthroughToOverlay()
36         {
37             if (activePassthroughID != 0)
38             {
39                 CompositionLayerPassthroughAPI.SetPassthroughLayerType(
                       activePassthroughID, LayerType.Overlay);
40                 currentActiveLayerType = LayerType.Overlay;
41             }
42         }
43
44         public void SetPassthroughToUnderlay()
45         {
46             if (activePassthroughID != 0)
47             {
48                 CompositionLayerPassthroughAPI.SetPassthroughLayerType(
                       activePassthroughID, LayerType.Underlay);
49                 currentActiveLayerType = LayerType.Underlay;
50             }
51         }
52
53         void StartPassthrough()
54         {
55             activePassthroughID = CompositionLayerPassthroughAPI.
                   CreatePlanarPassthrough(currentActiveLayerType,
                   OnDestroyPassthroughFeatureSession);
56         }
57
58         void OnDestroyPassthroughFeatureSession(int passthroughID)
59         {
60             CompositionLayerPassthroughAPI.DestroyPassthrough(passthroughID)
                   ;
61             activePassthroughID = 0;
62         }
63     }
64 }
```

B.6.4 Varjo XR plugin

The *Varjo XR Plugin* enables the integration of Varjo devices into Unity through the *OpenXR* standard, extending the available functionality with features specific to the company's professional headsets. The plugin was used to support devices from the *Varjo XR* product family, including the *Varjo XR-4*, providing access to tracking systems, advanced stereoscopic rendering, *video passthrough*, and *eye tracking* capabilities.

The plugin is integrated through Unity's *XR Plug-in Management* system. After installing the *Varjo XR Plugin* via the *Package Manager*, the *OpenXR* provider must be enabled for the target platform, and the Varjo-specific support must be activated within the *OpenXR Feature Groups*. Once the provider has been configured, the plugin allows developers to use Unity's standard XR components without requiring hardware-specific management. Headset and controller tracking are handled by the OpenXR runtime, while the Varjo extensions provide additional functionality through the components and scripts included in the plugin.

One of the main advantages of the plugin is the straightforward configuration of mixed reality features. The *Varjo XR Plugin* includes pre-built components and scripts that provide easy access to the headset's *video passthrough* capabilities. The main functions provided by the plugin are described in Section 5.1.1.

B.6.5 Mixed Reality Toolkit (MRTK)

The *Mixed Reality Toolkit* (MRTK) was used to develop applications for *HoloLens 2*, providing an abstraction layer over the native Windows Mixed Reality systems and simplifying the management of the device's main functionalities, such as hand tracking, spatial mapping, and gaze-based interactions. Integrating MRTK into Unity requires an initial project configuration that differs from that of standard XR packages. The first step consists of installing the required packages through the *Mixed Reality Feature Tool*, which adds the MRTK framework, OpenXR dependencies, and HoloLens 2-specific components to the project. Once the installation is complete, the required prefabs, scripts, and configuration profiles are imported into the project.

To use MRTK with HoloLens 2, the project must be configured for the *Universal Windows Platform (UWP)*. Within *Project Settings > XR Plug-in Management*, the *OpenXR* provider must be enabled for the UWP platform. Subsequently, the HoloLens 2 features must be enabled in the *OpenXR Feature Groups*, including support for hand tracking and gaze-based interactions. This configuration allows the Windows Mixed Reality runtime to provide Unity with sensor data from the device, including head pose, hand joint positions, and spatial information about the surrounding environment. After configuring the XR provider, MRTK is initialized through the *MixedRealityToolkit* prefab. This GameObject is added to the scene and acts as the central component of the framework, as it contains references to the services used during application execution. These services include the input system, the hand tracking system, the spatial awareness system, and the components responsible for converting device-generated events into events that can be processed by the application scripts.

The behavior of the toolkit is defined through the *MixedRealityToolkit Profile*. The main profile contains a set of subprofiles that allow developers to configure the features

enabled within the project. For HoloLens 2, the most relevant profiles are those related to input and spatial awareness. These profiles make it possible to enable hand tracking, specify which input devices should be recognized, and configure the behavior of the environment sensing system.

Once MRTK has been initialized, interactions with scene objects can be implemented using the components provided by the framework. To make an object manipulable using HoloLens 2 hand gestures, the *Object Manipulator* component is typically used. This script automatically manages the connection between the user's hand and the selected object, updating its position, rotation, and scale during manipulation. The component relies on events generated by the MRTK input system without requiring direct processing of the hand joint data. To allow an object to be detected during near interactions, the *NearInteractionGrabbable* component must also be added. This component identifies the object as grabbable by MRTK's near interaction system and provides the framework with the information required to generate grab events. The combination of *NearInteractionGrabbable* and *Object Manipulator* enables interactions that closely resemble the manipulation of a real object using the pinch gesture supported by HoloLens 2.

To test MRTK applications without repeatedly deploying them to HoloLens 2, the *Holographic Remoting* mode can be used. In this configuration, rendering is performed on the development computer, while HoloLens 2 streams tracking and input data to the PC. To enable this mode, the *Microsoft Mixed Reality OpenXR Plugin* must be installed, remoting support must be enabled in the project settings, and the *Holographic Remoting Player* application must be launched on the device. The connection between Unity and HoloLens 2 is established by specifying the device IP address in the remoting settings. Once the connection is active, the Unity application can be executed directly from the editor while using the headset as a remote XR device. This approach is particularly useful during development because it allows rapid testing of changes to scripts and scene configurations without recompiling and deploying a new UWP application. When running directly on the device, the project must instead be built as a UWP application and deployed to HoloLens 2. In this configuration, MRTK and all XR services execute locally on the headset, while the Windows Mixed Reality OpenXR runtime directly manages head tracking, hand tracking, spatial mapping, and user input.

B.6.6 Haptic Direct plugin

The *Haptic Direct Plugin* was used to integrate the *Geomagic Touch X* haptic device. Built on top of the OpenHaptics SDK, the plugin enables communication between Unity and the device control system. Unlike conventional input devices commonly used in XR systems, a haptic device supports bidirectional communication: it provides the kinematic configuration of the stylus, including its position and orientation in three-dimensional space, while simultaneously receiving force commands generated by the virtual environment.

Integration into the Unity project requires the installation of the *Geomagic Touch X* drivers on the host system, followed by the import of the software package provided with the plugin. This package includes the scripts and components required to manage the device and retrieve the data returned by the hardware. Within the scene, the *Haptic Direct Manager* component is responsible for initializing communication with the device and

establishing the connection with the haptic driver. During application startup, the manager detects the connected device and activates the haptic data update loop. The virtual representation of the device is provided through the *Geomagic Touch X Device* prefab, which associates the movement of the physical stylus with a corresponding object in the Unity scene. The device pose is continuously updated using the information supplied by the driver, expressed in the manipulator's local coordinate system and subsequently transformed into the virtual scene coordinate system.

From a software perspective, the plugin acts as an intermediate layer between Unity and the low-level OpenHaptics SDK libraries. Device management is therefore delegated to the driver and the dedicated APIs, while Unity handles the virtual scene rendering and interaction logic. This separation enables the use of the haptic device without requiring the direct implementation of low-level communication procedures. Communication between the device and the application follows a bidirectional workflow. The manipulator motion data are acquired by the driver, processed by the plugin software layer, and made available to the objects in the Unity scene. Conversely, the interaction forces computed by the application are transmitted back to the device actuators through the same communication infrastructure.

An important aspect of haptic device integration concerns the management of the *haptic loop*. Unlike Unity's rendering loop, which typically operates at lower frequencies, haptic control requires update rates on the order of 1 kHz to ensure stable force feedback and minimize delays in contact perception. For this reason, the plugin manages the device through a dedicated control loop that operates independently of the graphical rendering loop. Since the coordinate system of the physical device differs from that of the virtual environment, a coordinate transformation must also be applied. The plugin performs this conversion by aligning the Cartesian axes and applying an appropriate scaling factor, allowing the workspace of the *Geomagic Touch X* to be mapped onto the dimensions of the virtual scene.

The plugin does not directly define the physical interaction model; instead, it provides the functionality required to acquire the device state and return the forces computed by the application. In the developed system, the haptic response is determined according to the interaction between the virtual device and the objects in the scene, converted into the device reference frame, and finally applied to the user.

Appendix C

Vuforia Engine configuration for Unity Engine

The configuration of Vuforia within Unity is carried out through several sequential steps. First, the corresponding *.unitypackage* file must be installed. Subsequently, a database containing the targets to be used in the application is generated and imported into Unity. It is important to note that the database used for image-based targets differs from the one used for model-based targets, as the target creation workflows are fundamentally different (see Sections C.1 and C.2).

Once the database has been imported into Unity, Vuforia targets are instantiated as dedicated GameObjects within the scene hierarchy. Each Image Target is associated with a specific reference image and a defined physical size, which determines the scale of the virtual coordinate system relative to the real-world environment.

Prior to runtime, Vuforia integration requires the registration of a Vuforia License Key. This key is obtained from the Vuforia Developer Portal and linked to the Unity project through the Vuforia Configuration panel, accessible via *Edit* > *Project Settings* > *Vuforia Engine* > *App License Key*. The license key serves to authenticate the application and enables runtime access to Vuforia's tracking services, ensuring correct recognition of the configured database and image targets on supported devices.

At runtime, each Target GameObject acts as a dynamic reference frame whose transformation is continuously updated based on tracking results. Any virtual content attached to this object automatically follows the motion of the corresponding physical target, thereby ensuring consistent spatial alignment between the digital and physical domains. This hierarchical structure simplifies scene management and allows developers to focus primarily on interaction design rather than low-level tracking implementation details (Listing C.1).

From a software architecture perspective, Vuforia functions as a middleware layer integrated into the Unity rendering pipeline. It manages camera access, performs real-time image analysis, and updates the spatial transformations of tracked objects. During execution, the system continuously processes incoming camera frames, extracts visual features, matches them against active databases, and estimates the pose of detected targets. This information is then propagated to Unity's transformation system, enabling coherent rendering of virtual objects in the scene. The abstraction provided by the plugin

significantly reduces development complexity while maintaining adequate performance for real-time augmented reality applications.

Listing C.1: C# script used by Vuforia Engine for managing tracking and estimating the pose of the tracked image or model.

```
using UnityEngine;
using UnityEngine.Events;
using Vuforia;

/// <summary>
/// Default Vuforia event handler used to manage tracking events
/// associated with AR targets.
/// The script enables or disables virtual augmentations depending
/// on the tracking state of the observed target.
/// </summary>
public class DefaultObserverEventHandler : MonoBehaviour
{
    /// <summary>
    /// Defines which tracking states are considered valid
    /// for rendering the augmentation.
    /// </summary>
    public enum TrackingStatusFilter
    {
        Tracked,
        Tracked_ExtendedTracked,
        Tracked_ExtendedTracked_Limited
    }

    // Current tracking filter configuration
    public TrackingStatusFilter StatusFilter =
    TrackingStatusFilter.Tracked_ExtendedTracked_Limited;

    /// <summary>
    /// Enables pose smoothing to reduce jitter
    /// and abrupt pose variations.
    /// </summary>
    public bool UsePoseSmoothing = false;

    // Interpolation curve used for pose smoothing
    public AnimationCurve AnimationCurve =
    AnimationCurve.Linear(0, 0, LERP_DURATION, 1);

    /// <summary>
    /// Unity events triggered when the target
    /// is detected or lost.
    /// </summary>
    public UnityEvent OnTargetFound = new();
    public UnityEvent OnTargetLost = new();

    // Reference to the Vuforia Observer component
    protected ObserverBehaviour mObserverBehaviour;

    // Stores the previous tracking state
    protected TargetStatus mPreviousTargetStatus =
    TargetStatus.NotObserved;

    protected bool mCallbackReceivedOnce;

    // Duration of the interpolation process
    const float LERP_DURATION = 0.3f;

    PoseSmoother mPoseSmoother;

    /// <summary>
```

```
60      /// Initialization method automatically called by Unity.
61      /// Registers the tracking callbacks and initializes
62      /// pose smoothing.
63      /// </summary>
64      protected virtual void Start()
65      {
66          mObserverBehaviour = GetComponent<ObserverBehaviour>();
67
68          if (mObserverBehaviour)
69          {
70              // Register callback functions
71              mObserverBehaviour.OnTargetStatusChanged +=
72              OnObserverStatusChanged;
73
74              mObserverBehaviour.OnBehaviourDestroyed +=
75              OnObserverDestroyed;
76
77              // Force initial status update
78              OnObserverStatusChanged(
79              mObserverBehaviour,
80              mObserverBehaviour.TargetStatus);
81
82              // Initialize pose smoothing system
83              SetupPoseSmoothing();
84          }
85      }
86
87      /// <summary>
88      /// Called whenever the target tracking status changes.
89      /// </summary>
90      void OnObserverStatusChanged(
91      ObserverBehaviour behaviour,
92      TargetStatus targetStatus)
93      {
94          var name = mObserverBehaviour.TargetName;
95
96          // Print tracking status information in the Unity console
97          Debug.Log(
98          $"Target status: {name} " +
99          $"{targetStatus.Status} -- " +
100         $"{targetStatus.StatusInfo}");
101
102         // Handle tracking state transition
103         HandleTargetStatusChanged(
104         mPreviousTargetStatus.Status,
105         targetStatus.Status);
106
107         // Handle additional tracking information
108         HandleTargetStatusInfoChanged(
109         targetStatus.StatusInfo);
110
111         // Update previous tracking state
112         mPreviousTargetStatus = targetStatus;
113     }
114
115     /// <summary>
116     /// Determines whether the virtual augmentation
117     /// should be rendered according to the current
118     /// tracking state.
119     /// </summary>
120     protected bool ShouldBeRendered(Status status)
121     {
122         // Standard tracking mode
123         if (status == Status.TRACKED)
124         {
```

```
125                 return true;
126             }
127
128             // Extended tracking mode
129             if (StatusFilter ==
130             TrackingStatusFilter.Tracked_ExtendedTracked &&
131             status == Status.EXTENDED_TRACKED)
132             {
133                 return true;
134             }
135
136             // Limited tracking mode
137             if (StatusFilter ==
138             TrackingStatusFilter
139             .Tracked_ExtendedTracked_Limited &&
140             (status == Status.EXTENDED_TRACKED ||
141             status == Status.LIMITED))
142             {
143                 return true;
144             }
145
146             return false;
147         }
148
149         /// <summary>
150         /// Called when the target becomes visible.
151         /// Enables the associated virtual content.
152         /// </summary>
153         protected virtual void OnTrackingFound()
154         {
155             if (mObserverBehaviour)
156             SetComponentsEnabled(true);
157
158             // Trigger Unity event
159             OnTargetFound?.Invoke();
160         }
161
162         /// <summary>
163         /// Called when the target is lost.
164         /// Disables the associated virtual content.
165         /// </summary>
166         protected virtual void OnTrackingLost()
167         {
168             if (mObserverBehaviour)
169             SetComponentsEnabled(false);
170
171             // Trigger Unity event
172             OnTargetLost?.Invoke();
173         }
174
175         /// <summary>
176         /// Initializes the pose smoothing system.
177         /// Smoothing is only available when using
178         /// DEVICE world center mode.
179         /// </summary>
180         protected void SetupPoseSmoothing()
181         {
182             UsePoseSmoothing &=
183             VuforiaBehaviour.Instance.WorldCenterMode ==
184             WorldCenterMode.DEVICE;
185
186             // Create pose smoothing object
187             mPoseSmoother =
188             new PoseSmoother(
189             mObserverBehaviour,
```

```
190             AnimationCurve);
191
192             // Register state update callback
193             VuforiaBehaviour.Instance.World.OnStateUpdated +=
194             OnStateUpdated;
195         }
196
197         /// <summary>
198         /// Updates pose smoothing at each frame.
199         /// </summary>
200         void OnStateUpdated()
201         {
202             if (enabled && UsePoseSmoothing)
203             mPoseSmoother.Update();
204         }
205
206         /// <summary>
207         /// Enables or disables rendering-related components
208         /// associated with the augmentation.
209         /// </summary>
210         void SetComponentsEnabled(bool enable)
211         {
212             var components =
213             VuforiaRuntimeUtilities
214             .GetComponentsInChildrenExcluding
215             <Component, DefaultObserverEventHandler>(gameObject);
216
217             foreach (var component in components)
218             {
219                 switch (component)
220                 {
221                     // Enable or disable mesh rendering
222                     case Renderer rendererComponent:
223                     rendererComponent.enabled = enable;
224                     break;
225
226                     // Enable or disable colliders
227                     case Collider colliderComponent:
228                     colliderComponent.enabled = enable;
229                     break;
230
231                     // Enable or disable UI canvases
232                     case Canvas canvasComponent:
233                     canvasComponent.enabled = enable;
234                     break;
235
236                     // Enable or disable runtime meshes
237                     case RuntimeMeshRenderingBehaviour runtimeMeshComponent:
238                     runtimeMeshComponent.enabled = enable;
239                     break;
240                 }
241             }
242         }
243
244         /// <summary>
245         /// Internal class used to smooth abrupt pose transitions.
246         /// </summary>
247         class PoseSmoother
248         {
249             const float e = 0.001f;
250             const float MIN_ANGLE = 2f;
251             PoseLerp mActivePoseLerp;
252             Pose mPreviousPose;
253             readonly ObserverBehaviour mTarget;
254             readonly AnimationCurve mAnimationCurve;
```

```
255         TargetStatus mPreviousStatus;
256
257         public PoseSmoother(ObserverBehaviour target, AnimationCurve
                animationCurve)
258         {
259             mTarget = target;
260             mAnimationCurve = animationCurve;
261         }
262
263         /// <summary>
264         /// Updates the current pose and applies smoothing.
265         /// </summary>
266         public void Update()
267         {
268             var currentPose = new Pose(mTarget.transform.position,
                    mTarget.transform.rotation);
269
270             var currentStatus = mTarget.TargetStatus;
271
272             UpdatePoseSmoothing(currentPose, currentStatus);
273
274             mPreviousPose = currentPose;
275             mPreviousStatus = currentStatus;
276         }
277
278         /// <summary>
279         /// Applies interpolation between consecutive poses
280         /// to reduce tracking jitter.
281         /// </summary>
282         void UpdatePoseSmoothing(
283         Pose currentPose,
284         TargetStatus currentTargetStatus)
285         {
286             if (mActivePoseLerp == null &&
287             ShouldSmooth(
288             currentPose,
289             currentTargetStatus))
290             {
291                 // Create interpolation object
292                 mActivePoseLerp =
293                 new PoseLerp(
294                 mPreviousPose,
295                 currentPose,
296                 mAnimationCurve);
297             }
298
299             if (mActivePoseLerp != null)
300             {
301                 // Compute smoothed pose
302                 var pose =
303                 mActivePoseLerp.GetSmoothedPosition(
304                 Time.deltaTime);
305
306                 // Apply smoothed transformation
307                 mTarget.transform.SetPositionAndRotation(
308                 pose.position,
309                 pose.rotation);
310
311                 // Stop interpolation when completed
312                 if (mActivePoseLerp.Complete)
313                 {
314                     mActivePoseLerp = null;
315                 }
316             }
317         }
```

```
        }
    }
```

C.1 Image, multi and cylinder targets configuration

To define a new image-based target, it is first necessary to access the online database and select the *Add Target* option. From the corresponding interface, the user can specify the target type (i.e., Image Target, Multi Target, or Cylinder Target) and assign a name to the newly created element.

For an Image Target, the width of the square image must be defined. For a Multi Target, the three geometric dimensions are required, namely length, width, and height. For a Cylinder Target, the diameters of the upper and lower bases, together with the height, must be specified. In addition, depending on the selected target type, the appropriate number of images must be provided, accounting for texture wrapping in the case of Multi Target and Cylinder Target configurations.

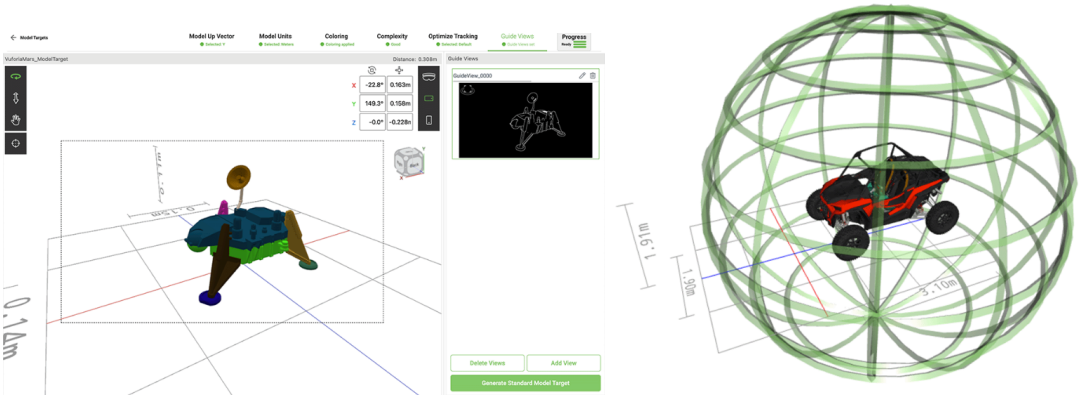
Once the targets have been defined within the database, the workflow continues in Unity, where a dedicated *GameObject* is created to represent the target in the scene. This is achieved by attaching the appropriate Vuforia components, namely the *ImageTargetBehaviour* and *ImageTargetPreview* for Image Targets, the *MultiTargetBehaviour* and *MultiTargetPreview* for Multi Targets, or the *CylinderTargetBehaviour* and *CylinderTargetPreview* for Cylinder Targets, together with the *DefaultObserverEventHandler* script (Listing C.1). Within the Inspector panel, under the corresponding *TargetBehaviour* component, the previously created database can be assigned and the specific target selected. This configuration links the Unity *GameObject* to the corresponding entry in the Vuforia database, enabling runtime recognition and tracking of the physical target.

C.2 Model target configuration

To define a new Model Target, the Vuforia extension software must be used: *Model Target Generator* (MTG). In this program, a project must be created by importing the three-dimensional model of the object that will be used for recognition. The file must be a CAD model or a mesh that is properly scaled and consistent with real-world units. Once the model is imported, the system starts a geometric analysis phase, which represents the core of the MTG. During this phase, the software interprets the three-dimensional structure and prepares the data required to build the recognition model. The first fundamental element to define is the tracking origin, i.e., the spatial reference point with respect to which tracking will be performed at runtime. Next, the model's bounding box is defined, representing the overall spatial extent of the object. This operation allows the system to understand the spatial limits within which the model is contained and contributes to the correct normalization of the geometry. In parallel, the model orientation is set in order to align the virtual coordinate system with the physical one.

An essential step in the process is the definition of the Guide View. This represents a reference view of the model used to facilitate the initialization of tracking. The Guide

View indicates the ideal position from which the user should observe the object so that the system can recognize it reliably, thus improving the application startup experience (Figure C.1a).



(a) Example of guide views generation.

(b) Example of full 360° training.

Figure C.1: Vuforia Engine model target and advanced model target generation.

Following the initial configuration, the MTG automatically generates a set of views of the three-dimensional model. These views constitute a multi-angular representation of the object and allow the system to recognize it from different perspectives.

Advanced Model Targets represent an extension of the traditional Model Target paradigm introduced by Vuforia Engine, designed to improve the flexibility and automation of the three-dimensional recognition process. Unlike standard Model Targets, which require the explicit presence of a Guide View to initialize tracking, Advanced Model Targets are able to learn the full three-dimensional structure of the object and recognize it from multiple viewpoints, often even at 360 degrees, without requiring precise initial user alignment (Figure C.1b). This is made possible by a cloud-based training process that uses synthetic views of the three-dimensional model to train a recognition system based on deep learning (Section C.2.1). The resulting model learns the geometric features of the object and enables its identification under highly variable viewing conditions.

The creation of an Advanced Model Target is still performed through the Model Target Generator, but it requires an additional processing stage compared to conventional Model Targets. After importing the three-dimensional model and defining the main geometric parameters, the system performs an advanced sampling procedure of the possible views of the object. During this phase, the software automatically generates a large number of virtual observations distributed around the model geometry, removing redundant or geometrically uninformative configurations. The final result is an optimized database that maximizes tracking robustness while reducing computational load.

After the targets have been generated and stored in the database, the implementation proceeds in Unity by creating a dedicated *GameObject* that acts as a scene representation of each target. The corresponding Vuforia components are then attached to this object, including *ModelTargetBehaviour*, *ModelTargetPreview*, and the *DefaultObserverEventHandler* script. The configuration is completed through the Inspector panel,

where the database is first assigned within the *ModelTargetBehaviour* component. Subsequently, the specific target defined in the database is selected. This setup establishes a direct link between the Unity *GameObject* and the associated database entry in Vuforia, enabling real-time detection and tracking of the physical target during runtime.

C.2.1 Neural networks and deep learning

Neural networks are a class of machine learning models inspired by the structure and functioning of the human brain [307]. They consist of layers of interconnected nodes, or neurons, each of which transforms input data and forwards the result to subsequent layers. A standard architecture includes an input layer, one or more hidden layers, and an output layer. Each neuron computes a weighted sum of its inputs, adds a bias term, and applies a non-linear activation function to produce its output. This mechanism allows the network to model complex, non-linear relationships in data. Activation functions play a key role in the expressive power of neural networks. Common choices include the sigmoid function, the hyperbolic tangent, and the Rectified Linear Unit (ReLU) [308]. Among these, ReLU is widely adopted in modern architectures due to its computational efficiency and its effectiveness in alleviating the vanishing gradient problem during training.

Training a neural network involves optimizing its parameters to minimize a loss function. The weights are typically initialized randomly and iteratively updated using gradient-based optimization methods such as gradient descent. The computation of gradients is efficiently performed through the backpropagation algorithm [309], which propagates the error from the output layer back through the network. Through this optimization process, the model learns a mapping from inputs to outputs that minimizes prediction error. The performance of the network depends on several factors, including its architecture, the choice of learning rate, the quantity and quality of training data, and the optimization strategy employed.

Neural networks have been successfully applied to a wide range of tasks, including image recognition [310], natural language processing [311], and time-series forecasting [312]. Their ability to learn representations directly from data reduces the need for manual feature engineering.

Deep learning refers to neural networks composed of multiple hidden layers. This depth enables the model to learn hierarchical representations of the input data, where successive layers capture increasingly abstract features. Compared to shallow architectures, deep neural networks can represent more complex functions, although they typically require larger datasets and greater computational resources. In practice, deep learning models are trained using the same backpropagation and gradient-based optimization techniques as standard neural networks, but applied to architectures with a significantly higher number of parameters. These models have become a cornerstone of modern artificial intelligence due to their strong empirical performance across diverse domains.

In summary, neural networks provide a flexible framework for learning from data and solving complex tasks, though their effectiveness often depends on sufficient data availability and computational resources.

Appendix D

Basic and kinematics constraints

The formulation of the constraint equations associated with the absolute position of bodies in space, as well as with the relative position between pairs of kinematically connected bodies, requires the preliminary definition of certain orthogonality and parallelism conditions between pairs of vectors. These conditions, which will be referred to as *basic constraints*, provide a tool for characterizing the constraints introduced by kinematic pairs [303].

D.1 First orthogonality constraint

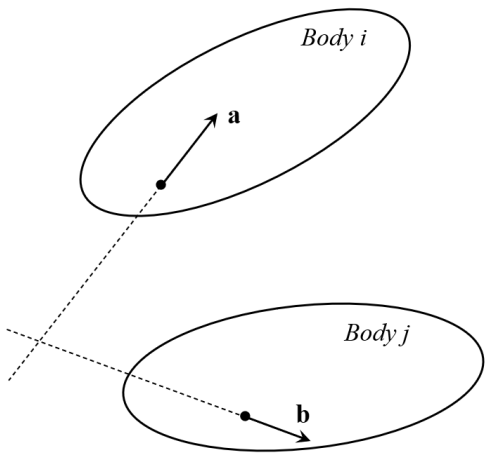


Figure D.1: First orthogonality constraint.

Given two non-zero vectors expressed with respect to the global reference frame, ${}^0\mathbf{a}$ and ${}^0\mathbf{b}$, rigidly attached to bodies i and j, respectively, the necessary and sufficient condition for the two vectors to be orthogonal (Figure D.1) is that their scalar product vanishes:

$$\Psi^{o1}\left({}^0\mathbf{a}, {}^0\mathbf{b}\right) = {}^0\mathbf{a}^T \cdot {}^0\mathbf{b} = 0, \tag{D.1}$$

where $\Psi^{o1}\left({}^0\mathbf{a}, {}^0\mathbf{b}\right)$ denotes the first orthogonality constraint. Expressing the vectors ${}^0\mathbf{a}$ and ${}^0\mathbf{b}$ in the reference frames attached to bodies i and j, respectively, the previous relation becomes:

$$\Psi^{o1}\left({}^0\mathbf{a}, {}^0\mathbf{b}\right) = {}^i\mathbf{a}^T \cdot {}^0_i\mathbf{R}^T \cdot {}^0_j\mathbf{R} \cdot {}^j\mathbf{b} = 0, \tag{D.2}$$

where ${}^0_i\mathbf{R}$ and ${}^0_j\mathbf{R}$ are the transformation matrices from the body-fixed reference frames to the global reference frame.

D.2 Second orthogonality constraint

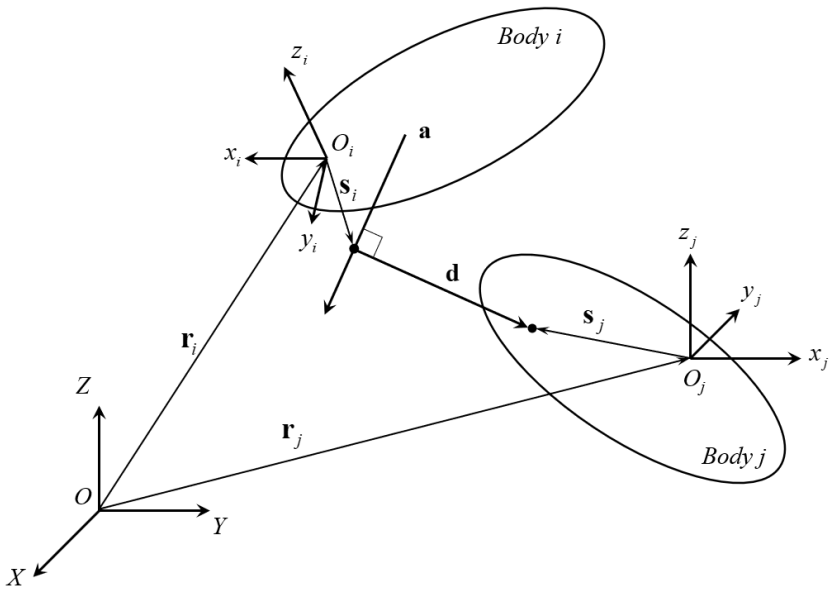


Figure D.2: Second orthogonality constraint.

The condition for a vector ${}^0\mathbf{a}$, rigidly attached to body i and expressed with respect to the global reference frame, and a non-zero vector ${}^0\mathbf{d}$, connecting two points belonging respectively to bodies i and j, to be orthogonal is obtained by imposing that their scalar product vanishes:

$$\Psi^{o2}\left({}^0\mathbf{a}, {}^0\mathbf{d}\right) = {}^0\mathbf{a}^T \cdot {}^0\mathbf{d} = 0, \tag{D.3}$$

where Ψ^{o2} denotes the second orthogonality constraint.
Referring to the geometry shown in Figure D.2, expressing the vector ${}^0\mathbf{d}$ as:

$${}^0\mathbf{d} = {}^0\mathbf{r}_j + {}^0_j\mathbf{R} \cdot {}^j\mathbf{s}_j - {}^0\mathbf{r}_i - {}^0_i\mathbf{R} \cdot {}^i\mathbf{s}_i, \tag{D.4}$$

Equation D.3 becomes:

$$\Psi^{o2}\left({}^0\mathbf{a}, {}^0\mathbf{d}\right) = {}^i\mathbf{a}^T \cdot {}^0_i\mathbf{R}^T \cdot ({}^0\mathbf{r}_j + {}^0_j\mathbf{R} \cdot {}^j\mathbf{s}_j - {}^0\mathbf{r}_i) - {}^i\mathbf{a}^T \cdot {}^i\mathbf{s}_i = 0 \tag{D.5}$$

The relation just derived is not symmetric with respect to the indices i and j. However, the condition that the vector ${}^0\mathbf{b}$, rigidly attached to body j, is orthogonal to the vector ${}^0\mathbf{d}$ can be obtained by exchanging the indices i and j in Equation D.5.

D.3 Spherical constraint

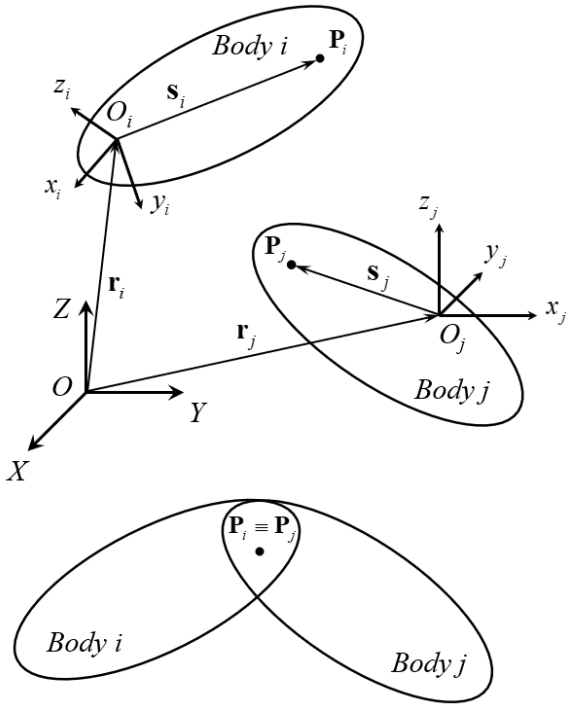


Figure D.3: Spherical constraint.

Referring to the geometry shown in Figure D.3, in order for two points $\mathbf{P}_i$ and $\mathbf{P}_j$, belonging respectively to bodies i and j, to be geometrically coincident, the following condition must be satisfied:

$$\Psi^s(\mathbf{P}_i, \mathbf{P}_j) = {}^0\mathbf{r}_j + {}^0_j\mathbf{R} \cdot {}^j\mathbf{s}_j - {}^0\mathbf{r}_i - {}^0_i\mathbf{R} \cdot {}^i\mathbf{s}_i = \mathbf{0}. \tag{D.6}$$

This relation is symmetric with respect to the indices i and j and gives rise to three scalar constraint equations.

D.4 Distance constraint

Given two points $\mathbf{P}_i$ and $\mathbf{P}_j$, belonging respectively to bodies i and j, the distance constraint imposes that the distance between them is equal to a constant value $c \neq 0$:

$$\Psi^{ss}(\mathbf{P}_i, \mathbf{P}_j; C) = {}^0\mathbf{d}^T \cdot {}^0\mathbf{d} - c^2 = 0, \tag{D.7}$$

where ${}^0\mathbf{d}$ represents the distance between the two points (Figure D.4).

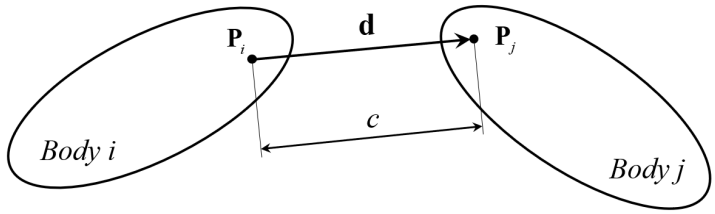


Figure D.4: Distance constraint.

D.5 First parallelism constraint

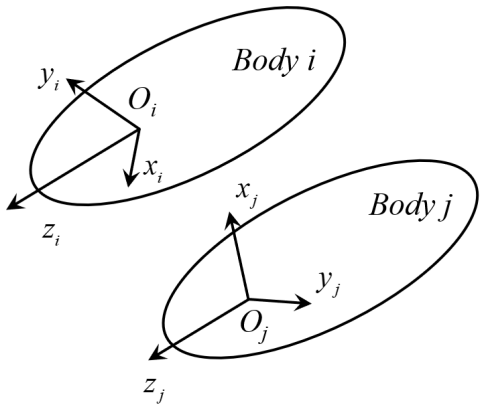


Figure D.5: First parallelism constraint.

Consider two bodies i and j whose joint reference frames have origins at points $\mathbf{P}_i$ and $\mathbf{P}_j$, respectively. Referring to the geometry shown in Figure D.5, the axes $\mathbf{z}_i$ and $\mathbf{z}_j$ remain parallel if and only if $\mathbf{z}_j$ is simultaneously orthogonal to both $\mathbf{x}_i$ and $\mathbf{y}_i$:

$$\Psi^{p1}(\mathbf{z}_i, \mathbf{z}_j) = \begin{Bmatrix} \Psi^{o1}(\mathbf{x}_i, \mathbf{z}_j) \\ \Psi^{o1}(\mathbf{y}_i, \mathbf{z}_j) \end{Bmatrix} = \begin{Bmatrix} \mathbf{x}_i^T \cdot \mathbf{z}_j \\ \mathbf{y}_i^T \cdot \mathbf{z}_j \end{Bmatrix} = \mathbf{0}. \tag{D.8}$$

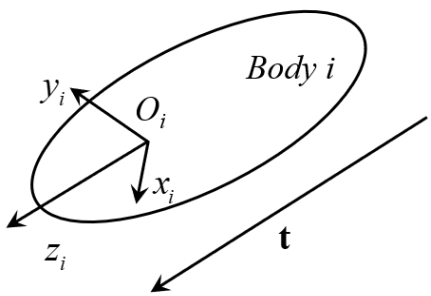


Figure D.6: Second parallelism constraint.

D.6 Second parallelism constraint

Consider the case in which the unit vector $\mathbf{z}_i$ must remain parallel to a generic vector $\mathbf{t}$, as shown in Figure D.6. For $\mathbf{t} \neq \{0\}$, it is parallel to $\mathbf{z}_i$ if and only if it is perpendicular to both $\mathbf{x}_i$ and $\mathbf{y}_i$. Therefore, the parallelism condition is expressed by the following two scalar equations:

$$\Psi^{p2}(\mathbf{z}_i, \mathbf{t}) = \begin{Bmatrix} \Psi^{o2}(\mathbf{x}_i, \mathbf{t}) \\ \Psi^{o2}(\mathbf{y}_i, \mathbf{t}) \end{Bmatrix} = \begin{Bmatrix} \mathbf{x}_i^T \cdot \mathbf{t} \\ \mathbf{y}_i^T \cdot \mathbf{t} \end{Bmatrix} = \mathbf{0}. \tag{D.9}$$

Table D.1 summarizes the number of scalar equations associated with each basic constraint.

Constraint	**Number of constraint equations**
First orthogonality constraint (Ψ^{o1})	1
Second orthogonality constraint (Ψ^{o2})	1
First parallelism constraint (Ψ^{p1})	2
Second parallelism constraint (Ψ^{p2})	2
Spherical constraint (Ψ^{s})	3
Distance constraint (Ψ^{ss})	1

Table D.1: Number of scalar equations associated with each basic constraint.

D.7 Revolute joint

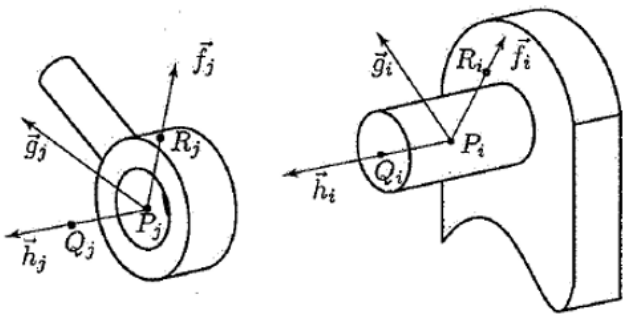


Figure D.7: Revolute joint [303].

$\mathbf{P}_i$ at the joint center.
$\mathbf{h}_i$ parallel to the joint axis.
$\mathbf{f}_i$ perpendicular to the joint axis.

$\mathbf{P}_j$ at the joint center.
$\mathbf{h}_j$ parallel to the joint axis.
$\mathbf{f}_j$ perpendicular to the joint axis.

A revolute joint is a kinematic constraint between two rigid bodies i and j that allows only a relative rotation about a common axis, referred to as the joint axis, while preventing all other relative motions. Referring to Figure D.7, the origins of the joint reference frames are identified on the two bodies by the points $\mathbf{P}_i$ and $\mathbf{P}_j$, whereas the axis of relative rotation passes through the points $\mathbf{Q}_i$ and $\mathbf{Q}_j$, and is defined by the unit vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ associated with the $\mathbf{z}$ axes of the corresponding joint reference frames. The orientation of the remaining axes of the joint reference frame can be chosen arbitrarily. The constraints introduced by a revolute joint require that:

- the absolute coordinates of $\mathbf{P}_i$ and $\mathbf{P}_j$ coincide;
- the unit vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ remain parallel.

Algebraically, these constraints are expressed by the following conditions:

$$\Psi^s(\mathbf{P}_i, \mathbf{P}_j) = \mathbf{0}, \tag{D.10a}$$

$$\Psi^{p1}({}^0\mathbf{h}_i, {}^0\mathbf{h}_j) = \mathbf{0}. \tag{D.10b}$$

where ${}^0\mathbf{h}_i$ and ${}^0\mathbf{h}_j$ denote the unit vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ expressed in the absolute reference frame. Each of the five scalar equations resulting from the above conditions introduces one kinematic constraint. The remaining degree of freedom allows only the relative rotation of the two bodies about the joint axis.

D.8 Cylindrical joint

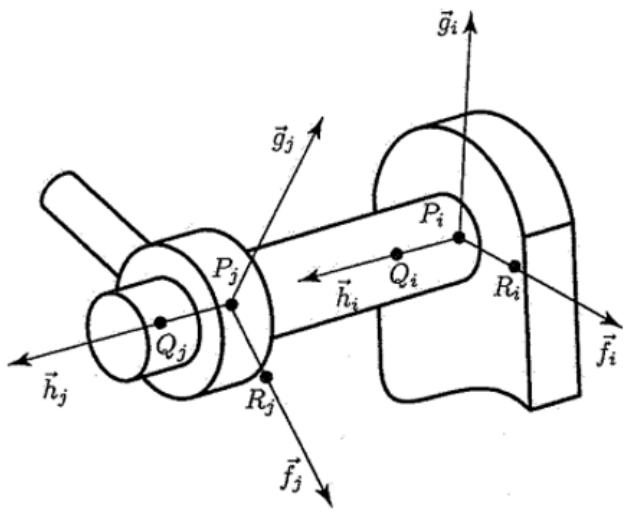


Figure D.8: Cylindrical joint [303].

$\mathbf{P}_i$ on the joint axis.	$\mathbf{P}_j$ on the joint axis.
$\mathbf{h}_i$ parallel to the joint axis.	$\mathbf{h}_j$ parallel to the joint axis.
$\mathbf{f}_i$ perpendicular to the joint axis.	$\mathbf{f}_j$ perpendicular to the joint axis.

A cylindrical joint performs the same function as a revolute joint, with the difference that translations along the axis of relative rotation between the two bodies are not constrained. Referring to Figure D.8, the joint definition points $\mathbf{P}_i$ and $\mathbf{P}_j$ lie on the common axis, which is defined by the points $\mathbf{Q}_i$ and $\mathbf{Q}_j$ and therefore by the vectors $\mathbf{h}_i$ and $\mathbf{h}_j$. The analytical formulation of the constraints imposed by a cylindrical joint is obtained by requiring that the unit vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ are collinear. Collinearity is ensured by imposing that $\mathbf{h}_i$ remains parallel both to $\mathbf{h}_j$ and to the vector $\mathbf{d}_{ij}$, whose endpoints are the points $\mathbf{P}_i$ and $\mathbf{P}_j$. This leads to the following equations

$$\Psi^{p1}({}^0\mathbf{h}_i, {}^0\mathbf{h}_j) = \mathbf{0}, \tag{D.11a}$$

$$\Psi^{p2}({}^0\mathbf{h}_i, {}^0\mathbf{d}_{ij}) = \mathbf{0}. \tag{D.11b}$$

where ${}^0\mathbf{h}_i$, ${}^0\mathbf{h}_j$ and ${}^0\mathbf{d}_{ij}$ are the vectors $\mathbf{h}_i$, $\mathbf{h}_j$ and $\mathbf{d}_{ij}$ respectively expressed in the absolute reference frame. Each of the four scalar equations introduces one constraint, and the remaining degrees of freedom correspond to rotation about the common axis and translation along it.

D.9 Prismatic joint

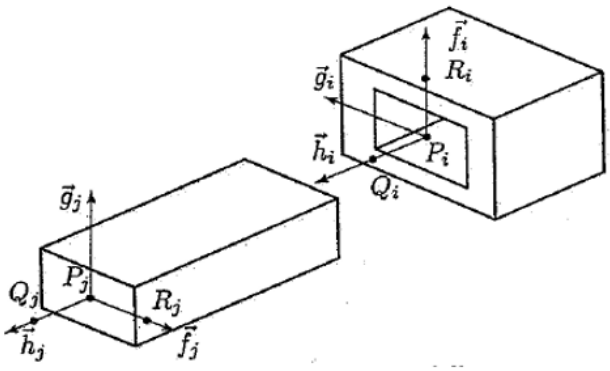


Figure D.9: Prismatic joint [303].

$\mathbf{P}_i$ on the joint axis.	$\mathbf{P}_j$ on the joint axis.
$\mathbf{h}_i$ parallel to the joint axis.	$\mathbf{h}_j$ parallel to the joint axis.
$\mathbf{f}_i$ perpendicular to the joint axis.	$\mathbf{f}_j$ perpendicular to the joint axis.

A prismatic joint is a kinematic constraint between two rigid bodies i and j that allows only relative translations along a single common axis and prevents relative rotations about the same axis. Referring to Figure D.9, the geometric constraints imposed by this joint are defined by introducing the points $\mathbf{P}_i$, $\mathbf{Q}_i$, $\mathbf{P}_j$ and $\mathbf{Q}_j$ along the translation axis. The axes $\mathbf{f}_i$ and $\mathbf{f}_j$ are chosen to be perpendicular. The analytical formulation of the constraints imposed by a prismatic joint is obtained by requiring that the unit vectors $\mathbf{f}_i$

and $\mathbf{f}_j$ are orthogonal and that the unit vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ are collinear, as in the case of the cylindrical joint D.11. This leads to the equations

$$\Psi^{p1}({}^0\mathbf{h}_i, {}^0\mathbf{h}_j) = \mathbf{0}, \tag{D.12a}$$

$$\Psi^{p2}({}^0\mathbf{h}_i, {}^0\mathbf{d}_{ij}) = \mathbf{0}, \tag{D.12b}$$

$$\Psi^{o1}({}^0\mathbf{f}_i, {}^0\mathbf{f}_j) = \mathbf{0}. \tag{D.12c}$$

where ${}^0\mathbf{h}_i$, ${}^0\mathbf{h}_j$ are the unit vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ respectively, ${}^0\mathbf{f}_i$, ${}^0\mathbf{f}_j$ are the unit vectors $\mathbf{f}_i$ and $\mathbf{f}_j$, and ${}^0\mathbf{d}_{ij}$ is the vector $\mathbf{d}_{ij}$ expressed in the absolute reference frame. In conclusion, the only remaining degree of freedom is associated with translation along the joint axis.

D.10 Spherical joint

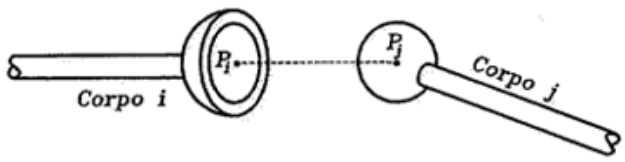


Figure D.10: Spherical joint [303].

$\mathbf{P}_i$ at the joint center.	$\mathbf{P}_j$ at the joint center.
$\mathbf{h}_i$, $\mathbf{f}_i$ arbitrary.	$\mathbf{h}_j$, $\mathbf{f}_j$ arbitrary.

A spherical joint is defined by the condition that the center of the sphere, represented by the point $\mathbf{P}_i$ belonging to body i, coincides with the center of its socket, represented by the point $\mathbf{P}_j$ belonging to body j (Figure D.10). This condition coincides with the spherical constraint

$$\Psi^s(\mathbf{P}_i, \mathbf{P}_j) = \mathbf{0}. \tag{D.13}$$

The three scalar equations impose restrictions exclusively on the relative position of the points $\mathbf{P}_i$ and $\mathbf{P}_j$ and not on the orientation of the connected bodies.

D.11 Screw joint

The screw joint can be regarded as a cylindrical joint in which rotation and translation along the axis are coupled by a constraint relation. Referring to the geometry in Figure D.11, setting $\mathbf{d}_{ij} = \mathbf{P}_i\mathbf{P}_j$ and denoting

- θ the angle, positive if counterclockwise, between the unit vectors $\mathbf{f}_i$ and $\mathbf{f}_j$;
- θ_0, the value of θ when $\mathbf{P}_i \equiv \mathbf{P}_j$;
- n the number of revolutions measured from $\mathbf{P}_i \equiv \mathbf{P}_j$;

- α the screw pitch;

this relation can be written as

$$ {}^{0}\mathbf{h}_i^T \mathbf{d}_{ij} = \alpha(\theta + 2n\pi - \theta_0). \tag{D.14} $$

where ${}^{0}\mathbf{h}_i$ is the unit vector $\mathbf{h}_i$ expressed in the absolute reference frame. Therefore, the presence of the screw joint is described, in addition to the cylindrical joint constraints D.11, also by the equation

$$ \Psi^{scr}({}^{0}\mathbf{h}_i, \mathbf{P}_i, \mathbf{P}_j, \alpha, \theta_0, n) \equiv {}^{0}\mathbf{h}_i^T \mathbf{d}_{ij} - \alpha(\theta + 2n\pi - \theta_0) = 0. \tag{D.15} $$

The screw joint contributes five scalar equations to the position analysis system.

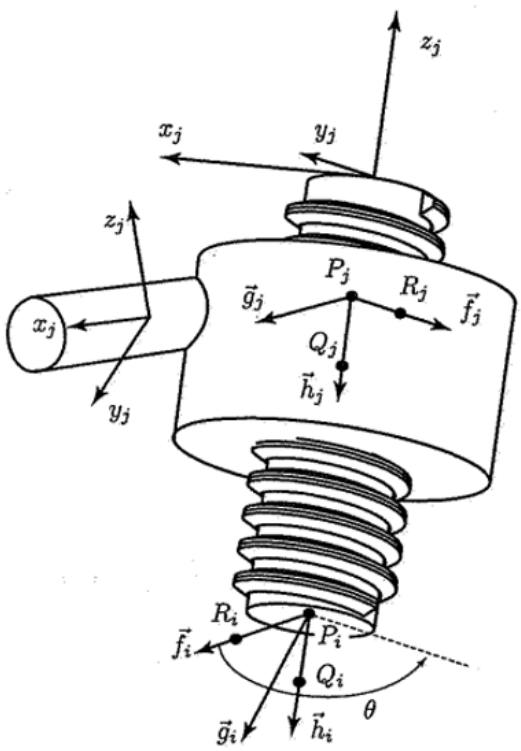


Figure D.11: Screw joint [303].

$\mathbf{P}_i$ on the joint axis.	$\mathbf{P}_j$ on the joint axis.
$\mathbf{h}_i$ parallel to the joint axis.	$\mathbf{h}_j$ parallel to the joint axis.
$\mathbf{f}_i$ perpendicular to the joint axis.	$\mathbf{f}_j$ perpendicular to the joint axis.
θ angle between $\mathbf{f}_i$ and $\mathbf{f}_j$.	$\mathbf{d}_{ij} = \mathbf{P}_i\mathbf{P}_j$

Appendix E

Catmull–Clark refinement algorithm

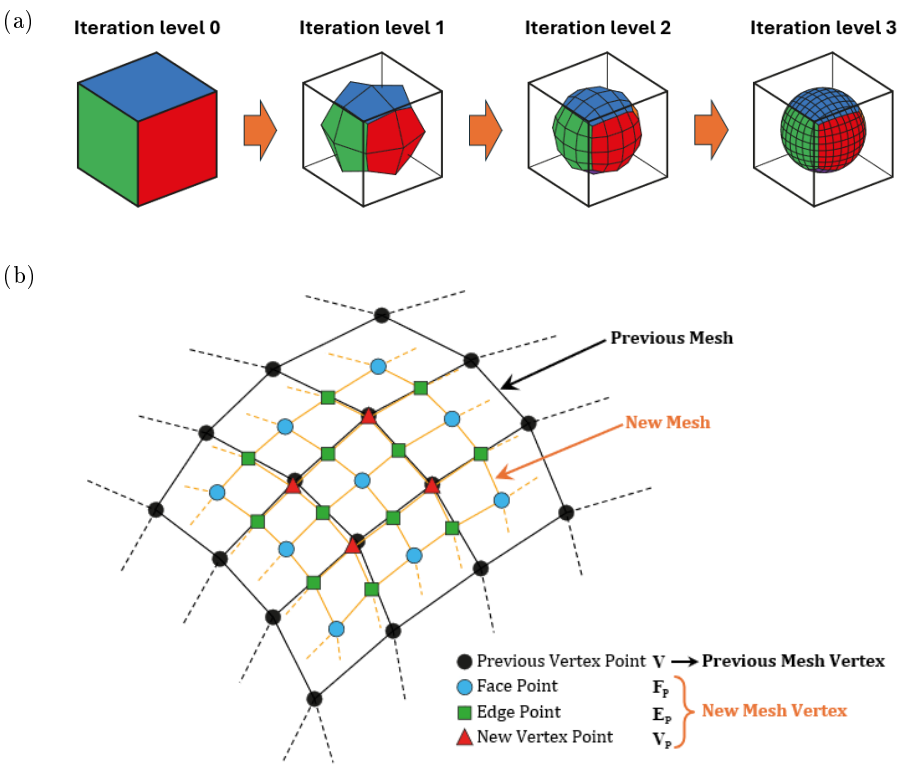


Figure E.1: (a) Iteration levels of the Catmull-Clark recursive procedure, (b) single iteration of the Catmull-Clark refinement algorithm.

The subdivision surface approach is a technique in Computer-Aided Design used to create smooth, high-resolution surfaces by iteratively refining a coarse polygonal mesh. At each subdivision step, new vertices are introduced, and existing vertices are repositioned according to specific rules to smooth out sharp edges and corners. This process produces a limit surface that appears smooth and organic, while the initial mesh provides control over the surface's general shape.

During the last decades, several refinement algorithms have been proposed. The

most commonly used are the Catmull-Clark [313], Doo-Sabin [314], and Loop [315] algorithms. The Catmull-Clark algorithm is designed to work with polygonal meshes composed mainly of quadrilaterals, and it generates limit surfaces with G^2 continuity everywhere and G^1 continuity at extraordinary vertices. The algorithm operates iteratively, with each iteration refining the surface by adding new vertices based on the positions of existing ones, as well as the surrounding edges and faces. The number of iterations directly influences the smoothness and detail of the resulting surface: as the iterations increase, the surface converges towards a smoother approximation of the desired geometry (Figure E.1a).

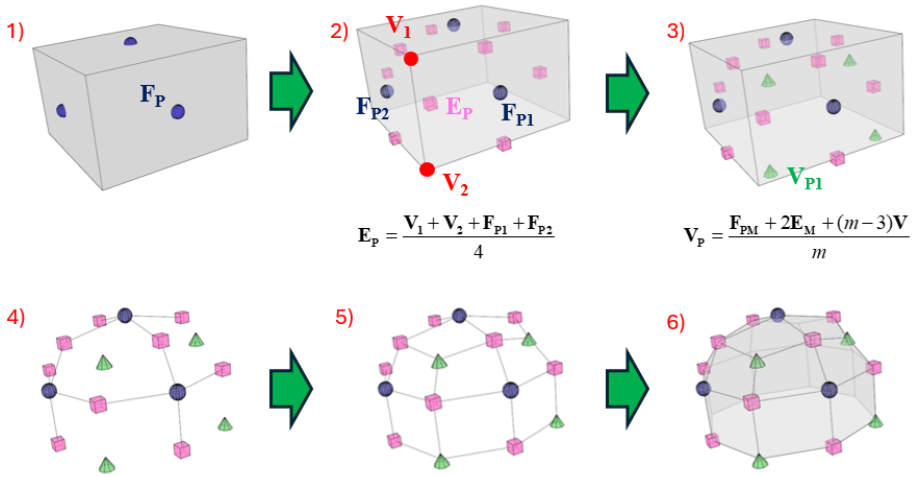


Figure E.2: Detailed step-by-step view of the first iteration.

At each iteration, the previous mesh is subdivided by calculating the positions of the new vertices based on the vertices of the previous mesh (Figure E.2). For each face of the mesh, its midpoint (Face Point) $\mathbf{F_P}$ is calculated as the average of the coordinates of the vertices that form the face. For each edge of the mesh, the Edge Point $\mathbf{E_P}$ is calculated as the average of the coordinates of the two vertices $\mathbf{V_1}$ and $\mathbf{V_2}$ that define it, along with the two Face Points $\mathbf{F_{P1}}$ and $\mathbf{F_{P2}}$ of the adjacent faces:

$$\mathbf{E_P} = \frac{1}{4}(\mathbf{V_1} + \mathbf{V_2} + \mathbf{F_{P1}} + \mathbf{F_{P2}}) \tag{E.1}$$

For each edge, the midpoint $\mathbf{E}$ is calculated as the average of the adjacent vertices. For each generic vertex $\mathbf{V}$ of the previous mesh, the average $\mathbf{E_M}$ of the m midpoints of the adjacent edges is calculated as follows:

$$\mathbf{E_M} = \frac{1}{m}\sum_{i=0}^{m} \mathbf{E}_i \tag{E.2}$$

and the average $\mathbf{F_{PM}}$ of the m Face Points $\mathbf{F_P}$ adjacent to the vertex is calculated as follows:

$$\mathbf{F_{PM}} = \frac{1}{m}\sum_{i=0}^{m} \mathbf{F_P}_i \tag{E.3}$$

For each vertex of the previous mesh, its new position $\mathbf{V_P}$ is calculated as the weighted average of the corresponding $\mathbf{F_{PM}}$, $\mathbf{E_M}$ and the vertex $\mathbf{V}$ itself:

$$\mathbf{V_P} = \frac{1}{m}\left(\mathbf{F_P} + 2 \cdot \mathbf{E_M} + (m-3) \cdot \mathbf{V}\right) \tag{E.4}$$

The new mesh is generated by connecting the Face Points with the Edge Points and linking the New Vertex Points to the Edge Points. Figure E.1b shows one iteration level, highlighting the previous mesh, the Face Points, the Edge Points, the new positions of the Vertex Points, and the connections between them that generate the new mesh. Listing E.1 shows a simplified implementation of the Catmull-Clark subdivision algorithm.

Listing E.1: Simplified implementation of the Catmull-Clark subdivision algorithm.

```
1
2   public Mesh Subdivide(Mesh mesh)
3   {
4       // 1. Compute face points
5       foreach (Face face in mesh.Faces)
6       {
7           face.FacePoint = Average(face.Vertices);
8       }
9
10      // 2. Compute edge points
11      foreach (Edge edge in mesh.Edges)
12      {
13          edge.EdgePoint = Average(
14          edge.Vertex1,
15          edge.Vertex2,
16          edge.Face1.FacePoint,
17          edge.Face2.FacePoint
18          );
19      }
20
21      // 3. Reposition original vertices
22      foreach (Vertex vertex in mesh.Vertices)
23      {
24          int n = vertex.AdjacentFaces.Count;
25          Point F = AverageFacePoints(vertex);
26          Point R = AverageEdgeMidpoints(vertex);
27          Point P = vertex.Position;
28
29          vertex.NewPosition =
30          (F + 2 * R + (n - 3) * P) / n;
31      }
32      Mesh refinedMesh = new Mesh();
33
34      // 4. Split each face into new quadrilateral faces
35      foreach (Face face in mesh.Faces)
36      {
37          foreach (Edge edge in face.Edges)
38          {
39              refinedMesh.AddFace(
40              edge.Vertex1.NewPosition,
41              edge.EdgePoint,
42              face.FacePoint,
43              edge.Previous.EdgePoint
44              );
45          }
46      }
47      return refinedMesh;
48  }
```

Appendix F

Delaunay triangulation

Delaunay triangulation is a method in computational geometry used to construct a triangulated network from a given set of points [316,317]. It is widely employed because it provides a robust and well-conditioned discretisation of geometric domains, which is particularly important in numerical methods where mesh quality directly affects stability and accuracy, such as finite element analysis, surface reconstruction, interpolation, and general numerical simulation.

Its fundamental property is the Delaunay condition: for every triangle in the triangulation, the circumcircle contains no other point from the set. This property ensures the generation of well-conditioned triangles by avoiding degenerate configurations or elements with high aspect ratios, making the resulting structure particularly suitable for numerical simulations, mesh generation, and interpolation problems. A natural extension of this concept to three-dimensional space is the Delaunay tetrahedralization, where the decomposition is performed using tetrahedra instead of triangles, while preserving the analogous empty circumsphere property.

Among the various algorithms available for constructing a Delaunay triangulation, one of the most widely used is the Bowyer-Watson algorithm, which is based on an incremental approach [318] (Listing F.1). As shown in Figure F.1, the general principle is as follows:

- An initial triangle (called the *super-triangle*) that encloses all the points in the dataset is constructed.
- The first point of the dataset is selected and connected to the vertices of the super-triangle, subdividing the region into new triangles.
- The second point is selected, and the corresponding local triangulation is formed; for these triangles, the circumcircle is considered.
- All triangles whose circumcircle contains the newly inserted point are identified and removed, generating a polygonal cavity.
- The procedure is repeated iteratively for all remaining points in the dataset, updating the triangulation at each insertion.

- At the end of the process, the initial super-triangle and all triangles and connections associated with it are removed.

This approach provides a simple and efficient way to obtain a valid Delaunay triangulation for the entire point set.

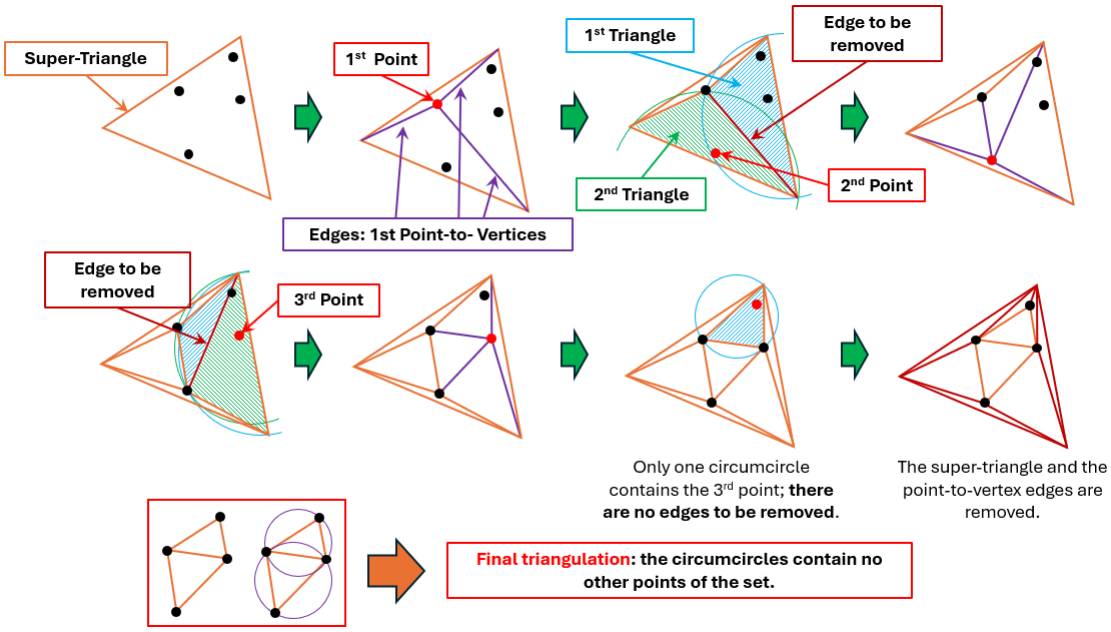


Figure F.1: Graphical representation of the incremental Bowyer-Watson procedure for the generation of a Delaunay triangulation from a discrete set of points in the plane.

An alternative and widely adopted approach for constructing the Delaunay triangulation is based on a geometric transformation that reduces the problem to the computation of a convex hull in a higher-dimensional space. This technique is known as the *lifting method* [319] (Listing F.2). The key idea is to embed the original 2D points into a 3D space using a quadratic transformation. Each point $\mathbf{P} = (x, y)$ is mapped to a point on a paraboloid defined by

$$\varphi(x, y) = (x, y, x^2 + y^2) \tag{F.1}$$

This mapping places all points on a smooth curved surface in 3D (Figure F.2). A fundamental property of this transformation is that geometric relationships in 2D (specifically, the Delaunay condition based on empty circumcircles) become equivalent to visibility relations on the convex hull of the lifted points. After the transformation, a standard 3D convex hull is computed on the set of lifted points. The convex hull consists of triangular faces forming a polyhedral surface. Each face can be classified according to the orientation of its normal vector. Let $\mathbf{n} = (n_x, n_y, n_z)$ be the normal vector of a face. The faces satisfying

$$n_z < 0 \tag{F.2}$$

are those oriented toward the negative direction of the lifting axis. These faces form the *lower convex hull*, which corresponds exactly to the Delaunay triangulation in the

original 2D space.

All remaining faces (with $n_z \geq 0$) belong to the upper hull and are discarded.

Finally, the selected faces are projected back to the original 2D space by simply removing the third coordinate:

$$(x, y, z) \rightarrow (x, y). \tag{F.3}$$

The resulting set of triangles defines the Delaunay triangulation of the original point set. This equivalence between the Delaunay triangulation and the lower convex hull is a direct consequence of the geometric duality induced by the paraboloid lifting. In particular, the empty circumcircle property in 2D corresponds to the supporting hyperplane property in 3D, which guarantees the correctness of the construction.

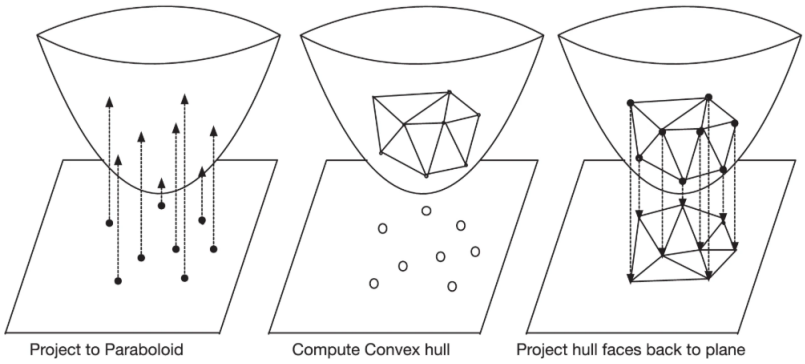


Figure F.2: Example illustrating the relationship between Delaunay triangulation and convex hull using a paraboloid lifting defined by $x^2 + y^2$ (left). The convex hull is computed in $\mathbb{R}^3$ by mapping each point (x, y) to $(x, y, x^2 + y^2)$ (center). The resulting convex hull is projected back onto $\mathbb{R}^2$, yielding the Delaunay triangulation (right) [320].

Listing F.1: Implementation of the Bowyer-Watson algorithm for 2D Delaunay triangulation.

```
1
2   public class BowyerWatsonTriangulator
3   {
4       /// <summary>
5       /// Computes the 2D Delaunay triangulation using the Bowyer-Watson
            algorithm.
6       /// </summary>
7       public List<Triangle> ComputeTriangulation(List<Point2D> points)
8       {
9           List<Triangle> mesh = new List<Triangle>();
10
11          // 1. Initialize with a super-triangle that contains all points
12          Triangle superTriangle = GenerateSuperTriangle(points);
13          mesh.Add(superTriangle);
14
15          // 2. Incremental insertion of points
16          foreach (Point2D point in points)
17          {
18              List<Triangle> badTriangles = new List<Triangle>();
19
20              // Find triangles whose circumcircle contains the point
21              foreach (Triangle triangle in mesh)
22              {
23                  if (triangle.CircumcircleContains(point))
```

```
24                  {
25                      badTriangles.Add(triangle);
26                  }
27              }
28
29              List<Edge> cavityBoundary = new List<Edge>();
30
31              // Extract boundary of the cavity
32              foreach (Triangle triangle in badTriangles)
33              {
34                  foreach (Edge edge in triangle.Edges)
35                  {
36                      int count = 0;
37
38                      foreach (Triangle t in badTriangles)
39                      {
40                          if (t.ContainsEdge(edge))
41                          count++;
42                      }
43
44                      // Keep only boundary edges
45                      if (count == 1)
46                      cavityBoundary.Add(edge);
47                  }
48              }
49
50              // Remove invalid triangles
51              foreach (Triangle triangle in badTriangles)
52              {
53                  mesh.Remove(triangle);
54              }
55
56              // Re-triangulate cavity
57              foreach (Edge edge in cavityBoundary)
58              {
59                  mesh.Add(new Triangle(edge.P1, edge.P2, point));
60              }
61          }
62
63          // 3. Remove triangles connected to the super-triangle
64          List<Triangle> finalMesh = new List<Triangle>();
65
66          foreach (Triangle triangle in mesh)
67          {
68              if (!triangle.SharesVertexWith(superTriangle))
69              {
70                  finalMesh.Add(triangle);
71              }
72          }
73
74          return finalMesh;
75      }
76
77      /// <summary>
78      /// Generates a super-triangle that bounds all input points.
79      /// </summary>
80      private Triangle GenerateSuperTriangle(List<Point2D> points)
81      {
82          double minX = points[0].X, maxX = points[0].X;
83          double minY = points[0].Y, maxY = points[0].Y;
84
85          foreach (Point2D p in points)
86          {
87              if (p.X < minX) minX = p.X;
88              if (p.X > maxX) maxX = p.X;
```

```
            if (p.Y < minY) minY = p.Y;
            if (p.Y > maxY) maxY = p.Y;
        }

        double dx = maxX - minX;
        double dy = maxY - minY;
        double deltaMax = dx > dy ? dx : dy;

        double midX = (minX + maxX) / 2.0;
        double midY = (minY + maxY) / 2.0;

        // Large triangle that encloses all points
        Point2D p1 = new Point2D(midX - 20 * deltaMax, midY - deltaMax);
        Point2D p2 = new Point2D(midX, midY + 20 * deltaMax);
        Point2D p3 = new Point2D(midX + 20 * deltaMax, midY - deltaMax);

        return new Triangle(p1, p2, p3);
    }
}
```

Listing F.2: Delaunay triangulation via lifting method.

```

public class LiftingTriangulator
{
    public struct Point2D
    {
        public double X;
        public double Y;

        public Point2D(double x, double y)
        {
            X = x;
            Y = y;
        }
    }

    public class Point3D
    {
        public double X;
        public double Y;
        public double Z;
        public Point2D BasePoint;

        public Point3D(double x, double y, double z, Point2D basePoint)
        {
            X = x;
            Y = y;
            Z = z;
            BasePoint = basePoint;
        }
    }

    public class HullFace3D
    {
        public Point3D[] Vertices = new Point3D[3];
        public double[] Normal = new double[3];
    }

    /// <summary>
    /// Computes the Delaunay triangulation using lifting to 3D
    /// and extraction of the lower convex hull.
    /// </summary>
    public List<HullFace3D> ComputeTriangulation(List<Point2D> points)
```

```
43      {
44          // Lift points to 3D paraboloid: z = x^2 + y^2
45          List<Point3D> liftedPoints = new List<Point3D>();
46
47          foreach (Point2D p in points)
48          {
49              double z = (p.X * p.X) + (p.Y * p.Y);
50              liftedPoints.Add(new Point3D(p.X, p.Y, z, p));
51          }
52
53          // Compute convex hull in 3D
54          List<HullFace3D> fullConvexHull = Compute3DConvexHull(
                liftedPoints);
55
56          List<HullFace3D> delaunayFaces = new List<HullFace3D>();
57
58          foreach (HullFace3D face in fullConvexHull)
59          {
60              ComputeFaceNormal(face);
61
62              // Keep only faces pointing downward
63              if (face.Normal[2] < 0.0)
64              delaunayFaces.Add(face);
65          }
66
67          return delaunayFaces;
68      }
69
70      /// Computes the normal of a triangular face.
71      private void ComputeFaceNormal(HullFace3D face)
72      {
73          Point3D p0 = face.Vertices[0];
74          Point3D p1 = face.Vertices[1];
75          Point3D p2 = face.Vertices[2];
76
77          double ux = p1.X - p0.X;
78          double uy = p1.Y - p0.Y;
79          double uz = p1.Z - p0.Z;
80
81          double vx = p2.X - p0.X;
82          double vy = p2.Y - p0.Y;
83          double vz = p2.Z - p0.Z;
84
85          double nx = (uy * vz) - (uz * vy);
86          double ny = (uz * vx) - (ux * vz);
87          double nz = (ux * vy) - (uy * vx);
88
89          double length = Math.Sqrt(nx * nx + ny * ny + nz * nz);
90
91          if (length > 1e-9)
92          {
93              face.Normal[0] = nx / length;
94              face.Normal[1] = ny / length;
95              face.Normal[2] = nz / length;
96          }
97      }
98
99      /// Convex hull computation (placeholder).
100     private List<HullFace3D> Compute3DConvexHull(List<Point3D> points)
101     {
102         return new List<HullFace3D>();
103     }
104 }
```

Appendix G

C# source code for the industrial virtual assembly case study

Listing G.1: C# code used for the update of the *Inertia Tensor Rotation*.

```
[...]
void Euler_Angles(GameObject Body, double[,] R0)
{
    double[,] R1 = new double[3, 3];
    double[,] R_213 = new double[3, 3];

    // Change of basis from a right-handed reference frame to a left-
        handed reference frame:
    // [R_213] = [M][R][M]^T
    alglib.rmatrixgemm(3, 3, 3, 1.0, M, 0, 0, 0, R0, 0, 0, 0, 0.0, R1,
        0, 0); // Matrix product: [R1] = [M][R]
    alglib.rmatrixgemm(3, 3, 3, 1.0, R1, 0, 0, 0, M, 0, 0, 1, 0.0, R_213
        , 0, 0); // Matrix product: [R_213] = [R1][M]^T

    // Euler angles extraction
    double th = -Mathf.Asin((float)R_213[1, 2]) * Mathf.Rad2Deg; //
        Theta

    if (th == 90)
    {
        // Psi
        double psi = 0;

        // Phi
        double phi = Mathf.Atan2((float)R_213[0, 1], (float)R_213[0, 0])
             * Mathf.Rad2Deg;

        // Assign Euler angles, expressed as a Quaternion, to the
            inertia tensor rotation
        Body.GetComponent<Rigidbody>().inertiaTensorRotation =
        Quaternion.Euler((float)th, (float)phi, (float)psi);
    }
    else if (th == -90)
    {
        // Psi
        double psi = 0;

        // Phi
        double phi = Mathf.Atan2((float)-R_213[0, 1], (float)R_213[0,
            0]) * Mathf.Rad2Deg;

```

```
35            // Assign Euler angles, expressed as a Quaternion, to the
                  inertia tensor rotation
36            Body.GetComponent<Rigidbody>().inertiaTensorRotation =
37            Quaternion.Euler((float)th, (float)phi, (float)psi);
38        }
39        else
40        {
41            // Phi
42            double phi = Mathf.Atan2(
43            (float)R_213[0, 2] / Mathf.Cos((float)th),
44            (float)R_213[2, 2] / Mathf.Cos((float)th)) * Mathf.Rad2Deg;
45
46            // Psi
47            double psi = Mathf.Atan2(
48            (float)R_213[1, 0] / Mathf.Cos((float)th),
49            (float)R_213[1, 1] / Mathf.Cos((float)th)) * Mathf.Rad2Deg;
50
51            // Assign Euler angles, expressed as a Quaternion, to the
                  inertia tensor rotation
52            Body.GetComponent<Rigidbody>().inertiaTensorRotation =
53            Quaternion.Euler((float)th, (float)phi, (float)psi);
54        }
55        [...]
```

Listing G.2: Computation of relative pose between pinch reference frame and grasped object.

```
1         private void DirectInteractorPosition(GameObject obj, GameObject
              hand, Transform Direct_Transf, bool isInside)
2         {
3             Collider[] Coll;
4
5             [...]
6
7             if (Coll.Length > 0)
8             {
9                 // Check whether the Direct Interactor is inside any
                      collider volume
10                foreach (var collider in Coll)
11                {
12                    // Compute closest point on collider surface to the hand
13                    Vector3 closestPoint = collider.ClosestPoint(
                          Direct_Transf.position);
14
15                    // If the closest point equals the hand position, the
                          hand is inside
16                    if (closestPoint == Direct_Transf.position)
17                    {
18                        isInside = true;
19                        break;
20                    }
21                }
22            }
23
24            // If the hand is not inside the object, compute closest surface
                  point
25            if (!isInside)
26            {
27                float minDistance = Mathf.Infinity;
28                Vector3 closestPoint = Vector3.zero;
29                Vector3 closestPointOnSurface = Vector3.zero;
30
31                // Check direct colliders
32                if (Coll.Length > 0)
33                {
```

```
34                  foreach (var collider in Coll)
35                  {
36                      closestPoint = collider.ClosestPoint(Direct_Transf.
                            position);
37                      float distance = Vector3.Distance(Direct_Transf.
                            position, closestPoint);
38
39                      // Keep the minimum distance among all colliders
40                      minDistance = Mathf.Min(minDistance, distance);
41                  }
42                  closestPointOnSurface = closestPoint;
43              }
44
45              // Translate object so that closest surface point matches
                    the hand position
46              if (minDistance < Mathf.Infinity)
47              {
48                  Vector3 translation = Direct_Transf.position -
                        closestPointOnSurface;
49                  obj.transform.position += translation;
50              }
51          }
52
53          // Compute relative pose between object and Direct Interactor (
                RIGHT HAND)
54          if (hand == Right_Hand && obj_moved_right != null)
55          {
56              DI_Start_pos_R = Direct_Transf.position;
57              DI_Start_rot_R = Direct_Transf.rotation;
58              DI_Start_scale_R = Direct_Transf.lossyScale;
59
60              obj_Start_pos_R = obj_moved_right.transform.position;
61              obj_Start_rot_R = obj_moved_right.transform.rotation;
62
63              // Compute relative position in the local frame of the
                    Direct Interactor
64              relativePosition_R =
65              DivideVectors(
66              Quaternion.Inverse(Direct_Transf.rotation) *
67              (obj_Start_pos_R - DI_Start_pos_R),
68              DI_Start_scale_R);
69          }
70      }
```

Listing G.3: C# code used to implement and integrate Wrist Trackers in the scene for the Windows platform.

```
1       [...]
2       public class XRViveTracker : XRTracker
3       {
4           /// <summary>
5           /// A <see cref="PoseControl"/> that represents information from
                 the <see cref="HTCViveTrackerProfile.grip"/> OpenXR binding
                .
6           /// </summary>
7           [Preserve, InputControl(offset = 0, aliases = new[] { "device",
                "gripPose" }, usage = "Device", noisy = true)]
8           public PoseControl devicePose { get; private set; }
9
10          /// <summary>
11          /// A [Vector3Control](xref:UnityEngine.InputSystem.Controls.
                Vector3Control) required for back compatibility with the
                XRSDK layouts. This is the device position. For the Oculus
                Touch device, this is both the grip and the pointer position
```

```
                . This value is equivalent to mapping devicePose/position.
            /// </summary>
            [Preserve, InputControl(offset = 8, alias = "gripPosition",
                noisy = true)]
            new public Vector3Control devicePosition { get; private set; }

            /// <summary>
            /// A [QuaternionControl](xref:UnityEngine.InputSystem.Controls.
                QuaternionControl) required for backwards compatibility with
                 the XRSDK layouts. This is the device orientation. For the
                Oculus Touch device, this is both the grip and the pointer
                rotation. This value is equivalent to mapping devicePose/
                rotation.
            /// </summary>
            [Preserve, InputControl(offset = 20, alias = "gripOrientation",
                noisy = true)]
            new public QuaternionControl deviceRotation { get; private set;
                }

            [Preserve, InputControl(offset = 60)]
            new public ButtonControl isTracked { get; private set; }

            [Preserve, InputControl(offset = 64)]
            new public IntegerControl trackingState { get; private set; }

            [...]
        }
    }

    [...]
```

Listing G.4: Screw tightening control with torque wrench alignment.

```
        [...]
        // Maintain Torque Wrench-Screw alignment and proceed with
            tightening
        if (alignment_End > thres_align_Avv && dist_plan_End < thres_O_plan
            && dist_ax_End < thres_O_ax && Target_End > thres_End)
        {
            // Activate feedback
            feedback_Screw.GetComponent<Renderer>().material.color = Color.
                green;

            // Debug executed only once
            if (All_Screw_TW == false)
            {
                // Compute reference unit vector for alpha angle calculation
                // (Torque Wrench x-axis projected onto the yz-plane of the
                    screw)
                float dot = Vector3.Dot(Torque_Wrench.transform.right,
                    go_triggerato.transform.right);
                // Magnitude of the projection of TW x-axis onto screw x-
                    axis (component of vector c)

                Vector3 c = dot * go_triggerato.transform.right;
                // Components of vector c (projection of TW x-axis onto
                    screw x-axis)

                vRef = (Torque_Wrench.transform.right - c);
                // Component of TW x-axis on the screw yz-plane (vector d,
                    orthogonal to c)

                vRef = vRef.normalized;
                // Normalize to obtain unit vector d
```

```
24
25                  All_Screw_TW = true;
26                  // Debug.Log($"Aligned with {go_triggerato.name}");
27              }
28
29              // Compute updated projected unit vector for alpha angle
                    calculation
30              float dot_new = Vector3.Dot(Torque_Wrench.transform.right,
                    go_triggerato.transform.right);
31              // Projection magnitude of TW x-axis onto screw x-axis (new
                    vector c magnitude)
32
33              Vector3 c_new = dot_new * go_triggerato.transform.right;
34              // Components of new vector c (projection of TW x-axis onto
                    screw x-axis)
35
36              Vector3 vNew = (Torque_Wrench.transform.right - c_new);
37              // Component of TW x-axis on the screw yz-plane (new vector d
                    orthogonal to c_new)
38
39              vNew = vNew.normalized;
40              // Normalize to obtain unit vector of new d
41
42              // Compute angle alpha between vRef and vNew using Vector3.
                    SignedAngle:
43              // Uses atan2: cosine from dot product and sine from cross
                    product
44              // alpha is positive if cross product aligns with reference
                    normal (right-hand rule)
45              float alpha = Vector3.SignedAngle(vRef, vNew, go_triggerato.
                    transform.right);
46
47              // Screw tightening phase
48              if (alpha < 0)
49              {
50                  // Screw rotation
51                  // Space.Self: local space of the GameObject
52                  // Rotate applies alpha to current transform
53                  go_triggerato.transform.Rotate(Vector3.right, alpha, Space.
                        Self);
54
55                  // Screw translation (forward motion)
56                  float avanzamento = passo / 360 * Mathf.Abs(alpha) * 15;
57                  go_triggerato.transform.position += go_triggerato.transform.
                        right * avanzamento;
58
59                  vRef = vNew;
60              }
61              else
62              {
63                  // Update reference vector
64                  vRef = vNew;
65              }
66          }
67          [...]
```

List of Figures

2.1 A general data flow architecture of a digital twin and its embedded predictive models. 8
2.2 (a) Ivan Sutherland's "Sword of Damocles" and (b) example of CAVE structure [64]. 14
2.3 (a) Outside-In and (b) Inside-Out tracking systems. 16
2.4 Vive base station 1.0 (left) and 2.0 (right). 17
2.5 Devices based on outside-in tracking. 18
2.6 Devices based on inside-out tracking. 18
2.7 Example of a NURBS surface representation [77]. 22
2.8 Example of a subdivision surface representation. 22
2.9 Example of an implicit surface representation. 23
2.10 Example of voxel grids representation [83]. 23
2.11 Example of a point-based representation. 24
2.12 Example of a polygonal mesh representation. 24
2.13 (a) Back-face culling and (b) Clockwise winding order in a left-handed reference system [22]. 25

3.1 (a) Image Target application example and (b) real 2D image target (left) and corresponding tracked features (right). 29
3.2 (a) Multi Target application example, (b) parallelepiped unwrapping and (c) real front image target (left) and corresponding tracked features (right). 30
3.3 (a) Example of a Cylinder Target application, (b) cylindrical unwrapping, and (c) real side image target (top) and corresponding tracked features (bottom). 31
3.4 Vuforia Engine rating system for visual features detected. 32
3.5 (a) Model Target application example and (b) guide views used for the tracking. 33
3.6 HTC VIVE tracker left-handed reference systems. 35
3.7 HTC VIVE Wrist tracker left-handed reference system. 36
3.8 HTC VIVE Ultimate tracker left-handed reference system. 36
3.9 Tracker application example. 37

4.1 Separating Axis Theorem for 2D OBB interaction along a candidate axis $\mathbf{s}$. 43
4.2 Application of the GJK algorithm for OBB-OBB interaction in both 2D and 3D cases. 48

4.3 Controllers of the devices in Figures 2.5 and 2.6. 52
4.4 Hand tracking joints in different framework. 53
4.5 Hand tracking application example. 54
4.6 Customized hand gestures. 55
4.7 Hand axis. 57
4.8 An example of the OAF/GAF methodology for the cylindrical grip [121]. . 59
4.9 GAF/OAF methodology adapted for cylindrical grasp [42,90]. 60
4.10 GAF/OAF methodology adapted for pinch grasp [42,90]. 61
4.11 GAF/OAF methodology adapted for spherical grasp [42]. 61
4.12 Types of haptic devices. 62
4.13 ((a)) Geomagic Touch X, ((b)) example of an application. 63
4.14 ((a)) SenseGlove Nova 2, ((b)) example of an application. 64

5.1 A general overview of DesignAR [137]. 66
5.2 AiRSculpt environment [146]. 66
5.3 Immersive and multimodal user interface of the VRAD/OpenCASCADE application [151]. 67
5.4 Pose recognition and interactive manipulation scheme. 69
5.5 Catmull-Clark algorithm applied within both the controlled diagonals framework and Delaunay triangulation. 70
5.6 Catmull–Clark algorithm adapted for triangular meshes preserving the diagonal direction when transitioning between triangular and quadrilateral meshes. 71
5.7 (a) Face indexing for each face of the base cube created in the application with the left-handed reference frame, and (b) the vertex naming order in the generic mesh, which allows the adaptation of the Catmull–Clark algorithm and the removal of not-showed vertices (red). 72
5.8 (a) Base cubes face indexing, (b) faces indexing of the new base cubes for an extrusion from F5 faces and (c) not-shown faces after the extrusion operation. 73
5.9 Snapshots from the interactive modelling experience using the adapted GAF/OAF methodology and the subdivision surfaces. Explanation of each snapshot is provided in the text. 79
5.10 Finalising the modelling with the conversion of the SubD surface (on the left) into NURBS patches (in the middle) and then a perfect hole is subtracted from the resulting B-Rep solid (on the right). The final modelling is achieved into Rhinoceros. 79

6.1 Training of an assembly procedure guided by textual instructions and visual cues (left); exploded view of the model to explore the individual components of the object (center); sectional view used to inspect the internal structure of the assembly (right) [171]. 81
6.2 AR application as a training tool in medical education [183]. 82
6.3 An example of the virtual environment of quantitative task 1. 89
6.4 An example of the virtual environment of quantitative task 2. 90
6.5 An example of the virtual environment of quantitative task 3 90

6.6 An example of the virtual environment of quantitative task 4. 91
6.7 An example of the virtual environment of quantitative task 5. 91
6.8 An example of the virtual environment of quantitative task 6. 92
6.9 An example of the virtual environment of quantitative task 7. 92
6.10 An example of the virtual environment of quantitative task 8. 93
6.11 An example of the virtual environment of quantitative task 9. 94
6.12 An example of the virtual environment of quantitative task 10. 94
6.13 An example of the virtual environment of qualitative task 11. 95
6.14 An example of the virtual environment of qualitative task 12. 95
6.15 Properties of directional lights for the 5th level of intensity 96
6.16 An example of the virtual environment of qualitative task 15. 96
6.17 KPI assessment function for the first seven quantitative tasks. 97
6.18 Mean KPI and standard deviation for Task 1 (near-field manipulation—pick and place). For pairwise statistical comparisons, see Table 6.7. 102
6.19 Mean KPI and standard deviation for Task 2 (far-field manipulation—pick and place). For pairwise statistical comparisons, see Table 6.8. 102
6.20 Mean KPI and standard deviation for Task 3 (near-field manipulation—pick, rotate and insert). For pairwise statistical comparisons, see Table 6.9. . . 103
6.21 Mean KPI and standard deviation for Task 4 (far-field manipulation—pick, rotate and insert). For pairwise statistical comparisons, see Table 6.10. . . 103
6.22 Mean KPI and standard deviation for Task 5 (two-hand dynamics). For pairwise statistical comparisons, see Table6.11. 104
6.23 Mean KPI and standard deviation for Task 6 (button interaction). For pairwise statistical comparisons, see Table 6.12. 104
6.24 Mean KPI and standard deviation for Task 7 (teleporting). For pairwise statistical comparisons, see Table 6.13. 105
6.25 Mean KPI and standard deviation for Task 8 (dynamics—pick and throw). For pairwise statistical comparisons, see Table 6.14. 105
6.26 Mean KPI and standard deviation for Task 9 (reading canvas at different distances). For pairwise statistical comparisons, see Table 6.15. 106
6.27 Mean KPI and standard deviation for Task 10 (interactive dynamics). For pairwise statistical comparisons, see Table 6.16. 106
6.28 Mean KPI and standard deviation for Task 11 (high-resolution model inspection). For pairwise statistical comparisons, see Table 6.17. 108
6.29 Mean KPI and standard deviation for Task 12 (sound listening). For pairwise statistical comparisons, see Table 6.18. 108
6.30 Mean KPI and standard deviation for Task 13 (mid-exposure tolerability). For pairwise statistical comparisons, see Table 6.19. 109
6.31 Mean KPI and standard deviation for Task 14 (ergonomics and comfort in wearing). For pairwise statistical comparisons, see Table 6.20. 109
6.32 Mean KPI and standard deviation for Task 15 (low-light environment sensibility). For pairwise statistical comparisons, see Table 6.21. 110
6.33 (b) Assembly of the Dummies on the Tray using M4×25 screws, (c) fully assembled Tray module, and (a) virtual assembly workstation. 112
6.34 (a) Real and virtual screwdriver, (b) real and virtual torque wrench. . . . 113

6.35 Possible hand configurations that trigger the customized pinch gesture. . . 114
6.36 Example of Dummy manipulation. 115
6.37 Example of alignment between Dummies and their corresponding Shadow objects. 118
6.38 (a) Position of the *Screw Start* and *Screw End* with respect to a test surface, (b) reference frame of the screw, (c) reference frame of the screwdriver, and (d) reference frame of the torque wrench. 119
6.39 (a) Spherical tolerance (left) and conical tolerance (right) for the *Screw–Screw Start* alignment, (b) cylindrical tolerance (left) and conical tolerance (right) for the *Screwdriver–Screw* alignment, (c) cylindrical tolerance (left) and conical tolerance (right) for the *Torque Wrench–Screw* alignment. . . 120
6.40 (a) Screwing conditions for tightening via torque wrench, (b) screwing conditions for tightening via screwdriver. 121
6.41 Flow diagram of the assembly procedure. 122
6.42 Example of a tightening procedure using the torque wrench. 122

7.1 Interaction among digital twin, sensors, actuators and augmented reality interface for achieving an AR digital twin. 124
7.2 Chrono-Unity examples: sandbox, mixer, earthquake and vehicle dynamics. 127
7.3 A multibody dynamic model based on sequential impulse solvers with communication ports with real actuators and sensors. 131
7.4 (a) Haptic feedback real-time/synchronous loop, (b) object manipulation through the virtual handle. 133
7.5 Flow diagram for computing virtual handle parameters and the haptic feedback perceived by the user. 134
7.6 (a) Unity left-handed reference frame of the four-bar mechanism, (b) rotational springs between the cranks and the ground. 136
7.7 (a) Manipulation of the four-bar mechanism using a haptic device in a VR environment, (b) virtual handle for motion transmission. 136
7.8 (a) Trajectory of the connecting rod point, (b) imposed motion of the crank point for force comparison between the two solvers. 137
7.9 Comparison between Chrono-Unity and sequential impulse solver of the force applied by the haptic device on the user for a prescribed motion path of the connecting rod point, with results expressed in the reference frame of Figure 7.6a. 138
7.10 (a) Unity left-handed reference frame of the slider-crank mechanism, (b) system configuration with an external resisting torque applied to the crank with respect to the ground frame. 140
7.11 (a) Interaction with the slider-crank mechanism in a VR environment using a haptic device, (b) virtual handle attached to the slider point for motion transmission. 140
7.12 Comparison between Chrono-Unity and sequential impulse solver of the force applied by the haptic device on the user for a prescribed motion path of the slider, with results expressed in the reference frame of Figure 7.10a. 141
7.13 (a) Unity left-handed reference frame of the flexible beam in the Unity environment, (b) discrete flexible beam representation. 142

7.14 (a) Deformation of the flexible beam through haptic interaction in a VR environment, (b) virtual handle used for transmitting the interaction to the multibody system. 142
7.15 Comparison between Chrono-Unity and Sequential Impulse of the haptic feedback force applied to the user during manipulation of the beam end, expressed in the reference frame of Figure 7.13a. 143
7.16 Interaction among digital shadows (based on ROM), sensors, actuators and augmented reality interface for achieving an augmented reality digital twin of a structural component. 145
7.17 Procedural steps for generating the complete reduced-order model 146
7.18 Comparison between samples produced via Random Sampling and Latin Hypercube Sampling for two generic parameters A and B, each with eight levels. 147
7.19 Example of mesh nodes position after mesh morphing phase for various configurations: (a) is the baseline mesh; (b) is a random hole diameter; (c) is the minimum hole diameter, and (d) the maximum one. 148
7.20 Singular values σ_i of the matrix $\mathbf{S}$ (a) and cumulative energy retained by the first k modes (b) for the cantilevered plate with a central hole subjected to transverse loading at the free end. 150

8.1 Communication paths between the physical and the digital twins in the training environment. The user communicates with the digital twin through the virtual reality trackers (COM Path 1) and the myoelectric sensors (COM Path 2). The digital twin, through the mathematical model, returns an updated virtual environment by means of the head-mounted display (COM Path 3). 155
8.2 Physical and digital twins of the VR training environment. The physical twin includes the kinematic sensors (trackers), the myoelectric sensors, and the head-mounted display. The digital twin is driven by the multibody dynamic model able to simulate real-time physics including collisions. . . . 156
8.3 Schematic overview of the dynamic model of the arm. The locations of the anchor points of the springs connecting the trackers and the body segment are computed during initial calibration. 157
8.4 The proposed procedure to manage the dynamic behaviour of the prosthetic arm digital twin. Both communication paths from the physical sensors go to the multibody model that updates the position and attitude of all the objects in the virtual environment, according to an accurate physics simulation. 159
8.5 An example of mesh skinning applied to a rigid-body rig for a cylindrical grasping pose. The pyramidal bodies represent the bones of the armature (rigid bodies of the simulation model), while the anatomical mesh represents the deformable virtual character. 160
8.6 Mesh vertex weights for realistic skinning of the virtual arm. Red corresponds to a weight of 1 and blue corresponds to a weight of 0. Higher weights indicate stronger coupling between mesh motion and bone motion. 160

8.7 The four hand poses and the corresponding skeleton and skinned mesh: (a) power grip, (b) three-finger grip, (c) pointing finger, and (d) flat hand. 162
8.8 VR-SHAP test implementation in the virtual environment. Tasks are divided into three groups according to difficulty level. A help guide, using virtual buttons, can be activated by the user during task execution. 163
8.9 An example of the first level of the VR-SHAP test: grasping and manipulation of a bottle. The user locates the bottle, grasps it, and places it stably on a cylindrical base. 163
8.10 An example of the second level of the VR-SHAP test: rotation of a handle. The user locates the handle, grasps it, and rotates it by 45°. 163
8.11 An example of the third level of the VR-SHAP test: manipulation of buttons. The user locates the buttons, grasps them one at a time, and releases them inside a box. 164
8.12 (a) Training for target interaction using the pointing finger posture; (b) free object manipulation environment; (c) Calibration Room used for tracker calibration and arm model alignment; (d) Hand-Move Room dedicated to familiarization with the classifier and the virtual environment. . 165
8.13 Usability analysis results on arm tracking (left) and ease of sensor placement (right). Both tracking fidelity and ease of use received high scores, confirming good overall usability. 166
8.14 Usability results for VR-SHAP tasks at different difficulty levels. First-level tasks were rated easiest due to simplicity, while second-level tasks were perceived as more difficult due to limited wrist degrees of freedom. . 167
8.15 Usability results on cybersickness (left) and help guide clarity (right). Reported sickness levels were low, and the graphical guidance system was considered highly useful. 167
8.16 Development framework and software libraries supporting the construction of the interactive augmented reality digital twin application. 169
8.17 The real system (left) and its corresponding 3D virtual digital twin model (right). 169
8.18 Communication scheme and supporting tools linking the physical twin, the digital twin, and the head-mounted display. 170
8.19 The operator wearing a HoloLens 2 headset interacting with the physical twin (left), and a sample view of the augmented scene (right). 172
8.20 Comparison between physical and digital twins through graphical overlays (left), and gesture-based interactive inspection of components (right): when the user points to the rod, a contextual panel appears displaying its system parameters (center of mass position, velocity, and acceleration). . . 172

8.21 (a) Motor-axis angular displacement driven by user input on the virtual knob; (b) comparison of driving torque computed via the Lagrangian method and the Sequential Impulse solver (time step: 1/60 s); (c) comparison between slider velocities computed using the standard multibody approach and the Sequential Impulse solver (time step: 0.01 s); (d) comparison of reaction force magnitude at the pin connecting the rod to Slider 1, computed using the standard multibody approach and the Sequential Impulse solver (time step: 0.01 s). 176
8.22 The physical system (a) and its 3D digital-twin model (b) for the considered use case; (c) the load application pointer used in the case study. . . . 178
8.23 Communication architecture and enabling tools connecting the physical twin, the digital twin, and the head-mounted display. 178
8.24 Integration of development environments and software libraries for implementing an interactive augmented-reality digital twin experience. 179
8.25 Real-time simulation in mode 1 under four different load magnitudes and application points: applied force of (a) 4.07 N, (b) 6.63 N, (c) 5.18 N, and (d) 4.66 N, with a hole diameter of 32 mm. 180
8.26 (a), (c), and (e) Real-time simulation in mode 2 with interactive variation of the central hole diameter (50, 25, and 10 mm, respectively); (b), (d), and (f) corresponding superimposed stress contour maps of the plate configurations after load application. 181
8.27 Mesh section view workflow. 182
8.28 Workflow from FEA nodal outputs to the mapped graphical mesh. 183
8.29 List of points for stress verification. 183
8.30 Accuracy-assessment test cases: load application locations and load magnitudes. 184
8.31 Example comparison of nodal stress results for case 1 obtained from the full FEM (a) and the ROM (b), both visualized using an identical colormap. 185
8.32 Example comparison of nodal displacement results for case 1 obtained fromthe full FEM (a) and the ROM (b), both visualized using an identical colormap. 185
8.33 Example of percentage error in the stress-result comparison for case 1 (a);normalized stress values obtained from FEA versus normalized predicted stress values (b); percentage error distribution across all ten test cases for von Mises stress (c) and vertical displacement (d). 187
8.34 Digital Twin architecture and communication diagram. 188
8.35 Nomenclature used for the external forces formulation, IMU reference system and finite element analysis reference system. 190
8.36 (a) Static masses mounted on the chassis, (b) cantilevered and distributed static masses, (c) IMU converted external forces, (d) external forces distributed on the suspension housings. 192
8.37 (a) CBUSH elements for damping units, (b) CBUSH elements for longitudinal and transversal constraints. 194
8.38 Integration of development environments and software libraries implemented in the case study. 195

8.39 (a) Starting connection to the server; (b), (c), and (d) displacements along the x, y, and z axes, respectively,caused by static masses mounted on the chassis. 196
8.40 Chassis displacements under external forces,measured via IMU, along (a) the y-axis and (b) the z-axis. 197

A.1 Position of a point P. 204
A.2 Planar displacement of a rigid body: homologous positions of a point P. . 205
A.3 Coordinate transformation. 206
A.4 Definition of Euler angles according to the 313 convention. 211
A.5 Notation adopted for Rodrigues parameters. 212
A.6 Cayley formula. 213
A.7 Rotation of $\mathbf{v}$ about an axis $\mathbf{u}$. 227

B.1 Unity Editor interface. 231
B.2 Collider used in Unity. 233
B.3 Input Debugger window to check whether a device is being tracked and whether its raw data are being sent to the engine. 240
B.4 Input Map of the HTC VIVE Wrist Tracker. 242
B.5 Input Map for XR devices. 243
B.6 2 different configurations of the XR Origin. 246

C.1 Vuforia Engine model target and advanced model target generation. . . . 263

D.1 First orthogonality constraint. 265
D.2 Second orthogonality constraint. 266
D.3 Spherical constraint. 267
D.4 Distance constraint. 268
D.5 First parallelism constraint. 268
D.6 Second parallelism constraint. 268
D.7 Revolute joint [303]. 269
D.8 Cylindrical joint [303]. 270
D.9 Prismatic joint [303]. 271
D.10 Spherical joint [303]. 272
D.11 Screw joint [303]. 273

E.1 (a) Iteration levels of the Catmull-Clark recursive procedure, (b) single iteration of the Catmull-Clark refinement algorithm. 274
E.2 Detailed step-by-step view of the first iteration. 275

F.1 Graphical representation of the incremental Bowyer-Watson procedure for the generation of a Delaunay triangulation from a discrete set of points in the plane. 278

F.2 Example illustrating the relationship between Delaunay triangulation and convex hull using a paraboloid lifting defined by x^2+y^2 (left). The convex hull is computed in $\mathbb{R}^3$ by mapping each point (x, y) to $(x, y, x^2 + y^2)$ (center). The resulting convex hull is projected back onto $\mathbb{R}^2$, yielding the Delaunay triangulation (right) [320]. 279

List of Tables

2.1 Technical specifications of the devices under investigation. 19

4.1 Reference configurations of hand shape and corresponding normalized values. 56
4.3 Alignment configurations. 58
4.2 Reference directions. 58

6.1 Users' list, with age, gender, and VR background experience. 87
6.2 Mapping between the proposed tasks and concrete industrial workflows. . 87
6.3 Reference time for the first seven quantitative tasks. 98
6.4 Results of Mauchly's test for the sphericity assumption and the corresponding corrections. 99
6.5 Results of RM-ANOVA: p-values and generalized η^2. 99
6.6 List of VR devices and their acronyms used throughout the tables. 101
6.7 Task 1 – Near field manipulation: pick and place. Tukey's post-hoc pairwise comparisons. 102
6.8 Task 2 – Far field manipulation: pick and place. Tukey's post-hoc pairwise comparisons. 102
6.9 Task 3 – Near field manipulation: pick, rotate and insert. Tukey's post-hoc pairwise comparisons. 103
6.10 Task 4 – Far field manipulation: pick, rotate and insert. Tukey's post-hoc pairwise comparisons. 103
6.11 Task 5 – Two hands dynamics. Tukey's post-hoc pairwise comparisons. . . 104
6.12 Task 6 – Button interaction. Tukey's post-hoc pairwise comparisons. . . . 104
6.13 Task 7 – Teleporting. Tukey's post-hoc pairwise comparisons. 105
6.14 Task 8 – Dynamics: pick and throw. Tukey's post-hoc pairwise comparisons. 105
6.15 Task 9 – Reading canvas at different distances. Tukey's post-hoc pairwise comparisons. 106
6.16 Task 10 – Interactive dynamics (catching falling spheres). Tukey's post-hoc pairwise comparisons. 106
6.17 Task 11 – Hi-res model inspection. Tukey's post-hoc pairwise comparisons. 108
6.18 Task 12 – Sound listening. Tukey's post-hoc pairwise comparisons. 108
6.19 Task 13 – Mid exposure tolerability Tukey's post-hoc pairwise comparisons. 109
6.20 Task 14 – Ergonomics and comfort in wearing. Tukey's post-hoc pairwise comparisons. 109

6.21 Task 15 – Low light environment sensibility. Tukey's post-hoc pairwise comparisons. 110
6.22 KPI mean values for each device and each task. 111

8.1 Mean time and standard deviation for the six VR-SHAP tasks. 164
8.2 Main input parameters for the multibody models 174
8.3 Peak percentage deviation between the Sequential Impulse solver and the conventional multibody solver. 177
8.4 Main geometrical and material features of the plate. 177
8.5 Stress estimation errors at six critical points, along with mean and maximum element distortion for three plate configurations. 185
8.6 Maximum percentage deviation between the ROM and the full FEM solver. 186

A.1 Multiplication rules for the set $\{1, -1, i, -i, j, -j, k, -k\}$. 220

D.1 Number of scalar equations associated with each basic constraint. 269

Listings

3.1 OpenVR Unity script to directly access the tracker poses at runtime. . . . 37
3.2 OpenXR Unity script to directly access the tracker poses at runtime. . . . 38
5.1 Implementation of the mesh generation procedure based on controlled face triangulation with front and back-face handling and indexed vertex mapping. 74
5.2 Implementation of the base control cage generation for a structured parallelepiped, using face based indexing and parametric vertex mapping. . . 74
B.1 C# script used to enable passthrough. 251
C.1 C# script used by Vuforia Engine for managing tracking and estimating the pose of the tracked image or model. 257
E.1 Simplified implementation of the Catmull-Clark subdivision algorithm. . . 276
F.1 Implementation of the Bowyer-Watson algorithm for 2D Delaunay triangulation. 279
F.2 Delaunay triangulation via lifting method. 281
G.1 C# code used for the update of the *Inertia Tensor Rotation*. 283
G.2 Computation of relative pose between pinch reference frame and grasped object. 284
G.3 C# code used to implement and integrate Wrist Trackers in the scene for the Windows platform. 285
G.4 Screw tightening control with torque wrench alignment. 286

Bibliography

[1] K. Y. H. Lim, P. Zheng, and C.-H. Chen, "A state-of-the-art survey of digital twin: techniques, engineering product lifecycle management and business innovation perspectives," *Journal of intelligent manufacturing*, vol. 31, no. 6, pp. 1313–1337, 2020.

[2] J. M. Hunde, T. S. Ochono, D. Senevirathne, D. D. Eneyew, G. T. Bitsuamlak, M. A. Capretz, and K. Grolinger, "Data-driven and physics-based modeling approaches and their integration in building digital twins: A systematic review," *Journal of Building Engineering*, p. 114214, 2025.

[3] D. Paes, E. Arantes, and J. Irizarry, "Immersive environment for improving the understanding of architectural 3d models: Comparing user spatial perception between immersive and traditional virtual reality systems," *Automation in construction*, vol. 84, pp. 292–303, 2017.

[4] C.-C. Dosoftei, "Simulation power vs. immersive capabilities: Enhanced understanding and interaction with digital twin of a mechatronic system," *Applied Sciences*, vol. 13, no. 11, p. 6463, 2023.

[5] M. Grieves, "Digital twin of physical systems: opportunities and challenges," in *Proceedings of the ASME 2002 International Mechanical Engineering Congress and Exposition*, pp. 17–22, 2002.

[6] M. Grieves *et al.*, "Digital twin: manufacturing excellence through virtual factory replication," *White paper*, vol. 1, no. 2014, pp. 1–7, 2014.

[7] M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," in *Transdisciplinary perspectives on complex systems: New findings and approaches*, pp. 85–113, Springer, 2016.

[8] D. Jones, C. Snider, A. Nassehi, J. Yon, and B. Hicks, "Characterising the digital twin: A systematic literature review," *CIRP journal of manufacturing science and technology*, vol. 29, pp. 36–52, 2020.

[9] C.-J. Liang, W. McGee, C. Menassa, and V. Kamat, "Bi-directional communication bridge for state synchronization between digital twin simulations and physical construction robots," in *Proceedings of the International Symposium on Automation and Robotics in Construction (IAARC)*, 2020.

[10] F. Tao, Q. Qi, A. Liu, and A. Kusiak, "Data-driven smart manufacturing," *Journal of manufacturing systems*, vol. 48, pp. 157–169, 2018.

[11] Q. Qi and F. Tao, "Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison," *IEEE access*, vol. 6, pp. 3585–3593, 2018.

[12] Z. Huang, Y. Shen, J. Li, M. Fey, and C. Brecher, "A survey on ai-driven digital twins in industry 4.0: Smart manufacturing and advanced robotics," *Sensors*, vol. 21, no. 19, p. 6340, 2021.

[13] J. Fitzgerald, C. Gomes, and P. G. Larsen, *The engineering of digital twins.* Springer, 2024.

[14] M. Cirelli, P. Canonico, A. Cellupica, and P. P. Valentini, "Reduced-order models and augmented reality for real-time interactive structural digital twin exploration and interrogation," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 19, no. 10, pp. 7263–7281, 2025.

[15] D. A. Paternina-Verona, O. E. Coronado-Hernández, V. S. Fuertes-Miquel, M. Saba, and H. M. Ramos, "Digital twin based on cfd modelling for analysis of two-phase flows during pipeline filling–emptying procedures," *Applied Sciences*, vol. 15, no. 5, p. 2643, 2025.

[16] M. Cirelli, A. Cellupica, P. Canonico, and P. P. Valentini, "Impulse dynamics and augmented reality for real-time interactive digital twin exploration and interrogation," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 18, no. 2, pp. 929–941, 2024.

[17] Z.-Q. Qu, *Model order reduction techniques with applications in finite element analysis: with applications in finite element analysis.* Springer Science & Business Media, 2004.

[18] A. Quarteroni, A. Manzoni, and F. Negri, *Reduced basis methods for partial differential equations: an introduction.* Springer, 2015.

[19] G. Zhou, C. Zhang, Z. Li, K. Ding, and C. Wang, "Knowledge-driven digital twin manufacturing cell towards intelligent manufacturing," *International journal of production research*, vol. 58, no. 4, pp. 1034–1051, 2020.

[20] M. Soori, B. Arezoo, and R. Dastres, "Digital twin for smart manufacturing, a review," *Sustainable Manufacturing and Service Economics*, vol. 2, p. 100017, 2023.

[21] N. Anwer, R. Stark, F. Tao, and J. A. Erkoyuncu, "Developing and leveraging digital twins in engineering design," *CIRP Annals*, vol. 74, no. 2, pp. 843–868, 2025.

[22] A. Cellupica, M. Tristano, M. Cirelli, P. P. Valentini, and F. Naets, "A reduced-order model-driven augmented reality digital twin for real-time vehicle chassis displacement monitoring," in *Proceedings of the International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS)*, 2025.

[23] E. J. Tuegel, A. R. Ingraffea, T. G. Eason, and S. M. Spottswood, "Reengineering aircraft structural life prediction using a digital twin," *International Journal of Aerospace Engineering*, vol. 2011, no. 1, p. 154798, 2011.

[24] O. Piñal and A. Arguelles, "Mixed reality and digital twins for astronaut training," *Acta Astronautica*, vol. 219, pp. 376–391, 2024.

[25] J. Corral-Acero, F. Margara, M. Marciniak, C. Rodero, F. Loncaric, Y. Feng, A. Gilbert, J. F. Fernandes, H. A. Bukhari, A. Wajdan, *et al.*, "The 'digital twin'to enable the vision of precision cardiology," *European heart journal*, vol. 41, no. 48, pp. 4556–4564, 2020.

[26] T. Sun, X. He, and Z. Li, "Digital twin in healthcare: Recent updates and challenges," *Digital health*, vol. 9, p. 20552076221149651, 2023.

[27] M. Jafari, A. Kavousi-Fard, T. Chen, and M. Karimi, "A review on digital twin technology in smart grid, transportation system and smart city: Challenges and future," *IEEe Access*, vol. 11, pp. 17471–17484, 2023.

[28] A. Aghazadeh Ardebili, M. Zappatore, A. I. H. A. Ramadan, A. Longo, and A. Ficarella, "Digital twins of smart energy systems: a systematic literature review on enablers, design, management and computational challenges," *Energy Informatics*, vol. 7, no. 1, p. 94, 2024.

[29] Z. Song, C. M. Hackl, A. Anand, A. Thommessen, J. Petzschmann, O. Kamel, R. Braunbehrens, A. Kaifel, C. Roos, and S. Hauptmann, "Digital twins for the future power system: An overview and a future perspective," *Sustainability*, vol. 15, no. 6, p. 5259, 2023.

[30] F. Tao, H. Zhang, A. Liu, and A. Y. Nee, "Digital twin in industry: State-of-the-art," *IEEE Transactions on industrial informatics*, vol. 15, no. 4, pp. 2405–2415, 2018.

[31] M. Hankel and B. Rexroth, "The reference architectural model industrie 4.0 (rami 4.0)," *Zvei*, vol. 2, no. 2, pp. 4–9, 2015.

[32] C. Wagner, J. Grothoff, U. Epple, R. Drath, S. Malakuti, S. Grüner, M. Hoffmeister, and P. Zimermann, "The role of the industry 4.0 asset administration shell and the digital twin during the life cycle of a plant," in *2017 22nd IEEE international conference on emerging technologies and factory automation (ETFA)*, pp. 1–8, IEEE, 2017.

[33] X. Xu, Y. Lu, B. Vogel-Heuser, and L. Wang, "Industry 4.0 and industry 5.0—inception, conception and perception," *Journal of manufacturing systems*, vol. 61, pp. 530–535, 2021.

[34] B. Wang, H. Zhou, X. Li, G. Yang, P. Zheng, C. Song, Y. Yuan, T. Wuest, H. Yang, and L. Wang, "Human digital twin in the context of industry 5.0," *Robotics and Computer-Integrated Manufacturing*, vol. 85, p. 102626, 2024.

[35] F. Merienne, "Human factors consideration in the interaction process with virtual environment," 2010.

[36] P. P. Valentini, "Enhancing user role in augmented reality interactive simulations," in *Human Factors in Augmented Reality Environments*, pp. 233–256, Springer, 2012.

[37] S. M. Hasan, K. Lee, D. Moon, S. Kwon, S. Jinwoo, and S. Lee, "Augmented reality and digital twin system for interaction with construction machinery," *Journal of Asian Architecture and Building Engineering*, vol. 21, no. 2, pp. 564–574, 2022.

[38] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre, "Recent advances in augmented reality," *IEEE computer graphics and applications*, vol. 21, no. 6, pp. 34–47, 2002.

[39] T.-C. Hsieh and S. Smith, "An innovative ar-assisted self-adjusting digital twin for speed control of a servomechanism," *Computers & Industrial Engineering*, vol. 193, p. 110328, 2024.

[40] L. Mariti and P. P. Valentini, "Efficiency and precise interaction for multibody simulations in augmented reality," in *Multibody Dynamics: Computational Methods and Applications*, pp. 173–192, Springer, 2013.

[41] A. Schäfer, G. Reis, and D. Stricker, "A survey on synchronous augmented, virtual, andmixed reality remote collaboration systems," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–27, 2022.

[42] P. P. Valentini, "Natural interface for interactive virtual assembly in augmented reality using leap motion controller," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 12, no. 4, pp. 1157–1165, 2018.

[43] M. Kim and J. Y. Lee, "Touch and hand gesture-based interactions for directly manipulating 3d virtual objects in mobile augmented reality," *Multimedia Tools and Applications*, vol. 75, no. 23, pp. 16529–16550, 2016.

[44] Y. Cai, Y. Wang, and M. Burnett, "Using augmented reality to build digital twin for reconfigurable additive manufacturing system," *Journal of Manufacturing Systems*, vol. 56, pp. 598–604, 2020.

[45] A. Künz, S. Rosmann, E. Loria, and J. Pirker, "The potential of augmented reality for digital twins: A literature review," in *2022 IEEE conference on virtual reality and 3D user interfaces (VR)*, pp. 389–398, IEEE, 2022.

[46] O. Bongomin, M. C. Mwape, N. S. Mpofu, B. K. Bahunde, R. Kidega, I. L. Mpungu, G. Tumusiime, C. A. Owino, Y. M. Goussongtogue, A. Yemane, *et al.*, "Digital twin technology advancing industry 4.0 and industry 5.0 across sectors," *Results in Engineering*, vol. 26, p. 105583, 2025.

[47] C. Andrews, M. K. Southworth, J. N. Silva, and J. R. Silva, "Extended reality in medical practice," *Current treatment options in cardiovascular medicine*, vol. 21, no. 4, p. 18, 2019.

[48] M. J. Liberatore and W. P. Wagner, "Virtual, mixed, and augmented reality: a systematic review for immersive systems research," *Virtual Reality*, vol. 25, no. 3, pp. 773–799, 2021.

[49] M. Slater and M. V. Sanchez-Vives, "Enhancing our lives with immersive virtual reality," *Frontiers in Robotics and AI*, vol. 3, p. 74, 2016.

[50] M. Billinghurst, A. Clark, and G. Lee, "A survey of augmented reality," *Foundations and Trends® in Human—Computer Interaction*, vol. 8, no. 2-3, pp. 73–272, 2015.

[51] M. Speicher, B. D. Hall, and M. Nebeling, "What is mixed reality?," in *Proceedings of the 2019 CHI conference on human factors in computing systems*, pp. 1–15, 2019.

[52] C. Wheatstone, "Contributions to the physiology of vision," *Philosoph Trans of the Royal Soc*, vol. 128, pp. 371–394, 1838.

[53] A. Artaud, "The theatre and its double, trans," *Mary Caroline Richards (New York: Grove, 1958)*, vol. 124, 1958.

[54] M. Heilig, "Sensorama simulator, us patent no. 3050870," *US Patent and Trademark Office. https://patents. google. com/patent/US3050870A/en*, 1962.

[55] I. E. Sutherland, "A head-mounted three dimensional display," in *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pp. 757–764, 1968.

[56] M. W. Krueger, T. Gionfriddo, and K. Hinrichsen, "Videoplace—an artificial reality," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 35–40, 1985.

[57] A. Lippman, "Movie-maps: An application of the optical videodisc to computer graphics," *Acm Siggraph Computer Graphics*, vol. 14, no. 3, pp. 32–42, 1980.

[58] T. A. Furness III, "The super cockpit and its human factors challenges," in *Proceedings of the human factors society annual meeting*, vol. 30, pp. 48–52, SAGE Publications Sage CA: Los Angeles, CA, 1986.

[59] S. S. Fisher, M. McGreevy, J. Humphries, and W. Robinett, "Virtual environment display system," in *Proceedings of the 1986 workshop on Interactive 3D graphics*, pp. 77–87, 1987.

[60] S. S. Fisher, "Virtual interface environment," *Space Station Human Factors Research Review. Volume 4: Inhouse Advanced Development and Research*, 1988.

[61] J. Lanier, "Virtual reality: The promise of the future," *Interactive Learning International*, vol. 8, no. 4, pp. 275–279, 1992.

[62] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti, "Surround-screen projection-based virtual reality: the design and implementation of the cave," in *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pp. 51–58, 2023.

[63] C. Cruz-Neira, *Virtual reality based on multiple projection screens: The CAVE and its applications to computational science and engineering.* University of Illinois at Chicago, 1995.

[64] F. Beji, W. d. P. Ferreira, I. Pivotto Dabat, and V. Matias, "Cave automatic virtual environment technology: A patent analysis," *Engineering Proceedings*, vol. 89, no. 1, p. 9, 2025.

[65] P. C. Thomas and W. David, "Augmented reality: An application of heads-up display technology to manual manufacturing processes," in *Hawaii international conference on system sciences*, vol. 2, pp. 659–669, ACM SIGCHI Bulletin, 1992.

[66] P. Milgram and F. Kishino, "A taxonomy of mixed reality visual displays," *IEICE TRANSACTIONS on Information and Systems*, vol. 77, no. 12, pp. 1321–1329, 1994.

[67] H. Kato and M. Billinghurst, "Marker tracking and hmd calibration for a video-based augmented reality conferencing system," in *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, pp. 85–94, IEEE, 1999.

[68] R. T. Azuma, "A survey of augmented reality," *Presence: teleoperators & virtual environments*, vol. 6, no. 4, pp. 355–385, 1997.

[69] T. Starner, "Project glass: An extension of the self," *IEEE Pervasive Computing*, vol. 12, no. 2, pp. 14–16, 2013.

[70] P. A. Rauschnabel, A. Brem, and B. S. Ivens, "Who will buy smart glasses? empirical results of two pre-market-entry studies on the role of personality in individual awareness and intended adoption of google glass wearables," *Computers in Human Behavior*, vol. 49, pp. 635–647, 2015.

[71] B. C. Kress and I. Chatterjee, "Waveguide combiners for mixed reality headsets: a nanophotonics design perspective," *Nanophotonics*, vol. 10, no. 1, pp. 41–74, 2020.

[72] J. P. Rolland and H. Hua, "Head-mounted display systems," *Encyclopedia of optical engineering*, vol. 2, pp. 1–14, 2005.

[73] R. W. Fisher, "Head-mounted projection display system featuring beam splitter and method of making same," Nov. 5 1996. US Patent 5,572,229.

[74] T. Akenine-Moller, E. Haines, and N. Hoffman, *Real-time rendering.* AK Peters/crc Press, 2019.

[75] M. E. Mortenson, *Geometric modeling.* John Wiley & Sons, Inc., 1997.

[76] L. Piegl and W. Tiller, *The NURBS book.* Springer Science & Business Media, 2012.

[77] H. Nguyen-Xuan, T. Hoang, and V. P. Nguyen, "An isogeometric analysis for elliptic homogenization problems," *Computers & Mathematics with Applications*, vol. 67, no. 9, pp. 1722–1741, 2014.

[78] W. Ma, "Subdivision surfaces for cad—an overview," *Computer-Aided Design*, vol. 37, no. 7, pp. 693–709, 2005.

[79] J. Bloomenthal and C. Bajaj, *Introduction to implicit surfaces*. Morgan Kaufmann, 1997.

[80] C. Xu, G. Sun, and R. Liang, "A survey of volume visualization techniques for feature enhancement," *Visual Informatics*, vol. 5, no. 3, pp. 70–81, 2021.

[81] A. E. Kaufman and K. Mueller, "Overview of volume rendering.," *The visualization handbook*, vol. 7, pp. 127–174, 2005.

[82] J. Beyer, M. Hadwiger, and H. Pfister, "State-of-the-art in gpu-based large-scale volume visualization," in *Computer Graphics Forum*, vol. 34, pp. 13–37, Wiley Online Library, 2015.

[83] A. O. Aremu, J. Brennan-Craddock, A. Panesar, I. Ashcroft, R. J. Hague, R. D. Wildman, and C. Tuck, "A voxel-based method of constructing and skinning conformal and functionally graded lattice structures suitable for additive manufacturing," *Additive Manufacturing*, vol. 13, pp. 1–13, 2017.

[84] L. Kobbelt and M. Botsch, "A survey of point-based techniques in computer graphics," *Computers & Graphics*, vol. 28, no. 6, pp. 801–814, 2004.

[85] L. Wang, X. Shi, and Y. Liu, "Foveated rendering: A state-of-the-art survey," *Computational visual media*, vol. 9, no. 2, pp. 195–228, 2023.

[86] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," in *Computer graphics forum*, vol. 26, pp. 80–113, Wiley Online Library, 2007.

[87] P. S. Heckbert, "Survey of texture mapping," *IEEE computer graphics and applications*, vol. 6, no. 11, pp. 56–67, 2007.

[88] G. Wong and J. Wang, *Real-Time Rendering*. CRC Press, 2013.

[89] J. D. Foley, *Computer graphics: principles and practice*, vol. 12110. Addison-Wesley Professional, 1996.

[90] A. Cellupica, M. Cirelli, O. Giannini, and P. P. Valentini, "Interactive modelling in augmented reality with subdivision surfaces and advanced user gesture recognition," *Applied Sciences*, vol. 14, no. 24, p. 11873, 2024.

[91] E. Lengyel, *Mathematics for 3D game programming and computer graphics*. Course Technology Press, 2011.

[92] E. G. Hemingway and O. M. O'Reilly, "Perspectives on euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments," *Multibody system dynamics*, vol. 44, no. 1, pp. 31–56, 2018.

[93] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014.

[94] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Ep n p: An accurate o (n) solution to the p n p problem," *International journal of computer vision*, vol. 81, no. 2, pp. 155–166, 2009.

[95] A. S. Shewail, N. A. Elsayed, and H. H. Zayed, "Survey of indoor tracking systems using augmented reality," *IAES International Journal of Artificial Intelligence*, vol. 12, no. 1, p. 402, 2023.

[96] L. Chen, *On-the-fly dense 3D surface reconstruction for geometry-aware augmented reality.* PhD thesis, Bournemouth University, 2019.

[97] A. I. Comport, É. Marchand, and F. Chaumette, "A real-time tracker for markerless augmented reality," in *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.*, pp. 36–45, IEEE, 2003.

[98] E. Marchand, H. Uchiyama, and F. Spindler, "Pose estimation for augmented reality: a hands-on survey," *IEEE transactions on visualization and computer graphics*, vol. 22, no. 12, pp. 2633–2651, 2015.

[99] I. Millington, *Game physics engine development.* CRC Press, 2007.

[100] L. Mariti, N. P. Belfiore, E. Pennestrì, and P. P. Valentini, "Comparison of solution strategies for multibody dynamics equations," *International Journal for Numerical Methods in Engineering*, vol. 88, no. 7, pp. 637–656, 2011.

[101] A. Schmitt and J. Bender, "Impulse-based dynamic simulation of multibody systems: numerical comparison with standard methods," *Proc. automation of discrete production engineering*, pp. 324–329, 2005.

[102] V. Cristea, F. Pop, C. Dobre, and A. Costan, "Distributed architectures for event-based systems," in *Reasoning in event-based distributed systems*, pp. 11–45, Springer, 2011.

[103] C. Castejón, "Nonlinear phenomena of contact in multibody systems dynamics: a review," *Nonlinear Dynamics*, 2021.

[104] F. Pfeiffer and C. Glocker, *Multibody dynamics with unilateral contacts*, vol. 421. Springer science & business media, 2000.

[105] G. v. d. Bergen, "Efficient collision detection of complex deformable models using aabb trees," *Journal of graphics tools*, vol. 2, no. 4, pp. 1–13, 1997.

[106] S. A. Gottschalk, *Collision queries using oriented bounding boxes.* The University of North Carolina at Chapel Hill, 2000.

[107] X. Weiping, Z. Qing, and Z. Yeting, "Real-time collision detection algorithm using capsule to model 3d city," *Wuhan Daxue Xuebao (Xinxi Kexue Ban)/Geomatics and Information Science of Wuhan University*, vol. 34, no. 9, pp. 1030–1034, 2009.

[108] J. T. Klosowski, M. Held, J. S. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-dops," *IEEE transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21–36, 1998.

[109] C. Ericson, *Real-time collision detection.* Crc Press, 2004.

[110] S. Gottschalk, "Separating axis theorem," 1996.

[111] K. Border, "Separating hyperplane theorems," *Notes, California Institute of Technology. www. hss. caltech. edu/˜ kcb/Notes/SeparatingHyperplane. pdf*, 2010.

[112] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 171–180, 1996.

[113] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[114] W. Cox, L. While, and M. Reynolds, "A review of methods to compute minkowski operations for geometric overlap detection," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 8, pp. 3377–3396, 2020.

[115] D. E. Stewart and J. C. Trinkle, "An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction," *International Journal for Numerical Methods in Engineering*, vol. 39, no. 15, pp. 2673–2691, 1996.

[116] K. G. Murty and F.-T. Yu, *Linear complementarity, linear and nonlinear programming*, vol. 3. Heldermann Berlin, 1988.

[117] H. Goldstein, C. P. Poole, J. Safko, *et al.*, *Classical mechanics*, vol. 2. Addison-wesley Reading, MA, 1950.

[118] B. V. Mirtich, *Impulse-based dynamic simulation of rigid body systems.* University of California, Berkeley, 1996.

[119] P. P. Valentini and E. Pezzuti, "Accuracy in fingertip tracking using leap motion controller for interactive virtual applications," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 11, no. 3, pp. 641–650, 2017.

[120] T. Piumsomboon, A. Clark, M. Billinghurst, and A. Cockburn, "User-defined gestures for augmented reality," in *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pp. 955–960, 2013.

[121] A. Cellupica, M. Cirelli, G. Saggio, E. Gruppioni, and P. P. Valentini, "An interactive digital-twin model for virtual reality environments to train in the use of a sensorized upper-limb prosthesis," *Algorithms*, vol. 17, no. 1, p. 35, 2024.

[122] P. P. Valentini, "Interactive virtual assembling in augmented reality," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 3, no. 2, pp. 109–119, 2009.

[123] C. B. Phillips, J. Zhao, and N. I. Badler, "Interactive real-time articulated figure manipulation using multiple kinematic constraints," in *Proceedings of the 1990 Symposium on interactive 3D Graphics*, pp. 245–250, 1990.

[124] E. J. Haug, *Computer aided kinematics and dynamics of mechanical systems*, vol. 1. Allyn and Bacon Boston, 1989.

[125] A. Adel, "Future of industry 5.0 in society: human-centric solutions, challenges and prospective research areas," *Journal of cloud computing*, vol. 11, no. 1, p. 40, 2022.

[126] J. Pizoń and A. Gola, "Human–machine relationship—perspective and future roadmap for industry 5.0 solutions," *Machines*, vol. 11, no. 2, p. 203, 2023.

[127] S. H. Choi, M. Kim, and J. Y. Lee, "Smart and user-centric manufacturing information recommendation using multimodal learning to support human-robot collaboration in mixed reality environments," *Robotics and Computer-Integrated Manufacturing*, vol. 91, p. 102836, 2025.

[128] C. Liu, Z. Zhang, D. Tang, Q. Nie, L. Zhang, and J. Song, "A mixed perception-based human-robot collaborative maintenance approach driven by augmented reality and online deep reinforcement learning," *Robotics and Computer-Integrated Manufacturing*, vol. 83, p. 102568, 2023.

[129] L. Gualtieri, M. Öhler, A. Revolti, and P. Dallasega, "A visual management and augmented-reality-based training module for the enhancement of short and long-term procedural knowledge retention in complex machinery setup," *Computers & Industrial Engineering*, vol. 196, p. 110478, 2024.

[130] R. Palmarini, J. A. Erkoyuncu, R. Roy, and H. Torabmostaedi, "A systematic review of augmented reality applications in maintenance," *Robotics and Computer-Integrated Manufacturing*, vol. 49, pp. 215–228, 2018.

[131] A. C. P. Martins, I. R. Castellano, K. M. L. C. Júnior, J. M. F. de Carvalho, F. G. Bellon, D. S. de Oliveira, and J. C. L. Ribeiro, "Bim-based mixed reality application for bridge inspection," *Automation in Construction*, vol. 168, p. 105775, 2024.

[132] C. Chen, J. Hong, and S. Wang, "Automated positioning of 3d virtual scene in ar-based assembly and disassembly guiding system," *The International Journal of Advanced Manufacturing Technology*, vol. 76, no. 5, pp. 753–764, 2015.

[133] R. Rosati, P. Senesi, B. Lonzi, A. Mancini, and M. Mandolini, "An automated cad-to-xr framework based on generative ai and shrinkwrap modelling for a user-centred design approach," *Advanced Engineering Informatics*, vol. 62, p. 102848, 2024.

[134] D. Kim, J. Park, and K. H. Ko, "Development of an ar based method for augmentation of 3d cad data onto a real ship block image," *Computer-Aided Design*, vol. 98, pp. 1–11, 2018.

[135] S. M. Feeman, L. B. Wright, and J. L. Salmon, "Exploration and evaluation of cad modeling in virtual reality," *Computer-Aided Design and Applications*, vol. 15, no. 6, pp. 892–904, 2018.

[136] D. Hecht, M. Reiner, and G. Halevy, "Multimodal virtual environments: response times, attention, and presence," *Presence: Teleoperators and virtual environments*, vol. 15, no. 5, pp. 515–523, 2006.

[137] P. Reipschläger and R. Dachselt, "Designar: Immersive 3d-modeling combining augmented reality with interactive displays," in *Proceedings of the 2019 ACM international conference on interactive surfaces and spaces*, pp. 29–41, 2019.

[138] G.-P. Bonneau and S. Hahmann, "3d sketching in immersive environments: Shape from disordered ribbon strokes," *Computers & Graphics*, vol. 123, p. 103978, 2024.

[139] E. Rosales, J. Rodriguez, and A. Sheffer, "Surfacebrush: from virtual reality drawings to manifold surfaces," *arXiv preprint arXiv:1904.12297*, 2019.

[140] R. Arora, R. H. Kazi, F. Anderson, T. Grossman, K. Singh, and G. W. Fitzmaurice, "Experimental evaluation of sketching on surfaces in vr.," in *CHI*, vol. 17, pp. 5643–5654, 2017.

[141] J. Kang, K. Zhong, S. Qin, H. Wang, and D. Wright, "Instant 3d design concept generation and visualization by real-time hand gesture recognition," *Computers in Industry*, vol. 64, no. 7, pp. 785–797, 2013.

[142] Y. Jiang, C. Zhang, H. Fu, A. Cannavò, F. Lamberti, H. Y. Lau, and W. Wang, "Handpainter-3d sketching in vr with hand-based physical proxy," in *Proceedings of the 2021 CHI conference on human factors in computing systems*, pp. 1–13, 2021.

[143] T. A. Galyean and J. F. Hughes, "Sculpting: An interactive volumetric modeling technique," *ACM SIGGRAPH Computer Graphics*, vol. 25, no. 4, pp. 267–274, 1991.

[144] J. Dorman and A. Rockwood, "Surface design using hand motion with smoothing," *Computer-Aided Design*, vol. 33, no. 5, pp. 389–402, 2001.

[145] X. Zhu and Y. Yang, "Interactive mesh sculpting with arbitrary topologies in head-mounted vr environments," *Mathematics*, vol. 12, no. 15, p. 2428, 2024.

[146] S.-A. Jang, H.-i. Kim, W. Woo, and G. Wakefield, "Airsculpt: A wearable augmented reality 3d sculpting system," in *International Conference on Distributed, Ambient, and Pervasive Interactions*, pp. 130–141, Springer, 2014.

[147] M. Attene, M. Campen, and L. Kobbelt, "Polygon mesh repairing: An application perspective," *ACM Computing Surveys (CSUR)*, vol. 45, no. 2, pp. 1–33, 2013.

[148] F. D. Ix, H. Qin, and A. Kaufman, "A novel haptics-based interface and sculpting system for physics-based geometric design," *Computer-Aided Design*, vol. 33, no. 5, pp. 403–420, 2001.

[149] P. P. Valentini and M. E. Biancolini, "Interactive sculpting using augmented-reality, mesh morphing, and force feedback: Force-feedback capabilities in an augmented reality environment," *IEEE Consumer Electronics Magazine*, vol. 7, no. 2, pp. 83–90, 2018.

[150] B. P. Perles and J. M. Vance, "Interactive virtual tools for manipulating nurbs surfaces in a virtual environment," *J. Mech. Des.*, vol. 124, no. 2, pp. 158–163, 2002.

[151] P. Bourdot, T. Convard, F. Picon, M. Ammi, D. Touraine, and J.-M. Vézien, "Vr–cad integration: Multimodal immersive interaction and advanced haptic paradigms for implicit edition of cad models," *Computer-Aided Design*, vol. 42, no. 5, pp. 445–461, 2010.

[152] J. Butterworth, A. Davidson, S. Hench, and M. T. Olano, "3dm: A three dimensional modeler using a head-mounted display," in *Proceedings of the 1992 symposium on Interactive 3D graphics*, pp. 135–138, 1992.

[153] P. Benkő, R. R. Martin, and T. Várady, "Algorithms for reverse engineering boundary representation models," *Computer-Aided Design*, vol. 33, no. 11, pp. 839–851, 2001.

[154] C. S. Geometry, "Constructive solid geometry," 2008.

[155] A. Malik, H. Lhachemi, and R. Shorten, "A cyber-physical system to design 3d models using mixed reality technologies and deep learning for additive manufacturing," *PloS one*, vol. 18, no. 7, p. e0289207, 2023.

[156] M. Fuge, M. E. Yumer, G. Orbay, and L. B. Kara, "Conceptual design and modification of freeform surfaces using dual shape representations in augmented reality environments," *Computer-Aided Design*, vol. 44, no. 10, pp. 1020–1032, 2012.

[157] V. I. Pavlovic, R. Sharma, and T. S. Huang, "Visual interpretation of hand gestures for human-computer interaction: A review," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 19, no. 7, pp. 677–695, 2002.

[158] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey," *Artificial intelligence review*, vol. 43, no. 1, pp. 1–54, 2015.

[159] P. P. Valentini, "Natural interface in augmented reality interactive simulations," *Virtual and Physical Prototyping*, vol. 7, no. 2, pp. 137–151, 2012.

[160] H. Zhang and K. E. Hoff III, "Fast backface culling using normal masks," in *Proceedings of the 1997 symposium on Interactive 3D graphics*, pp. 103–ff, 1997.

[161] A. C. Boud, D. J. Haniff, C. Baber, and S. Steiner, "Virtual reality and augmented reality as a training tool for assembly tasks," in *1999 IEEE international conference on information visualization (Cat. No. PR00210)*, pp. 32–36, IEEE, 1999.

[162] F. Bellalouna, "Virtual-reality-based approach for cognitive design-review and fmea in the industrial and manufacturing engineering," in *2019 10th IEEE international conference on cognitive infocommunications (CogInfoCom)*, pp. 41–46, IEEE, 2019.

[163] S. Jayaram, H. I. Connacher, and K. W. Lyons, "Virtual assembly using virtual reality techniques," *Computer-aided design*, vol. 29, no. 8, pp. 575–584, 1997.

[164] C. Qiu, S. Zhou, Z. Liu, Q. Gao, and J. Tan, "Digital assembly technology based on augmented reality and digital twins: a review," *Virtual Reality & Intelligent Hardware*, vol. 1, no. 6, pp. 597–610, 2019.

[165] T. W. Thurner, E. Dyachenko, and A. Mironenko, "Diffusion of new technologies in medical research: The case of virtual reality," *Health and Technology*, vol. 10, no. 5, pp. 1221–1227, 2020.

[166] A. H. Behzadan and V. R. Kamat, "Visualization of construction graphics in outdoor augmented reality," in *Proceedings of the Winter Simulation Conference, 2005.*, pp. 7–pp, IEEE, 2005.

[167] F. Bellalouna, "New approach for industrial training using virtual reality technology," *Procedia CIRP*, vol. 93, pp. 262–267, 2020.

[168] V. Di Pasquale, P. Cutolo, C. Esposito, B. Franco, R. Iannone, and S. Miranda, "Virtual reality for training in assembly and disassembly tasks: a systematic literature review," *Machines*, vol. 12, no. 8, p. 528, 2024.

[169] B. Xie, H. Liu, R. Alghofaili, Y. Zhang, Y. Jiang, F. D. Lobo, C. Li, W. Li, H. Huang, M. Akdere, *et al.*, "A review on virtual reality skill training applications," *Frontiers in Virtual Reality*, vol. 2, p. 645153, 2021.

[170] J. J. Roldán, E. Crespo, A. Martín-Barrio, E. Peña-Tapia, and A. Barrientos, "A training system for industry 4.0 operators in complex assemblies based on virtual reality and process mining," *Robotics and computer-integrated manufacturing*, vol. 59, pp. 305–316, 2019.

[171] J. Wolfartsberger, R. Zimmermann, G. Obermeier, and D. Niedermayr, "Analyzing the potential of virtual reality-supported training for industrial assembly tasks," *Computers in Industry*, vol. 147, p. 103838, 2023.

[172] P. Fraga-Lamas, T. M. Fernandez-Carames, O. Blanco-Novoa, and M. A. Vilar-Montesinos, "A review on industrial augmented reality systems for the industry 4.0 shipyard," *Ieee Access*, vol. 6, pp. 13358–13375, 2018.

[173] Z. Wang, X. Bai, S. Zhang, M. Billinghurst, W. He, P. Wang, W. Lan, H. Min, and Y. Chen, "A comprehensive review of augmented reality-based instruction in manual assembly, training and repair," *Robotics and Computer-Integrated Manufacturing*, vol. 78, p. 102407, 2022.

[174] Y. Itoh, T. Langlotz, J. Sutton, and A. Plopski, "Towards indistinguishable augmented reality: A survey on optical see-through head-mounted displays," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–36, 2021.

[175] J. Blattgerste, B. Strenge, P. Renner, T. Pfeiffer, and K. Essig, "Comparing conventional and augmented reality instructions for manual assembly tasks," in *Proceedings of the 10th international conference on pervasive technologies related to assistive environments*, pp. 75–82, 2017.

[176] Y. Yin, P. Zheng, C. Li, and L. Wang, "A state-of-the-art survey on augmented reality-assisted digital twin for futuristic human-centric industry transformation," *Robotics and Computer-Integrated Manufacturing*, vol. 81, p. 102515, 2023.

[177] Z. Zhu, C. Liu, and X. Xu, "Visualisation of the digital twin data in manufacturing by using augmented reality," *Procedia Cirp*, vol. 81, pp. 898–903, 2019.

[178] J. P. Ballor, O. L. McClain, M. A. Mellor, A. Cattaneo, T. A. Harden, P. Shelton, E. Martinez, B. Narushof, F. Moreu, and D. D. Mascareñas, "Augmented reality for next generation infrastructure inspections," in *Model Validation and Uncertainty Quantification, Volume 3: Proceedings of the 36th IMAC, A Conference and Exposition on Structural Dynamics 2018*, pp. 185–192, Springer, 2018.

[179] A. Chouchene, A. Ventura Carvalho, F. Charrua-Santos, and W. Barhoumi, "Augmented reality-based framework supporting visual inspection for automotive industry," *Applied System Innovation*, vol. 5, no. 3, p. 48, 2022.

[180] V. Chheang, S. Narain, G. Hooten, R. Cerda, B. Au, B. Weston, B. Giera, P.-T. Bremer, and H. Miao, "Enabling additive manufacturing part inspection of digital twins via collaborative virtual reality," *Scientific Reports*, vol. 14, no. 1, p. 29783, 2024.

[181] S. Yang, S. A. Mirahmadi, E. Zhu, and B. Solanki, "Live digital twin with virtual reality for accessible and immersive manufacturing education," *The International Journal of Advanced Manufacturing Technology*, vol. 136, no. 7, pp. 3577–3590, 2025.

[182] M. M. Mahdi, M. S. Bajestani, S. Do Noh, and D. B. Kim, "Digital twin-based architecture for wire arc additive manufacturing using opc ua," *Robotics and computer-integrated manufacturing*, vol. 94, p. 102944, 2025.

[183] S. G. Izard, J. A. Juanes, F. J. García Peñalvo, J. M. G. Estella, M. J. S. Ledesma, and P. Ruisoto, "Virtual reality as an educational and training tool for medicine," *Journal of medical systems*, vol. 42, no. 3, p. 50, 2018.

[184] S. Choi, K. Jung, and S. D. Noh, "Virtual reality applications in manufacturing industries: Past research, present findings, and future directions," *Concurrent Engineering*, vol. 23, no. 1, pp. 40–63, 2015.

[185] H. Eschen, T. Kötter, R. Rodeck, M. Harnisch, and T. Schüppstuhl, "Augmented and virtual reality for inspection and maintenance processes in the aviation industry," *Procedia manufacturing*, vol. 19, pp. 156–163, 2018.

[186] A. A. Halim, "Applications of augmented reality for inspection and maintenance process in automotive industry," *Journal of Fundamental and Applied Sciences*, vol. 10, no. 3S, pp. 412–421, 2018.

[187] M. M. Kouijzer, H. Kip, Y. H. Bouman, and S. M. Kelders, "Implementation of virtual reality in healthcare: a scoping review on the implementation process of virtual reality in various healthcare settings," *Implementation science communications*, vol. 4, no. 1, p. 67, 2023.

[188] S. Torosian, V. Mousakhani, S. Wehsener, V. Ramnauth, and G. Walcott-Bedeau, "Virtual reality and preclinical medical education: a systematic review of its application and effectiveness," *Discover Education*, vol. 4, no. 1, p. 191, 2025.

[189] A. J. Lungu, W. Swinkels, L. Claesen, P. Tu, J. Egger, and X. Chen, "A review on the applications of virtual reality, augmented reality and mixed reality in surgical simulation: an extension to different kinds of surgery," *Expert review of medical devices*, vol. 18, no. 1, pp. 47–62, 2021.

[190] D. Avola, R. Caronna, L. Cinque, G. L. Foresti, and M. R. Marini, "Toward the future of surgery: an immersive, virtual-reality-based endoscopic prototype," *IEEE Systems, Man, and Cybernetics Magazine*, vol. 4, no. 3, pp. 6–13, 2018.

[191] D. Avola, L. Cinque, G. L. Foresti, M. R. Marini, and D. Pannone, "Vrheab: a fully immersive motor rehabilitation system based on recurrent neural network," *Multimedia Tools and Applications*, vol. 77, no. 19, pp. 24955–24982, 2018.

[192] D. Avola, L. Cinque, G. L. Foresti, and M. R. Marini, "An interactive and low-cost full body rehabilitation framework based on 3d immersive serious games," *Journal of biomedical informatics*, vol. 89, pp. 81–100, 2019.

[193] D. G. Morín, A. G. Armada, and P. Pérez, "Cutting the cord: Key performance indicators for the future of wireless virtual reality applications," in *2020 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, pp. 1–6, IEEE, 2020.

[194] A. Bangor, P. T. Kortum, and J. T. Miller, "An empirical evaluation of the system usability scale," *Intl. Journal of Human–Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008.

[195] S. G. Fussell, J. L. Derby, J. K. Smith, W. J. Shelstad, J. D. Benedict, B. S. Chaparro, R. Thomas, and A. R. Dattel, "Usability testing of a virtual reality tutorial," in *Proceedings of the Human Factors and Ergonomics Society annual meeting*, vol. 63, pp. 2303–2307, SAGE Publications Sage CA: Los Angeles, CA, 2019.

[196] P. Barve and R. De Amicis, "A user-centered evaluation of a vr hmd-based harvester training simulator," *Multimodal Technologies and Interaction*, vol. 10, no. 2, p. 15, 2026.

[197] J. Gornall, A. Peña, H. Pinto, J. Rojas, F. Correa, and J. García, "Extended realityin construction 4.0: A systematic review of applications, implementation barriers, and research trends," *Applied Sciences*, vol. 16, no. 1, p. 9, 2025.

[198] F. Malva, R. Nogueira, J. Marinho, N. Cid Martins, A. Malta, M. Mendes, and N. Ferreira, "An extended comparative analysis of ar/mr equipment for physical asset management applications," *Maintenance, Reliability and Condition Monitoring*, 2026.

[199] M. Cirelli, A. Cellupica, P. P. Valentini, L. Cinque, and M. R. Marini, "A comprehensive method to evaluate the usability of virtual reality headset devices for industrial applications," *Sensors*, vol. 26, no. 13, p. 4038, 2026.

[200] C. P. Kamm, R. Blättler, R. Kueng, and T. Vanbellingen, "Feasibility and usability of a new home-based immersive virtual reality headset-based dexterity training in multiple sclerosis," *Multiple sclerosis and related disorders*, vol. 71, p. 104525, 2023.

[201] S. Khorasani, B. V. Syiem, S. Nawaz, J. Knibbe, and E. Velloso, "Hands-on or hands-off: Deciphering the impact of interactivity on embodied learning in vr," *Computers & Education: X Reality*, vol. 3, p. 100037, 2023.

[202] E. Chang, H. T. Kim, and B. Yoo, "Virtual reality sickness: a review of causes and measurements," *International Journal of Human–Computer Interaction*, vol. 36, no. 17, pp. 1658–1682, 2020.

[203] D. Kamińska, G. Zwoliński, and A. Laska-Leśniewicz, "Usability testing of virtual reality applications—the pilot study," *Sensors*, vol. 22, no. 4, p. 1342, 2022.

[204] F. Nenna, D. Zanardi, and L. Gamberini, "Enhanced interactivity in vr-based telerobotics: an eye-tracking investigation of human performance and workload," *International Journal of Human-Computer Studies*, vol. 177, p. 103079, 2023.

[205] Y. Luo, S. Ahn, A. Abbas, J. Seo, S. H. Cha, and J. I. Kim, "Investigating the impact of scenario and interaction fidelity on training experience when designing immersive virtual reality-based construction safety training," *Developments in the Built Environment*, vol. 16, p. 100223, 2023.

[206] J. W. Mauchly, "Significance test for sphericity of a normal n-variate distribution," *The annals of mathematical statistics*, vol. 11, no. 2, pp. 204–209, 1940.

[207] S. W. Greenhouse and S. Geisser, "On methods in the analysis of profile data," *Psychometrika*, vol. 24, no. 2, pp. 95–112, 1959.

[208] H. Huynh and L. S. Feldt, "Estimation of the box correction for degrees of freedom from sample data in randomized block and split-plot designs," *Journal of educational statistics*, vol. 1, no. 1, pp. 69–82, 1976.

[209] A. Nanda, B. B. Mohapatra, A. P. K. Mahapatra, A. P. K. Mahapatra, and A. P. K. Mahapatra, "Multiple comparison test by tukey's honestly significant difference (hsd): Do the confident level control type i error," *International Journal of Statistics and Applied Mathematics*, vol. 6, no. 1, pp. 59–65, 2021.

[210] L. St, S. Wold, *et al.*, "Analysis of variance (anova)," *Chemometrics and intelligent laboratory systems*, vol. 6, no. 4, pp. 259–272, 1989.

[211] R. G. Miller Jr, *Beyond ANOVA: basics of applied statistics.* CRC press, 1997.

[212] B. G. Tabachnick and L. S. Fidell, *Experimental designs using ANOVA*, vol. 724. Thomson/Brooks/Cole Belmont, CA, 2007.

[213] R. Likert, "A technique for the measurement of attitudes.," *Archives of psychology*, 1932.

[214] G. Norman, "Likert scales, levels of measurement and the "laws" of statistics," *Advances in health sciences education*, vol. 15, no. 5, pp. 625–632, 2010.

[215] D. Avola, L. Cinque, G. L. Foresti, and M. R. Marini, "A novel low cybersickness dynamic rotation gain enhancer based on spatial position and orientation in virtual environments," *Virtual Reality*, vol. 27, no. 4, pp. 3191–3209, 2023.

[216] J. Sauro and J. R. Lewis, *Quantifying the user experience: Practical statistics for user research.* Morgan Kaufmann, 2016.

[217] H. Hoehle and V. Venkatesh, "An assessment of software usability using a 1-item scale," *IEEE Transactions on Professional Communication*, vol. 58, no. 4, pp. 377–392, 2015.

[218] M. Galesic and M. Bosnjak, "Effects of questionnaire length on participation and indicators of response quality in a web survey," *Public opinion quarterly*, vol. 73, no. 2, pp. 349–360, 2009.

[219] H. Mazhar, T. Heyn, A. Pazouki, D. Melanz, A. Seidl, A. Bartholomew, A. Tasora, and D. Negrut, "Chrono: a parallel multi-physics library for rigid-body, flexible-body, and fluid dynamics," *Mechanical Sciences*, vol. 4, no. 1, pp. 49–64, 2013.

[220] A. Tasora, D. Negrut, and M. Anitescu, "Large-scale parallel multi-body dynamics with frictional contact on the graphical processing unit," *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics*, vol. 222, no. 4, pp. 315–326, 2008.

[221] V. Acary and B. Brogliato, *Numerical methods for nonsmooth dynamical systems: applications in mechanics and electronics.* Springer Science & Business Media, 2008.

[222] P. Flores, M. Machado, M. T. Silva, and J. M. Martins, "On the continuous contact force models for soft materials in multibody dynamics," *Multibody system dynamics*, vol. 25, no. 3, pp. 357–375, 2011.

[223] R. W. Cottle, J.-S. Pang, and R. E. Stone, *The linear complementarity problem.* SIAM, 2009.

[224] M. Frâncu and F. Moldoveanu, "Unified simulation of rigid and flexible bodies using position based dynamics.," *VRIPHYS*, vol. 49, p. 58, 2017.

[225] A. Tasora, D. Mangoni, and S. Benatti, "Solving non-smooth dynamic problems using the alternating direction method of multipliers," in *Advances in Nonlinear Dynamics: Proceedings of the Second International Nonlinear Dynamics Conference (NODYCON 2021), Volume 2*, pp. 563–571, Springer, 2021.

[226] M. R. Hestenes, E. Stiefel, *et al.*, "Methods of conjugate gradients for solving linear systems," *Journal of research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409–436, 1952.

[227] Y. Saad, *Iterative methods for sparse linear systems.* SIAM, 2003.

[228] J. Nocedal and S. J. Wright, *Numerical optimization.* Springer, 2006.

[229] D. Negrut, *On the implicit integration of differential-algebraic equations of multibody dynamics.* The University of Iowa, 1998.

[230] P. P. Valentini and E. Pezzuti, "Interactive multibody simulation in augmented reality," *Journal of Theoretical and Applied Mechanics*, vol. 48, pp. 733–750, 2010.

[231] J. Baumgarte, "Stabilization of constraints and integrals of motion in dynamical systems," *Computer methods in applied mechanics and engineering*, vol. 1, no. 1, pp. 1–16, 1972.

[232] M.-D. Tsai, M.-S. Hsieh, and C.-H. Tsai, "Bone drilling haptic interaction for orthopedic surgical simulator," *Computers in Biology and Medicine*, vol. 37, no. 12, pp. 1709–1718, 2007.

[233] Q. Ha-Van, H. Schwendinger, Y. Kim, and M. Harders, "Design and characterization of an actuated drill mockup for orthopedic surgical training," *IEEE transactions on haptics*, vol. 13, no. 4, pp. 655–667, 2020.

[234] M. Patel, A. Unmesh, A. Ipsita, L. Erickson, P. Maheshwari, R. Jain, J. Shi, L. H. Blumenschein, and K. Ramani, "Powvrtool: a handheld haptic device for realistic power tool feedback in vr-based manufacturing training," *Virtual Reality*, 2025.

[235] J. Lim and Y. Choi, "Force-feedback haptic device for representation of tugs in virtual reality," *Electronics*, vol. 11, no. 11, p. 1730, 2022.

[236] H.-G. Lee, J.-H. Kim, M.-G. Jang, J. Kim, and H.-G. Kim, "A novel augmented reality (ar) haptic feedback system using real-time finite element analysis and visualization," *Virtual Reality*, 2026.

[237] A. F. Abate, M. Guida, P. Leoncini, M. Nappi, and S. Ricciardi, "A haptic-based approach to virtual training for aerospace industry," *Journal of Visual Languages & Computing*, vol. 20, no. 5, pp. 318–325, 2009.

[238] A. Corigliano, M. Dossi, and S. Mariani, "Model order reduction and domain decomposition strategies for the solution of the dynamic elastic–plastic structural problem," *Computer Methods in Applied Mechanics and Engineering*, vol. 290, pp. 127–155, 2015.

[239] P. Koutsovasilis and M. Beitelschmidt, "Model order reduction of finite element models: improved component mode synthesis," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 16, no. 1, pp. 57–73, 2010.

[240] G. H. Yoon, "Structural topology optimization for frequency response problem using model reduction schemes," *Computer Methods in Applied Mechanics and Engineering*, vol. 199, no. 25-28, pp. 1744–1763, 2010.

[241] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 42, no. 1, pp. 55–61, 2000.

[242] G. Rozza, D. B. P. Huynh, and A. T. Patera, "Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations," *Archives of computational methods in engineering*, vol. 15, no. 3, pp. 1–47, 2007.

[243] G. Kerschen, J.-c. Golinval, A. F. Vakakis, and L. A. Bergman, "The method of proper orthogonal decomposition for dynamical characterization and order reduction of mechanical systems: an overview," *Nonlinear dynamics*, vol. 41, no. 1, pp. 147–169, 2005.

[244] F. Chinesta, P. Ladeveze, and E. Cueto, "A short review on model order reduction based on proper generalized decomposition," *Archives of Computational Methods in Engineering*, vol. 18, no. 4, pp. 395–404, 2011.

[245] J. M. Silva, J. F. Villena, P. Flores, and L. M. Silveira, "Outstanding issues in model order reduction," in *Scientific Computing in Electrical Engineering*, pp. 139–152, Springer, 2007.

[246] T. Lassila, A. Manzoni, A. Quarteroni, and G. Rozza, "Model order reduction in fluid dynamics: challenges and perspectives," *Reduced Order Methods for modeling and computational reduction*, pp. 235–273, 2014.

[247] W. H. Schilders, H. A. Van der Vorst, and J. Rommes, *Model order reduction: theory, research aspects and applications*, vol. 13. Springer, 2008.

[248] A. C. Antoulas, *Approximation of large-scale dynamical systems.* SIAM, 2005.

[249] H. Chernoff, *Sequential analysis and optimal design.* SIAM, 1972.

[250] H. Hotelling, "Some improvements in weighing and other experimental techniques," *The Annals of Mathematical Statistics*, vol. 15, no. 3, pp. 297–306, 1944.

[251] M. N. Das and N. C. Giri, *Design and analysis of experiments.* New Age International, 1979.

[252] R. Iman, J. Davenport, and D. Zeigler, "Latin hypercube sampling (program user's guide) 1980," *Sandia Laboratories, Department of Energy.*

[253] K. Q. Ye, "Orthogonal column latin hypercubes and their application in computer experiments," *Journal of the American Statistical Association*, vol. 93, no. 444, pp. 1430–1439, 1998.

[254] X. Du, J. Liang, J. Lei, J. Xu, and P. Xie, "A radial-basis function mesh morphing and bayesian optimization framework for vehicle crashworthiness design: X. du et al.," *Structural and Multidisciplinary Optimization*, vol. 66, no. 3, p. 64, 2023.

[255] S. Boyaval, C. Le Bris, T. Lelievre, Y. Maday, N. C. Nguyen, and A. T. Patera, "Reduced basis techniques for stochastic problems," *Archives of Computational methods in Engineering*, vol. 17, no. 4, pp. 435–454, 2010.

[256] P. Benner, S. Gugercin, and K. Willcox, "A survey of projection-based model reduction methods for parametric dynamical systems," *SIAM review*, vol. 57, no. 4, pp. 483–531, 2015.

[257] Y. Choi, P. Brown, W. Arrighi, R. Anderson, and K. Huynh, "Space–time reduced order model for large-scale linear dynamical systems with application to boltzmann transport problems," *Journal of Computational Physics*, vol. 424, p. 109845, 2021.

[258] H.-S. Choi, J. An, S. Han, J.-G. Kim, J.-Y. Jung, J. Choi, G. Orzechowski, A. Mikkola, and J. H. Choi, "Data-driven simulation for general-purpose multibody dynamics using deep neural networks," *Multibody System Dynamics*, vol. 51, no. 4, pp. 419–454, 2021.

[259] C. K. Williams, "Prediction with gaussian processes: From linear regression to linear prediction and beyond," in *Learning in graphical models*, pp. 599–621, Springer, 1998.

[260] L. Le Gratiet and J. Garnier, "Recursive co-kriging model for design of computer experiments with multiple levels of fidelity," *International Journal for Uncertainty Quantification*, vol. 4, no. 5, 2014.

[261] M. L. Stein, *Interpolation of spatial data: some theory for kriging.* Springer Science & Business Media, 1999.

[262] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*, vol. 2. MIT press Cambridge, MA, 2006.

[263] R. M. Neal, *Bayesian learning for neural networks*, vol. 118. Springer Science & Business Media, 2012.

[264] C. Castellini, "Upper limb active prosthetic systems—overview," *Wearable robotics*, pp. 365–376, 2020.

[265] M. Controzzi, C. Cipriani, and M. C. Carrozza, "Design of artificial hands: A review," *The Human Hand as an Inspiration for Robot Hand Development*, pp. 219–246, 2014.

[266] I. Vujaklija, D. Farina, and O. C. Aszmann, "New developments in prosthetic arm systems," *Orthopedic research and reviews*, pp. 31–39, 2016.

[267] H. Bouwsema, C. K. van der Sluis, and R. M. Bongers, "Learning to control opening and closing a myoelectric hand," *Archives of physical medicine and rehabilitation*, vol. 91, no. 9, pp. 1442–1446, 2010.

[268] E. D'Anna, G. Valle, A. Mazzoni, I. Strauss, F. Iberite, J. Patton, F. M. Petrini, S. Raspopovic, G. Granata, R. Di Iorio, *et al.*, "A closed-loop hand prosthesis with simultaneous intraneural tactile and position feedback," *Science robotics*, vol. 4, no. 27, p. eaau8892, 2019.

[269] T. Pistohl, C. Cipriani, A. Jackson, and K. Nazarpour, "Abstract and proportional myoelectric control for multi-fingered hand prostheses," *Annals of biomedical engineering*, vol. 41, no. 12, pp. 2687–2698, 2013.

[270] C. Pylatiuk, S. Schulz, and L. Döderlein, "Results of an internet survey of myoelectric prosthetic hand users," *Prosthetics and orthotics international*, vol. 31, no. 4, pp. 362–370, 2007.

[271] E. A. Biddiss and T. T. Chau, "Upper limb prosthesis use and abandonment: a survey of the last 25 years," *Prosthetics and orthotics international*, vol. 31, no. 3, pp. 236–257, 2007.

[272] A. Bicchi, "Hands for dexterous manipulation and robust grasping: A difficult road toward simplicity," *IEEE Transactions on robotics and automation*, vol. 16, no. 6, pp. 652–662, 2000.

[273] N. E. Bunderson, "Real-time control of an interactive impulsive virtual prosthesis," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 22, no. 2, pp. 363–370, 2013.

[274] L. Hargrove, L. Miller, K. Turner, and T. Kuiken, "Control within a virtual environment is correlated to functional outcomes when using a physical prosthesis," *Journal of neuroengineering and rehabilitation*, vol. 15, no. Suppl 1, p. 60, 2018.

[275] G. Barresi, A. Marinelli, G. Caserta, M. De Zambotti, J. Tessadori, L. Angioletti, N. Boccardo, M. Freddolini, D. Mazzanti, N. Deshpande, *et al.*, "Exploring the embodiment of a virtual hand in a spatially augmented respiratory biofeedback setting," *Frontiers in Neurorobotics*, vol. 15, p. 683653, 2021.

[276] J. M. Lambrecht, C. L. Pulliam, and R. F. Kirsch, "Virtual reality environment for simulating tasks with a myoelectric prosthesis: an assessment and training tool," *JPO: Journal of Prosthetics and Orthotics*, vol. 23, no. 2, pp. 89–94, 2011.

[277] M. Semprini, N. Boccardo, A. Lince, S. Traverso, L. Lombardi, A. Succi, M. Canepa, V. Squeri, J. A. Saglia, P. Ariano, *et al.*, "Clinical evaluation of hannes: Measuring the usability of a novel polyarticulated prosthetic hand," in *Tactile Sensing, Skill Learning, and Robotic Dexterous Manipulation*, pp. 205–225, Elsevier, 2022.

[278] D. Di Domenico, A. Marinelli, N. Boccardo, M. Semprini, L. Lombardi, M. Canepa, S. Stedman, A. D. Bellingegni, M. Chiappalone, E. Gruppioni, *et al.*, "Hannes prosthesis control based on regression machine learning algorithms," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5997–6002, IEEE, 2021.

[279] P. Weiner, J. Starke, F. Hundhausen, J. Beil, and T. Asfour, "The kit prosthetic hand: design and control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3328–3334, IEEE, 2018.

[280] C. Cipriani, M. Controzzi, and M. C. Carrozza, "The smarthand transradial prosthesis," *Journal of neuroengineering and rehabilitation*, vol. 8, no. 1, p. 29, 2011.

[281] M. Zecca, S. Micera, M. C. Carrozza, and P. Dario, "Control of multifunctional prosthetic hands by processing the electromyographic signal," *Critical Reviews™ in Biomedical Engineering*, vol. 30, no. 4-6, 2002.

[282] M. Todisco, G. Saggio, L. Bianchi, and L. R. Quitadamo, "Svm classification of eeg signals for brain computer interface.,"

[283] F. Riillo, L. R. Quitadamo, F. Cavrini, G. Saggio, C. A. Pinto, N. C. Pastò, L. Sbernini, and E. Gruppioni, "Evaluating the influence of subject-related variables on emg-based hand gesture classification," in *2014 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, pp. 1–5, IEEE, 2014.

[284] A. Kecskeméthy and A. Weinberg, "An improved elasto-kinematic model of the human forearm for biofidelic medical diagnosis," *Multibody System Dynamics*, vol. 14, no. 1, pp. 1–21, 2005.

[285] P. P. Valentini, "Effects of the dimensional and geometrical tolerances on the kinematic and dynamic performances of the rzeppa ball joint," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 228, no. 1, pp. 37–49, 2014.

[286] N. A. Rumman and M. Fratarcangeli, "State of the art in skinning techniques for articulated deformable characters," in *International Conference on Computer Graphics Theory and Applications*, vol. 2, pp. 200–212, SciTePress, 2016.

[287] A. Sujar, J. J. Casafranca, A. Serrurier, and M. Garcia, "Real-time animation of human characters' anatomy," *Computers & Graphics*, vol. 74, pp. 268–277, 2018.

[288] E. De Aguiar and N. Ukita, "Representing mesh-based character animations," *Computers & graphics*, vol. 38, pp. 10–17, 2014.

[289] P. Kyberd, P. Chappell, and C. Light, "Assessment of hand functionality using the southampton hand assessment procedure," 2007.

[290] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre, "Recent advances in augmented reality," *IEEE computer graphics and applications*, vol. 21, no. 6, pp. 34–47, 2001.

[291] F. Liarokapis, "An augmented reality interface for visualizing and interacting with virtual content," *Virtual Reality*, vol. 11, no. 1, pp. 23–43, 2007.

[292] D. Chicco, M. J. Warrens, and G. Jurman, "The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation," *Peerj computer science*, vol. 7, p. e623, 2021.

[293] S. Li, M. Frey, and F. Gauterin, "Model-based condition monitoring of the sensors and actuators of an electric and automated vehicle," *Sensors*, vol. 23, no. 2, p. 887, 2023.

[294] M. Tristano, B. Lenzo, X. Xu, B. Forrier, T. D'hondt, E. Risaliti, and E. Wilhelm, "Hardware-in-the-loop real-time implementation of a vehicle stability control through individual wheel torques," *IEEE Transactions on Vehicular Technology*, vol. 73, no. 4, pp. 4683–4693, 2024.

[295] V. D. S. Committee *et al.*, *Vehicle dynamics terminology*. SAE International, 2022.

[296] O. Gilles, D. G. Pérez, P.-A. Brameret, and V. Lacroix, "Securing iiot communications using opc ua pubsub and trusted platform modules," *Journal of Systems Architecture*, vol. 134, p. 102797, 2023.

[297] D. E. Comer, *Internetworking con TCP/IP*, vol. 1. Pearson Italia Spa, 2006.

[298] A. C. Smit, E. Schat, and E. Ceulemans, "The exponentially weighted moving average procedure for detecting changes in intensive longitudinal data in psychological research in real-time: A tutorial showcasing potential applications," *Assessment*, vol. 30, no. 5, pp. 1354–1368, 2023.

[299] R. Rajamani, *Vehicle dynamics and control*. Springer, 2006.

[300] F. Naets, I. Noemi Uva, and M. Tristano, "Vehicle dynamics soft sensing, estimation and identification," in *Vehicle Dynamics, Control and Design*, pp. 133–176, Springer, 2025.

[301] W. B. Riley and A. R. George, "Design, analysis and testing of a formula sae car chassis," tech. rep., SAE Technical Paper, 2002.

[302] B. J. Mac Donald, *Practical stress analysis with finite elements*. Glasnevin publishing, 2007.

[303] E. Pennestrì, E. Cheli, *et al.*, *Cinematica e dinamica dei sistemi multibody*, vol. 1. Casa Editrice Ambrosiana, 2006.

[304] O. A. Bauchau and L. Trainelli, "The vectorial parameterization of rotation," *Nonlinear dynamics*, vol. 32, no. 1, pp. 71–92, 2003.

[305] J. J. Rotman, *An introduction to the theory of groups*. Springer Science & Business Media, 2012.

[306] H. G. Dales, *Banach algebras and automatic continuity.* Oxford University Press, 2001.

[307] M. Rafiq, G. Bugmann, and D. Easterbrook, "Neural network design for engineering applications," *Computers & Structures*, vol. 79, no. 17, pp. 1541–1552, 2001.

[308] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[309] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*, pp. 65–93, Elsevier, 1992.

[310] B. B. Traore, B. Kamsu-Foguem, and F. Tangara, "Deep convolution neural network for image recognition," *Ecological informatics*, vol. 48, pp. 257–268, 2018.

[311] Y. Goldberg, "A primer on neural network models for natural language processing," *Journal of Artificial Intelligence Research*, vol. 57, pp. 345–420, 2016.

[312] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.

[313] E. Catmull and J. Clark, "Recursively generated b-spline surfaces on arbitrary topological meshes," in *Seminal graphics: pioneering efforts that shaped the field*, pp. 183–188, 1998.

[314] D. Doo and M. Sabin, "Behaviour of recursive division surfaces near extraordinary points," in *Seminal graphics: pioneering efforts that shaped the field*, pp. 177–181, 1998.

[315] C. Loop, "Smooth subdivision surfaces based on triangles," 1987.

[316] B. Delaunay, "Bulletin de l'académie des sciences de l'urss, classe des sciences mathématiques et naturelles," 1934.

[317] D.-T. Lee and B. J. Schachter, "Two algorithms for constructing a delaunay triangulation," *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, 1980.

[318] A. Bowyer, "Computing dirichlet tessellations," *The computer journal*, vol. 24, no. 2, pp. 162–166, 1981.

[319] H. Edelsbrunner, "Triangulations and meshes in computational geometry," *Acta numerica*, vol. 9, pp. 133–213, 2000.

[320] Y. Chen, P. Goetsch, M. A. Hoque, J. Lu, and S. Tarkoma, "*d* d-simplexed: Adaptive delaunay triangulation for performance modeling and prediction on big data analytics," *IEEE Transactions on Big Data*, vol. 8, no. 2, pp. 458–469, 2019.