Master's Degree in Telecommunication Engineering

Master's Thesis

Immersive Virtual Reality Platform for Sign Language Learning Through Automatic Gesture Recognition and Translation

Author
Blanca María de Pedro Morejón

Supervised by
Dr. Ming Lin

Madrid
May 2026

I declare, under my responsibility, that the Project presented with the title **Immersive Virtual Reality Platform for Sign Language Learning Through Automatic Gesture Recognition and Translation**

at the ETS de Ingeniería — ICAI of the Universidad Pontificia Comillas in the academic year 2025/2026, is of my authorship, original and unpublished, and has not been submitted previously for any other purpose. The Project is not plagiarised from another, either totally or partially, and the information that has been taken from other documents is duly referenced.

lanca de Pedro Morejón

Fdo.: Blanca de Pedro Morejón Fecha: 18 /04 / 2026

Authorised for submission of the project

THE PROJECT SUPERVISOR

Signed by

Ming C. Lin

Fdo.: Dr. Ming Lin Fecha: 4 / 18 / 2026

Abstract

IMMERSIVE VIRTUAL REALITY PLATFORM FOR SIGN LANGUAGE LEARNING THROUGH AUTOMATIC GESTURE RECOGNITION AND TRANSLATION

Author: Blanca María de Pedro Morejón
Supervisor: Dr. Ming Lin
Collaborating Institution: ICAI - Universidad Pontificia Comillas

PROJECT SUMMARY

The project developed consists of the design and implementation of an immersive virtual reality platform aimed at guided learning of American Sign Language (ASL), named ASL LearnVR. The system allows the user to practice manual signs through real-time optical hand tracking, receive immediate feedback on their execution, and compare their gesture with a visual reference within the immersive environment itself. The proposal is not limited to recognizing whether a sign is correct or incorrect, but rather focuses on facilitating a guided practice process in which the user can identify specific errors and progressively correct them. To achieve this, the platform integrates deterministic gesture recognition, finger-level analysis, guiding hands, and different interaction modes oriented towards learning.

Keywords: Virtual reality, sign language, hand tracking, deterministic recognition, real-time feedback, ASL

1. Introduction

The incorporation of immersive technologies in educational contexts has opened new possibilities for learning based on active practice [1, 2]. In particular, virtual reality enables the creation of three-dimensional environments in which the user interacts directly with the system and receives immediate feedback on their actions. These characteristics are especially relevant in the learning of motor skills, where repetition, observation, and progressive correction play an essential role.

In the field of sign language, most existing work has focused on automatic gesture recognition, often through machine learning models oriented towards classification [3, 4]. However, this approach does not always address the needs of an educational system, as recognizing a sign does not necessarily imply explaining how it has been performed or indicating what should be corrected. Therefore, the problem addressed in this work is not limited to detecting signs, but rather to designing a platform that enables the practice, analysis, and correction of manual gestures in a structured way within an immersive environment.

2. Project Definition

The purpose of this Master's Thesis is to develop an immersive ASL learning platform in virtual reality that allows the user to practice manual signs through direct interaction with their hands and receive useful feedback during practice. The system has been conceived as a guided learning tool, in which the user not only performs gestures but also has access to visual support, real-time evaluation, and mechanisms for progressive correction.

The proposed solution has been developed for Meta Quest 3 using Unity 6 and the XR Hands package as the basis for controller-free hand tracking. In contrast to machine learning-based approaches, a deterministic recognition method has been adopted, in which each sign is defined through explicit geometric and temporal conditions. This decision allows control over system behavior, the definition of adjustable tolerances, and ensures an interpretable and reproducible evaluation.

The implemented content includes a total of 84 signs organized into seven pedagogical categories: alphabet, digits, basic communication, colors, days of the week, months, and verbs. Of these, 34 are validated exclusively based on hand pose, while 50 require temporal analysis of movement, allowing both static, dynamic, and sequential gestures to be covered within a single platform.

3. System Description

The developed system is structured into several coordinated modules: hand motion capture, gesture validation, detailed execution analysis, and real-time feedback generation. The platform relies on the skeletal model provided by the device, from which hand joints are obtained and geometric parameters related to finger flexion, relative separation, and overall hand orientation are computed.

Based on this information, the system compares the user's execution with the definition of the active sign. In static gestures, validation is based on hand configuration, while in dynamic gestures, additional conditions related to trajectory, movement direction, or temporal stability are included. Sequential gestures validate each phase progressively. In this way, the platform integrates different types of signs within a single architecture without breaking system coherence.

One of the central elements of the project is the separation between global recognition decision and finger-level diagnosis. While the recognizer determines whether the sign can be considered correct, the geometric analysis system independently examines the configuration of each finger to identify deviations from the target pose. This enables the generation of specific feedback indicating which finger should be corrected and in what way.

The platform also incorporates a virtual reference hand that shows the correct execution of the sign and serves as a visual guide during practice. This representation is decoupled from user tracking, allowing a stable and clear reference to be maintained within the environment. In addition, the system provides textual messages, visual indicators, and explanatory feedback during execution, creating a continuous flow between observation, practice, and correction.

4. Results

The results obtained show that the system is able to consistently recognize the implemented signs when the user correctly performs the configuration and, where applicable, the associated movement or sequence. It also allows incorrect executions to be rejected and distinguishes between structurally similar signs through specific geometric and temporal conditions.

In static gestures, the system distinguishes between similar hand configurations based on explicit finger-level constraints. In dynamic gestures, validation depends not only on the initial pose but also on the movement performed, allowing differentiation between signs that share the same initial configuration. In sequential gestures, progressive phase validation enables step-by-step tracking of the user's progress and prevents completion when the executed sequence is incorrect.

The feedback system constitutes one of the most relevant results of the work, as it transforms gesture validation into an active learning tool. The user not

only receives a global indication of success or error, but also specific information about finger configuration or the movement that needs to be adjusted. However, limitations associated with optical tracking are observed, especially in configurations involving finger occlusion or in signs requiring particularly precise thumb placement.

5. Conclusions

Once the development of the project has been completed, it is observed that it is possible to build an immersive sign language learning platform that integrates hand tracking, gesture recognition, and real-time feedback generation within a coherent and functional system. The work demonstrates that virtual reality can be used not only as a visualization medium, but also as an environment for guided practice in the learning of manual gestures.

It has also been shown that a deterministic approach is particularly suitable when the goal of the system is not only to recognize signs, but to facilitate their practice and provide interpretable feedback. The explicit definition of geometric and temporal conditions allows control over system behavior, identification of specific deviations, and the provision of useful information to help users progressively improve their execution.

Overall, the proposed platform goes beyond isolated sign recognition and is presented as a solution oriented towards practical ASL learning, establishing a solid foundation for future extensions of the sign catalog, improvements in robustness against tracking limitations, and the evolution towards more complex content within immersive environments.

Abstract

INMERSIVA DE REALIDAD VIRTUAL PARA EL APRENDIZAJE DE LENGUA DE SIGNOS MEDIANTE RECONOCIMIENTO Y TRADUCCIÓN AUTOMÁTICA DE GESTOS.

Autor: Blanca María de Pedro Morejón
Director: Dr. Ming Lin
Entidad Colaboradora: ICAI - Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

El proyecto desarrollado consiste en el diseño e implementación de una plataforma inmersiva en realidad virtual orientada al aprendizaje guiado de lengua de signos americana (ASL), denominada ASL LearnVR. El sistema permite al usuario practicar signos manuales mediante seguimiento óptico de la mano en tiempo real, recibir retroalimentación inmediata sobre su ejecución y comparar su gesto con una referencia visual dentro del propio entorno inmersivo. La propuesta no se centra únicamente en reconocer si un signo es correcto o incorrecto, sino en facilitar un proceso de práctica guiada en el que el usuario pueda identificar errores concretos y corregirlos progresivamente. Para ello, la plataforma integra reconocimiento determinista de gestos, análisis por dedo, manos guía y distintos modos de interacción orientados al aprendizaje.

Palabras clave: Realidad virtual, lengua de signos, hand tracking, reconocimiento determinista, feedback en tiempo real, ASL

1. Introducción

La incorporación de tecnologías inmersivas en contextos educativos ha abierto nuevas posibilidades para el aprendizaje basado en la práctica activa [1, 2]. En particular, la realidad virtual permite crear entornos tridimensionales en los que el usuario interactúa directamente con el sistema y recibe retroalimentación inmediata sobre sus acciones. Estas características resultan especialmente relevantes en el aprendizaje de habilidades motoras, donde la repetición, la observación y la corrección progresiva desempeñan un papel esencial.

En el ámbito de la lengua de signos, la mayoría de trabajos existentes se han centrado en el reconocimiento automático de gestos, frecuentemente mediante modelos de aprendizaje automático orientados a clasificación [3, 4]. Sin embargo, este enfoque no siempre responde a las necesidades de un sistema educativo, ya que reconocer un signo no implica necesariamente explicar cómo se ha ejecutado ni indicar qué debe corregirse. Por ello, el problema que se plantea en este trabajo no es únicamente detectar signos, sino diseñar una plataforma que permita practicar, analizar y corregir gestos manuales de forma estructurada dentro de un entorno inmersivo.

2. Definición del proyecto

El propósito de este Trabajo Fin de Máster es desarrollar una plataforma inmersiva de aprendizaje de ASL en realidad virtual que permita al usuario practicar signos manuales mediante interacción directa con sus manos y recibir retroalimentación útil durante la práctica. El sistema se ha concebido como una herramienta de aprendizaje guiado, en la que el usuario no solo ejecuta gestos, sino que dispone de apoyo visual, evaluación en tiempo real y mecanismos de corrección progresiva.

La solución propuesta se ha desarrollado para Meta Quest 3 utilizando Unity 6 y el paquete XR Hands como base para el seguimiento manual sin controladores. Frente a enfoques basados en aprendizaje automático, se ha optado por un reconocimiento determinista en el que cada signo se define mediante condiciones geométricas y temporales explícitas. Esta decisión permite controlar el comportamiento del sistema, definir tolerancias ajustables y garantizar una evaluación interpretable y reproducible.

El contenido implementado incluye un total de 84 signos organizados en siete categorías pedagógicas: alfabeto, dígitos, comunicación básica, colores, días de la semana, meses y verbos. De ellos, 34 se validan exclusivamente a partir de la pose de la mano y 50 requieren análisis temporal del movimiento, lo que permite cubrir tanto gestos estáticos como dinámicos y secuenciales dentro de una misma plataforma.

3. Descripción del sistema

El sistema desarrollado se estructura en distintos módulos coordinados entre sí, captura del movimiento de la mano, validación del gesto, análisis detallado de la ejecución y generación de feedback en tiempo real. La plataforma parte del modelo esquelético proporcionado por el dispositivo, a partir del cual se

obtienen las articulaciones de la mano y se calculan parámetros geométricos relacionados con la flexión de los dedos, su separación relativa y la orientación general de la mano.

A partir de esta información, el sistema compara la ejecución del usuario con la definición del signo activo. En los gestos estáticos, la validación se basa en la configuración de la mano, mientras que en los gestos dinámicos, se añaden condiciones relativas a trayectoria, dirección del movimiento o estabilidad temporal. Los gestos secuenciales validan cada fase de forma progresiva. De este modo, la plataforma integra en una misma arquitectura distintos tipos de signo sin romper la coherencia del sistema.

Uno de los elementos centrales del proyecto es la separación entre la decisión global de reconocimiento y el diagnóstico por dedo. Mientras que el reconocedor determina si el signo puede considerarse correcto, el sistema de análisis geométrico estudia de forma independiente la configuración de cada dedo para identificar desviaciones respecto a la pose objetivo. Esto permite generar feedback específico sobre qué dedo debe corregirse y en qué sentido.

La plataforma incorpora además una mano virtual de referencia que muestra la ejecución correcta del signo y sirve como guía visual durante la práctica. Esta representación está desacoplada del tracking del usuario, lo que permite mantener una referencia estable y clara dentro del entorno. Junto con ello, el sistema presenta mensajes textuales, indicadores visuales y retroalimentación explicativa durante la ejecución, creando un flujo continuo entre observación, práctica y corrección.

4. Resultados

Los resultados obtenidos muestran que el sistema es capaz de reconocer de forma consistente los signos implementados cuando el usuario ejecuta correctamente la configuración y, cuando corresponde, el movimiento o la secuencia asociada. Asimismo, permite rechazar ejecuciones incorrectas y discriminar entre signos estructuralmente similares mediante condiciones geométricas y temporales específicas.

En los gestos estáticos, el sistema distingue configuraciones manuales próximas entre sí a partir de restricciones explícitas por dedo. En los gestos dinámicos, la validación no depende únicamente de la pose inicial, sino también del movimiento realizado, lo que permite diferenciar signos que comparten configuración de partida. En los gestos secuenciales, la validación progresiva por fases permite seguir

el avance del usuario paso a paso y evita completar signos cuando la secuencia ejecutada no es correcta.

El sistema de feedback constituye uno de los resultados más relevantes del trabajo, ya que transforma la validación del gesto en una herramienta activa de aprendizaje. El usuario no solo recibe una indicación global de éxito o error, sino también información específica sobre la configuración de los dedos o sobre el movimiento que debe ajustar. No obstante, se observan limitaciones asociadas al seguimiento óptico, especialmente en configuraciones con oclusión entre dedos o en signos que requieren una colocación especialmente precisa del pulgar.

5. Conclusiones

Una vez concluido el desarrollo del proyecto, se observa que es posible construir una plataforma inmersiva de aprendizaje de lengua de signos que integre captura manual, reconocimiento de gestos y generación de feedback en tiempo real dentro de un sistema coherente y funcional. El trabajo desarrollado demuestra que la realidad virtual puede utilizarse no solo como medio de visualización, sino también como entorno de práctica guiada para el aprendizaje de gestos manuales.

Asimismo, se ha comprobado que un enfoque determinista resulta especialmente adecuado cuando el objetivo del sistema no es únicamente reconocer signos, sino facilitar su práctica y proporcionar retroalimentación interpretable. La definición explícita de condiciones geométricas y temporales permite controlar el comportamiento del sistema, identificar desviaciones concretas y ofrecer al usuario información útil para mejorar progresivamente su ejecución.

En conjunto, la plataforma propuesta va más allá del reconocimiento aislado de signos y se plantea como una solución orientada al aprendizaje práctico de ASL, sentando una base sólida para futuras ampliaciones del catálogo, mejoras de robustez frente a limitaciones del tracking y evolución hacia contenidos de mayor complejidad dentro de entornos inmersivos.

Bibliography

[1] Laura Freina and Michela Ott. "A Literature Review on Immersive Virtual Reality in Education: State Of The Art and Perspectives". In: *Proceedings of eLearning and Software for Education (eLSE)*. Vol. 1. 2015, pp. 133–141.

[2] Jaziar Radianti et al. "A systematic review of immersive virtual reality applications for higher education: Design elements, lessons learned, and research agenda". In: *Computers & Education* 147 (2020), p. 103778.

[3] Danielle Bragg et al. "Sign Language Recognition, Generation, and Translation: An Interdisciplinary Perspective". In: *The 21st International ACM SIGACCESS Conference on Computers and Accessibility* (2019), pp. 16–31.

[4] Necati Cihan Camgöz et al. "Sign Language Transformers: Joint End-to-End Sign Language Recognition and Translation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 10023–10033.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 General Context

More than 5% of the global population, around 430 million people, experience some degree of hearing loss, of whom 34 million are children, according to the World Health Organization [1]. The WHO estimates that, without adequate measures for prevention, early detection and intervention, and access to hearing care services, this number could exceed 700 million by 2050 [1]. This reality not only poses a healthcare challenge, but also an educational and social one, as it directly affects access to communication and inclusion across multiple areas of daily life.

Sign language is a complete linguistic system, with its own grammatical structure and a strong cultural component within the deaf community. Its promotion aligns with the principles of inclusion and accessibility established by international organizations such as UNESCO [2] and with the Sustainable Development Goals defined by the United Nations [3]. However, its learning among the hearing population remains limited in many contexts, which contributes to maintaining a persistent communication gap.

In parallel, the last decade has been marked by significant advances in computer vision (*Computer Vision*), machine learning (*Machine Learning*), and immersive technologies. Improvements in hand tracking systems within virtual reality environments have made it possible to capture specific poses and analyze movements with greater precision [4, 5]. These advances have expanded the range of possible educational applications within immersive environments.

Recent studies indicate that virtual reality can facilitate practice-based learning, especially in tasks that require motor precision, by providing interactive environments with immediate feedback [6]. In the specific context of sign language education, approaches based on augmented and mixed reality have been proposed, integrating artificial intelligence to improve interaction and visual understanding

of gestures [7, 8]. Additionally, recent research highlights the potential of visual guidance systems that display the correct execution of a gesture to support progressive learning [9].

The combination of these technological advances and the educational needs described creates a favorable context for the development of tools that leverage current hand tracking capabilities in immersive environments.

These tools can not only detect gestures, but also support the user throughout the learning process in a structured, progressive, and understandable way.

1.2 Problem Statement

Despite recent advances in automatic sign language recognition, most approaches developed to date have adopted a predominantly recognition- or classification-oriented perspective, focusing on determining whether a given hand configuration corresponds to a specific label [10, 11]. Although this approach is effective for automatic recognition tasks, it presents limitations when the primary objective is learning.

When learning sign language, it is not sufficient to reproduce a gesture approximately. Hand position, orientation, and movement must be performed with precision, as changes in shape or trajectory can alter the intended meaning. In this context, a significant portion of existing systems provide only binary feedback (correct/incorrect), without specifying which component of the gesture requires correction or offering progressive guidance to improve execution [8, 9].

At the same time, immersive environments have shown that active practice with real-time feedback can support learning in educational contexts [12, 13, 6]. This is further enhanced by the incorporation of controller-free hand tracking in standalone headsets, which enables the capture and analysis of complex hand configurations in real time [4, 5]. However, the integration of hand tracking, structured gesture validation, and pedagogical feedback within a single immersive platform remains relatively uncommon in solutions specifically designed for learning [8, 11].

Therefore, the problem addressed in this work is not limited to sign recognition, but rather the need to design a platform that enables the structured practice, analysis, and correction of gestures in virtual reality, combining real-time hand tracking with immediate and detailed feedback mechanisms oriented towards progressive learning.

1.3 Project Scope

This Master's Thesis aims to design and implement an immersive virtual reality platform for guided learning of American Sign Language (ASL), based on the real-time analysis of hand configuration and movement through optical tracking without physical controllers.

The system, named *ASL LearnVR*, has been developed for Meta Quest 3 using Unity 6 as the implementation environment and the *Unity XR Hands* package as the access layer to the hand tracking subsystem. The platform implements a total of 84 signs organized into seven pedagogical categories: alphabet (letters A–Z), digits (0–9), basic communication (greetings and everyday expressions), colors (11), days of the week, months of the year, and basic verbs (10). Of these signs, 50 require temporal analysis of movement (dynamic gestures), while 34 are validated exclusively based on the geometric configuration of the hand (static gestures).

The technical scope of the system covers four main functions. First, the definition of poses and gestures that constitute the pedagogical content of the platform, through the explicit specification of hand configurations and movement trajectories. Second, the presentation of reference animations that show the correct execution of each gesture and serve as guidance during the learning phase. Third, the capture of hand motion through the skeletal model provided by the device's hand tracking system. Based on this information, the system performs structured gesture validation using a deterministic approach based on geometric and temporal rules. Finally, the system generates visual feedback aimed at progressive correction during the practice phase.

The system also distinguishes between the global recognition decision and finger-level diagnosis. This distinction allows the user to be informed not only whether the gesture is correct, but also which specific component needs to be corrected.

The project is defined with the following constraints: the analysis is limited to the right hand; the system does not employ machine learning models, but instead uses a fully interpretable deterministic approach; the content is restricted to a subset of basic ASL and does not address dialectal variations or a broader linguistic context; and tracking data is processed locally on the device, without transmission to external services.

1.4 Contributions

The main contributions of this work are as follows.

First, the development of a complete virtual reality-based system for learning sign language, integrating hand motion capture, gesture recognition, and real-time feedback generation.

Second, the design of a deterministic approach for manual gesture recognition, based on the explicit definition of geometric conditions at the finger level. This approach avoids the need for training data and enables full control over system behavior.

Another relevant contribution is the detailed feedback system, capable of identifying specific errors in finger configuration and providing useful information for their correction. This transforms gesture recognition into an active learning tool.

Additionally, a visual guidance system based on reference hands is incorporated, allowing the user to compare their execution with a correct representation of the gesture within the virtual environment, reinforcing the learning process through observation and imitation.

Furthermore, the system supports static, dynamic, and sequential gestures within a unified architecture, maintaining consistency across different gesture types.

Finally, the work demonstrates how all these elements can be integrated into a virtual reality environment, ensuring a continuous interaction flow and a coherent visual representation.

Chapter 2

Technology Overview

2.1 Virtual Reality and the OpenXR Standard

Virtual reality (VR) enables the creation of immersive three-dimensional environments in which the user interacts through real-time spatial tracking. Unlike conventional two-dimensional interfaces, VR integrates depth perception, head and hand tracking, and sensory feedback within a single interaction loop. These characteristics make it particularly well suited for learning motor skills that require spatial precision, such as the execution of hand configurations in sign language.

To ensure interoperability and compatibility across devices, the system is based on the **OpenXR** standard, promoted by the Khronos Group [14]. OpenXR defines a unified API for virtual and augmented reality applications, abstracting the underlying hardware through a device-specific *runtime*. This runtime acts as an intermediate layer between the application and the hardware, translating application calls into device-specific instructions and allowing the same application to run on different compatible devices without structural modifications.

The main alternative within the Meta ecosystem is the proprietary *OVRPlugin* SDK, which provides direct access to device-specific functionalities. However, its use introduces a direct dependency on the manufacturer and limits system portability. The adoption of OpenXR allows the system to remain independent from proprietary SDKs, ensures native compatibility with Meta Quest 3 through the Meta runtime, and supports future scalability to other compatible devices without requiring changes to the application architecture.

2.2 Development Engine: Unity 6

The system has been developed using **Unity 6 (version 6000.2.13f1)** as the graphics engine and runtime environment [15]. Unity is a widely used cross-

platform development engine for interactive and virtual reality applications, combining a visual editing environment with a C# scripting system and native support for OpenXR through the XR Plugin Management package [16].

For virtual reality management, the project uses the following official packages:

- **XR Plugin Management 4.5.3**: configures OpenXR as the XR rendering backend, automatically handles stereoscopic rendering (generation of a separate image for each eye), and provides a unified interface for accessing XR devices such as headsets, hands, and controllers.
- **XR Interaction Toolkit 3.2.2**: provides standardized components for managing interactions in virtual reality environments [17].
- **Meta OpenXR 2.3.0**: Meta-specific package that adds direct compatibility with Quest devices and optimizations for hand tracking [18].
- **Shader Graph 17.2.0**: visual system for creating custom shaders used in environment rendering.
- **TextMesh Pro**: advanced text rendering system used in the VR user interface.

Regarding the rendering pipeline, the project uses Unity's **Built-in Render Pipeline**. This pipeline is the engine's standard rendering system and offers direct compatibility with the XR tools used in the project, including OpenXR and XR Hands, without requiring additional configuration.

Since the application focuses on hand interaction and gesture validation rather than complex graphics or advanced visual effects, the Built-in Render Pipeline is sufficient to maintain stable performance on a *standalone* device such as Meta Quest 3.

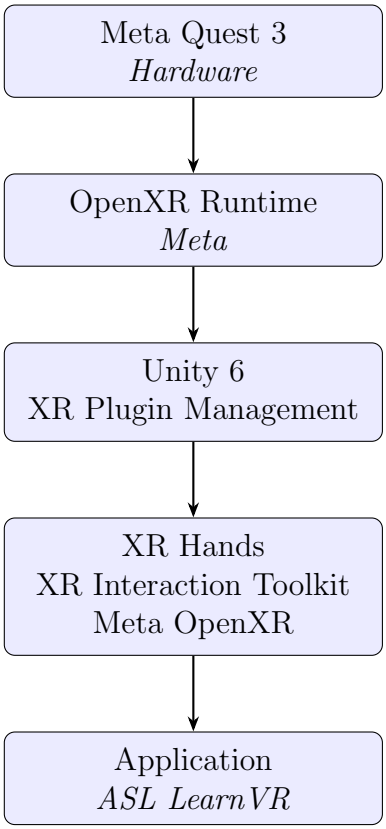


Figure 2.1: System technology stack, from hardware to application layer.

2.3 Hand Tracking System: XR Hands

Hand tracking is implemented using the official Unity package **XR Hands 1.7.2** [19], which exposes the `XRHandSubsystem` as a unified interface for accessing hand tracking data from the device.

On Meta Quest 3, tracking is performed using integrated cameras that reconstruct in real time a skeletal model composed of 26 joints per hand, hierarchically organized from the wrist to the fingertips [20]. For each joint, the subsystem provides three-dimensional position, orientation represented by quaternions, and tracking validity status.

Joint updates occur at an approximate frequency of **60 Hz**, which, in combination with the temporal mechanisms implemented, is sufficient for the validation of hand configurations within the context of this system.

Access to this data is achieved through two complementary mechanisms: the `jointsUpdated` event, which is triggered when the subsystem updates joint information, and direct access to the `XRHandSubsystem` via `XRGeneralSettings`. The

combination of both mechanisms allows the system to handle update events and perform direct queries of the hand state within Unity's execution cycle.

The use of XR Hands enables direct interaction with the kinematic structure of the hand without requiring additional physical devices, a key feature for the precise validation of hand configurations in the educational environment targeted by this system.

2.4 Execution Device: Meta Quest 3

Meta Quest 3 is a *standalone* virtual reality headset that integrates processing, rendering, and tracking within the device itself, without requiring connection to an external computer [21].

Among its most relevant features for this project are:

- **Optical hand tracking**: the device incorporates cameras that enable real-time reconstruction of the skeletal model of both hands without the need for physical controllers.
- **Standalone processing**: the Snapdragon XR2 Gen 2 *system-on-chip* provides sufficient computational capacity to execute in real time the deterministic recognition logic implemented in ASL LearnVR, influencing design decisions such as the absence of machine learning models at runtime.
- **Immersive visualization**: the headset features a resolution of 2064×2208 pixels per eye and an approximate field of view of 110°, facilitating precise observation of manual gestures and interface elements.

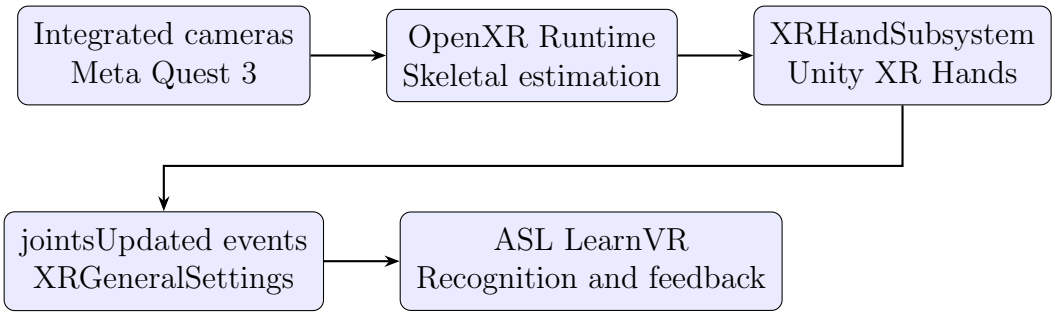


Figure 2.2: Data flow from the device cameras to the recognition system.

2.5 Development Environment

The development was carried out on a computer running Windows 11, using Unity 6 as the main development environment and Visual Studio Community as the code editor. Version control was managed using Git with a remote repository on GitHub, enabling modular code organization, change traceability, and development stability.

Deployment and testing were performed directly on the device via USB connection and Android Debug Bridge (ADB), compiling the application as an Android APK using Unity's Android Build Support module. This setup allowed rapid iteration between implementation and testing cycles on real hardware, which was particularly relevant for fine-tuning tolerances and parameters of the recognition system.

Chapter 3

State of the Art

3.1 Immersive Technologies in Education

The incorporation of immersive technologies in educational contexts has increased significantly over the past decade. Several studies have analyzed the use of virtual reality (VR) in teaching and learning processes, highlighting its ability to create interactive environments in which students can practice skills directly within a three-dimensional space.

Freina and Ott [12] identify that VR enables the design of learning environments based on direct experimentation, facilitating the understanding of complex concepts through active interaction within a three-dimensional setting. Subsequently, Radianti et al. [13] analyze the use of VR in higher education and observe that immersive environments can increase student engagement when they are coherently integrated with learning objectives.

Beyond general applications, VR has shown particular relevance in inclusive education contexts. The DiVRsity project [6] uses group-based *role-play* scenarios in VR to promote awareness of disability, demonstrating that immersion supports empathetic understanding and perspective-taking. Similarly, recent studies such as VR-Shopping [5] analyze improvements in accessibility and manual interaction for users with motor limitations, highlighting that well-designed natural interactions can reduce usability barriers. In addition, several works point out that immersive environments can support the learning of motor skills when they combine direct interaction with immediate feedback [8].

Virtual reality offers several advantages in educational contexts, including practice-based learning, controlled environments with immediate feedback, reduced physical and logistical barriers, and increased student motivation. These characteristics support the development of applications aimed at learning precise motor skills, such as sign language, where the spatial dimension of movement is essential.

3.2 Hand Tracking and Gesture Analysis

The development of *hand tracking* systems in virtual and mixed reality environments has evolved significantly in recent years. Recent research shows that controller-free hand tracking enables more natural interactions and simplifies user interaction [4].

Devices such as Meta Quest 3 integrate optical cameras capable of reconstructing a real-time skeletal model of the hand, composed of multiple hierarchically structured joints and updated at approximately 60 Hz, a frequency sufficient for smooth interaction. This capability allows the analysis of complex hand configurations in greater detail and enables gesture validation with sufficient precision for educational applications.

In educational contexts, *hand tracking* reduces usability barriers and improves accessibility, particularly for users with motor limitations [5]. It also enables real-time observation of gesture execution and the provision of immediate feedback, which is a key element in motor learning processes.

However, optical tracking presents technical limitations. Finger occlusion can reduce accuracy, performance depends on adequate lighting conditions, there is variability in hand size and morphology across users, and latency inherent to visual processing can introduce small temporal deviations. These limitations must be considered when designing gesture validation systems.

Accurate hand tracking therefore constitutes a technological prerequisite for the development of systems capable of analyzing and evaluating gesture execution in sign language.

3.3 Sign Language Recognition Systems

Research on automatic sign language recognition has predominantly focused on approaches based on *deep learning* and computer vision (*computer vision*). Camgoz et al. [10] present neural models oriented towards automatic sign language translation, prioritizing accuracy in classification tasks and sequential sign recognition. These systems typically employ architectures capable of modeling temporal information, such as recurrent networks, attention mechanisms, or *Transformer*-based architectures, which allow capturing the evolution of gestures over time.

The development of specialized *datasets* has been key to these advances. Datasets such as WLASL [22] enable the training of word-level recognition models using large collections of sign language video data. Similarly, the RWTH-PHOENIX-Weather dataset [23] has been widely used in research for continuous sign language recognition and automatic translation tasks.

These approaches have achieved significant progress in recognition and automatic translation. However, their primary focus lies in correctly identifying the sign, rather than helping the user learn how to perform it correctly or indicating which aspects of the gesture should be corrected during execution. This distinction is particularly relevant in educational applications, where the objective is not only to recognize the gesture, but to analyze its execution and guide the user through progressive correction.

3.4 Educational Platforms for Sign Language Learning

More recently, approaches specifically oriented towards learning have emerged. Wen et al. [8] present a mixed reality approach for sign language teaching that integrates immersive learning with multidimensional feedback. Their results indicate that spatial visualization improves the perception of hand position, orientation, and movement.

Similarly, Gillamac and Figueroa [9] explore augmented visual guidance systems that provide explicit references during gesture execution, reducing the user's cognitive load and facilitating real-time correction.

These platforms show that immersive environments offer clear advantages for learning ASL, including three-dimensional visualization of gestures, active practice with immediate feedback, unlimited repetition without social pressure, and improved spatial understanding of movement.

However, limitations remain. Feedback on specific components of the gesture is often limited, detailed validation at the joint level is not fully integrated, and few platforms combine hand tracking, structured gesture validation, and pedagogical feedback within a single system.

3.5 Synthesis and Identified Limitations

Four main lines of development have been identified:

1. Immersive technologies in education, which demonstrate effectiveness in experiential learning and inclusive education.
2. Advances in hand tracking that enable accurate capture of complex hand configurations.
3. Sign language recognition systems primarily based on *deep learning*, oriented towards classification or translation.

4. Immersive educational platforms with visual guidance and multidimensional feedback.

Despite these advances, several relevant limitations can be identified:

L1. Gap between recognition and learning. Most systems prioritize recognition accuracy over pedagogical value and lack training mechanisms that guide the user during practice.

L2. Limited feedback. Feedback is often binary and does not identify specific components of the gesture that require correction.

L3. Incomplete integration. Few platforms integrate hand tracking, structured gesture validation, and detailed pedagogical feedback within a single coherent system.

L4. Lack of transparency in evaluation. *Deep learning*-based models make it difficult to extract clear and adjustable pedagogical explanations.

These limitations highlight the opportunity to develop an immersive platform that combines precise hand tracking, structured and transparent gesture validation, and pedagogical feedback aimed at progressive improvement.

The system developed in this work is positioned at this intersection, leveraging current virtual reality and *hand tracking* capabilities to build a system focused on guided sign language learning. In contrast to predominant *deep learning*-based approaches, the proposed solution adopts a deterministic and interpretable approach that enables the analysis of specific gesture components and the delivery of detailed pedagogical feedback during practice.

The next chapter defines the objectives, methodology, and technical decisions derived from the limitations identified in the literature.

Chapter 4

Project Definition

4.1 Justification

As described in Chapter 1, a significant portion of current sign language recognition systems prioritizes recognition accuracy over pedagogical value, providing limited feedback during the learning process [11]. This work aims to develop a platform focused on guided gesture learning within an immersive environment.

While systems such as those presented in [7, 8] have explored the integration of augmented reality and artificial intelligence in sign language education, this work specifically focuses on structured gesture validation with detailed visual feedback within a fully immersive virtual reality environment.

Virtual reality offers specific advantages for this educational context. Unlike two-dimensional tools, it allows the integration of active practice, three-dimensional visualization of gestures, and real-time feedback within a single interactive environment [12, 13, 6].

The objective of this project is to move from tools that only detect gestures to a platform that integrates teaching, practice, and structured evaluation of gesture execution. Specifically, the system integrates:

- Teaching through three-dimensional visual references.
- Active practice with controller-free hand tracking.
- Structured gesture validation through joint-level analysis.
- Detailed feedback oriented towards progressive improvement.

In this way, technology is used as a learning support tool rather than solely as a gesture recognition system.

4.2 Objectives

The main objective of this work is to design and implement an immersive system for learning American Sign Language (ASL), based on virtual reality and real-time hand tracking. The system is oriented towards guided practice of manual gestures and their structured validation, providing immediate visual feedback during the learning process.

Based on this general objective, the following specific objectives are defined:

4.2.1 Specific Objectives

O1. Design an immersive virtual reality environment that enables natural interaction through controller-free hand tracking, without requiring additional physical devices.

O2. Implement a hand tracking system capable of capturing the user's hand configuration and movement during sign execution.

O3. Develop differentiated validation mechanisms for static and dynamic gestures, evaluating user performance by comparison with predefined references and considering spatial, temporal, and motion sequence criteria.

O4. Integrate an immediate visual feedback system that informs the user about gesture correctness and indicates which components need to be corrected.

O5. Design and incorporate animated reference hands within the immersive environment to serve as visual guidance during practice.

O6. Evaluate system functionality by verifying the consistency of the validation process, the stability of the virtual reality environment, and the clarity of the feedback, taking into account basic usability and accessibility criteria.

4.3 Methodology

The project is developed following an experimental approach focused on the design and implementation of a functional system. The methodology combines conceptual design, iterative development, and progressive integration of modules, together with functional testing and usability evaluation. The objective is to build an immersive platform that allows users to practice manual gestures, receive immediate feedback, and progressively improve their execution.

4.3.1 Development Stages

Stage 1: Conceptual and technical design. In this phase, the overall system architecture and the user interaction flow within the virtual reality envi-

ronment were defined. The main system modules were structured, namely hand tracking, gesture validation, and feedback generation, and the static poses and dynamic sequences used as references for the validation process were formalized.

Stage 2: Development and integration. In this stage, the modules defined in the previous phase were implemented within the Unity environment. The *hand tracking* system, deterministic validation logic, and visual feedback system were integrated. Development was carried out iteratively, allowing adjustments to tolerances, temporal parameters, and interaction aspects based on observed behavior during testing.

Stage 3: Testing and validation. Finally, functional tests were conducted to verify correct gesture detection and validation, the stability of the immersive environment, and the consistency of the feedback provided to the user. In addition, usability and interaction clarity were evaluated to ensure that the system was understandable and easy to use.

4.3.2 Technical Decision: Deterministic Approach

In an initial phase, the use of *Machine Learning* models trained on public sign language datasets, such as WLASL [22], was considered. However, since the main objective of the system is to support gesture learning and provide clear feedback during practice, a deterministic approach based on the direct analysis of the hand's skeletal structure was selected.

This approach uses the data provided by the *hand tracking* system (XR Hands) to analyze hand configuration and compare it with previously defined reference poses.

This approach allows:

- Direct analysis of hand configuration.
- Definition of explicit tolerances for gesture validation.
- Identification of deviations in finger positioning.
- Generation of clear feedback during practice.

This approach is better aligned with the system's objective, which is not only to recognize a sign, but to support guided practice and progressive improvement of gesture execution.

4.4 Planning and Cost Estimation

4.4.1 Project Timeline

The project was developed over a period of six months (September 2025 – February 2026), covering literature review, architectural design, implementation, testing, and thesis writing.

Activity	**Sep**	**Oct**	**Nov**	**Dec**	**Jan**	**Feb**
State of the art review	•	•				
Tool selection	•	•				
Architecture design		•	•			
VR environment design			•			
Design of poses and dynamic gestures			•	•		
Implementation of validation system			•	•		
Feedback integration					•	
Reference hand design					•	•
Testing and evaluation						•
Thesis writing		•	•	•	•	•

Table 4.1: Project timeline.

4.4.2 Cost Estimation

As this work is a Master's Thesis developed within an academic and research context, the cost estimation is presented as an approximate valuation of the resources used during system development.

Hardware resources:

- Meta Quest 3 virtual reality headset (128GB): 550 €
- Personal development computer: 1,500 €
- Laboratory workstation: available resource

Software and development tools:

- Unity 6 Personal: 0 €
- Visual Studio Community: 0 €
- Git and GitHub: 0 €
- Blender: 0 €
- LaTeX: 0 €

Development cost

System development was carried out over an approximate period of six months within the context of a research stay. During this period, development work was conducted under an approximate monthly compensation of 2,500 USD.

Considering a six-month development period, the total cost associated with engineering work can be estimated as:

- System development (6 months × 2,500 USD/month): 15,000 USD

Total estimated project cost

- Hardware resources: 2,050 €
- System development: 15,000 USD

This estimation reflects the approximate cost of the material resources and development effort required for the implementation of the prototype presented in this work.

4.5 Considerations

The system is conceived as a tool to support learning, not as a replacement for instruction provided by native deaf users or certified instructors.

The scope of the project is limited to a set of basic American Sign Language (ASL) gestures. It is acknowledged that sign language presents regional and dialectal variations that are not covered in this implementation.

Hand tracking data is processed locally on the device, without transmission to external servers, ensuring user privacy.

Accessibility is considered a design principle from the system architecture, prioritizing natural interaction without additional physical devices and clear feedback that facilitates gesture understanding.

The next chapter describes the developed system, presenting its architecture, the modules that compose it, and the main implementation decisions adopted.

Chapter 5

Developed System

This chapter provides a detailed description of the system developed in this project. First, Section 5.1 presents the overall system architecture and the interaction flow during gesture practice.

Next, Section 5.2 describes the hand tracking and representation system based on XR Hands. Section 5.3 explains how static poses are defined and managed to represent individual signs.

Section 5.4 then introduces the recognition of dynamic gestures and the mechanisms used to analyze movement trajectories. Subsequently, Section 5.5 describes the pedagogical feedback system that informs the user about the correctness of the performed gesture.

Finally, Sections 5.6 and 5.7 address, respectively, the guide-hand system used to display reference signs and the design of user interaction within the virtual reality environment.

5.1 General system architecture

The system developed in this project allows the user to learn and practice American Sign Language (ASL) gestures within a virtual reality environment, by observing a visual reference of the sign and receiving immediate feedback on the execution of their own gesture.

The system architecture combines hand tracking, hand configuration analysis, and pedagogical feedback generation within a single interactive loop. In this way, the user can observe the reference gesture, attempt to reproduce it, and receive real-time guidance on possible execution errors. To achieve this, the system must address three specific technical functions:

1. Capture the configuration and movement of the user's hand in real time.
2. Determine whether this configuration corresponds to the target sign.

3. Provide a clear response indicating whether the sign is correct and, if not, which specific component needs to be corrected.

This process runs continuously while the user practices a gesture, enabling learning based on repetition and immediate correction.

The architecture is organized into different modules that manage data acquisition, pose analysis, and visual feedback generation within the virtual reality environment, following the flow:

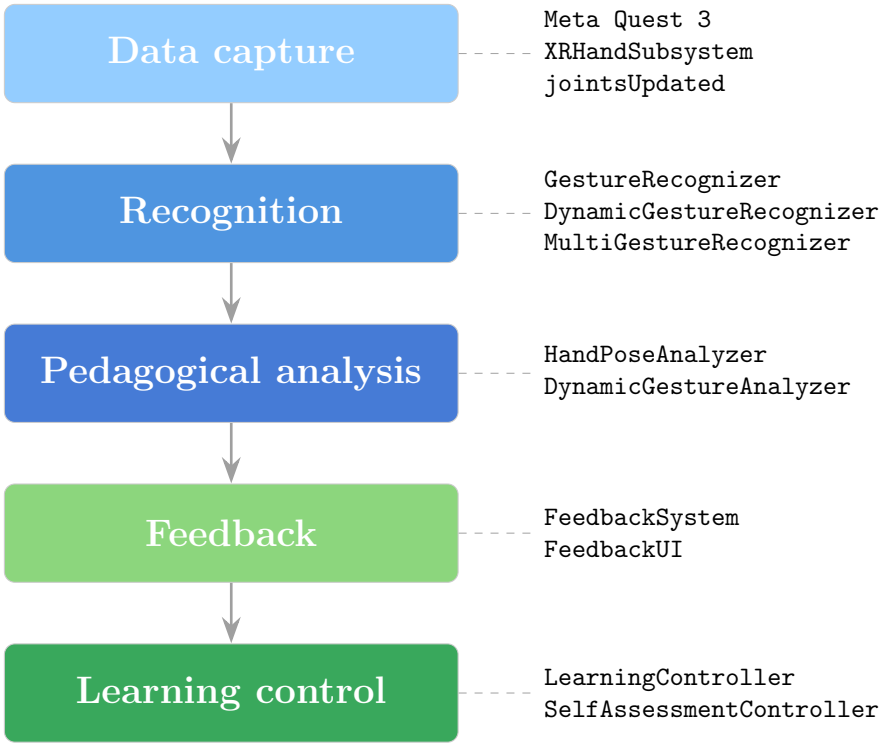


Figure 5.1: General system architecture organized into functional layers.

This separation avoids mixing data capture, recognition, and presentation, and makes debugging and parameter tuning easier by allowing each module to be adjusted independently.

5.1.1 Scope and structure of the system

In its current state, ASL_LearnVR implements a total of **84 signs** organized into **7 pedagogical categories**: alphabet (A–Z), digits (0–9), basic communication, colors, days of the week, months of the year, and basic verbs. Of these, **50**

are dynamic (requiring temporal analysis of movement trajectories) and **34 are static**.

The system is organized as a progressive practice platform in which the user learns individual gestures within each category. During each exercise, a visual reference of the gesture is shown through animated guide hands that illustrate the correct hand configuration. The user then attempts to reproduce the gesture and receives immediate visual feedback for each finger, indicating which components of the hand configuration need to be corrected.

Additionally, the platform includes a self-assessment mode that allows the user to check their progress across the full set of signs in each category.

5.1.2 Rule-based recognition

The recognition implemented in ASL_LearnVR is not based on machine learning models or trained classifiers. Instead, it uses a deterministic approach based on configurable geometric and temporal conditions for each sign. Each gesture is described through adjustable parameters such as tolerances, minimum confirmation times, and movement thresholds. As a result, for the same hand configuration and the same parameter set, the system always produces the same evaluation of the gesture.

This approach makes it possible not only to determine whether a sign has been performed correctly, but also to identify which specific component does not satisfy the expected conditions. This property is especially relevant from a pedagogical point of view, as it enables the generation of concrete visual feedback for the user.

The following sections describe in more detail how this recognition is implemented for both static and dynamic gestures.

5.1.3 General interaction flow

During practice, the system continuously executes the following cycle:

1. **Capture:** the headset obtains the hand configuration and movement through controller-free optical tracking.
2. **Analysis:** geometric parameters derived from the hand configuration are computed, such as finger flexion (*curl*), finger separation (*spread*), or orientation, and are compared with the definition of the target gesture.
3. **Evaluation:** the system determines whether the observed configuration satisfies the constraints defined for the sign, analyzing each finger independently.
4. **Feedback:** visual cues are generated within the virtual reality environment to indicate whether the gesture is correct and which components require adjustment.

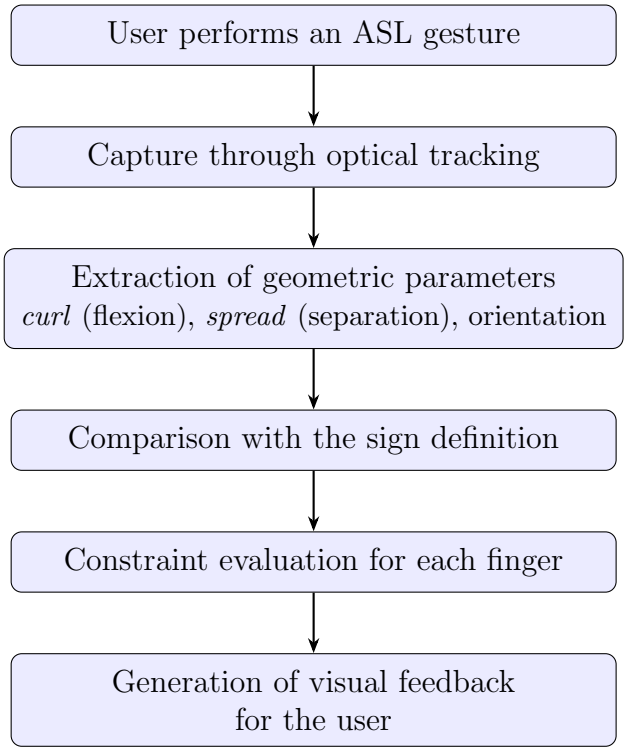


Figure 5.2: Interaction cycle during gesture practice.

This cycle is repeated continuously while the user practices, enabling immediate correction and supporting a learning process based on repetition and progressive adjustment.

5.1.4 Main system components

At the implementation level, the system is composed of several specialized modules responsible for hand data acquisition, gesture recognition, execution analysis, and pedagogical feedback generation within the virtual reality environment. Figure 5.3 shows the relationship between these modules.

XRHandSubsystem. This component provides access to the headset's hand tracking system through the *XR Hands* API. Through this subsystem, the position and orientation of the hand joints (*joints*) are obtained, forming the basis for gesture recognition.

GestureRecognizer. Component responsible for static gesture recognition. Each instance is configured with a single target sign and continuously evaluates whether the current hand pose matches the expected definition, applying a minimum hold time before confirming detection. When a match occurs, it emits events that are received and processed by the `FeedbackSystem`.

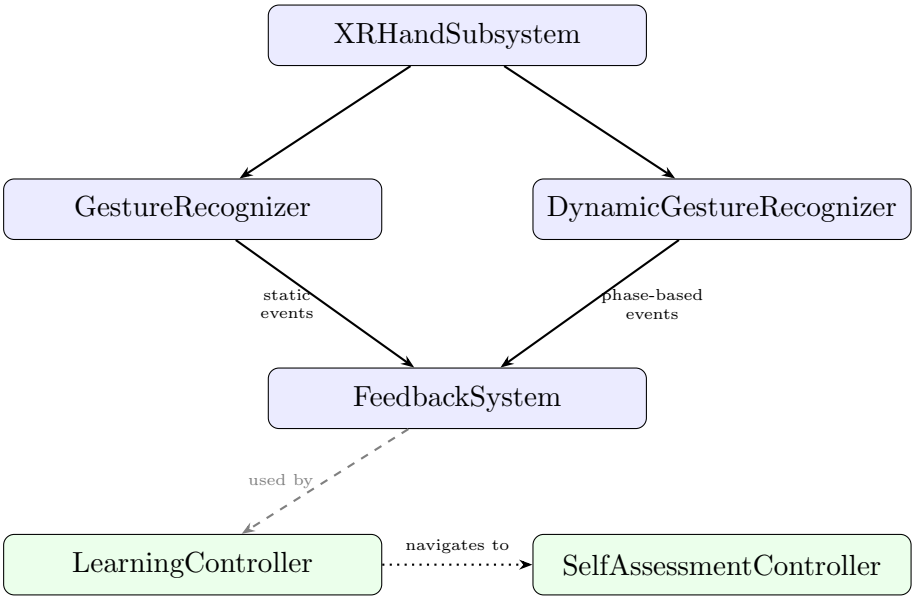


Figure 5.3: Relationship between the main modules of *ASL_LearnVR*.

DynamicGestureRecognizer. Responsible for detecting gestures that involve movement over time. This component analyzes the temporal evolution of hand position and emits structured events by phase (start, progress, completion, and failure), which the `FeedbackSystem` listens to in order to generate feedback adapted to each stage of the movement.

FeedbackSystem. Central feedback module and main orchestrator of the architecture. It subscribes to and processes the events emitted by `GestureRecognizer` and `DynamicGestureRecognizer`, and coordinates the remaining subcomponents `HandPoseAnalyzer`, `DynamicGestureAnalyzer`, `FeedbackUI`, and the finger visualizers. It applies hysteresis and debounce mechanisms to stabilize messages and manages the different feedback states (*Waiting*, *InProgress*, *ShowingErrors*, *Success*).

HandPoseAnalyzer. Internal subcomponent of the `FeedbackSystem` specialized in static gestures. It computes geometric parameters derived from the hand configuration, such as finger flexion (*curl*), finger separation (*spread*), and overall orientation, and compares them with the constraint profiles defined for each sign. Its function is to identify which fingers deviate from the expected pose and generate finger-specific correction messages in order to provide descriptive feedback to the user.

DynamicGestureAnalyzer. Internal subcomponent of the `FeedbackSystem` specialized in dynamic gestures. It manages feedback progression through the movement phases, as waiting for the initial pose (*Idle*), detection of movement

onset (*StartDetected*), movement tracking (*InProgress*), and final outcome (*Completed* or *Failed*). At each phase, it generates contextual messages that guide the user during gesture execution.

LearningController. Controller for the learning scene. It manages the gesture practice logic, configuring the `FeedbackSystem` with the active sign, controlling the playback of the animated guide hand, and coordinating navigation between signs within each category.

SelfAssessmentController. Controller for the self-assessment scene. It allows the user to practice all the signs of a category freely and without a fixed order. It uses `MultiGestureRecognizer` to simultaneously detect any sign in the active category, records the obtained results, and updates the global learning progress view.

The set of modules described above operates in a coordinated manner. The `XRHandSubsystem` continuously captures the position and orientation of the hand joints and notifies `GestureRecognizer` and `DynamicGestureRecognizer` through events. The `GestureRecognizer` evaluates whether the current pose matches the target sign and, after maintaining it for the required minimum time, emits a detection event. In parallel, the `DynamicGestureRecognizer` monitors the temporal evolution of movement through its internal state machine, emitting phase-structured events. In both cases, the `FeedbackSystem` receives these events and immediately generates the corresponding visual and textual feedback, thus closing the interaction loop between the user and the system.

5.2 Hand capture and representation

Gesture recognition in *ASL_LearnVR* is based on the real-time analysis of the user's hand configuration. To achieve this, the system uses the optical hand tracking provided by the Meta Quest 3 headset, which supplies the position and orientation of the hand joints through the *XR Hands* API.

From these data, the system derives a geometric representation of the hand pose based on parameters such as finger flexion, finger separation, and palm orientation. This representation constitutes the basis on which sign execution is evaluated.

This section describes the capture mechanism, the hand model provided by *XR Hands*, the derived geometric parameters used for pose analysis, and the limitations of optical tracking that have influenced the system design.

5.2.1 Hand tracking with XR Hands

Hand tracking in *ASL_LearnVR* is performed using the *Unity XR Hands* package, which provides access to the hand tracking data delivered by the Meta Quest 3 headset without the need for physical controllers. The system relies on the `XRHandSubsystem`, which acts as an abstraction layer between the headset hardware and the recognition modules implemented in the application.

Hand information is received through the `XRHandTrackingEvents` component, which triggers the `jointsUpdated` event at each subsystem update. This event provides an `XRHandJointsUpdatedEventArgs` object containing the full configuration of the hand joints at that moment.

The gesture recognition modules (`GestureRecognizer` and `DynamicGestureRecognizer`) subscribe to this event in order to automatically receive updates of the hand configuration whenever the tracking system produces new data.

In addition, the `HandTrackingStatus` component monitors the tracking state of both hands and emits events when both become correctly detected (`onBothHandsTracked`) or when tracking of either hand is lost (`onHandsLost`). This allows the application to properly manage tracking interruptions during gesture practice.

5.2.2 Internal hand model

The hand data model provided by *XR Hands* represents each hand through a set of 26 *joints*. As illustrated in Figure 5.4, these points cover the wrist (*Wrist*), the palm (*Palm*), and, for each of the five fingers, the metacarpal (*Metacarpal*), proximal phalanx (*Proximal*), intermediate phalanx (*Intermediate*), distal phalanx (*Distal*), and fingertip (*Tip*). In the case of the thumb, which has no intermediate phalanx, the distal phalanx is used instead.

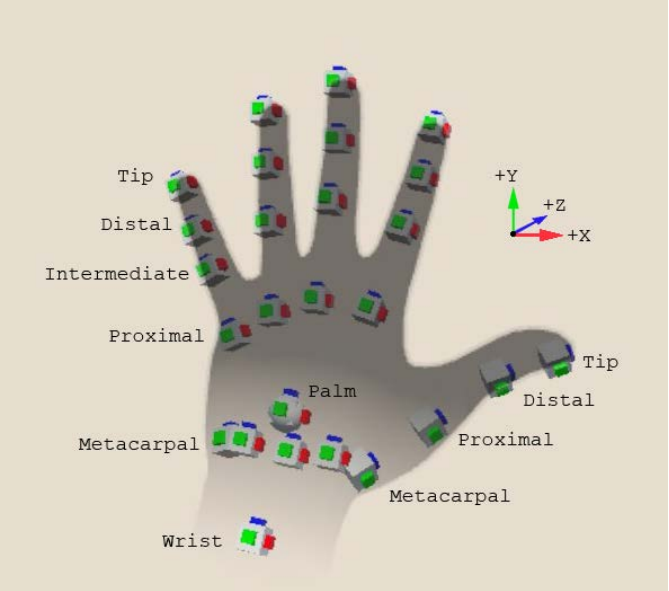


Figure 5.4: Hand data model exposed by *Unity XR Hands*, showing the 26 tracked *joints* labeled by anatomical level. Source official Unity XR Hands documentation [24].

Each *joint* provides a `Pose` composed of position (`Vector3`) and orientation (`Quaternion`) in the headset reference space. Access to a specific joint is performed through the `TryGetPose` method of the `XRHandJoint` object, which returns `false` if the joint is not available at that moment, allowing partial tracking losses to be handled without interrupting the analysis cycle.

5.2.3 Representation of hand configuration

The geometric description of hand configuration in *ASL_LearnVR* is based on two complementary representations, the *finger shapes* defined by the *XR Hands* API, used for the global detection of each sign, and a set of internally computed geometric parameters, used for finger-level pedagogical diagnosis.

XR Hands finger shapes. The *XR Hands* API defines a set of parameters, referred to as *finger shapes*, that describe the configuration of each finger based on the hand joints [25]. These parameters are expressed as normalized values and allow the hand shape to be characterized in a compact way. They include measures of finger flexion (*Full Curl*), the distribution of that flexion between the base joint and the distal phalanges (*Base Curl* and *Tip Curl*), the proximity between the thumb tip and other fingers (*Pinch*), and the angular separation between adjacent

fingers (*Spread*). These parameters are used as the basis for the global evaluation of hand configuration in each sign.

Each sign is defined in an asset of type `XRHandShape` or `XRHandPose`, where the relevant finger shape to evaluate, the target value, and the upper and lower tolerances are specified for each relevant finger. Validation is performed internally by `XRHandShape.CheckConditions()`, which returns a global yes/no result without indicating which specific finger failed to satisfy the conditions.

Table 5.1 shows the configuration of four representative signs that illustrate the use of the different *finger shapes* available in *XR Hands*.

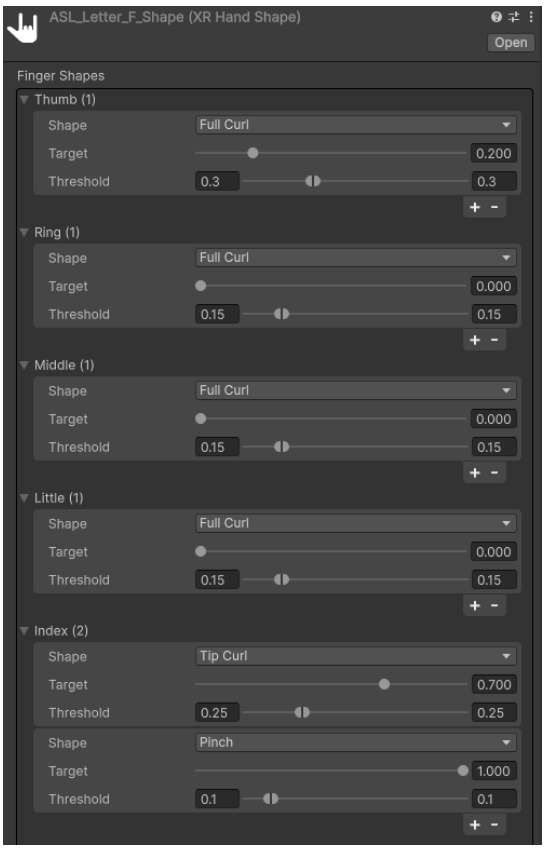
Table 5.1: Configuration of `XRHandShape` for four representative letters of the ASL alphabet.

Finger	**Shape**	**Target**	**Tolerance**
A *(closed fist, thumb at the side)*			
Thumb	Full Curl	0.10	±0.20
Index	Full Curl	1.00	±0.15
Middle	Full Curl	1.00	±0.15
Ring	Full Curl	1.00	±0.15
Little	Full Curl	1.00	±0.15
E *(tips curved toward the knuckles, base extended)*			
Thumb	Full Curl	0.70	±0.35
Index	Tip Curl	0.90	±0.20
	Base Curl	0.10	±0.20
Middle	Tip Curl	0.90	±0.20
	Base Curl	0.10	±0.20
Ring	Tip Curl	0.90	±0.20
	Base Curl	0.10	±0.20
Little	Tip Curl	0.90	±0.20
	Base Curl	0.10	±0.20
F *(curved index finger touching the thumb, others extended)*			
Thumb	Full Curl	0.20	±0.30
Index	Tip Curl	0.70	±0.25
	Pinch	1.00	±0.10
Middle	Full Curl	0.00	±0.15
Ring	Full Curl	0.00	±0.15
Little	Full Curl	0.00	±0.15

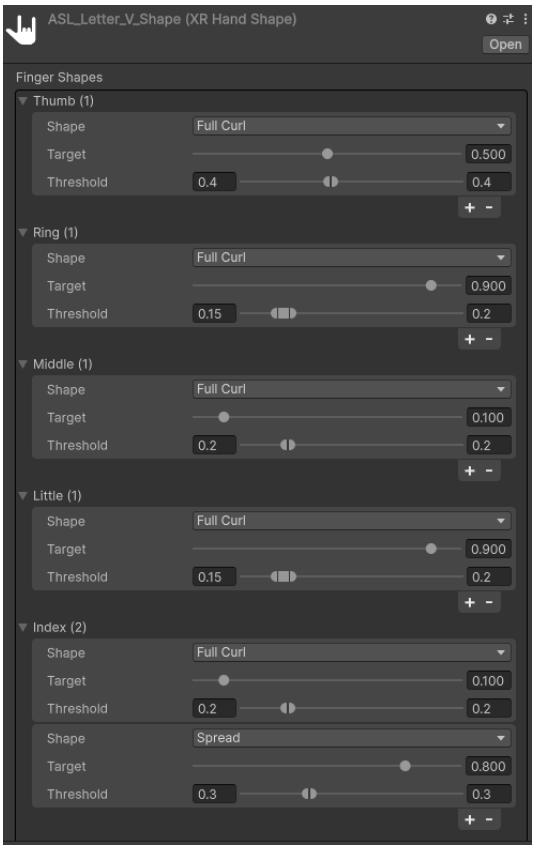
(Continuation of Table 5.1)

Finger	Shape	Target	Tolerance
V	*(index and middle fingers extended and separated, others closed)*		
Thumb	Full Curl	0.50	±0.40
Index	Full Curl	0.10	±0.20
	Spread	0.80	±0.30
Middle	Full Curl	0.10	±0.20
Ring	Full Curl	0.90	±0.20
Little	Full Curl	0.90	±0.20

To illustrate how these configurations are implemented within Unity, Figure 5.5 presents two representative examples using `XRHandShape`.



(a) Letter F configuration: combination of `Tip Curl` and `Pinch` constraints.



(b) Letter V configuration: `Spread` together with `Full Curl` constraints.

Figure 5.5: Representative examples of `XRHandShape` configurations in Unity.

Custom geometric parameters for diagnosis. Since `XRHandShape.CheckConditions()` only returns a global validation result and does not identify which specific finger caused the failure, the system implements its own set of geometric calculations within the `HandPoseAnalyzer` in order to generate finger-specific feedback. From the hand joints, four main geometric parameters are computed.

The first is **curl**, which quantifies the degree of flexion of each finger as a normalized value in the range $[0, 1]$. For each finger, three direction vectors are computed between consecutive segments (proximal→intermediate, intermediate→distal, and distal→tip). From these, the two angles formed between consecutive finger segments are obtained. The final *curl* value is calculated as the average of both angles, normalized through linear interpolation between 180° (fully extended finger) and 60° (fully closed finger). In the case of the thumb, whose range of motion is different, the angle is computed between the proximal→intermediate vector and the distal→tip vector, omitting the intermediate segment, and normalized between 180° and 50°.

The second parameter is **spread**, which measures the angular separation between adjacent fingers. For each pair of consecutive fingers, the proximal→tip vector is computed for each finger and the angle between both vectors is obtained. This parameter makes it possible to distinguish configurations such as the letter *V*, where the index and middle fingers are separated, from the letter *U*, where both fingers remain together.

The third parameter is the **distance between fingertips**, which evaluates the proximity between the thumb tip and the tip of another finger through the three-dimensional distance between both positions. This parameter is used in signs where the thumb must make contact with another finger, such as the letters *F*, *K*, *M*, *N*, *P*, or *T*.

The fourth parameter is **palm orientation**, which computes the angle between the *forward* vector of the `Palm` joint and the expected direction defined in the sign profile. This parameter is only evaluated when the profile explicitly enables the `checkOrientation` condition, since most signs do not require a specific hand orientation.

These four parameters are compared against the ranges defined in `AlphabetConstraintProfiles` and `DigitConstraintProfiles`, which describe the static signs of the alphabet and the digits from 0 to 9. These same parameters also form the basis of dynamic gesture recognition, since such gestures include an initial static pose followed by a specific movement trajectory. The separation between both layers is detailed in Section 5.4.

5.2.4 Limitations of optical tracking

Optical hand tracking presents inherent limitations related to the headset hardware that affect the design of the recognition system. These limitations do not depend on the application design itself, but on the capture technology used, and they have been taken into account in the design of the different modules.

The main limitation is **finger occlusion**, when one finger is hidden behind another from the perspective of the headset cameras, the subsystem may not have reliable data for that *joint* at that moment. To mitigate this effect, the `HandPoseAnalyzer` preserves the last valid *curl* value of each finger whenever `TryGetPose` returns `false`, avoiding abrupt changes in the visual feedback.

The second relevant limitation is **tracking loss under certain hand orientations**, especially when the palm faces the user's body or when the wrist is in extreme positions. The `DynamicGestureRecognizer` incorporates a tracking-loss tolerance of 0,5 seconds (`TRACKING_LOSS_TOLERANCE`), so that brief interruptions do not cancel an ongoing gesture.

Finally, **noise in position and orientation measurements** may generate variations in *curl* values and in the hand trajectory. This effect is mitigated through two mechanisms, the additional tolerance (`curlTolerance`) applied to the constraint ranges in the `HandPoseAnalyzer`, and the position and orientation smoothing applied in the `DynamicGestureRecognizer` before computing kinematic metrics.

In addition to these general limitations of optical tracking, certain hand configurations in the ASL alphabet present further recognition difficulties. In particular, the letter *R* requires the index and middle fingers to cross, a configuration that optical tracking cannot reliably capture, since from the camera perspective the fingers appear together rather than crossed. The system therefore defines this letter as a configuration with the fingers together and minimal separation, distinct from the letter *U*, and informs the user through the sign description how it should actually be performed.

Similarly, the letters *M*, *N*, and *T*, which require placing the thumb between different fingers, present greater ambiguity in estimating the thumb's relative position and produce more variability in the geometric values used for recognition.

Finally, the signs *P* and *Q*, whose configuration involves a downward hand orientation, are more sensitive to momentary tracking loss. Although these signs can be correctly recognized and have been defined in the system, their execution requires greater precision from the user.

5.3 Definition and management of static poses

In the context of the developed system, a *static pose* is defined as a stable hand configuration in which the relative shape of the fingers and the overall orientation of the hand remain approximately constant over a short period of time. This type of sign does not depend on a movement trajectory, but rather on the geometric position of the fingers.

Many basic units of American Sign Language (ASL), such as alphabet letters or numbers, can be described through this type of configuration. In these cases, sign recognition consists of verifying whether the current hand configuration satisfies a set of predefined geometric constraints for each finger.

Within the system, each static pose is described by a set of conditions specifying finger flexion, finger separation, possible fingertip contacts and, when necessary, palm orientation. These conditions make it possible to distinguish between similar configurations, for example, between the letters V and U, which differ only in the separation between the index and middle fingers.

Unlike dynamic gestures, which require analyzing the trajectory of the hand over time, static poses are identified by evaluating only the geometric configuration of the hand at each moment. However, to avoid incorrect detections caused by tracking noise or momentary matches, the system requires the pose to remain stable for a short interval before confirming it as a recognized sign.

This type of sign forms the basis of the learning system: 34 of the 84 implemented signs are identified as static poses, corresponding to the alphabet, with the exception of the letters J and Z, which involve movement, and to the digits from 0 to 9.

5.3.1 Sign representation

Each static sign is encapsulated in a *ScriptableObject* of type `SignData`, which groups all the information required for its detection and pedagogical presentation. Its main fields are:

- `signName`: textual identifier of the sign, used to associate it with its corresponding constraint profile in `HandPoseAnalyzer`.
- `description`: optional descriptive text shown to the user during learning.
- `handShapeOrPose`: reference to the asset of type `XRHandShape` or `XRHandPose` that defines the geometric conditions of the sign.
- `requiresMovement`: indicates whether the sign requires a movement trajectory, which determines which recognition pipeline is activated.

- `minimumHoldTime`: minimum time in seconds that the pose must be maintained before being confirmed (default value: 0,3 s).
- `icon`: sprite associated with the sign for its representation in the interface.

The distinction between `XRHandShape` and `XRHandPose` is relevant, since `XRHandShape` evaluates only the finger configuration, whereas `XRHandPose` adds a constraint on the relative orientation of the hand with respect to the camera. In the system, signs whose correct execution depends on hand direction, such as G, H, K, L, P, Q, or U, are defined using `XRHandPose`.

Figure 5.6 shows a screenshot of the inspector for the `SignData` asset corresponding to the letter A. The detection fields `handShapeOrPose`, `requiresMovement`, and `minimumHoldTime` can be seen, as well as the reference to the `ASL_Letter_A_Shape` asset of type `XRHandShape`.

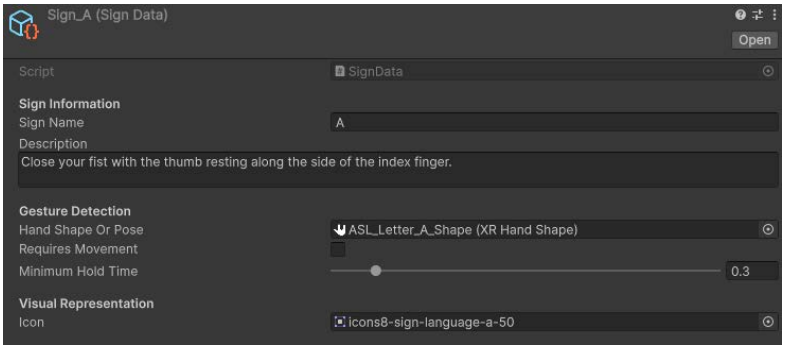
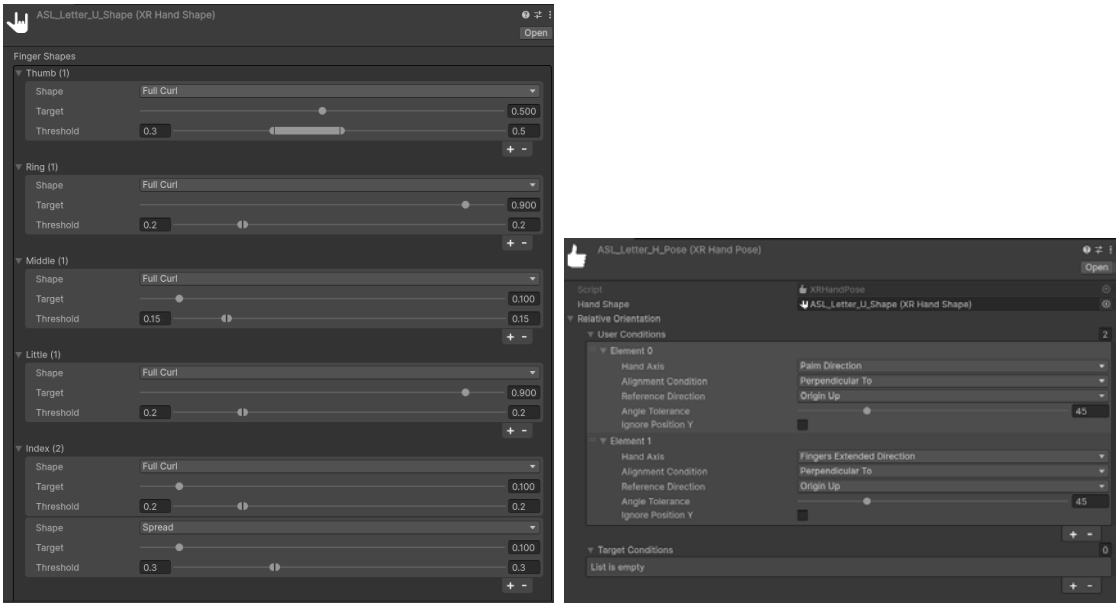


Figure 5.6: Inspector of the `SignData` asset for the letter A.

Figure 5.7 illustrates this distinction with two specific signs. The letter U is defined only through an `XRHandShape` that specifies the configuration of the index and middle fingers extended together, with the remaining fingers closed. The letter H shares exactly the same finger configuration (referencing the same `XRHandShape` as U) but is defined as an `XRHandPose`, adding two orientation constraints, where the palm must be perpendicular to the vertical axis (*Palm Direction*) and the fingers must point horizontally (*Fingers Extended Direction*), both with a tolerance of 45°.

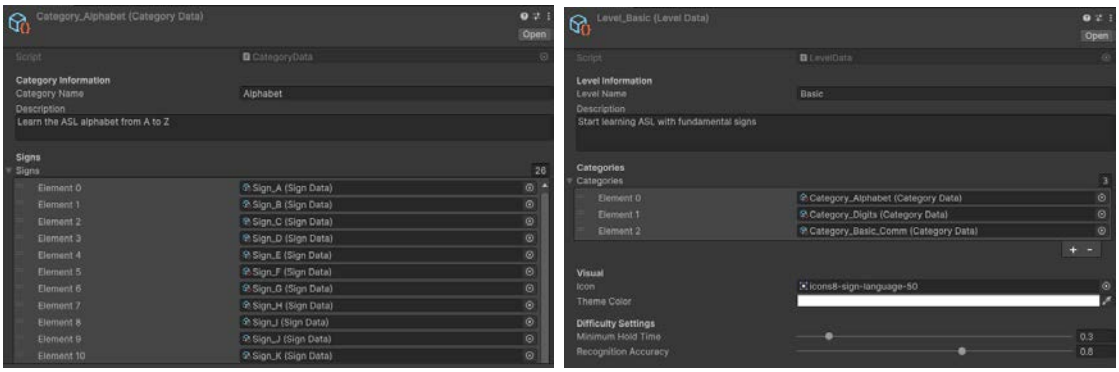
The `SignData` assets do not contain level or category information. This information is managed through an independent hierarchy of *ScriptableObjects*. `LevelData` defines a learning level with its name, description, icon, interface theme color, and a global `minimumHoldTime` that establishes the minimum pose hold time for that difficulty level. Each `LevelData` groups a list of `CategoryData`, and each `CategoryData` groups a set of `SignData` under a specific category name and description. This structure enables reuse of the same sign across different contexts without data duplication.



(a) `XRHandShape` for the letter U. (b) `XRHandPose` for the letter H.

Figure 5.7: Distinction between `XRHandShape` and `XRHandPose`.

Figure 5.8 illustrates the hierarchical organization of the data structures used in the system.



(a) `CategoryData` for the alphabet, referencing 26 `SignData` assets. (b) `LevelData` for the basic level, grouping three categories.

Figure 5.8: `LevelData` groups categories, and each `CategoryData` references a list of `SignData`.

5.3.2 Validation through XRHandShape

The global validation of a static pose is performed through the method `XRHandShape.CheckConditions(XRHandJointsUpdatedEventArgs)`, which compares the current state of the hand *joints* with the conditions defined in the asset and returns `true` if all conditions are satisfied simultaneously, or `false` otherwise [25]. When the sign is defined through `XRHandPose`, validation additionally includes a check of the relative orientation of the hand with respect to the main camera, which is automatically assigned as `targetTransform` at runtime.

The `GestureRecognizer` component encapsulates this check and integrates it into the detection flow. The evaluation is not performed on every *frame*, but at regular intervals of 0,1 s (`detectionInterval`), which reduces computational load without compromising the response perceived by the user.

5.3.3 Temporal pose confirmation

To avoid false positives caused by momentary tracking fluctuations, pose detection is not confirmed at the first instant when `CheckConditions` returns true. The `GestureRecognizer` implements a temporal confirmation mechanism with two independent thresholds.

The first is `minimumHoldTime`, read directly from the `SignData` of the target sign. The pose must be held continuously during this interval for the `onGestureDetected` event to be emitted. If the condition is lost before the required time has elapsed, the timer is reset.

The second is `detectionLossTolerance` (0,2 s by default), which introduces a tolerance window before considering that the pose has been definitively lost. This prevents momentary tracking interruptions (caused by occlusion or noise) from cancelling an already initiated detection and forcing the user to repeat the gesture from the beginning.

The complete confirmation flow is as follows:

1. If `CheckConditions` returns true for the first time, the start time (`holdStartTime`) is recorded and the `wasDetected` state is activated.
2. In each subsequent evaluation, if the condition continues to hold, the system checks whether the `minimumHoldTime` has elapsed.
3. If the required time is reached, `onGestureDetected` is emitted and `performedTriggered` is activated.
4. If the condition is no longer satisfied, the `detectionLossTolerance` countdown starts. Only if the loss exceeds this threshold is `onGestureEnded` emitted and the state reset.

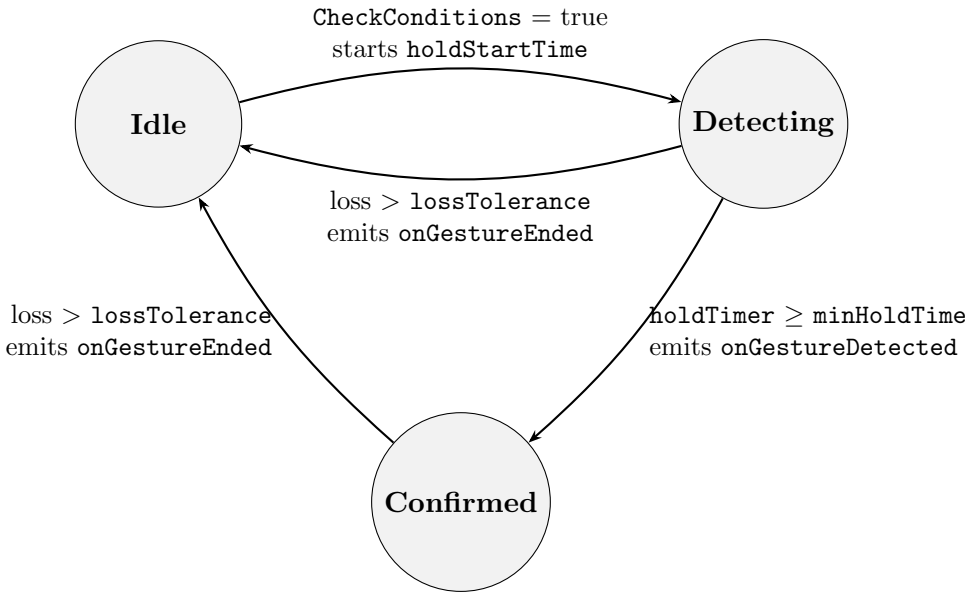


Figure 5.9: State machine of the `GestureRecognizer` for the temporal confirmation of static poses.

5.3.4 Global detection vs finger-level diagnosis

The recognition decision is binary and rests with the `GestureRecognizer`, which evaluates the conditions defined for each sign through `CheckConditions`. In parallel, the `HandPoseAnalyzer` performs an independent geometric analysis for each finger, computing the parameters described in Section 5.2.3 with the goal of identifying which fingers deviate from the target configuration and in what way.

This analysis is independent and does not modify or condition the recognizer's global decision. If the recognizer confirms the sign, the `HandPoseAnalyzer` directly returns success without computing errors. If the recognizer does not confirm it, the `HandPoseAnalyzer` analyzes each finger and generates a list of `FingerError` objects. Each element contains the affected finger, the type of deviation (`FingerErrorType`) and its severity (`Severity`: *Minor* or *Major*).

Semantic error types include, among others, `NeedsExtend` (the finger is too closed), `NeedsCurve` (it should curve without closing completely), `NeedsFist` (it should close into a fist), `SpreadTooNarrow` or `SpreadTooWide` (incorrect finger separation), and `ShouldTouch` (the thumb should make contact with another finger). This list of errors is used by the feedback system to build the correction message shown to the user.

Finger-level pedagogical analysis is parameterized through `FingerConstraint-`

`Profile`, which defines for each finger the expected ranges of *curl*, *spread*, thumb contact, and palm orientation. The profiles corresponding to the alphabet and digits are generated at runtime from the static classes `AlphabetConstraintProfiles` and `DigitConstraintProfiles`, and are automatically registered in the `HandPoseAnalyzer` during scene initialization.

Each `FingerConstraint` within a profile defines, in addition to the numerical ranges, the expected semantic state of the finger (`FingerShapeState`: *Extended*, *Curved*, or *Closed*). This state is automatically derived from the midpoint of the *curl* range if it is not explicitly specified.

The thresholds used to delimit these states are: *Extended* for *curl* values below 0,25, *Curved* for values between 0,25 and 0,75, and *Closed* for values above 0,75. This semantic state is what the system uses to generate correction messages that are understandable for the user, such as *"close the index finger into a fist"* or *"straighten the middle finger"*, instead of displaying raw numerical values.

5.4 Dynamic gesture recognition

Unlike static poses, whose validation is reduced to checking an instantaneous hand configuration, dynamic gestures require analyzing the temporal evolution of movement. In *ASL_LearnVR*, a dynamic gesture is modeled as a sequence composed of an initial pose that triggers recognition, a movement trajectory described through kinematic metrics, and an optional final pose that confirms the correct execution of the sign.

5.4.1 Integration architecture with static recognition

Dynamic gesture recognition does not operate as an isolated module, but rather as an extension of the static recognition system already described in Section 5.3. The system does not attempt to identify a hand configuration from scratch; instead, it uses static detection as the starting point to decide when movement analysis can begin.

To achieve this integration, `DynamicGestureRecognizer` does not directly query the static recognizer, but accesses it through the `IPoseAdapter` interface. This interface defines a minimal communication layer in which the name of the currently detected static pose is first obtained (`GetCurrentPoseName()`) and then it is checked whether the hand is being correctly tracked (`IsHandTracked()`).

In the current implementation, this interface is implemented through `StaticPoseAdapter`, a component that acts as a bridge between `MultiGestureRecognizer` and `DynamicGestureRecognizer`. Its function is to translate the internal state of the static recognizer into a simple and homogeneous form that the dynamic recognizer can query on each *frame*. To do so, `StaticPoseAdapter` uses two complementary mechanisms: subscription to the events `onGestureRecognized` and `onGestureLost` emitted by `MultiGestureRecognizer`, and direct synchronization on each `Update()` with the recognizer's `CurrentActiveSign` property, which guarantees the consistency of the active pose name in each iteration of the main loop.

This design decision has two advantages. First, it separates the dynamic recognition logic from the mechanism used to detect static poses, so the dynamic module does not depend directly on the static recognizer. Second, it makes it possible to replace the source of the information with other compatible adapters without modifying the main dynamic recognition logic.

Based on this integration layer, `DynamicGestureRecognizer` implements recognition through a finite state machine executed continuously in `Update()`. The system distinguishes three states: `Idle`, `PendingConfirmation`, and `InProgress`, represented in Figure 5.10.

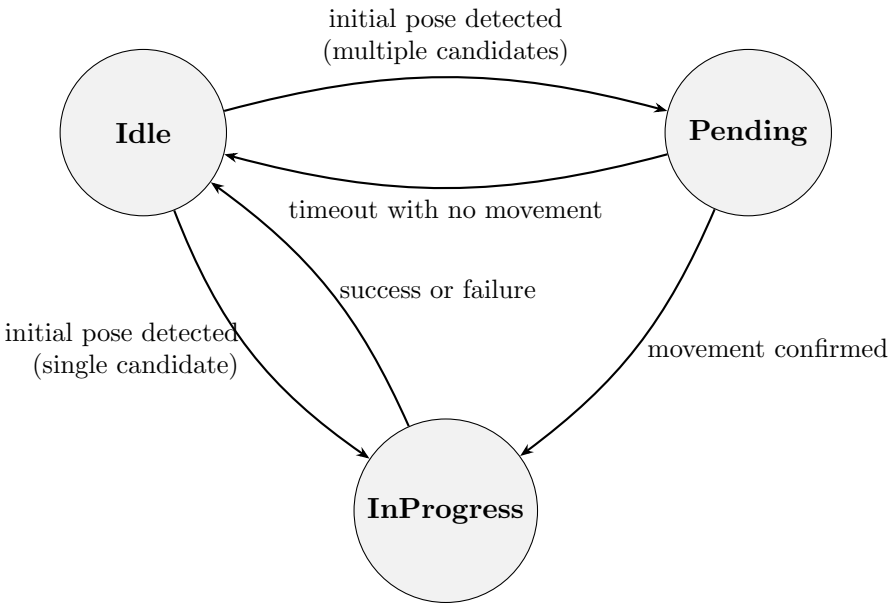


Figure 5.10: State machine of the `DynamicGestureRecognizer`.

In the `Idle` state, the system waits for an initial pose compatible with some dynamic gesture. To do so, it queries the active pose through the adapter in each iteration and compares that information with the start poses defined in the *ScriptableObjects* of type `DynamicGestureDefinition`. In learning mode, this search is restricted to the sign the user is practicing at that moment. In self-assessment mode, by contrast, all available dynamic gestures are considered, since the system must infer which one the user is trying to perform.

When an initial pose matches the start requirements of some gesture, the behavior depends on the number of candidates found. If there is a single compatible gesture, the recognizer moves directly to the `InProgress` state, since there is no ambiguity to resolve. If, on the other hand, several gestures share the same initial pose, the system first goes through `PendingConfirmation`. This intermediate state is necessary to determine, from the initial movement, which of the candidates the user is trying to perform.

During the `PendingConfirmation` state, the system observes whether enough movement appears to consider that the gesture has started. To evaluate this, two indicators computed by `MovementTracker` are used, the distance traveled by the palm and its instantaneous speed.

The criteria are evaluated in priority order and, when one of them matches a compatible candidate, the system transitions to `InProgress`. If the maximum waiting time elapses without being able to identify a compatible candidate, the system returns to the `Idle` state.

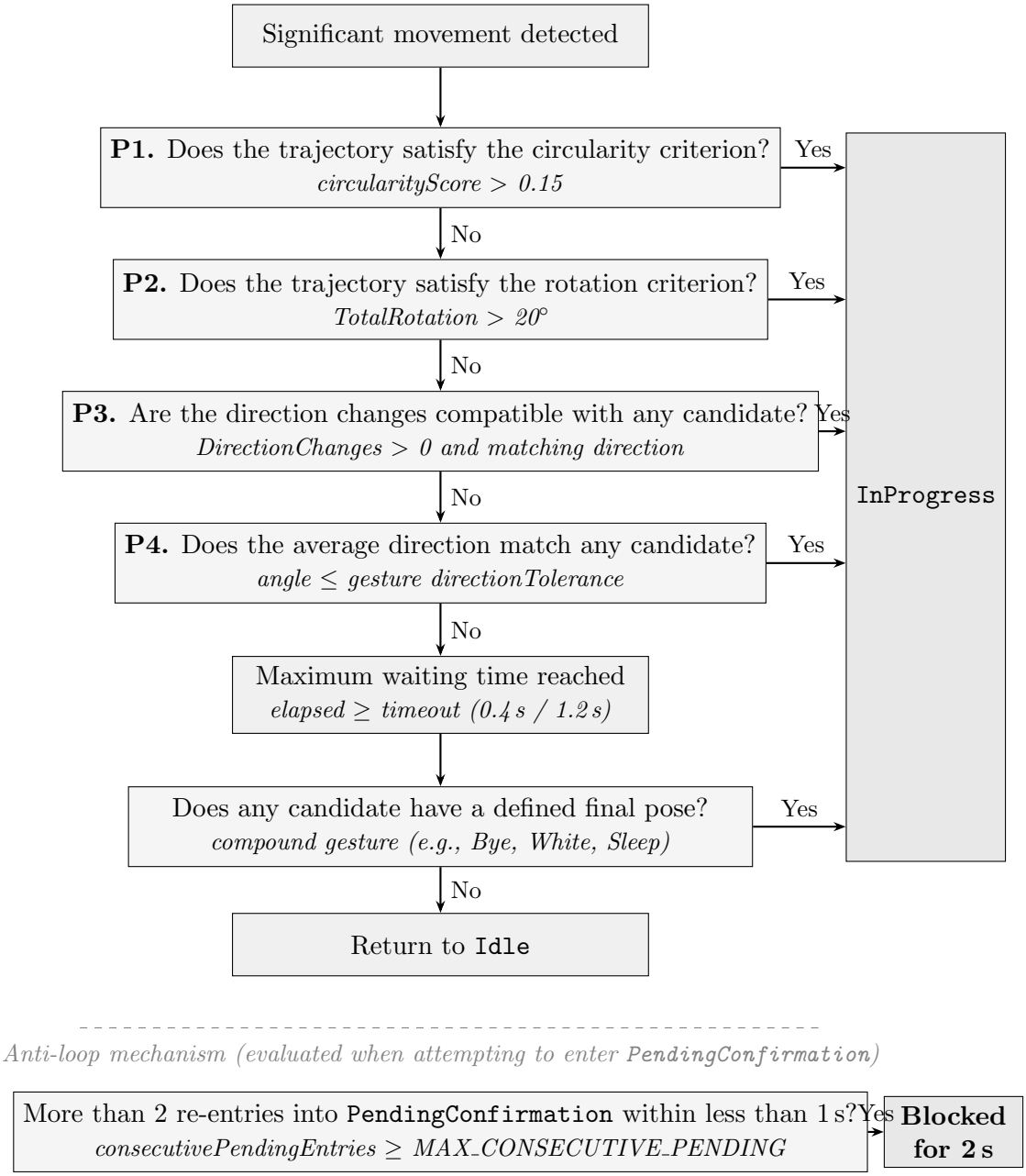


Figure 5.11: Disambiguation logic in `PendingConfirmation`.

Movement is considered significant when the palm moves by approximately more than 0,025 m or when the speed exceeds 0,08 m/s. These values were adjusted empirically through system testing, with the aim of distinguishing between small tracking variations and intentional user movement.

When several gestures share the same initial pose, this first segment of move-

ment is also used to differentiate them. To do so, basic trajectory characteristics are analyzed, such as the main direction of movement, possible rotations, direction changes, or the degree of circularity, depending on the requirements defined for each gesture.

The maximum time spent in `PendingConfirmation` depends on the context. When there is only one candidate gesture, the waiting time is short (0,4 s), since it is only necessary to check that the user has started moving. When several gestures share the same initial pose, the margin is extended to 1,2 s in order to gather more information and disambiguate correctly. If no evidence of movement appears within that interval, the system interprets that no dynamic gesture has been initiated and returns to `Idle`.

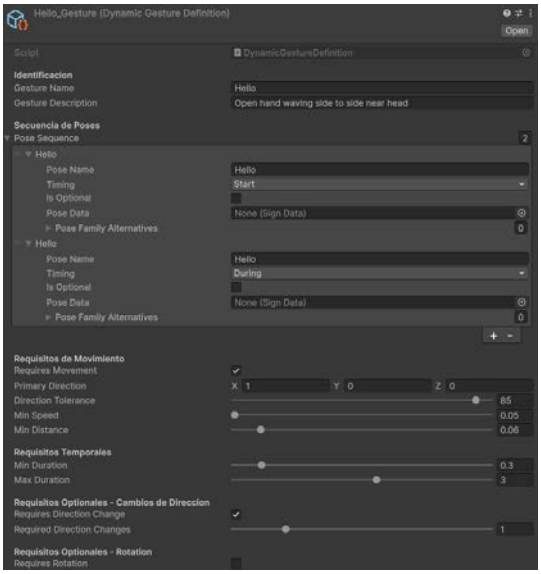
In addition, the system incorporates a protection mechanism to avoid repeated activations when the hand remains close to the gesture start threshold. In these situations, small tracking variations can cause rapid transitions between the `Idle` and `PendingConfirmation` states.

If more than two re-entries into `PendingConfirmation` occur within a one-second window, dynamic recognition is temporarily disabled for two seconds. This prevents the system from entering activation and cancellation cycles when the user maintains a pose close to the detection threshold.

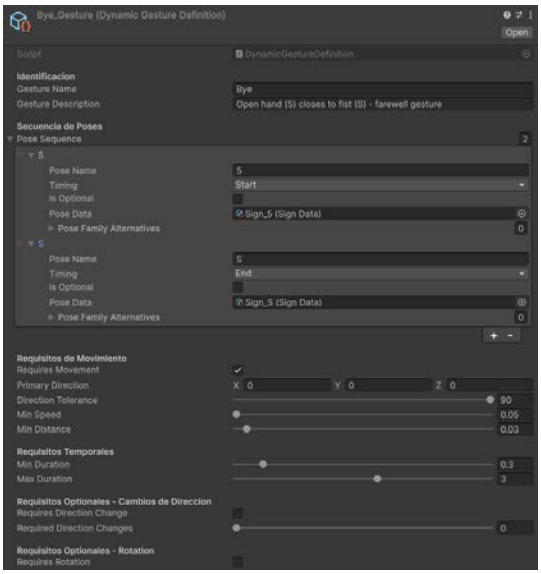
Once the start of the movement has been confirmed, the system enters `InProgress`. In this state, the system no longer evaluates whether the gesture has started, but whether it is being executed correctly. For this purpose, `DynamicGestureRecognizer` continuously analyzes the trajectory of the palm, the duration of the gesture, and, when applicable, the final pose.

Validation is not based on a single condition, but on the combination of the requirements defined in each `DynamicGestureDefinition`. These include the main movement direction, minimum speed, total distance traveled, direction changes, accumulated rotation, trajectory circularity, or the spatial area in which the hand is located.

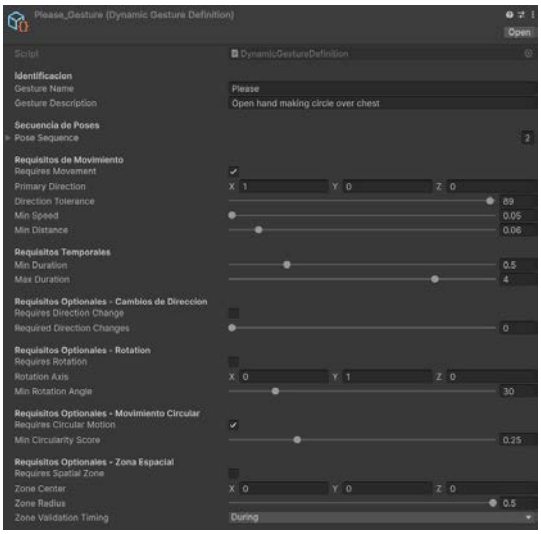
`DynamicGestureDefinition` makes it possible to describe dynamic gestures with different movement patterns using the same recognition mechanism. Figure 5.12 shows four examples taken from the application. The gesture *Hello* is based on a lateral change of direction while keeping the hand open. *Bye* is a compound gesture that begins with the `5` pose (open palm) and ends with the `S` pose (closed palm). *Please* requires a circular movement over the chest, while *Monday* combines wrist rotation and circularity.



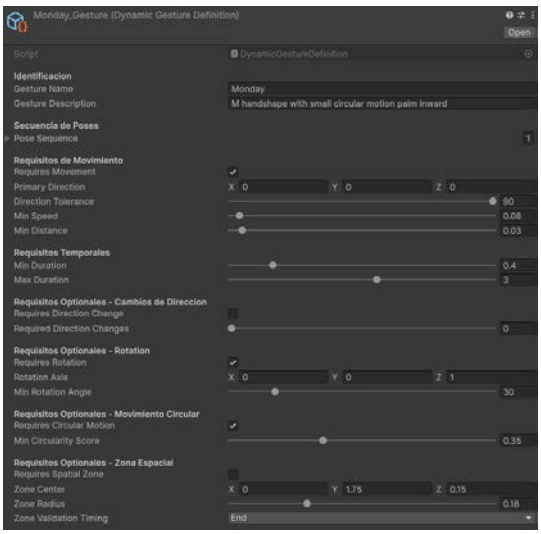
(a) *Hello*: lateral movement with a change of direction.



(b) *Bye*: compound gesture with a pose transition (`5` start → `S` end).



(c) *Please*: circular movement over the chest ($circularityScore \geq 0.25$).



(d) *Monday*: combination of wrist rotation and circular movement.

Figure 5.12: Examples of `DynamicGestureDefinition` in the Inspector.

Movement tracking is performed through `MovementTracker`, which records at each *frame* the position and rotation of the palm (`XRHandJointID.Palm`). Internally, this component maintains a history limited by two simultaneous criteria, a maximum of 120 samples and a maximum time window of 3 s. In this way, the

system retains enough recent information to compute kinematic metrics without the history growing indefinitely.

From these samples, `MovementTracker` performs the calculations required for validation, namely total distance traveled, instantaneous speed, average movement direction, number of direction changes, accumulated rotation, and, when necessary, a measure of trajectory circularity. These metrics are continuously queried by `DynamicGestureRecognizer` to determine whether the gesture progressively matches its definition.

During dynamic recognition, a momentary loss of tracking does not cause an immediate failure. If hand tracking is lost for less than 0,5 s (`TRACKING_LOSS_TOLERANCE`), the system preserves the current state and waits for tracking to recover. Only when the loss exceeds this threshold is the ongoing gesture cancelled. This tolerance reduces false failures caused by brief occlusions or by instabilities inherent to headset-based hand tracking.

These examples illustrate how the system can recognize gestures with very different movement profiles without modifying the logic of the recognizer, simply by adjusting the parameters defined in `DynamicGestureDefinition`.

5.4.2 Computation of kinematic metrics

The system computes five kinematic metrics from the information stored in the buffer. These were chosen to distinguish the system's dynamic gestures without resorting to machine learning models. Each gesture can be characterized deterministically through its speed, the direction and direction changes of the movement, the accumulated wrist rotation, and, where appropriate, the shape of its trajectory.

Instantaneous speed. Speed is estimated from the last three samples in the buffer as the accumulated distance between them divided by the corresponding time interval:

$$v = \frac{\|p_{n-1} - p_{n-2}\| + \|p_{n-2} - p_{n-3}\|}{t_{n-1} - t_{n-3}} \tag{5.1}$$

where p_i is the palm position at sample i and t_i is the capture time. Using three samples instead of two provides a more stable estimate and reduces the effect of optical tracking noise.

Average direction. At each update, the average direction of movement is computed from the last 30 % of the samples in the buffer. For each pair of consecutive samples in that segment, the normalized displacement vector is obtained, and the

resulting direction is the normalized average of all of them:

$$\hat{d}_{\text{avg}} = \frac{1}{M} \sum_{i=\lfloor 0.7N \rfloor}^{N-1} \frac{p_{i+1} - p_i}{\|p_{i+1} - p_i\|} \tag{5.2}$$

where N is the total number of samples stored in the buffer and M is the number of vectors considered in that segment. This metric is the basis of criterion P4 in the disambiguation algorithm and is used to verify that the movement follows the main direction defined in each `DynamicGestureDefinition`.

Direction changes. At each sample, the normalized displacement vector with respect to the previous sample is computed:

$$\hat{d}_i = \frac{p_i - p_{i-1}}{\|p_i - p_{i-1}\|} \tag{5.3}$$

If the angle between $\hat{d}_i$ and $\hat{d}_{i-1}$ exceeds 30°, the direction-change counter is incremented. This threshold was reduced from 45° during testing in order to improve sensitivity to smooth *waving*-type movements. This metric makes it possible to distinguish gestures with segmented trajectories from gestures with continuous trajectories. For example, the letter Z presents two direction changes that form the characteristic zigzag, whereas J describes a continuous downward trajectory without direction changes.

Accumulated rotation. At each frame, the incremental rotation angle of the palm with respect to the previous sample is accumulated, computed through `Quaternion.Angle`:

$$\theta_{\text{total}} = \sum_{i=1}^{N} \angle(q_i,\, q_{i-1}) \tag{5.4}$$

where q_i is the orientation of the palm at sample i. This metric makes it possible to detect gestures that involve wrist rotation, such as *Monday* (rotation around the Z axis) or *Blue*.

Circularity. For gestures with a circular trajectory, a minimum of 10 samples is required. The algorithm computes the trajectory center $\mathbf{c}$ as the average of the positions in the buffer, the mean radius $\bar{r}$ as the average distance from each point to the center, and the variance of the radii as a measure of regularity:

$$\mathbf{c} = \frac{1}{N} \sum_{i=1}^{N} p_i \qquad \bar{r} = \frac{1}{N} \sum_{i=1}^{N} \|p_i - \mathbf{c}\| \qquad \sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (\|p_i - \mathbf{c}\| - \bar{r})^2 \tag{5.5}$$

The circularity score is obtained by normalizing the variance with respect to the square of the mean radius:

$$s_{\text{circ}} = 1 - \text{clamp}\left(\frac{\sigma^2}{\bar{r}^2},\, 0,\, 1\right) \tag{5.6}$$

A value close to 1 indicates an almost perfectly circular trajectory, whereas a value close to 0 indicates an irregular or essentially linear trajectory.

The combination of these five metrics makes it possible to characterize deterministically the dynamic gestures implemented in the system. J is validated by a downward direction accompanied by wrist rotation and no direction changes. Z is validated by exactly two direction changes in the horizontal plane. Gestures such as *Hello* are validated through the combination of speed, direction, and number of direction changes.

None of these cases requires a trained model. The geometric metrics are sufficient and computationally efficient for real-time execution on Meta Quest 3.

In gestures where the final pose differs from the initial pose or does not correspond to any sign in the pedagogical catalog, the system directly validates the final configuration by calling `XRHandShape.CheckConditions()` on the last *jointsUpdated* snapshot, without going through the static pose adapter. This mechanism makes it possible to model compound gestures with different start and end configurations without the need to register the final pose as an independent sign.

5.4.3 Performance considerations

During recognition, two operations of the `MovementTracker` generate temporary memory allocations on each frame, `CalculateSpeed()` and `CalculateAverageDirection()`, which internally convert the sample queues into arrays for indexed access. The circularity calculation does not present this problem, since it uses pre-allocated lists in the constructor (`cachedPositions`, `cachedRadii`).

These allocations are short-lived and are automatically released by the C# garbage collector. In the tests carried out, they did not compromise system performance in *standalone* mode on Meta Quest 3. Nevertheless, replacing the temporary arrays with equivalent pre-allocated structures would eliminate these allocations and reduce garbage collector activity during active recognition.

5.5 Feedback system

The feedback system is not limited to indicating whether a gesture is correct or incorrect. Its purpose is to tell the user which part of the gesture must be corrected and at what moment the error occurs. To achieve this, two complementary mechanisms are used. On the one hand, a static finger-level analysis evaluates hand configuration in manual poses. On the other hand, a dynamic phase-based analysis evaluates the execution of gestures in motion.

Both mechanisms operate independently of the recognizer. Coordination between them is handled through `FeedbackSystem`, which centralizes the generation of the feedback shown to the user.

5.5.1 System architecture

`FeedbackSystem` acts as the coordinator of all feedback subsystems. It subscribes to the events of `GestureRecognizer` (static poses) and `DynamicGestureRecognizer` (dynamic gestures), and delegates the analysis to `HandPoseAnalyzer` and `DynamicGestureFeedbackAnalyzer`, respectively. The results are distributed to three outputs: correction text (`TextMeshProUGUI`), finger-level visual indicators (`FingerIndicatorVisualizer`), and a color overlay on the finger segments (`XRFingerOverlayRenderer`).

This separation between recognition and explanation is a deliberate design decision, since `HandPoseAnalyzer` never replaces the global recognizer in determining whether a gesture is correct, but instead uses the result already emitted by `GestureRecognizer` through the `isDetectedByRecognizer` parameter. If the recognizer confirms the gesture, the analyzer shows success without recomputing anything. If it does not confirm it, the analyzer explains why through finger-level errors.

5.5.2 Static finger-level analysis

For static poses, the analysis is performed with a sampling interval of 200 ms (`analysisInterval`), which avoids evaluations on every *frame* and reduces computational cost. This mechanism acts as a *debounce*, since the pose is only analyzed if the interval since the last analysis has elapsed.

Finger-level analysis is based on `FingerConstraintProfile`, a *ScriptableObject* that defines for each finger the acceptable *curl* range (0 = extended, 1 = closed), the constraints on separation between adjacent fingers (*spread*), thumb position constraints, and, optionally, palm orientation. The profiles for all the letters of the alphabet (A–Z) and the digits (0–9) are loaded automatically at startup.

Curl computation. The *curl* value of each long finger is computed geometrically from the flexion angles at the intermediate and distal joints. For each joint, the angle is obtained between the inverse of the incoming segment and the outgoing segment, which corresponds to the flexion angle at that joint: 180° for a fully extended finger and approximately 60° for a finger closed into a fist. The *curl* value is obtained by normalizing the average of both angles:

$$\text{curl}_i = \text{clamp}\left(\text{InverseLerp}\left(180,\ 60,\ \frac{\angle(-\hat{d}_{\text{prox}},\ \hat{d}_{\text{inter}}) + \angle(-\hat{d}_{\text{inter}},\ \hat{d}_{\text{dist}})}{2}\right),\ 0,\ 1\right) \tag{5.7}$$

where $\hat{d}_{\text{prox}}$, $\hat{d}_{\text{inter}}$, and $\hat{d}_{\text{dist}}$ are the normalized vectors of the proximal–intermediate, intermediate–distal, and distal–tip segments, respectively. A value of 0 corresponds to a fully extended finger and 1 to a fully closed one.

Algorithm 1 Computation of *curl* for a long finger

Require: Poses of the proximal, intermediate, distal, and tip joints
Ensure: *curl* value $\in [0, 1]$

$\hat{d}_{\text{prox}} \leftarrow \text{normalize}(p_{\text{inter}} - p_{\text{prox}})$
$\hat{d}_{\text{inter}} \leftarrow \text{normalize}(p_{\text{dist}} - p_{\text{inter}})$
$\hat{d}_{\text{dist}} \leftarrow \text{normalize}(p_{\text{tip}} - p_{\text{dist}})$
$\alpha_1 \leftarrow \angle(-\hat{d}_{\text{prox}},\ \hat{d}_{\text{inter}})$
$\alpha_2 \leftarrow \angle(-\hat{d}_{\text{inter}},\ \hat{d}_{\text{dist}})$
$\bar{\alpha} \leftarrow (\alpha_1 + \alpha_2)/2$
return clamp(InverseLerp(180, 60, $\bar{\alpha}$), 0, 1)

The thumb uses a simplified calculation that reflects its smaller range of motion. Instead of averaging two flexion angles, it uses a single angle between the inverse of the proximal segment and the distal–tip segment:

$$\text{curl}_{\text{thumb}} = \text{clamp}\Big(\text{InverseLerp}\Big(180,\ 50,\ \angle(-\hat{d}_{\text{prox}},\ \hat{d}_{\text{dist}})\Big),\ 0,\ 1\Big) \tag{5.8}$$

Algorithm 2 Computation of *curl* for the thumb

Require: Poses of the proximal, distal, and tip joints of the thumb
Ensure: *curl* value $\in [0, 1]$

$\hat{d}_{\text{prox}} \leftarrow \text{normalize}(p_{\text{dist}} - p_{\text{prox}})$
$\hat{d}_{\text{dist}} \leftarrow \text{normalize}(p_{\text{tip}} - p_{\text{dist}})$
$\alpha \leftarrow \angle(-\hat{d}_{\text{prox}},\ \hat{d}_{\text{dist}})$
return clamp(InverseLerp(180, 50, α), 0, 1)

Semantic classification and finger-level errors. The system classifies the state of each finger into three categories: *Extended* (curl < 0.25), *Curved* (curl between 0.25 and 0.75), and *Closed* (curl > 0.75). Based on the transition between the current and expected state, a semantic error type is generated, as summarized in Table 5.2.

Current state	Expected state	Semantic error
Extended	Curved	`NeedsCurve`
Extended	Closed	`NeedsFist`
Curved	Closed	`NeedsFist`
Curved	Extended	`NeedsExtend`
Closed	Extended	`NeedsExtend`
Closed	Curved	`TooMuchCurl`

Table 5.2: Semantic error types generated from the transition between the current and expected finger state.

Error level and pedagogical adjustments. The error level depends on the deviation with respect to the acceptable range. This range is computed by first applying the tolerance used by the system, that is, the original tolerance adjusted to guarantee a minimum margin before evaluating the deviation. A deviation greater than 0.18 is classified as *Major*, between 0.08 and 0.18 as *Minor*, and values below 0.08 are considered to lie within the acceptable range.

The effective tolerance is computed from the tolerance defined for the gesture, while ensuring a minimum margin. For fingers, $\max(\texttt{curlTolerance} \times 0.5,\ 0.08)$ is used, so that the tolerance is never below 0.08. For the thumb, the system applies $\max(\text{normal\_tolerance},\ 0.12)$, guaranteeing a minimum value of 0.12 due to the greater variability of thumb movement.

The system also applies two pedagogical adjustments. If a finger is in the *Curved* state when *Closed* is expected, the error is reduced from *Major* to *Minor*, since the finger is already in an intermediate position consistent with the correct direction of movement. Similarly, if the thumb presents a *Major* error with a deviation below 0.25, its level is also downgraded to *Minor*, acknowledging the greater natural variability of the thumb.

Spread constraints. When defined in the profile, the system evaluates the separation between adjacent fingers by measuring the angle between the proximal–tip vectors of each consecutive pair:

$$\theta_{\text{spread}} = \angle\Big(\hat{d}^{(i)}_{\text{prox}\to\text{tip}},\ \hat{d}^{(i+1)}_{\text{prox}\to\text{tip}}\Big) \tag{5.9}$$

where $\hat{d}^{(i)}_{\text{prox}\to\text{tip}}$ is the normalized vector from the proximal joint to the fingertip of finger i. The angle is computed through `Vector3.Angle`, which always returns a value in $[0, 180]$.

If θ_{spread} falls outside the range [`minSpreadAngle`, `maxSpreadAngle`] defined in the profile, the system generates a diagnosis indicating that the separation between the fingers is too small (`SpreadTooNarrow`) or too large (`SpreadTooWide`). The warning level is determined by the profile configuration. This constraint is evaluated only if it is explicitly enabled in the profile (`isEnabled = true`), so not all signs use it.

Thumb constraints. The thumb has three additional types of constraints. First, the profile may require the thumb to touch one or several specific fingers. If the distance between the thumb tip and the tip of the target finger exceeds 3 cm, a `ShouldTouch` error of *Major* severity is generated. This check is automatically skipped if the target finger has curl < 0.25, since in that case the main error lies in the finger position rather than in the thumb. Second, if the profile indicates that the thumb should be positioned over the other fingers (`shouldBeOverFingers`) and its curl is below 0.35, a *Major* position error is generated. Third, if it is expected to be positioned beside the fingers (`shouldBeBesideFingers`) and its curl exceeds 0.6, the system also generates a *Major* position error.

Palm orientation. When required by the profile (`checkOrientation`), the system evaluates palm orientation by comparing the *forward* direction of the palm joint with the expected direction defined in the profile. If the angle between them exceeds `orientationTolerance`, an error of type `RotationWrong` with *Minor* severity is generated.

Global score. Once all finger-level errors, spread constraints, thumb position constraints, and palm orientation have been evaluated, the system aggregates the results into a match score that reflects how close the current pose is to the target pose of the sign being practiced:

$$\text{matchScore} = \text{clamp}(1 - 0.3 \cdot N_{\text{major}} - 0.1 \cdot N_{\text{minor}},\ 0,\ 1) \tag{5.10}$$

where N_{major} and N_{minor} are the numbers of detected *Major* and *Minor* errors, respectively. Each *Major* error penalizes the score by 0.3 points, and each *Minor* error by 0.1, so that a pose with two severe errors obtains a `matchScore` of 0.4, whereas a pose with only fine adjustments still pending remains close to 1.

This score is not used to validate the gesture, as that responsibility rests exclusively with `GestureRecognizer`, but rather as an internal pedagogical signal that allows the system to distinguish three situations, absence of errors but no

recognizer confirmation yet, presence of only *Minor* errors (*near match*, the pose is almost correct), and presence of *Major* errors that block recognition. In the first and second cases, the message shown to the user is one of fine adjustment; in the third, it is one of active correction.

If `GestureRecognizer` confirms detection, the analyzer returns success directly with `matchScore = 1` without recomputing finger-level errors.

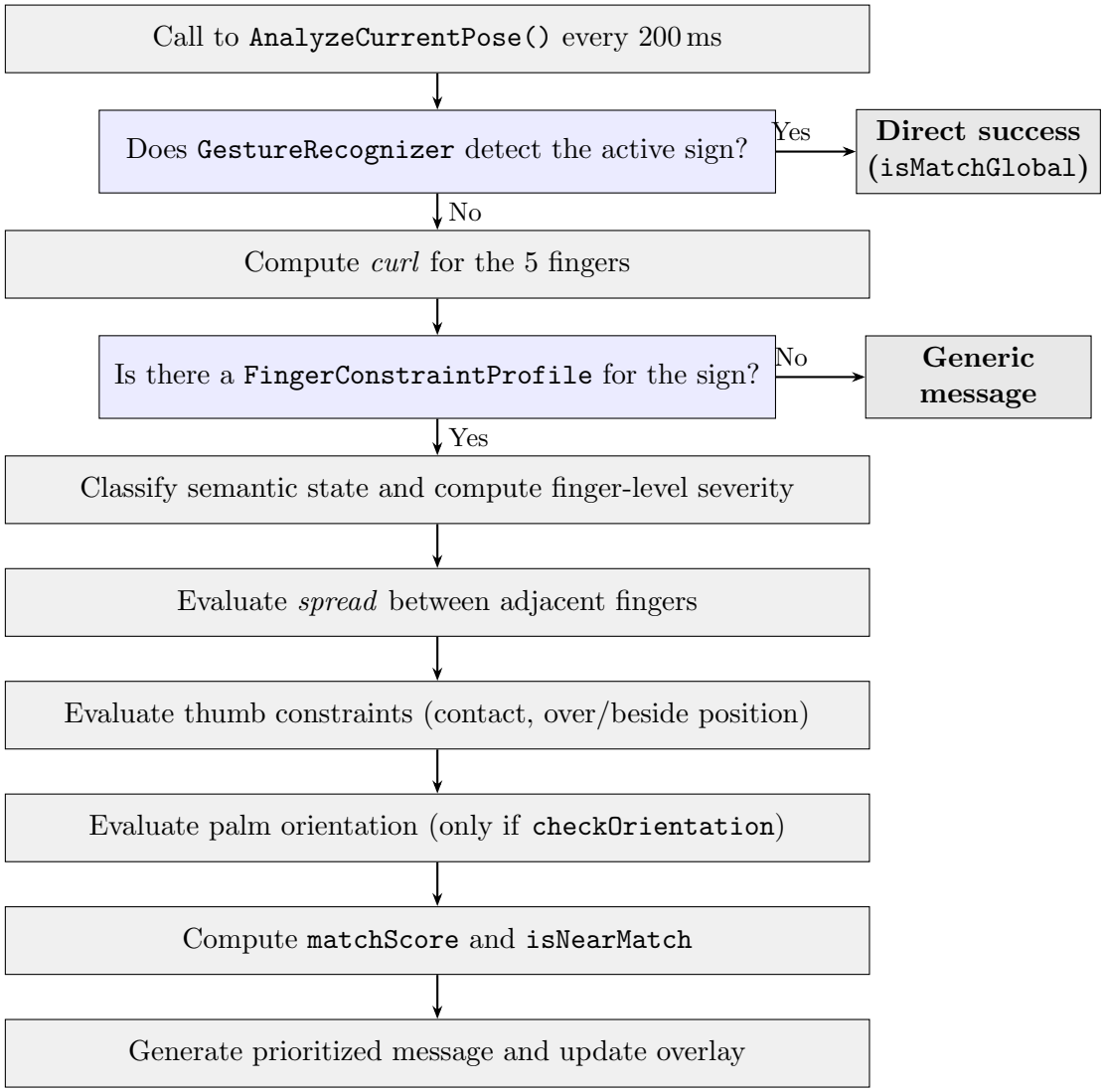


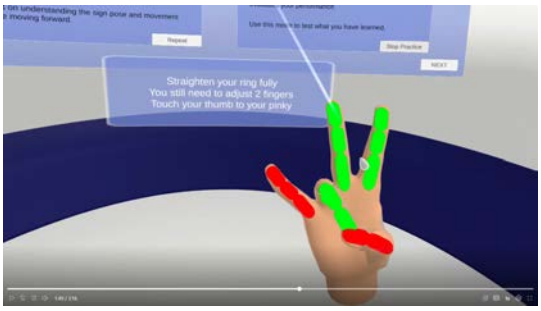
Figure 5.13: Static finger-level analysis flow.

5.5.3 Finger-level visual overlay

Visual feedback is implemented through `XRFingerOverlayRenderer`, which draws independent capsules over each segment of each finger (proximal–intermediate, intermediate–distal, distal–tip). Each segment is colored green when the finger is in the expected state, and red when it contains an error, accompanied by a textual message explaining how to correct it.

(a) Active errors: fingers in red with specific corrections for each finger and the thumb.

(b) Correct gesture: all segments in green and confirmation message.



(c) *Active correction*: adjustment message.

Figure 5.14: States of the feedback system for the sign *6*.

Color application is performed through `MaterialPropertyBlock`, which avoids material instantiation at runtime and keeps rendering cost low in *standalone* mode on Meta Quest 3.

The overlay is activated only during static pose analysis. When the user starts a dynamic gesture, the overlay is automatically disabled so as not to interfere with movement, and is reactivated when the system returns to the waiting state.

Figure 5.14 shows the three main states of the system for the sign *6*. In the first state, the overlay identifies incorrect fingers in red and the text panel details the necessary corrections. In the second state, once `GestureRecognizer` confirms

detection, all segments turn green and the success message is shown. The third state shows an example of active correction, where the user is performing a pose that is close but incorrect (the sign *7* instead of *6*), and the overlay marks in red the fingers that do not satisfy the profile requirements, while the text details the necessary adjustments finger by finger.

5.5.4 Phase-based feedback in dynamic gestures

For dynamic gestures, `DynamicGestureFeedbackAnalyzer` organizes feedback into six phases corresponding to the key moments of the gesture from a pedagogical perspective. These phases are independent of the recognizer state machine states (`Idle`, `PendingConfirmation`, `InProgress`).

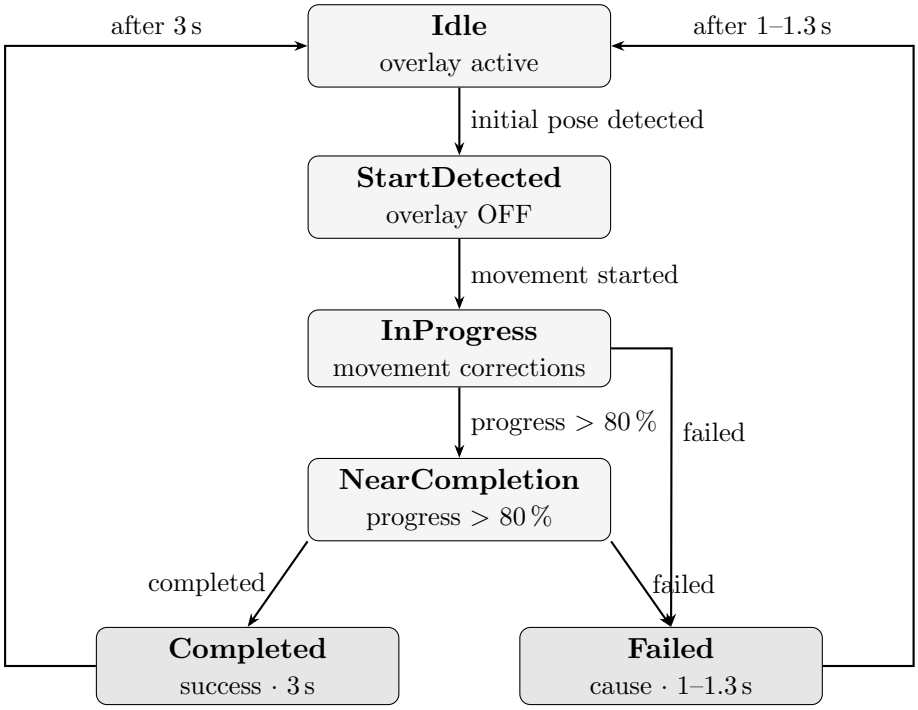


Figure 5.15: Dynamic feedback phase diagram managed by `DynamicGestureFeedbackAnalyzer`.

1. **Idle (phase 0).** The system waits for the user to place the hand in the initial pose. The visual overlay remains active to help position the fingers.
2. **StartDetected (phase 1).** The initial pose has been detected. The system confirms it with a positive message and tells the user to begin the movement. The overlay is disabled.

3. **InProgress (phase 2).** The movement is underway. The system continuously analyzes the kinematic metrics and emits corrections related to direction, speed, amplitude, or continuity. No finger-level corrections are shown during this phase.
4. **NearCompletion (phase 3).** Progress exceeds 80 %. The system emits an encouraging message so that the user completes the movement without stopping.
5. **Completed (phase 4).** The gesture was completed correctly. A fixed success message is shown for 3 s (`dynamicSuccessHoldDuration`).
6. **Failed (phase 5).** The gesture failed. A specific explanation of the cause is shown (incorrect direction, insufficient speed, not enough direction changes, etc.) for between 1 and 1.3 s, after which the system returns to the waiting phase.

During the *InProgress* phase, the `DetectMovementIssue()` method checks in each analysis cycle the metrics accumulated by `MovementTracker` described in Section 5.4.2 and compares them with the thresholds defined in `DynamicGestureDefinition`. The checks are applied in priority order, starting with hand-shape stability, speed, direction, amplitude, rotation, circularity, and number of direction changes. The method returns the first cause that exceeds its threshold, thus prioritizing the main problem. If a problem is detected, the message is latched during a random interval between 1 and 1.3 s (*message latch*), preventing the system from returning to `Idle` until the user has had time to read it. This mechanism also prevents the system from returning to `Idle` while the correction message is visible. Table 5.3 lists the messages emitted during this phase.

Problem (`DynamicMovementIssue`)	**Message shown to the user**
`StartPoseDegrading`	*"Keep the hand shape."*
`TooSlow`	*"Move faster."*
`TooFast`	*"Slow down."*
`DirectionWrong`	*"Move your hand [direction]."*
`TooShort`	*"Make the movement bigger."*
`RotationInsufficient`	*"Rotate your wrist more."*
`NotCircular`	*"Make the movement more circular."*
`NeedMoreDirectionChanges`	*"Move side to side more times."*
`None`	*"Keep going."*

Table 5.3: Messages emitted during the *InProgress* phase.

When the gesture is not completed, `NotifyFailed()` generates an explanatory message depending on `FailureReason` and on the phase in which the failure occurred (`GesturePhase`). Table 5.4 lists the causes and their corresponding messages.

Cause (`FailureReason`)	**Message shown to the user**
`SpeedTooLow`	*"The gesture was too slow."*
`SpeedTooHigh`	*"The gesture was too fast."*
`DistanceTooShort`	*"The movement was too short."*
`DirectionWrong`	*"Direction was incorrect. [detail]."*†
`DirectionChangesInsufficient`	*"Not enough direction changes."*
`RotationInsufficient`	*"You did not rotate your wrist enough."*
`NotCircular`	*"The movement was not circular enough."*
`Timeout`	*"The gesture took too long."*
`PoseLost` (start)	*"You started moving your hand too soon."*
`PoseLost` (during)	*"You lost the hand shape during the movement."*
`EndPoseMismatch`	*"The final pose was incorrect."*
`TrackingLost`	*"Keep your hand visible to the sensors."*
`OutOfZone`	*"Your hand left the tracking zone."*

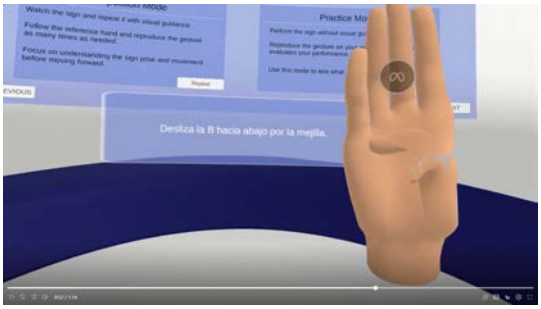
Table 5.4: Failure causes and messages generated by `NotifyFailed()` at the end of the gesture. All messages are completed with the suffix *"Try again while keeping the hand shape."* †`DirectionWrong` includes a gesture-specific description if defined, or the expected direction derived from the movement vector.

Figure 5.16 illustrates the behavior of the system during practice of the sign *Brown*. In the *Idle* phase, the lower panel asks the user to place the hand in the initial pose. Once the movement has started, the system shows progress in real time, and when the gesture is completed, the success message remains visible for 3 s.

5.5.5 Success confirmation

For both static and dynamic gestures, the system keeps the success state visible for a configurable time (`successDisplayDuration` = 3 s for static gestures, `dynamicSuccessHoldDuration` = 3 s for dynamic gestures) before resuming analysis. This interval guarantees that the user perceives the positive reinforcement before the system moves to the next sign or restarts the evaluation.

(a) *Idle* phase: initial pose instruction.



(b) *StartDetected* phase: pose confirmation and movement instruction.

(c) *InProgress* phase: real-time progress (89 %).

(d) *Completed* phase: success confirmation.

Figure 5.16: Feedback for the sign *Brown*.

5.6 Guide-hand animation

The system incorporates guide hands (*guide hands*) that show the user how to perform each sign. They do not represent the user's hand and are not part of the feedback system. They serve as a visual reference whose sole purpose is to illustrate the target pose or movement before the user practices it. This distinction is deliberate. The user's hand is detected by `XRHandSubsystem` and evaluated by the recognizer. The guide hands are scene objects completely decoupled from tracking, with their own model, fixed position, and independent skeleton.

5.6.1 Design decision

Guide-hand animation does not use Unity's conventional animation system. No `Animator Controllers`, `Animation Clips`, or Playables are used. Instead, `GuideHandPoseApplier` directly applies, on each *frame*, the local rotations (`localRotation`) of the joints of the hand skeleton.

This choice is based on three key ideas. First, it maintains coherence with

the recognition system. The same data structures that describe a pose can be used both to evaluate it and to display it, without the need for transformations or additional representations.

Second, it provides direct control over each *joint*. This makes it possible to adjust the hand configuration precisely and to build clear visual references, without depending on predefined animations or motion capture.

Finally, transitions between poses are performed progressively. Finger rotations evolve from the current configuration towards the target one, avoiding abrupt jumps and making the gesture easier for the user to understand.

5.6.2 Pose representation

The pose of a guide hand is stored in `HandPoseData`, a class that defines the complete hand configuration through four data structures: `FingerPoseData` for each of the four fingers, `ThumbPoseData` for the thumb, and a wrist position and rotation offset (`wristRotationOffset`, `wristPositionOffset`).

`FingerPoseData` defines the state of a long finger through five parameters: the *curl* values of the metacarpal, proximal, intermediate, and distal joints (all in $[0, 1]$), and the lateral separation angle (*spread*) with respect to the adjacent finger. `ThumbPoseData` uses a different structure, with three *curl* values (metacarpal, proximal, distal) and four additional angular parameters that capture the greater freedom of movement of the thumb: the abduction angle (`abductionAngle`), the opposition angle (`oppositionAngle`), the distal twist (`distalTwist`), and the pitch (`thumbPitch`).

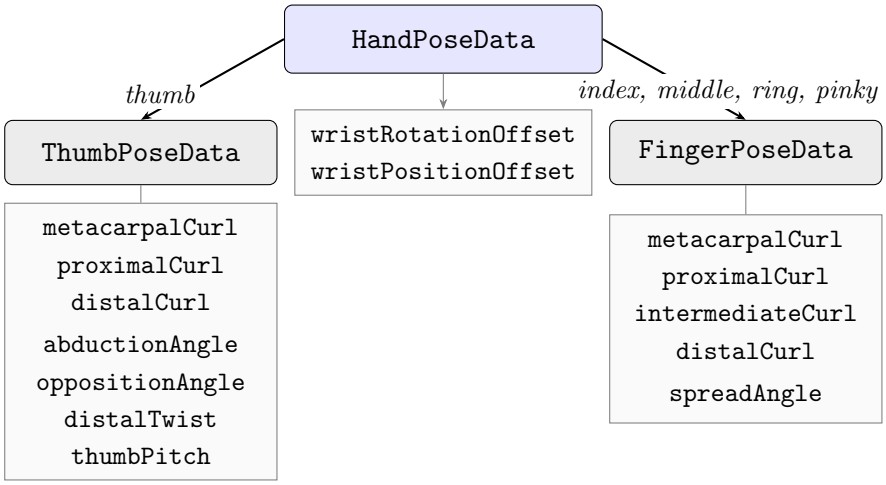


Figure 5.17: Data structure of `HandPoseData`.

The relationship with the recognition system is direct, since both subsystems

share the same pose definitions. When `GuideHandPoseApplier` needs to display a sign, it queries `ASLPoseLibrary` using the same sign name that the recognizer uses to validate it.

5.6.3 Pose construction

Visual poses are not generated automatically from the recognition parameters. The construction process is manual and iterative. It starts from the *curl*, *spread*, and *pinch* values defined in `HandPose` / `HandShape` as a starting point; these are applied to the model in the Unity editor, and then adjusted joint by joint until a visually correct reference is obtained. The pose must be recognizable and easy for the user to imitate, even if this implies slightly exaggerating finger separation or the extension of specific joints. The final result is stored in `HandPoseData` and registered in `ASLPoseLibrary`.

Figure 5.18 shows two examples of the final result. In both cases, it can be seen that manual adjustment of wrist orientation is part of the process. In *O*, if the palm were oriented directly towards the user, the circular shape formed between the thumb and the fingertips would not be clearly visible. In *G*, the separation and parallelism between the index finger and the thumb require adjusting the angles so that both fingers are correctly positioned and the user can understand the pose.

(a) Sign *G*: index finger extended with the thumb parallel and separated.

(b) Sign *O*: fingertips joined with the thumb.

Figure 5.18: Examples of guide-hand poses. Wrist orientation is manually adjusted to maximize the legibility of the finger configuration, even if this implies deviating from the exact execution position.

At runtime, `GuideHandPoseApplier` applies `HandPoseData` directly to the skeleton by modifying the local rotations of each *joint*. To prevent offsets from accumulating between poses, all rotations are always computed from the original values cached during initialization. The transition between poses is not instantaneous,

since the fingers move progressively until they reach the target configuration, which avoids abrupt jumps and makes the visual reference easier to follow.

5.6.4 Flow in learning mode

The complete flow is orchestrated by `GhostHandPlayer`, which manages the playback sequence, visibility, and transition to practice mode.

When the user accesses a sign, the guide hands are shown in a neutral pose (open hand, sign *5*). After a pause of 0.5 s (`delayBeforeGesture`), `GhostHandPlayer` applies the target pose to the right hand, which remains visible for 3 s (`gestureDisplayTime`). The user may repeat the animation as many times as needed through the *Repeat* button. The left hand remains at rest on the table at all times, except for two-handed signs, where it is positioned actively next to the right hand and performs its own sequence in parallel.

When the user presses *Practice*, `GhostHandPlayer` executes a *fade out* that disables the guide hands, and the recognizer begins evaluating the user's real hand with the feedback system active. If the user leaves practice mode, the guide hands reappear through a *fade in*, showing the pose of the current sign without automatically replaying the animation.

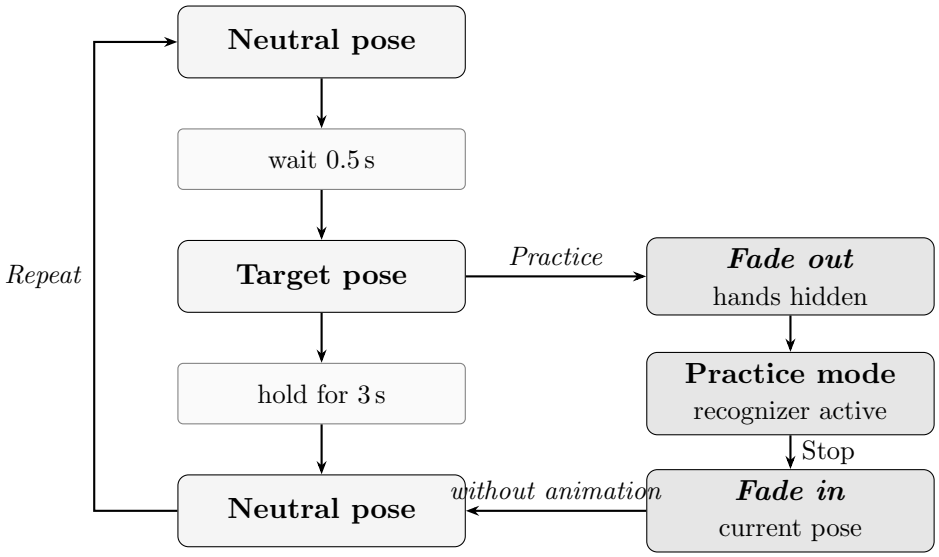


Figure 5.19: Flow of the learning mode managed by `GhostHandPlayer`. The loop on the left corresponds to observation mode (free repetition), while the branch on the right shows the transition to practice mode and the return.

5.6.5 Animated sequences

Signs that require movement, such as the letters J or Z, are not represented by a single pose, but as a temporal sequence of poses (`AnimatedPoseSequence`). This sequence is composed of several *keyframes*, where each one defines a complete hand configuration (`HandPoseData`) associated with a time instant within the gesture.

During playback, `GhostHandPlayer` traverses this sequence in real time. On each *frame*, the system computes the elapsed time and retrieves the corresponding pose within the sequence using `SampleAtTime(elapsed)`. This pose is applied directly to the guide hand through `ApplyPoseImmediate()`.

In this way, the hand successively adopts the different configurations defined in the keyframes, generating the movement of the gesture. Once the last keyframe has been reached, the final pose remains visible for a brief interval (`gestureDisplayTime`) before playback ends.

Special case: Months of the year

The signs corresponding to the months of the year constitute a particular case within the animation system. Although they are represented as a continuous animation, they are not defined through a trajectory, but rather as a concatenation of poses corresponding to letters. Each month is modeled as a sequence of abbreviated letters (for example, *A-U-G* for August), where each letter corresponds to a `HandPoseData`. The sequence is defined in `MonthSequenceData` and managed by `MonthPracticeController`, which coordinates both the presentation of each pose on the guide hands and the validation of each letter through the recognizer. Unlike dynamic gestures such as J or Z, where movement is defined through keyframes along a trajectory, in this case movement emerges from the transition between poses. The result is a continuous animation that corresponds to the sign through its first three letters.

Figure 5.20 shows the complete sequence for the month of August ($A \rightarrow U \rightarrow G$).

(a) Letter *A*. (b) Letter *U*. (c) Letter *G*.

Figure 5.20: Guide-hand sequence for the sign *August* ($A \rightarrow U \rightarrow G$).

The guide hands present each letter independently, in the same order in which

they must be performed, while the top panel indicates to the user the full sequence and the current letter.

5.6.6 Integration with the rest of the system

The guide hands share data structures with the recognition and feedback subsystems. `HandPoseData` is built from the same parameters as `FingerConstraintProfile`. In turn, `ASLPoseLibrary` acts as the single source of truth for both subsystems. The recognition–feedback–visual guidance cycle operates on the same `SignData`, avoiding duplicate definitions and guaranteeing that the visual reference shown to the user is consistent with the criteria that the recognizer applies to validate execution.

5.7 Interaction design in VR

The interaction design is based on a simple idea. The user learns by performing the gesture, not just by observing it. The application does not present signs as static content, but as actions that the user must perform, receiving progressive support until they can complete them. To achieve this, the experience is organized into two operating modes, allowing the user both to learn how to perform the sign and to practice that execution, between which the user can switch at any time.

5.7.1 Operating modes

In **repeat mode**, the guide hands are visible and the recognizer remains inactive, preventing the feedback system from intervening in the scene. The user observes the pose or movement of the sign and can repeat it as many times as desired through the *Repeat* option. In this mode there is no evaluation; the goal is to clearly understand the hand configuration before practicing the execution.

In **practice mode**, the guide hands disappear through *fade out* and the recognizer is activated. From that moment on, the user performs the sign with their own hand and receives real-time feedback on the execution, as described in Section 5.5. At any moment, the user can leave this mode, which causes the guide hands to reappear through *fade in*, once again showing the reference for the current sign.

In this way, the user always has access to a reference before being evaluated and can return to it as many times as needed without interrupting the practice flow.

5.7.2 Interaction flow

The experience is organized into three navigation levels. From the main screen, the user accesses the learning module, where a difficulty level is selected first and, within it, a thematic category (colors, numbers, family, etc.). This layered structure allows the user to focus on a limited subset of signs before progressing further, avoiding the cognitive overload that would result from presenting the full vocabulary all at once. Once the category has been selected, the user enters the learning session, where the signs can be browsed one by one.

When *Start Learning* is selected, the user reaches the level selection screen. Each level groups categories of increasing complexity, in line with the process of learning a new language. *Basic* includes the alphabet, digits, and basic communication. *Intermediate* includes colors, days, and months. *Advanced* focuses on verbs. The available categories and the number of signs in each one are displayed when selecting the level.

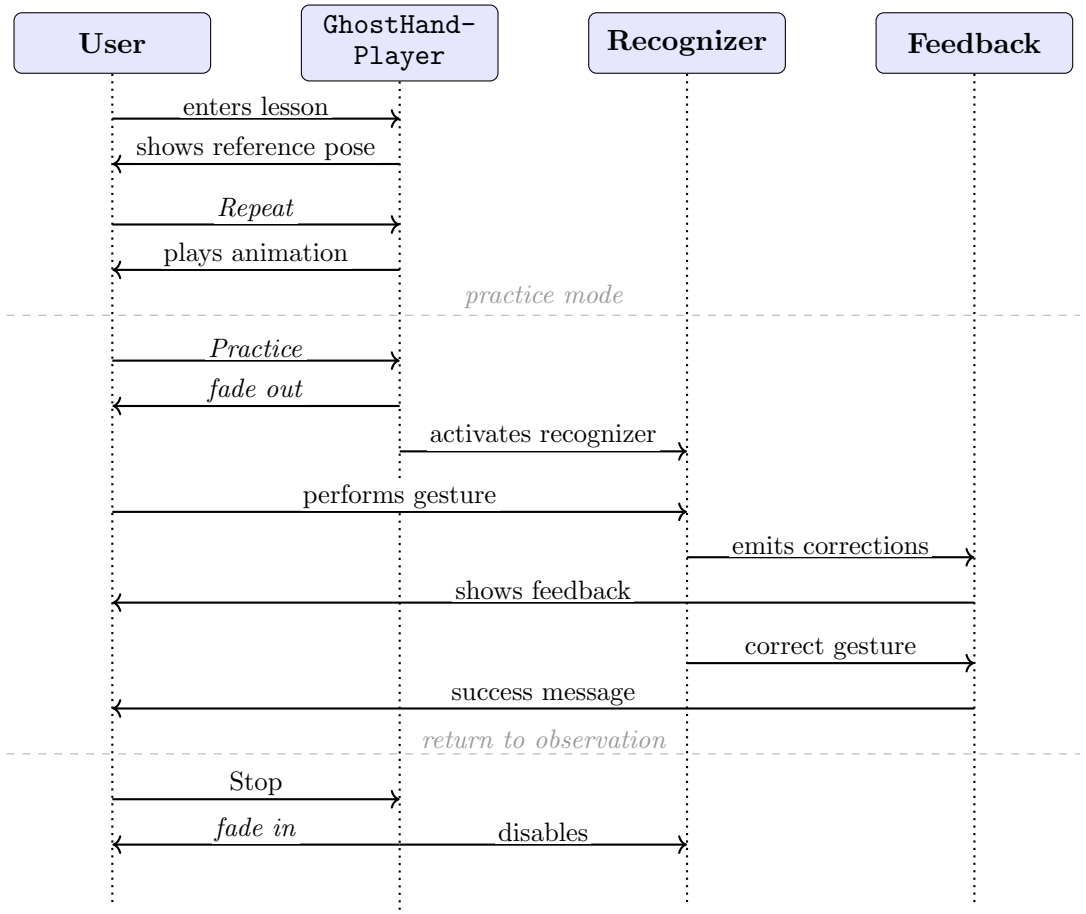


Figure 5.21: Sequence diagram of the interaction flow. Repeat mode (upper part) and practice mode (lower part) alternate under explicit user control.

From the user's perspective, interaction with each sign follows a clear sequence. Upon entering the lesson, the top panel shows the name of the sign and a brief description. The guide hands appear performing the reference pose or movement. The user observes the gesture and can repeat the animation as many times as desired through *Repeat*.

When the user decides to attempt the sign, *Practice* is pressed. At that moment, the guide hands disappear and the system begins evaluating the execution. If the sign is a static pose, the color overlay on the fingers indicates in real time which parts are correctly positioned and which ones need adjustment. If it is a dynamic gesture, the system provides guidance on the movement, such as direction, speed, or amplitude. When the recognizer confirms the sign, a success message is shown.

The user can move at any moment to the previous or next sign through the

Previous and *Next* buttons, located on the side panels of the VR environment. It is not necessary to complete a sign in order to continue; the pace of the session is always controlled by the user. From within the same learning session, a side panel allows direct access to self-assessment mode without needing to return to the selection screen.

Once the category has been completed, the user can access self-assessment mode. In this mode, all the signs of the category are shown in a grid, without guide hands or instructions. The recognizer remains active for all signs at once, so the user can perform them in any order. Each correct sign activates its indicator in green in the grid, and the progress counter is updated in real time. The goal is for the user to check independently which signs have already been mastered and which ones still need further practice.

Figure 5.22 summarizes the complete navigation structure between the different scenes of the application.

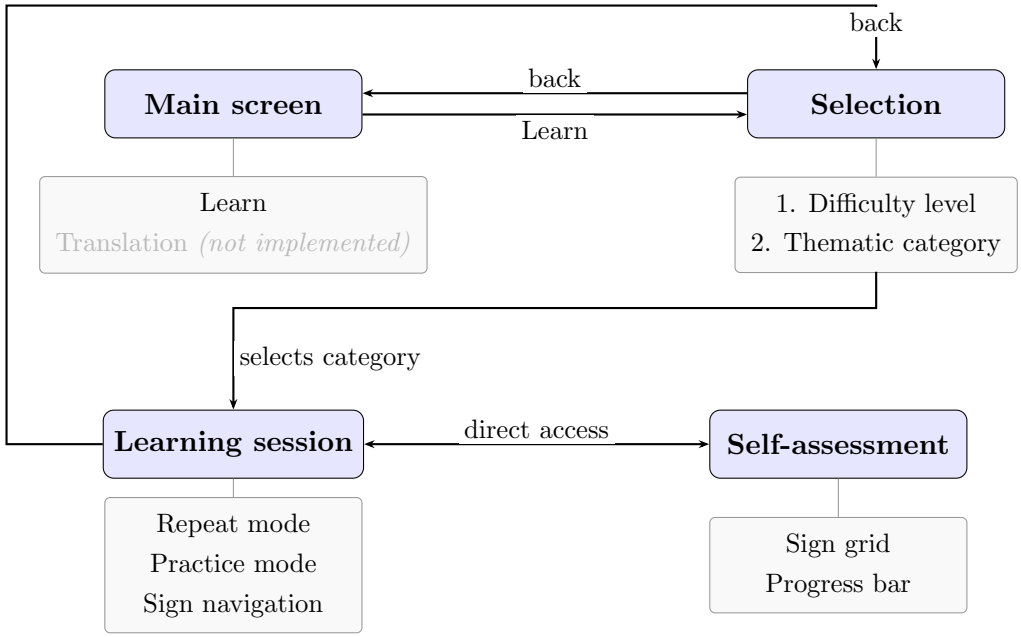


Figure 5.22: Navigation between scenes.

All interaction is performed through the hands, without additional physical devices. Interface elements (buttons, information panels, and navigation controls) respond to *pinch* gestures detected by the XR Hands tracking system. This choice fits the intended use of the application itself, since the user uses the hands both to perform the signs and to interact with the interface.

Representative examples of the different interface screens are shown in Figure 5.23, illustrating the main stages of the interaction flow.

(a) Main screen: access to the learning module.

(b) Access to self-assessment from the learning session.

(c) *Basic*

(d) *Intermediate*

(e) *Advanced*

(f) Initial state (Progress: 0/10).

(g) State with progress (Progress: 5/10).

Figure 5.23: Representative user interface screens across the different stages of the interaction flow.

5.7.3 User Interaction in the Virtual Environment

The developed system is not limited to the representation of a virtual hand, but is based on direct interaction between the user and the environment, where each gesture is physically performed and interpreted in real time. In this sense, the user plays an active role in the process, as gesture execution, capture, evaluation, and system response are continuously connected within a unified interaction flow.

The interaction is carried out using a virtual reality device with integrated hand tracking, specifically Meta Quest 3 [26], which makes it possible to dispense with physical controllers. This is particularly relevant in the context of sign language learning, as the user directly uses their own hands, maintaining a natural correspondence between the real gesture and its virtual representation. As illustrated in Figure 5.24, the physical gesture performed by the user is captured and mapped in real time onto the virtual environment, where finger-level visual feedback indicates which parts of the hand configuration are correct and which require adjustment.

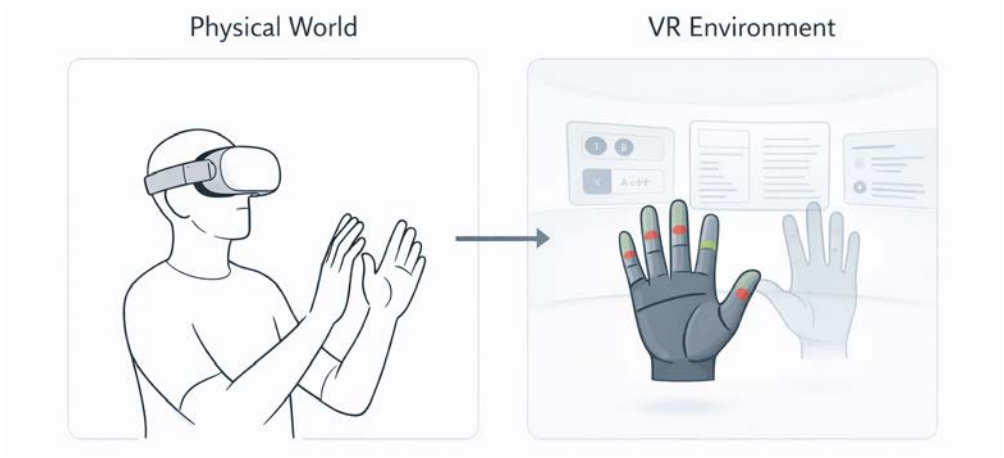


Figure 5.24: Physical interaction and its virtual representation in *ASL_LearnVR*.

Figure 5.25: Virtual reality device used in the platform.

This direct correspondence between the user's real hand and its virtual repre-

sentation is key to an intuitive interaction, as the user can immediately see how their movements are interpreted by the system. Overall, this interaction model ensures that the system operates not only as a gesture recognition mechanism, but as an active learning tool, where each user attempt generates meaningful feedback that supports gradual improvement.

At a technical level, the system relies on hand tracking data provided by the `XRHandSubsystem`, which captures the position and configuration of the hand joints in real time. As described in previous sections, this information is used by the recognition modules to evaluate the performed gesture and detect possible deviations.

Figure 5.26 illustrates the complete interaction flow within the system. The process begins with the physical execution of a gesture by the user. The system then captures the hand movement, processes it using the defined geometric and temporal rules, and generates a response in the form of visual feedback within the virtual environment. This feedback includes both the virtual representation of the hand and additional visual cues that assist the user in correcting the gesture.

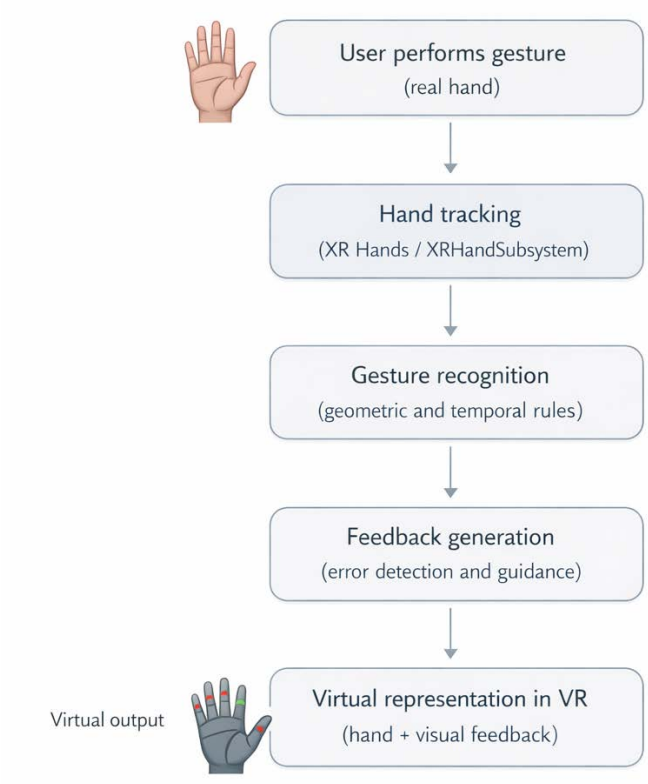


Figure 5.26: User interaction flow, from the physical execution of the gesture to its evaluation and representation in the virtual environment.

As a result, the virtual hand maintains a direct correspondence with the user's real hand, both in terms of position and configuration. This direct mapping is key to achieving an intuitive interaction, as it allows the user to immediately understand how their physical movements are interpreted by the system.

Overall, this approach enables the system to function not only as a gesture recognition mechanism, but also as an active learning tool. Each user attempt produces an interpretable response, facilitating progressive correction and reinforcing the association between the physical gesture and its correct execution.

Chapter 6

Results Analysis

This chapter analyzes the behavior of the system once all its components have been implemented, with the goal of validating its correctness, consistency, and suitability as a learning tool.

Specifically, the analysis focuses on verifying that:

- Gestures are correctly recognized.
- The system rejects incorrect executions.
- The generated feedback is coherent and useful for the user.
- The interaction flow works as defined.
- The visual elements (such as the reference hand or the animations) correspond to the expected gesture.

Since the system is not based on machine learning, validation is not carried out through statistical evaluation or dataset-based metrics. Instead, it is performed through direct analysis of system behavior in a set of representative scenarios.

This validation approach focuses on verifying that the system behaves consistently and correctly under predefined conditions, ensuring that gestures are properly recognized, incorrect executions are rejected, and feedback is generated in a coherent and interpretable manner. Therefore, the evaluation is qualitative and system-oriented, aligned with the deterministic nature of the proposed solution.

6.1 Correct gesture recognition

After implementing the platform gestures, the system was observed to correctly recognize all of them when the user performs the target pose and, in the case of dynamic gestures, the corresponding movement.

This validation has been carried out for both static and dynamic gestures. In static gestures, recognition depends on the hand satisfying the conditions defined

for each finger within the established tolerance margins. In dynamic gestures, in addition to a correct initial pose, the movement must follow the defined pattern. If any of these conditions is not met, the gesture is not validated.

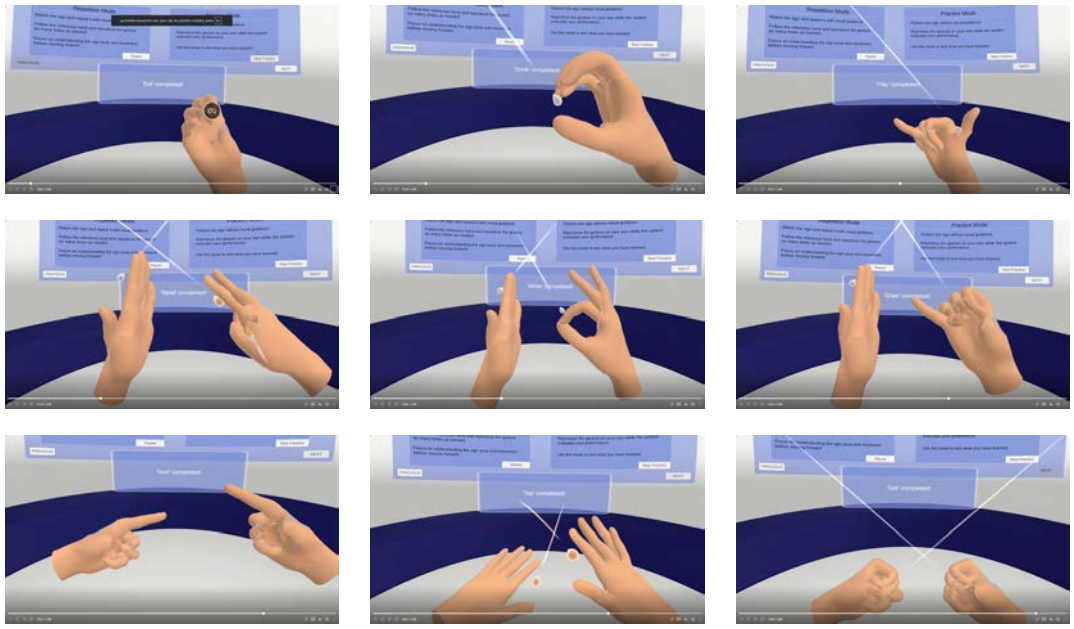


Figure 6.1: Correct recognition of the verbs in the *Advanced* category.

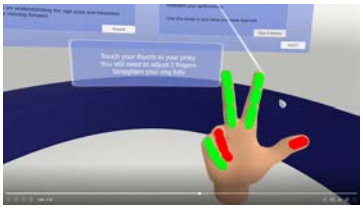
Although individual screenshots of all 84 implemented gestures are not included for brevity, correct recognition has been verified across the full set.

Beyond this general verification, it is also relevant to analyze how the system responds when the user performs a gesture different from the one being evaluated in learning mode. This makes it possible to confirm that the recognizer does not validate incorrect configurations, even when they may seem reasonable from the user's point of view.

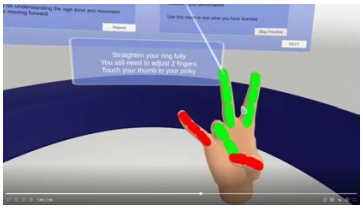
A representative example is given by digits 6 and 7, whose configurations are structurally similar, since both require three extended fingers and differ only in which finger is flexed. The system distinguishes both configurations accurately, rejecting executions that deviate from the criteria defined for the target sign.

Figure 6.2 illustrates this behavior for the sign *6*. When the user extends the fingers corresponding to *3* or *7*, the overlay identifies the incorrect segments in red and the text panel details the necessary corrections. Once the configuration is correct, all segments turn green and the system confirms recognition.

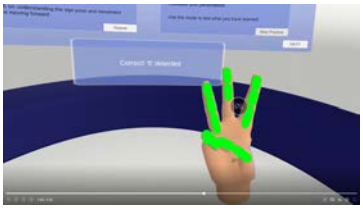
The overlay identifies the incorrect segments in red and the text panel details the necessary corrections finger by finger. Once the configuration has been corrected, the system confirms recognition.



(a) Configuration of *3* when attempting *6*: errors in the ring finger and thumb.



(b) Configuration of *7* when attempting *6*: errors in the thumb and little finger.

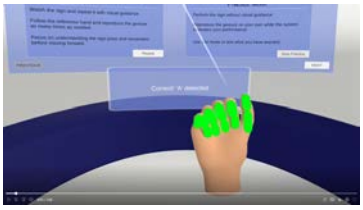


(c) Correct execution of *6*: all segments in green and confirmation from the system.

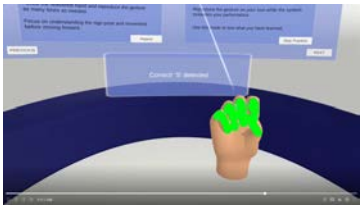
Figure 6.2: Recognition of the sign *6* under similar configurations.

This behavior is especially important in a learning context, since it prevents incorrect executions from being reinforced when the user confuses one sign with another.

In addition, the system correctly distinguishes between configurations with high structural similarity. The letters *A*, *S*, and *E* share a generally closed-hand pose, differing mainly in the position of the thumb and the degree of finger flexion. In all three cases, the system accurately evaluates each joint and confirms recognition when the defined criteria for the target sign are satisfied.



(a) Letter *A*: closed fist with the thumb at the side.



(b) Letter *S*: closed fist with the thumb over the fingers.

(c) Letter *E*: curved fingers with the thumb tucked in.

Figure 6.3: Correct recognition of the letters *A*, *S*, and *E*.

In these cases, the system evaluates the configuration of each finger precisely, which makes it possible to distinguish between signs that may look very similar at first sight.

It should be noted that the main difficulties during recognition have been found in gestures such as N, M, and T, where the thumb must be placed between two fingers, as well as in the gesture R, due to the need to cross the index and middle fingers. These limitations have already been discussed previously and are taken into account in the interpretation of the results.

This same behavior is observed in dynamic gestures. In these cases, it is not

enough to begin from a correct initial pose; the performed movement must also match the defined one.

A representative example is provided by the gestures *Blue* and *Brown*, which share the same initial pose (the *B* configuration, with the fingers extended and together) but differ in the movement. *Blue* requires rotating the wrist from right to left, whereas *Brown* involves sliding the hand downward. The system does not validate the gesture if the movement does not correspond to the expected one, even when the initial pose is the same.

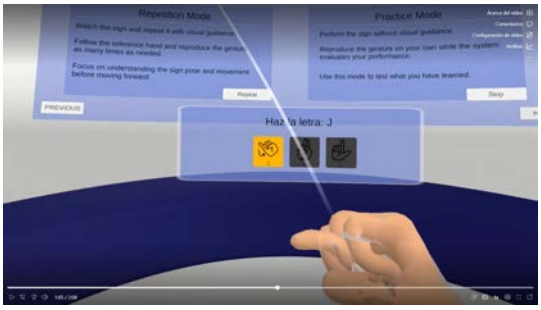
(a) *Blue* completed: wrist rotation from right to left.

(b) *Brown* completed: downward sliding movement along the cheek.

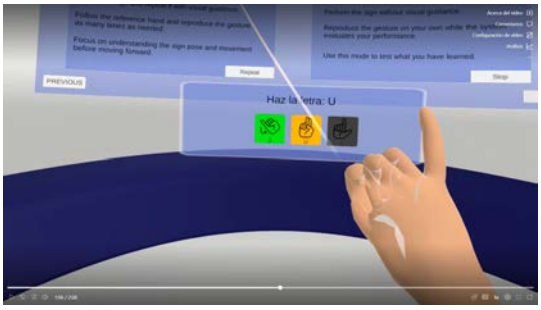
Figure 6.4: Correct recognition of *Blue* and *Brown*. Both start from the same *B* configuration, but the system distinguishes them based on the executed movement.

In the case of sequential gestures, such as those corresponding to the months, it has also been verified that the system behaves as expected. The gesture does not depend on a single configuration, but on a sequence of letters that must be performed in the correct order. The system validates each step progressively and only marks the gesture as completed when the entire sequence has been performed correctly. If the user performs an incorrect letter or changes the order, the system neither validates the gesture nor allows the user to move on to the next step.

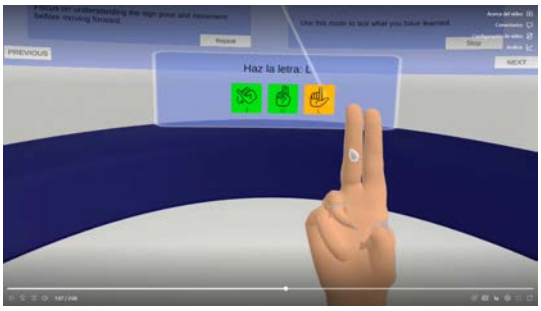
Figure 6.5 shows this behavior for the month of July (*J-U-L*). The indicator for each letter changes from black to yellow when it is in progress and to green when it is validated, so the user can track progress at all times. The message *MONTH COMPLETED!* appears only when all three letters have been executed correctly and in order. Each letter is validated independently, and the system only confirms the month when the complete sequence has been executed correctly and in order.



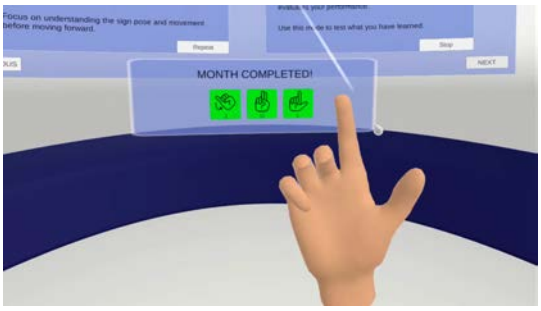
(a) Start of the sequence: *J* in progress (yellow), *U* and *L* pending.



(b) *J* validated (green); the system requests the letter *U*.



(c) *J* and *U* validated; the system requests the letter *L*.



(d) Complete sequence: *MONTH COMPLETED!* with all three letters validated.

Figure 6.5: Progressive validation of the *J*-*U*-*L* sequence for the month of July.

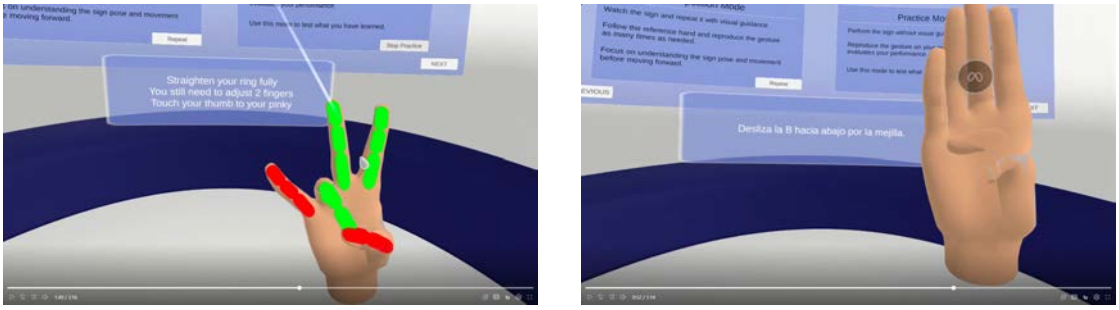
Overall, these checks show that the system correctly recognizes the implemented gestures and does not validate incorrect executions, even when they may appear similar from the user's point of view.

6.2 Feedback analysis

In addition to gesture recognition, one of the key elements of the system is the generation of real-time feedback. The goal is not only to indicate whether a gesture is correct or incorrect, but also to provide information that allows the user to understand what needs to be corrected.

When a gesture is correctly recognized, the system does not generate additional feedback. In these cases, the execution is considered to satisfy all defined conditions, and no corrections are required.

However, when the gesture is not validated, the system activates the geometric analysis of the hand and generates detailed feedback for each finger. This analysis identifies which fingers do not meet the target configuration and in what way they deviate from it.



(a) Static feedback: per-finger overlay with specific corrections.

(b) Dynamic feedback: movement correction during gesture execution.

Figure 6.6: Feedback generated for an incorrect execution of a static pose (left) and during the execution of a dynamic gesture (right). In both cases, the system provides specific information about what needs to be corrected.

As shown in the figure, the system does not simply indicate that the gesture is incorrect, but provides specific information about which fingers need to be corrected. This allows the user to immediately identify which part of the configuration is not correct.

Feedback is generated independently for each finger, identifying not only which fingers do not meet the target configuration, but also in what way they deviate: whether they are too extended, too flexed, or incorrectly oriented. In this way, two incorrect executions may produce different feedback depending on the specific errors in each case.

This level of detail is especially useful in a learning context, as it allows the user to correct specific errors instead of repeating the gesture without understanding what is wrong.

In practice, this behavior facilitates progressive learning, as the user can gradually adjust the position of the fingers until reaching the correct configuration.

Finally, it is important to highlight that the system keeps the recognition decision and feedback generation separate. A gesture is only validated when all conditions are satisfied, but feedback can indicate errors even when the execution is close to being correct. This separation makes it possible to provide useful information without affecting the consistency of recognition.

Overall, the feedback system transforms recognition into a learning tool, providing clear, specific, and actionable information for the user.

6.3 Interaction flow and real-time behavior

After analyzing recognition and feedback, it is necessary to examine the system's behavior during real use. It has been verified that the interaction flow defined in the previous chapter is executed continuously and without interruptions.

During practice, the system follows a constant cycle in which it captures the hand configuration, evaluates the gesture, and, depending on the result, either validates it or generates feedback. This process is repeated at every frame, enabling an immediate response to any change in execution. The user does not need to complete an attempt to know whether the gesture is correct, but can progressively correct it while the system updates the information in real time.

This behavior is especially evident in the self-assessment mode, where the recognizer remains active for all the signs in the category simultaneously, and progress is updated in real time as the user performs each sign. Figure 6.7 shows this behavior across different categories. In digits and colors, progress reflects an ongoing session, with several signs already validated and others still pending.

(a) Digit self-assessment: 5/10 recognized in real time.

(b) Color self-assessment: 5/11 recognized, with the user's hand visible in the scene.

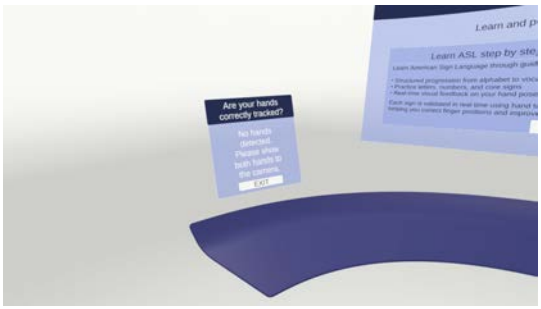
Figure 6.7: Self-assessment mode.

In the case of dynamic gestures, the system also responds continuously during movement, evaluating both the initial pose and the trajectory followed. The transition between states (waiting, evaluation, and validation) occurs smoothly, without perceptible interruptions, allowing the user to focus on execution without losing context during practice.

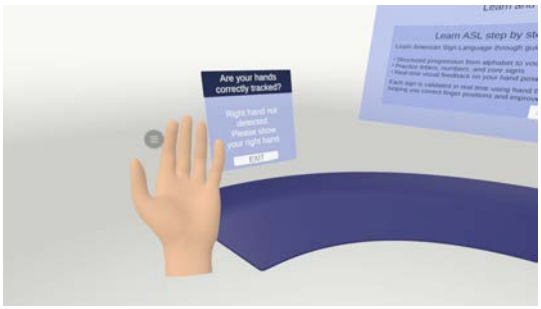
6.4 Tracking and visual coherence

In addition to recognition and feedback, the correct functioning of the hand tracking system has also been verified. The system continuously detects the tracking status and informs the user whether their hands are properly visible to the sensors. This behavior is particularly relevant at the beginning of the session, where the user must ensure that both hands are being captured before starting.

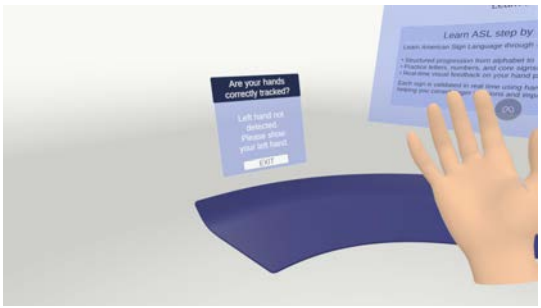
Figure 6.8 shows the four possible states: no hands detected, only the left hand detected, only the right hand detected, and both hands correctly detected. In each case, the system displays a specific message that guides the user in correcting the situation.



(a) No hands detected: the system prompts the user to show both hands.



(b) Only left hand detected: the system indicates that the right hand is missing.



(c) Only right hand detected: the system indicates that the left hand is missing.

(d) Both hands detected: the system confirms that the user is ready to start.

Figure 6.8: Tracking system states at the beginning of the session. The central panel informs the user about the detection status of each hand and guides them to correct it if necessary.

In addition, it has been verified that all animations are correctly assigned to their corresponding gestures. In every case, the displayed animation matches the target sign that the user is expected to reproduce. In dynamic gestures, although

the animations are constructed from a sequence of poses, the final result is a smooth transition that correctly represents the expected movement.

Overall, these elements ensure that the visual representation of the system is consistent with recognition and feedback, contributing to a clear and coherent user experience.

6.5 User Study

In addition to the technical evaluation presented in the previous sections, a user study was conducted to assess the system from an end-user perspective. The objective of this study was to evaluate both the usability of the interaction and the system's effectiveness as a learning tool for sign language gestures.

The study combines quantitative and qualitative measures. Objective metrics such as the number of attempts and the time required to correctly perform each gesture were recorded during the sessions, while subjective feedback was collected through the System Usability Scale (SUS) and a set of open-ended questions.

The following sections describe the participants, the experimental procedure, the results, and their implications. A detailed description of the study protocol, including the session guide, questionnaires, and task definitions, is provided in Appendix F.

6.5.1 Participants

A total of 8 participants took part in the study (4 women and 4 men). Table 6.1 summarizes the demographic profile of the sample.

ID	Age	Gender	VR Experience	ASL Knowledge
P1	24	Male	None	None
P2	25	Male	Low	None
P3	24	Male	None	None
P4	21	Female	Low	None
P5	23	Female	Low	None
P6	29	Female	None	None
P7	45	Female	None	None
P8	59	Male	None	None

Table 6.1: Demographic profile of the study participants.

Participants were recruited on a voluntary basis from the researcher's immediate social environment. The age distribution covered two groups: six participants

were between 21 and 29 years old, and two participants were between 45 and 59 years old. None of the participants had prior knowledge of American Sign Language. Regarding virtual reality experience, five participants had no prior exposure to VR headsets, while three reported low prior experience. This profile is consistent with the intended target user of the application: beginners approaching sign language learning in an introductory and self-guided way.

6.5.2 Procedure

Each session was conducted individually and followed a structured sequence of steps, with an approximate duration of 45 to 60 minutes.

First, participants completed a short pre-session questionnaire to collect demographic information and assess their prior experience with virtual reality and American Sign Language.

Participants were then introduced to the system and asked to freely explore the main screen for approximately two minutes in order to become familiar with the interaction. After this initial exploration, they performed a set of guided tasks designed to cover the main functionalities of the application.

Participants first navigated to the basic level and accessed the alphabet category, where they practiced a set of representative signs. This set included simple static signs (A, B, Y), signs that depend on hand orientation for correct recognition (G and Q), and a dynamic sign (Z), which involves following a spatial trajectory. They then navigated to the basic communication category and performed a set of dynamic gestures: Hello, Bye, Thank You, and Please. Additionally, one sequential sign (the abbreviated spelling of a month) was included to evaluate the system's behavior in multi-phase gesture execution. This design allowed the assessment of the system under different levels of gesture complexity, covering static, orientation-dependent, dynamic, and sequential gesture types.

After the guided practice, participants accessed the self-assessment mode and attempted to perform the previously practiced signs without visual guidance, allowing the evaluation of both recognition performance and the user's ability to recall the gestures independently.

Finally, participants completed the System Usability Scale (SUS) questionnaire [27] and answered a set of open-ended questions about their experience.

Throughout each session, an observer recorded the number of attempts required to correctly perform each sign, the time to first correct recognition, and notable behaviors or difficulties encountered during the interaction.

6.5.3 Results

Recognition performance

Table 6.2 presents the recognition performance for each gesture, expressed as the mean number of attempts and mean time to first correct recognition across all eight participants.

Sign	Type	Mean attempts	SD	Min	Max
A	Static	2.25	0.46	2	3
B	Static	2.25	0.46	2	3
C	Static	2.62	0.52	2	3
Y	Static	2.12	0.35	2	3
G	Static (orientation)	2.00	0.00	2	2
Q	Static (orientation)	2.38	0.52	2	3
Z	Dynamic	3.00	0.93	2	5
Hello	Dynamic	1.00	0.00	1	1
Bye	Dynamic	1.00	0.00	1	1
Thank You	Dynamic	1.62	0.52	1	2
Please	Dynamic	2.38	0.52	2	3
DEC (month)	Sequential	1.00	0.00	1	1

Table 6.2: Recognition performance per gesture: mean number of attempts and standard deviation across 8 participants.

Simple static signs such as A, B, and Y required between 2 and 3 attempts on average, with low variance across participants. The sign G, despite requiring a specific horizontal orientation, did not present significant difficulties once participants observed the guide hand demonstration, achieving a constant mean of 2 attempts with no variance. The sign Q showed slightly higher variability, particularly among older participants (P7 and P8), who reported difficulties maintaining a consistent downward hand orientation.

Among dynamic gestures, Hello and Bye were recognized on the first attempt by all participants without exception, reflecting the straightforward nature of their movement trajectories. Thank You showed slightly higher variability, with some participants requiring two attempts. Please presented the highest difficulty among dynamic gestures, with a mean of 2.38 attempts, as participants found the circular movement harder to execute consistently. Z showed the widest variability ($\sigma = 0.93$), with P1 requiring 5 attempts due to initial speed calibration difficulties. The sequential gesture (DEC) was completed on the first attempt by all participants.

System Usability Scale

Table 6.3 presents the SUS scores obtained for each participant, calculated following the standard scoring method defined by Brooke [27].

	P1	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**Mean**	**SD**
SUS score	85.0	85.0	87.5	95.0	92.5	75.0	65.0	80.0	**83.1**	9.7

Table 6.3: SUS scores per participant and overall statistics.

The mean SUS score of 83.1 ($\sigma = 9.7$) places the system in the *Good* category according to the standard SUS interpretation scale, approaching the *Excellent* threshold of 85. Six out of eight participants scored above 75, indicating good usability across most of the sample. The two lowest scores corresponded to P7 (65.0) and P6 (75.0), the two oldest participants in the study. Participants in the 21–29 age group obtained a mean SUS score of 86.7, compared to 72.5 for participants aged 45 and above, suggesting that prior familiarity with digital and interactive environments may influence the perceived usability of the system.

Overall satisfaction and qualitative feedback

In addition to the SUS questionnaire, participants rated their overall experience on a scale from 1 to 5. The mean overall rating was 4.4 out of 5, with six participants assigning a score of 4 and two assigning a score of 5, indicating a consistently positive evaluation across the sample.

Regarding the open-ended questions, the responses highlighted several recurring themes. Concerning the finger-level visual feedback, all participants reported that the color-coded overlay was easy to understand and useful for correcting errors in real time. The guide hands were consistently mentioned as a helpful reference for understanding both static poses and dynamic movements.

The most frequently cited difficulty was the recognition of certain gestures, particularly among participants who reported that performing the sign correctly did not always lead to immediate recognition. This observation was more pronounced for orientation-dependent signs such as Q, and for dynamic gestures involving trajectory constraints such as Z and Please. Some participants also mentioned initial uncertainty about the expected speed or number of repetitions for circular gestures.

When asked whether they would use the application independently to learn sign language, all eight participants answered affirmatively. Representative responses included descriptions of the system as *"very useful, simple but complete"*, *"very interesting and a good use of technology"*, and *"a good tool for self-guided learning"*.

6.5.4 Limitations

Although the results are encouraging, the study presents several limitations that should be acknowledged. The sample size is small (8 participants), which limits the statistical power of the quantitative results. The evaluation was conducted in a controlled environment over a single session, which does not capture long-term learning effects or the impact of repeated use. In addition, the age distribution is unbalanced, with only two participants above 40 years old.

Future work could address these limitations through a larger and more diverse sample, longitudinal studies to evaluate retention and progression over time, and comparative evaluations across different user profiles.

6.5.5 Design Implications

The observations from the user study highlight several aspects that can be improved in future iterations of the system. The variability observed in Z and Please suggests that the speed thresholds and trajectory constraints for dynamic gestures may benefit from adaptive calibration based on the user's movement profile. The difficulties reported with Q indicate that orientation-dependent signs could benefit from enhanced visual guidance that more explicitly communicates the expected palm direction.

The consistent positive reception of the finger-level feedback and the guide hand system confirms that these two components are central to the learning experience and should be preserved and refined in future versions. The results also reinforce the importance of aligning the demonstration speed of the guide hands with the recognition thresholds of the system, as users tend to replicate the movement speed shown in the animation.

Overall, these results show that the system behaves consistently across different types of gestures and execution scenarios, providing reliable recognition and meaningful feedback.

More importantly, these results indicate that the proposed approach is not only technically sound, but also suitable for learning purposes. By combining gesture validation with detailed feedback and visual guidance, the system supports a structured and progressive learning process, going beyond simple recognition and enabling users to actively improve their performance.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This work has presented the development of an immersive virtual reality platform for sign language learning, focused on manual gesture recognition and real-time feedback generation.

The implemented system allows users to practice different signs and receive immediate evaluation of their execution. Unlike other approaches based on machine learning models, recognition has been formulated through a deterministic approach based on configurable geometric and temporal conditions. This enables consistent and fully interpretable behavior.

Throughout the development, it has been verified that the system is capable of correctly recognizing the implemented gestures, rejecting incorrect executions, and providing detailed feedback that helps the user correct them. In addition, the interaction flow and visual representation behave coherently, which contributes to a continuous learning experience.

Overall, the developed system demonstrates that it is possible to build a virtual reality sign language learning tool that not only validates gestures, but also actively helps users improve them.

7.2 Contributions of the work

The main contributions of this work are the following.

First, the development of a complete virtual reality sign language learning platform that allows users to actively practice manual gestures within an immersive environment and receive immediate feedback during execution. The system is not limited to displaying content, but instead places the user in a learning process based on direct practice.

Second, the proposal of a deterministic and interpretable approach for manual gesture recognition. Instead of using machine learning models, each sign is defined through explicit geometric and temporal conditions, making it possible to evaluate transparently how the gesture is constructed and to guarantee consistent system behavior.

Another key contribution is the design of an explainable feedback system, capable of identifying specific errors in the configuration of each finger and providing targeted guidance for correction. In this way, recognition ceases to be a binary validation process and becomes an active learning tool.

Likewise, the system incorporates a visual reference representation through a virtual hand that shows the correct execution of the gesture. This guide allows the user to learn through direct imitation, comparing their own execution with the reference in real time, which reinforces the motor learning process.

In addition, a unified architecture has been developed that makes it possible to integrate static, dynamic, and sequential gestures within a single system, while maintaining coherence in their definition, validation, and evaluation.

Finally, this work demonstrates how to effectively integrate motion capture, gesture recognition, and feedback generation within a virtual reality environment, ensuring a continuous interaction flow and a consistent user experience.

7.3 Future work

Several possible extensions of the developed system are proposed as future work.

First, the vocabulary available in the platform could be expanded. This would not only involve increasing the number of signs, but also moving towards more complex gestures that include the coordinated use of both hands, arm movement, and, at a more advanced stage, the incorporation of facial expressions, which form an essential part of sign language.

Second, the development of a user-tracking system is proposed in order to record user progress over time. This system could include learning metrics, identification of the gestures that present the greatest difficulty, and analysis of the most frequent errors. In this way, it would be possible to personalize the learning experience and provide the user with useful information to improve more efficiently.

Finally, the incorporation of a translation mode is proposed, allowing the user to perform signs and obtain their text representation through the platform, as well as the inverse process. This functionality would represent an additional step towards more practical applications of the system beyond guided learning.

In addition, as a natural continuation of the work carried out, it is proposed to improve those aspects that presented limitations during development and could

not be fully resolved. These include certain complex recognition cases related to finger interaction or tracking limitations, which could be addressed in future versions of the system in order to increase its robustness.

Bibliography

[1] World Health Organization. *Deafness and Hearing Loss – Fact Sheet*. 2025. URL: https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss (visited on 11/03/2025).

[2] UNESCO. *Sign Languages Are for Everyone!* 2023. URL: https://www.unesco.org/en/articles/sign-languages-are-everyone (visited on 11/03/2025).

[3] United Nations. *Transforming Our World: The 2030 Agenda for Sustainable Development*. 2015. URL: https://sdgs.un.org/2030agenda.

[4] S. Geetha et al. "Human Interaction in Virtual and Mixed Reality Through Hand Tracking". In: *IEEE International Conference on Electronics, Computing and Communication Technologies*. 2024.

[5] Rubén Grande et al. "Enhancing Hand Interactions and Accessibility in Virtual Reality Environments for Users With Motor Disabilities: A Practical Case Study on VR-Shopping". In: *IEEE Access* (2025). DOI: 10.1109/ACCESS.2025.3549527.

[6] J. Ahn and S. Park. "DiVRsity: Design and Development of Group Role-Play VR Platform for Disability Awareness Education". In: *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2023.

[7] R. Kaviyaraj, P. Sathya, P. Nandhini, et al. "Augmented Reality and Artificial Intelligence in Sign Language Expression". In: *Third International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS)*. 2024.

[8] Hongli Wen et al. "Enhancing Sign Language Teaching: A Mixed Reality Approach for Immersive Learning and Multi-Dimensional Feedback". In: *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2024.

[9] Liam Iñaki B. Gillamac and Roberto B. Figueroa Jr. "Exploring Augmented Visual Guidance for Sign Language Learning". In: *IEEE ECTI-CON*. 2025.

[10] Necati Cihan Camgoz et al. "Neural Sign Language Translation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[11] Fidaa Khandaqji, Huthaifa I. Ashqar, and Abdelrahem Atawnih. "A Survey of Using Artificial Intelligence in Sign Language for Deaf and Hard-of-Hearing Students". In: *International Conference on Smart Learning Courses*. 2025.

[12] Laura Freina and Michela Ott. "A Literature Review on Immersive Virtual Reality in Education". In: *eLearning & Software for Education* (2015).

[13] Jaziar Radianti et al. "A Systematic Review of Immersive Virtual Reality Applications for Higher Education: Design Elements, Lessons Learned, and Research Agenda". In: *Education and Information Technologies* (2020). DOI: 10.1007/s10639-019-09959-1.

[14] Khronos Group. *OpenXR Specification*. 2024. URL: https://www.khronos.org/openxr/ (visited on 02/17/2026).

[15] Unity Technologies. *Unity 6 Documentation*. 2024. URL: https://docs.unity3d.com.

[16] Unity Technologies. *XR Plugin Management - Unity Manual*. 2024. URL: https://docs.unity3d.com/Manual/com.unity.xr.management.html (visited on 02/17/2026).

[17] Unity Technologies. *XR Interaction Toolkit Documentation*. 2024. URL: https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit.

[18] Meta. *Meta OpenXR Package Documentation*. 2024. URL: https://developer.oculus.com/documentation/unity/unity-openxr.

[19] Unity Technologies. *XR Hands Package Documentation*. 2024. URL: https://docs.unity3d.com/Packages/com.unity.xr.hands@latest (visited on 02/17/2026).

[20] Meta. *Meta Hand Tracking Overview*. 2024. URL: https://developer.oculus.com/documentation/unity/hand-tracking.

[21] Meta. *Meta Quest 3 Specifications*. 2024. URL: https://www.meta.com/quest/quest-3.

[22] Dongxu Li et al. "Word-Level Deep Sign Language Recognition from Video: A New Large-Scale Dataset and Methods Comparison". In: *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2020.

[23] Oscar Koller, Hermann Ney, and Richard Bowden. "Deep Hand: How to Train a CNN on 1 Million Hand Images When Your Data is Continuous and Weakly Labelled". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[24] Unity Technologies. *XR Hand Data Model — Unity XR Hands documentation*. `https://docs.unity3d.com/Packages/com.unity.xr.hands@1.4/manual/hand-data/xr-hand-data-model.html`. Accedido: 2025. 2024.

[25] Unity Technologies. *Finger Shapes — Unity XR Hands documentation*. `https://docs.unity3d.com/Packages/com.unity.xr.hands@1.4/manual/gestures/finger-shapes.html`. Accedido: 2025. 2024.

[26] Meta. *Meta Quest 3*. 2023. URL: `https://www.meta.com/quest/quest-3/`.

[27] John Brooke. "SUS: A quick and dirty usability scale". In: *Usability Evaluation in Industry*. Ed. by P. W. Jordan et al. Taylor & Francis, 1996, pp. 189–194.

[28] United Nations. *Transforming our world: the 2030 Agenda for Sustainable Development*. Resolution A/RES/70/1. United Nations General Assembly. 2015. URL: `https://sdgs.un.org/2030agenda`.

[29] World Health Organization. *World report on hearing*. Geneva: WHO Press, 2021. URL: `https://www.who.int/publications/i/item/world-report-on-hearing`.

[30] Marit Hoem Kvam, Mitchell Loeb, and Kristian Tambs. "Mental health in deaf adults: symptoms of anxiety and depression among hearing and deaf individuals". In: *Journal of Deaf Studies and Deaf Education* 12.1 (2007), pp. 1–7.

[31] Steven Barnett et al. "Community participatory research with deaf sign language users to identify health inequities". In: *American Journal of Health Behavior* 35.6 (2011), pp. 765–775.

[32] UNESCO. *Convention on the Rights of Persons with Disabilities, Article 24: Education*. United Nations. New York, 2006. URL: `https://www.un.org/development/desa/disabilities/%20convention-on-the-rights-of-persons-with-disabilities.html`.

[33] Ibrahim Abaker Adeyanju, Oluwaseun Oyinlola Bello, and Modinat Abolore Adegboye. "Methods of sign language recognition: a review". In: *Heliyon* 7.10 (2021), e08203.

[34] Razieh Rastgoo, Kambiz Kiani, and Sergio Escalera. "Sign language recognition: a deep survey". In: *Expert Systems with Applications* 164 (2021), p. 113794.

[35] Munir Oudah, Ali Al-Naji, and Javaan Chahl. "Hand gesture recognition based on computer vision: a review of techniques". In: *Journal of Imaging* 6.8 (2020), p. 73.

[36] Pramod Kumar Pisharady and Martin Saerbeck. "Recent methods and databases in vision-based hand gesture recognition: a review". In: *Computer Vision and Image Understanding* 141 (2015), pp. 152–165.

[37] Brenda Schick, Marc Marschark, and Patricia Elizabeth Spencer. *Advances in the Sign Language Development of Deaf Children*. New York: Oxford University Press, 2006.

[38] Michael McKee, Denise Schlehofer, and Denise Thew. "Ethical issues in conducting research with deaf populations". In: *American Journal of Public Health* 103.12 (2015), pp. 2174–2178.

Appendix A: Alignment with the Sustainable Development Goals

This Master's Thesis is developed within the framework of the United Nations 2030 Agenda for Sustainable Development [28], whose Sustainable Development Goals (SDGs) guide the global commitment towards more equitable, inclusive, and sustainable societies. *ASL LearnVR* is a technological platform with a clear educational and social purpose, aimed at facilitating the autonomous learning of American Sign Language for hearing individuals, thereby reducing the communication barriers faced by deaf people or individuals with speech impairments in their daily lives. The SDGs to which the project is directly and meaningfully connected are outlined below.

SDG 3: Good Health and Well-being

SDG 3 aims to ensure healthy lives and promote well-being for all [28]. The World Health Organization estimates that more than 1.5 billion people worldwide live with some degree of hearing loss, of whom approximately 430 million require audiological care, and this number is projected to exceed 700 million by 2050 [29]. The communicative isolation resulting from the lack of competent sign language interlocutors has well-documented consequences for mental health. It increases the risk of depression, anxiety, and social exclusion within the deaf community [30, 31].

ASL LearnVR contributes to this goal by enabling hearing individuals to acquire ASL skills autonomously, thereby increasing the number of potential interlocutors for deaf individuals and reducing communicative isolation. By removing the need for an instructor and allowing practice at any time and place, the platform also reduces the economic and logistical barriers that limit access to sign language education.

SDG 4: Quality Education

SDG 4 aims to ensure inclusive, equitable, and quality education, and to promote lifelong learning opportunities for all [28]. UNESCO recognizes sign language as the natural language of deaf individuals and emphasizes the need to develop educational resources that promote its teaching and dissemination [32]. However, access to sign language education remains limited in many contexts, as the availability of in-person courses is often restricted outside major cities. In addition, existing digital materials typically do not incorporate structured gesture evaluation, and many automatic recognition systems focus primarily on detection rather than pedagogical feedback [33, 34].

ASL LearnVR directly addresses this gap. The platform integrates, within a single immersive environment, teaching through three-dimensional visual references, controller-free active practice, and structured gesture evaluation with detailed articulation-level feedback. This approach goes beyond the binary detection model (correct or incorrect) and provides users with specific information on what needs to be corrected and how, without requiring an external instructor. Autonomous access and the ability to practice at one's own pace make the platform consistent with the principles of lifelong learning and accessible education promoted by SDG 4.

SDG 9: Industry, Innovation and Infrastructure

SDG 9 promotes the development of resilient infrastructures, inclusive industrialization, and innovation [28]. *ASL LearnVR* represents a concrete technical contribution at the intersection of educational virtual reality and gesture recognition. The developed system demonstrates that it is possible to implement an interpretable, efficient, and pedagogically valuable hand gesture recognition engine without relying on machine learning models or cloud-based infrastructure.

The deterministic approach adopted, based on configurable geometric and temporal conditions, constitutes a reproducible and transparent alternative to neural network-based approaches, with the added advantage of operating in real time on embedded hardware with limited computational resources [35, 36]. The modular system architecture, which separates capture, recognition, pedagogical analysis, and feedback into independent components, facilitates its extension to new sign languages or categories of signs without the need to retrain models or modify the core logic. This capacity for replication and adaptation is particularly relevant from the perspective of technological transferability to other educational contexts.

SDG 10: Reduced Inequalities

SDG 10 promotes the social, economic, and political inclusion of all individuals, regardless of their condition [28]. The deaf community faces structural inequalities in access to communication, education, and employment, largely due to the lack of competent sign language interlocutors among the hearing population [37, 38]. This communicative asymmetry not only limits the autonomy of deaf individuals but also hinders their full participation in professional, healthcare, and educational environments.

ASL LearnVR addresses this inequality from both an educational and technological perspective. By facilitating access to ASL learning for hearing individuals without barriers of schedule, cost, or geographic location, the platform contributes to expanding the base of competent sign language interlocutors and reducing the communicative gap between deaf and hearing communities. The platform design (including guided feedback, three-dimensional visual references, and autonomous practice) enables individuals with no prior training in sign language to progressively acquire basic skills, which has a direct impact on the communicative inclusion of deaf individuals in their daily environments.

Appendix B: Formal Definition of Static Hand Gestures

This appendix provides a complete formal specification of all static gesture recognition profiles implemented in ASL LearnVR. Each gesture is defined through a `FingerConstraintProfile` ScriptableObject that specifies, for each of the five fingers, a curl constraint and, where applicable, a spread constraint and a thumb-contact condition.

B.1 Constraint Model

The system uses a deterministic, constraint-based recognition model. Joint data are provided per-frame by the Unity `XR Hands` subsystem (`XRHandJointsUpdatedEventArgs`), and the final accept/reject decision is taken by the `XRHandShape.CheckConditions()` or `XRHandPose.CheckConditions()` API. The `FingerConstraintProfile` operates as a parallel pedagogical layer, as it evaluates the same joint data to identify which fingers deviate from the target configuration and to generate corrective feedback messages.

Two constraint types are defined per finger:

- **Curl constraint** (`CurlConstraint`): a continuous normalised value $c \in [0, 1]$, where 0 denotes a fully extended finger and 1 a fully closed finger (touching the palm). A finger passes the constraint if and only if $c_{\min} \leq c \leq c_{\max}$.
- **Spread constraint** (`SpreadConstraint`): an angular separation θ (degrees) between adjacent fingers, with acceptable range $[\theta_{\min}, \theta_{\max}]$. Enabled only where phonemically relevant (e.g. V, W, K, B).

The thumb additionally supports four boolean contact flags: `shouldBeBesideFingers`, `shouldBeOverFingers`, `shouldTouchIndex`, `shouldTouchMiddle`, `shouldTouchRing`, and `shouldTouchPinky`. Each constraint carries a severity level (*Major* or *Minor*) that determines the priority of the feedback message shown to the user.

B.2 Three-State Finger Taxonomy

Each finger is assigned a semantic state derived from the midpoint of its curl range, $\bar{c} = (c_{\min} + c_{\max})/2$:

Table 1: Semantic finger states derived from the curl midpoint

State	Condition on $\bar{c}$	Description
Extended	$\bar{c} < 0.30$	Finger is straight, with no flexion
Curved	$0.30 \leq \bar{c} \leq 0.72$	Controlled bend, fingertips do not contact the palm
Closed	$\bar{c} > 0.72$	Full flexion, fingertips contact the palm

B.3 Shared Curl Threshold Constants

The following named constants are defined in `AlphabetConstraintProfiles` and `DigitConstraintProfiles` and are reused across all gesture profiles:

Table 2: Named curl threshold constants used in constraint profiles

Constant	$c_{\min}$	$c_{\max}$	Semantic meaning
`EXTENDED`	0.00	0.45	Finger straight, no flexion
`PARTIAL`	0.30	0.65	Intermediate bend; used for circles (C, O, F)
`TIP_CURL`	0.55	0.78	Fingertips curl to the knuckles only (E, M, N)
`CURLED`	0.55	1.00	General closed position
`FULL_CURL`	0.85	1.00	Complete fist; palm contact (A, S, T)

The distinction between *tip curl* and *full curl* is meaningful at the level of sign distinction. A and S require $c \geq 0.85$ across all four fingers, whereas E, M, N and T require only $c \leq 0.78$. The gap between these two ranges ($0.78 < c < 0.85$) acts as a disambiguation margin.

B.4 Complete Alphabet Constraint Profiles (A–Z)

Table 3 lists the curl range $[c_{\min}, c_{\max}]$ for each finger of every static letter. Letters G, H, P, and Q also carry an orientation constraint (`checkOrientation = true`) noted in the last column. The symbol • in the Thumb column indicates an active contact flag, its target finger is specified in parentheses.

Table 3: Finger curl constraints for all 26 ASL alphabet signs. Curl values: 0 = extended, 1 = fully closed. Abbreviations: E = Extended [0.00, 0.45]; Cu = Curled [0.55, 1.00]; FC = Full Curl [0.85, 1.00]; TC = Tip Curl [0.55, 0.78].

Sign	Thumb	Index	Middle	Ring	Pinky	Notes
A	[0.00, 0.35] beside	FC	FC	FC	FC	—
B	[0.50, 1.00]	E + spread $[-2, +8]$	E + spread $[-2, +8]$	E + spread $[-2, +8]$	E	—
C	[0.20, 0.50]	[0.35, 0.65]	[0.35, 0.65]	[0.35, 0.65]	[0.35, 0.65]	—
D	[0.35, 0.75]	E	[0.35, 0.78]	[0.35, 0.78]	[0.35, 0.78]	—
E	[0.35, 0.75]	TC	TC	TC	TC	—
F	[0.30, 0.60] touch idx	[0.30, 0.60]	E	E	E	—
G	[0.00, 0.30]	E	Cu	Cu	Cu	Orientation: sideways
H	[0.30, 1.00]	E	E	Cu	Cu	Orientation: sideways
I	[0.30, 1.00]	Cu	Cu	Cu	E	—
K	[0.30, 0.60] touch mid	E + spread $[8, +30]$	E	Cu	Cu	—
L	[0.00, 0.30]	E	Cu	Cu	Cu	—
M	[0.35, 0.80] touch mid+ring	TC	TC	TC	[0.70, 1.00]	—
N	[0.35, 0.80] touch mid+ring	TC	TC	[0.70, 1.00]	[0.70, 1.00]	—
O	[0.35, 0.70]	[0.35, 0.70]	[0.35, 0.70]	[0.35, 0.70]	[0.35, 0.70]	—
P	[0.30, 0.60] touch mid	E + spread $[8, +30]$	E	Cu	Cu	Orientation: downward
Q	[0.00, 0.30]	E	Cu	Cu	Cu	Orientation: downward
R	[0.30, 1.00]	[0.30, 0.65]	[0.30, 0.65]	Cu	Cu	—
S	[0.30, 0.70] over	FC	FC	FC	FC	—
T	[0.30, 0.70] touch pinky	FC	FC	[0.70, 1.00]	[0.70, 1.00]	—
U	[0.35, 1.00]	E	E	Cu	Cu	—
V	[0.30, 1.00]	E + spread $[8, +30]$	E	Cu	Cu	—
W	[0.30, 1.00]	E + spread $[6, +28]$	E + spread $[6, +28]$	E + spread $[4, +24]$	Cu	—
X	[0.30, 1.00]	[0.40, 0.70] Curved	Cu	Cu	Cu	Index hook shape
Y	[0.00, 0.30]	Cu	Cu	Cu	E	—

B.5 Digit Constraint Profiles (0–9)

Table 4 lists the constraints for the ten ASL digits. Digit 0 shares the same hand shape as the letter O. Digits 1 through 5 are defined by progressive extension of additional fingers. Digits 6 through 9 introduce thumb-to-finger contact conditions on specific fingertips.

Several structural patterns are worth noting. First, digits 3 and 5 share the same curl ranges as digits 2 and 4 respectively, but require the thumb to be extended ($c \leq 0.35$) rather than tucked, which is the distinguishing feature in each pair. Second, digits 6–9 implement a *thumb-pinch* pattern: the thumb contacts a specific finger (pinky, ring, middle, or index respectively) while the remaining three fingers are extended. This makes them structurally analogous to the letter F (thumb–index contact, others extended).

B.6 Relationship to the XR Hands Recognition Pipeline

The `FingerConstraintProfile` is used exclusively by the pedagogical feedback system (`HandPoseAnalyzer` → `FeedbackSystem`). The binary recognition decision (accepted or rejected) is taken upstream by the Unity `XRHandShape` or

Table 4: Finger curl constraints for ASL digits 0–9. Abbreviations as in Table 3. •(−−) denotes a thumb-contact flag with the specified finger.

Digit	**Thumb**	**Index**	**Middle**	**Ring**	**Pinky**
0	[0.35, 0.70]	[0.35, 0.70]	[0.35, 0.70]	[0.35, 0.70]	[0.35, 0.70]
1	[0.30, 1.00]	E	Cu	Cu	Cu
2	[0.35, 1.00]	E	E	Cu	Cu
3	[0.00, 0.35]	E	E	Cu	Cu
4	[0.35, 1.00]	E	E	E	E
5	[0.00, 0.35]	E	E	E	E
6	[0.20, 0.60] •(pinky)	E	E	E	[0.30, 0.65]
7	[0.20, 0.60] •(ring)	E	E	[0.30, 0.65]	E
8	[0.20, 0.60] •(middle)	E	[0.30, 0.65]	E	E
9	[0.20, 0.60] •(index)	[0.30, 0.65]	E	E	E

`XRHandPose` assets referenced in each `SignData` ScriptableObject. These two layers are intentionally decoupled, as recognition determines *whether* a gesture is correct, while the constraint profile explains *why* it is not and *how* to correct it (see B 5 for the complete feedback architecture).

Feedback messages produced from these profiles are subject to a hysteresis filter in `FeedbackSystem` an error must persist for at least 250 ms before being displayed (`messageEnterDelay`), and must be absent for at least 450 ms before being hidden (`messageExitDelay`). At most three correction messages are shown simultaneously, sorted by severity (*Major* > *Minor*) and number of affected fingers.

Appendix C: System Architecture and Class Structure

This appendix describes the software architecture of ASL LearnVR from an engineering perspective. The system follows a modular design in which each component has a clearly defined responsibility. This section documents the principal classes, their relationships, and the key design decisions that govern the overall structure.

C.1 Application Scenes and Navigation

The application is organised into four Unity scenes that map directly onto the four main interaction modes. Navigation between scenes is managed by the singleton `SceneLoader`, which performs asynchronous loading via `SceneManager.LoadSceneAsync` and prevents concurrent loads. Session state (selected level, category, and sign) persists across scenes through the singleton `GameManager`, which survives scene transitions via `DontDestroyOnLoad`.

Table 5: Application scenes and their controlling components

Scene	Name	Main controller
1	`01_MainMenu`	`MenuController`
2	`02_LevelSelection`	`LevelSelectionController`
3	`03_LearningModule`	`LearningController`
4	`04_SelfAssessmentMode`	`SelfAssessmentController`

C.2 Data Layer

The content of the application is represented through a three-level ScriptableObject hierarchy. All assets are configured in the Unity Inspector and are loaded at runtime without requiring code changes.

- `SignData`: represents a single ASL sign. Stores the sign name, a reference to the corresponding `XRHandShape` or `XRHandPose` asset (used for recognition), the `requiresMovement` flag that determines whether the sign requires a dynamic gesture, the minimum hold time for confirmation, and an optional icon.
- `CategoryData`: groups a list of `SignData` assets under a thematic name (e.g., *Alphabet*, *Colors*, *Verbs*). Provides a `GetSignByName()` method for lookup by name.
- `LevelData`: groups a list of `CategoryData` assets and defines difficulty parameters (`minimumHoldTime`, `recognitionAccuracy`). The three levels defined in the project are Basic, Intermediate, and Advanced.

A specialised subclass `MonthSequenceData` extends `SignData` to represent month signs, which are practised as sequences of three letters managed by `MonthPracticeController`.

C.3 Static Gesture Recognition Pipeline

Static signs are recognised by comparing the live joint data of the user's hand against `XRHandShape` or `XRHandPose` assets configured per sign. Two recogniser classes are provided depending on whether a single sign or multiple signs must be detected simultaneously.

- `GestureRecognizer` monitors a single `SignData` target. Subscribes to `XRHandTrackingEvents.jointsUpdated` and calls `XRHandShape.CheckConditions()` or `XRHandPose.CheckConditions()` every `detectionInterval` seconds (default: 0.1 s). A gesture is confirmed only after the user holds the pose for at least `minimumHoldTime` seconds (default: 0.3 s). A loss-tolerance window of 0.2 s prevents false negatives from momentary tracking drops. Used in Scene 3 (Learning Module).
- `MultiGestureRecognizer`: monitors a list of `SignData` targets simultaneously and raises events for the best-matching sign at each analysis tick. Coordinates with `DynamicGestureRecognizer` via the `OnPendingConfirmationChanged` event to avoid simultaneous static/dynamic confirmations on the same hand shape. Implements a grace-period counter (`MAX_CONSECUTIVE_MISSES` = 2) to suppress flickering. Used in Scene 4 (Self-Assessment).

C.4 Dynamic Gesture Recognition Pipeline

Dynamic gestures (signs that require movement) are handled by a dedicated subsystem composed of three collaborating classes.

- `DynamicGestureRecognizer`: the central component. Implements a three-state internal state machine (`Idle` → `PendingConfirmation` → `InProgress`) that drives the recognition lifecycle. On each `Update` frame it reads the palm joint position and rotation from `XRHandSubsystem`, applies exponential smoothing (`positionSmoothingFactor` = 0.7 by default), and delegates motion analysis to `MovementTracker`. Detection of the initial hand pose is delegated through the `IPoseAdapter` interface.
- `MovementTracker`: maintains a sliding window of position and rotation samples (default: 3 s, 120 samples) and computes five motion metrics at each frame: total distance (m), current speed (m/s, averaged over 3 frames), average direction (last 30% of trajectory), total rotation (degrees), direction-change count (threshold: 30), and circularity score (geometric variance of radii from centroid).
- `DynamicGestureDefinition`: a ScriptableObject that encodes all requirements of a dynamic gesture: initial and end static pose references (`StaticPoseRequirement` list with `PoseTimingRequirement` enum), movement direction and tolerance, minimum speed and distance, temporal bounds (`minDuration`/`maxDuration`), and optional flags for direction-change, rotation, circular motion, and spatial zone requirements.

IPoseAdapter interface

`DynamicGestureRecognizer` accesses the current detected pose name through the `IPoseAdapter` interface (single method: `GetCurrentPoseName()`), which decouples it from the concrete recogniser in use:

- `StaticPoseAdapter`: wraps `MultiGestureRecognizer`. Used in Scene 4 where many signs are active simultaneously.
- `SingleGestureAdapter`: wraps a single `GestureRecognizer`. Used in Scene 3 where only the currently practised sign is active. Includes a 0.3 s loss-tolerance window.

Disambiguation logic

When the initial pose matches more than one gesture definition, `DynamicGestureRecognizer` enters `PendingConfirmation` state and uses movement onset to dis-

ambiguate. The timeout is 0.4 s in Scene 3 and 1.2 s in Scene 4. For compound gestures (those with mandatory end poses, e.g., *White*: 5→S, *Thursday*: T→H), disambiguation falls back to end-pose validation via `XRHandShape.CheckConditions()` using cached `XRHandJointsUpdatedEventArgs`.

C.5 Pedagogical Feedback System

The feedback system is fully decoupled from gesture recognition. Recognition determines *whether* a gesture is correct, while the feedback system explains *why* it is not and *how* to correct it.

- `FeedbackSystem`: the main orchestrator. Activated and deactivated by `LearningController` when the user enters or leaves practice mode. Subscribes to events from both `GestureRecognizer` (static) and `DynamicGestureRecognizer` (dynamic). Runs `HandPoseAnalyzer.Analyze()` every 0.2 s for static gestures. Applies a message-hysteresis filter (enter delay: 250 ms; exit delay: 450 ms) to prevent flickering. For dynamic gestures, manages a five-phase feedback lifecycle (Idle → StartDetected → InProgress → NearCompletion → Completed/Failed) with a message-latch mechanism, where success locks the message for 3 s and failure locks it for 1.0–1.3 s (randomised).
- `HandPoseAnalyzer`: reads live curl values from `XRHandSubsystem` joints and evaluates them against the active `FingerConstraintProfile`. Curl is computed geometrically from joint angles between consecutive finger segments (proximal, intermediate, distal, tip), normalised to [0, 1]. Applies an effective tolerance of 0.08 (thumb: 0.12). Errors are classified by severity (*Major*: deviation > 0.18; *Minor*: deviation > 0.08) and by semantic error type (see Appendix B).
- `FingerConstraintProfile`: ScriptableObject defining the per-finger curl, spread, and thumb-contact constraints for a given sign. Profiles for all 26 letters and 10 digits are generated programmatically by `AlphabetConstraintProfiles` and `DigitConstraintProfiles` and loaded automatically on startup by `HandPoseAnalyzer`.
- `FingerError` (struct): represents a single detected deviation. Fields: `finger` (`Finger` enum), `errorType` (`FingerErrorType` enum), `severity` (`Severity` enum), `currentValue`, `expectedValue` (both $\in [0, 1]$), and `correctionMessage` (user-facing string).
- `StaticGestureResult` / `DynamicGestureResult`: data-transfer objects that carry the full analysis output. `StaticGestureResult` includes `isMatchGlobal`, `perFingerErrors`, `matchScore` ($= 1 - 0.3 \cdot N_{\text{major}} - 0.1 \cdot N_{\text{minor}}$), and `isNearMatch`.

`DynamicGestureResult` includes `isSuccess`, `failureReason` (`FailureReason` enum), `failedPhase` (`GesturePhase` enum), and the full `DynamicMetrics` struct.

C.6 Visual Feedback Components

`XRFingerOverlayRenderer` renders capsule-shaped overlays along each finger segment. The components expose a `SetVisible(bool)` method. `FeedbackSystem` turns them off during dynamic gesture execution (when the user is moving) and restores them in Idle state.

The `GuideHandPoseApplier` drives a decoupled ghost hand mesh (the guide hand) by applying `HandPoseData` structs from `ASLPoseLibrary` to a hierarchy of 21 joint transforms using smooth interpolation (`transitionSpeed` = 5). The guide hand is faded out when practice mode begins and faded back in when it ends.

C.7 Key Design Decisions

Three architectural decisions deserve explicit mention because they have a direct impact on correctness and maintainability:

Separation of recognition and feedback. The binary recognition decision is always taken by `XRHandShape.CheckConditions()` or `XRHandPose.CheckConditions()`, which evaluate the joint data provided by the Unity XR Hands API. The `HandPoseAnalyzer` and constraint profiles are used exclusively by `FeedbackSystem` to generate explanatory messages, they do not influence the recognition outcome. This ensures that the thresholds used for feedback can be tuned for pedagogical purposes without affecting recognition accuracy.

IPoseAdapter interface. Rather than hard-coding a dependency on `MultiGestureRecognizer` or `GestureRecognizer`, `DynamicGestureRecognizer` communicates with the static pose detector through the `IPoseAdapter` interface. This allows the same dynamic recogniser to be used in both Scene 3 (single active sign, `SingleGestureAdapter`) and Scene 4 (all signs active, `StaticPoseAdapter`) without code changes.

ScriptableObject data model. All gesture definitions, constraint profiles, and content data are ScriptableObjects serialised as Unity assets. This means that adding a new sign, adjusting a threshold, or reorganising categories requires no code changes, only asset edits in the Inspector. It also enables the `DynamicGestureRecognizer` to filter and restore gesture definitions at runtime (`FilterGesturesBy-`

`Names` / `RestoreAllGestures`), which is used by `SelfAssessmentController` to scope recognition to the current category.

Appendix D: Adding New Signs to ASL LearnVR

This appendix documents the concrete steps required to extend the sign vocabulary of ASL LearnVR. Because all gesture definitions are encoded as Unity ScriptableObject assets, with no hard-coded gesture logic in the recognition engine. Adding a new sign does not require modifying any C# source file. The process reduces to creating and wiring a small set of assets through the Unity Inspector.

D.1 Adding a Static Sign

A static sign is one whose recognition depends solely on the instantaneous hand configuration (`requiresMovement = false`).

1. **Create the hand shape asset.** In the Unity project, use *Assets* → *Create* → *XR* → *Hand Interaction* → *Hand Shape* (or *Hand Pose*) to generate an `XRHandShape` or `XRHandPose` asset. Configure each finger's curl, spread, and orientation thresholds using the built-in sliders, optionally recording a live sample with the Quest 3 via the XR Hands Hand Capture tool.
2. **Create a `SignData` asset.** Use *Assets* → *Create* → *ASL Learn VR* → *Sign Data*. Fill in the following fields:
   - `signName`: unique identifier used throughout the system (e.g. `"Á"` or `"10"`).
   - `handShapeOrPose`: drag the asset created in step 1.
   - `requiresMovement`: leave `false`.
   - `minimumHoldTime`: duration in seconds the pose must be held for confirmation (default: 0.3 s).
3. **Register in a `CategoryData` asset.** Open the target category asset (e.g. `Category_Alphabet`) and add the new `SignData` to its `signs` list. The sign becomes immediately available in both the Learning Module (Scene 3) and Self-Assessment mode (Scene 4) with no further changes.
4. **(Optional) Create a `FingerConstraintProfile`.** Use *Assets* → *Create* → *ASL Learn VR* → *Finger Constraint Profile* to define per-finger curl ranges and spread constraints for the pedagogical feedback layer (see Appendix B). Set `signName` to match the `SignData` exactly. The `HandPoseAnalyzer` loads all profiles automatically on startup via `AlphabetConstraintProfiles` and `DigitConstraintProfiles`. For signs outside those categories, call `HandPoseAnalyzer.RegisterProfile()` at runtime or add the asset directly to the `constraintProfiles` list in the Inspector. This step is optional, as the

recognition pipeline operates independently of the constraint profile, which is used only to generate corrective feedback messages.

D.2 Adding a Dynamic Sign

A dynamic sign requires both a hand pose and a movement trajectory (`requiresMovement = true`). Steps 1–3 from Section D.1 apply, with the following additions.

2. **Create the `SignData` asset** as before, but set `requiresMovement = true`. This flag tells `LearningController` to activate `DynamicGestureRecognizer` instead of `GestureRecognizer` when this sign is loaded, and tells `MultiGestureRecognizer` not to confirm the sign on hold time alone in Scene 4.
4. **Create a `DynamicGestureDefinition` asset.** Use *Assets* → *Create* → *ASL* → *Dynamic Gesture Definition*. Table 6 lists the fields that must be configured.
5. **Register the definition in the scene.** Open the scene (`03_LearningModule` and/or `04_SelfAssessmentMode`), select the `DynamicGestureRecognizer` component, and add the new asset to its `gestureDefinitions` list. No code change is required. Alternatively, call `DynamicGestureRecognizer.AddGestureDefinition()` at runtime.

D.3 Adding a New Category or Level

1. **Create a `CategoryData` asset.** Use *Assets* → *Create* → *ASL Learn VR* → *Category Data*. Set `categoryName` and populate the `signs` list with the relevant `SignData` assets. The category becomes selectable in the application as soon as it is added to a `LevelData` asset.
2. **Register in a `LevelData` asset.** Open the target level asset (`Level_Basic`, `Level_Intermediate`, or `Level_Advanced`) and add the new `CategoryData` to its `categories` list. If a fourth level is needed, create a new `LevelData` asset via *Assets* → *Create* → *ASL Learn VR* → *Level Data* and register it in `LevelSelectionController`.

D.4 Summary

Table 7 summarises the assets involved and the code changes required for each extension scenario.

Table 6: Key fields of `DynamicGestureDefinition` and their meaning

Field	**Default**	**Description**
`gestureName`	—	Must match `SignData.signName` exactly.
`poseSequence`	empty	List of `StaticPoseRequirement` entries. Each entry specifies a `poseName`, a `PoseTimingRequirement` (`Start`, `During`, `End`, or `Any`), and an optional `poseData` reference for compound gestures (e.g. *White*: 5→S).
`primaryDirection`	forward	Expected movement direction in local space (normalised).
`directionTolerance`	45°	Angular tolerance around `primaryDirection`. Minimum 40° recommended for Quest 3.
`minSpeed`	0.12 m/s	Minimum hand speed. Values below 0.12 m/s may be undetectable for slow users.
`minDistance`	0.08 m	Minimum total trajectory length.
`minDuration`	0.4 s	Minimum gesture duration before success can be declared.
`maxDuration`	3.0 s	Timeout after which the gesture is failed.
`requiresDirectionChange`	false	Enable for zigzag or waving gestures.
`requiredDirectionChanges`	0	Number of direction changes required (threshold: 30°).
`requiresRotation`	false	Enable for wrist-twist gestures (e.g. *Blue*, *Purple*).
`minRotationAngle`	30°	Minimum cumulative wrist rotation. Keep ≤ 45 on Quest 3.
`requiresCircularMotion`	false	Enable for circular gestures (e.g. *Please*).
`minCircularityScore`	0.6	Minimum circularity score (0 = straight line, 1 = perfect circle).
`requiresSpatialZone`	false	Enable if the gesture must be performed in a specific body region.

Table 7: Assets required per extension scenario. No C# source file needs to be modified in any of the cases below.

Scenario	**Assets to create and/or modify**
New static sign	Create `XRHandShape` + `SignData`; add `SignData` to `CategoryData`.
New static sign with per-finger feedback	Above, plus create `FingerConstraintProfile` with matching `signName`.
New dynamic sign	Create `XRHandShape` + `SignData` (`requiresMovement = true`) + `DynamicGestureDefinition`; add `SignData` to `CategoryData`; register `DynamicGestureDefinition` in the `DynamicGestureRecognizer` Inspector list.
New category	Create `CategoryData`; add it to the `categories` list of the target `LevelData`.
New level	Create `LevelData`; register it in `LevelSelectionController`. A UI button may need to be added, but the data layer requires no code changes.

Appendix E: Dynamic Gesture Profiles

This appendix provides the complete parameter specification for all 38 dynamic gesture definitions implemented in ASL LearnVR, extracted directly from the `DynamicGestureDefinition` ScriptableObject assets. Each definition is evaluated at runtime by `DynamicGestureRecognizer` via `MovementTracker` (see Appendix C, Section C.4).

E.1 Parameter Reference

Table 8 describes each configurable parameter. For the complete field listing including default values, see Appendix D, Table D.1.

Table 8: Column descriptions for Tables 10–13

Column	**Meaning**
Pose sequence	Static pose requirements. *Start* = `PoseTimingRequirement.Start`; *Dur* = `During`; *End* = `End`. Compound gestures (e.g. Bye: 5→S) have both Start and End entries with direct `poseData` references.
Dir	Primary movement direction in local space (+X = right, −X = left, +Y = up, −Y = down, +Z = forward, −Z = toward user). Indicates no directional constraint.
Tol	`directionTolerance` in degrees.
MinSpd	`minSpeed` in m/s.
MinDist	`minDistance` in m.
$t_{\min}$ / $t_{\max}$	`minDuration` / `maxDuration` in seconds.
DirChg	`requiresDirectionChange` with `requiredDirectionChanges` count (threshold: 30°).
Rot	`requiresRotation` with `minRotationAngle` in degrees.
Circ	`requiresCircularMotion` with `minCircularityScore`.

E.2 Alphabet

Table 9: Dynamic gesture profiles — Alphabet (J, Z)

Sign	Pose sequence	Dir	Tol	MinSpd	MinDist	$t_{\min}$	$t_{\max}$	DirChg	Rot	Circ
J	Start: J	−Y	70°	0.08	0.06	0.5	2.0	—	70°	—
Z	Start: Z, Dur: Z	+X	89°	0.05	0.04	0.3	3.0	2	—	—

E.3 Basic Communication

Table 10: Dynamic gesture profiles — Basic Communication

Sign	Pose sequence	Dir	Tol	MinSpd	MinDist	$t_{\min}$	$t_{\max}$	DirChg	Rot	Circ
Bad	Start+Dur: Bad	−Y	95°	0.02	0.02	0.2	3.0	—	—	—
Bye	Start: 5, End: S	—	90°	0.05	0.03	0.3	3.0	—	—	—
Good	Start+Dur: Good	+Y	95°	0.02	0.02	0.2	3.0	—	—	—
Hello	Start+Dur: Hello	+X	85°	0.05	0.06	0.3	3.0	1	—	—
No	Start+Dur: No	+X	75°	0.05	0.06	0.3	3.0	1	—	—
Please	Start+Dur: Please	+X	89°	0.05	0.06	0.5	4.0	—	—	0.25
Thank You	—	+Z	70°	0.05	0.08	0.3	3.0	—	—	—
Yes	Start+Dur: Yes	+Y	70°	0.05	0.05	0.3	3.0	1	—	—

E.4 Colors

Table 11: Dynamic gesture profiles — Colors

Sign	Pose sequence	Dir	Tol	MinSpd	MinDist	$t_{\min}$	$t_{\max}$	DirChg	Rot	Circ
Black	Start+Dur: Black	+X	130°	0.015	0.03	0.2	3.5	—	—	—
Blue	Start: Blue	—	90°	0.06	0.02	0.4	3.0	1	20°	—
Brown	Start: B	−Y	45°	0.08	0.05	0.4	2.5	—	—	—
Gray	Start: Gray	+X	55°	0.12	0.08	0.5	3.0	2	—	—
Green	Start: Green	—	90°	0.08	0.03	0.4	3.0	1	25°	—
Orange	Start: O, End: S	—	90°	0.08	0.03	0.4	3.0	—	—	—
Pink	Start: Pink	−Y	45°	0.10	0.04	0.4	2.5	1	—	—
Purple	Start: Purple	—	90°	0.08	0.03	0.4	3.0	1	25°	—
Red	Start: Red	−Y	45°	0.10	0.04	0.4	2.5	1	—	—
White	Start: 5, End: White	—	90°	0.08	0.03	0.4	3.0	—	—	—
Yellow	Start: Yellow	—	90°	0.08	0.03	0.4	3.0	1	25°	—

E.5 Days of the Week

Table 12: Dynamic gesture profiles — Days of the Week

Sign	Pose sequence	Dir	Tol	MinSpd	MinDist	$t_{\min}$	$t_{\max}$	DirChg	Rot	Circ
Monday	Start: Monday	—	90°	0.08	0.03	0.4	3.0	—	30°	0.35
Tuesday	Start: Tuesday	—	90°	0.08	0.03	0.4	3.0	—	30°	0.35
Wednesday	Start: Wednesday	—	90°	0.08	0.03	0.4	3.0	—	30°	0.35
Thursday	Start: T, End: H	—	90°	0.08	0.03	0.4	3.0	—	—	—
Friday	Start: Friday	—	90°	0.08	0.03	0.4	3.0	—	30°	0.35
Saturday	Start: Saturday	—	90°	0.08	0.03	0.4	3.0	—	30°	0.35
Sunday	Start: Sunday	—	90°	0.10	0.05	0.5	3.0	—	—	0.40

E.6 Verbs

Table 13: Dynamic gesture profiles — Verbs

Sign	Pose sequence	Dir	Tol	MinSpd	MinDist	$t_{\min}$	$t_{\max}$	DirChg	Rot	Circ
Draw	Start: I	+X	80°	0.03	0.03	0.4	3.0	1	—	—
Drink	Start: C	−Z	80°	0.05	0.02	0.3	3.0	1	—	—
Eat	Start: White	−Z	80°	0.05	0.02	0.3	3.0	1	—	—
Get	Start: Get	−Z	60°	0.08	0.06	0.4	2.5	—	—	—
Hurt	Start: Hurt	—	90°	0.05	0.02	0.4	2.5	1	15°	—
Play	Start: Play	—	90°	0.05	0.02	0.4	2.5	—	15°	—
Read	Start: V	−Y	80°	0.03	0.02	0.4	3.0	1	—	—
Sleep	Start: 5, End: White	—	180°	0.00	0.00	0.2	3.0	—	—	—
Tap	Start: Tap	−Y	60°	0.05	0.02	0.4	2.5	2	—	—
Write	Start: F	+X	80°	0.03	0.03	0.4	3.0	1	—	—

E.7 Structural Patterns

Several cross-category patterns emerge from the data above.

Compound gestures (Bye, Orange, Thursday, White, Sleep) are distinguished by an explicit End pose constraint validated via direct `XRHandShape.CheckConditions()` against a cached `XRHandJointsUpdatedEventArgs` (see Appendix E, Section E.4). Their `minDistance` and `minSpeed` are low because recognition depends on the pose transition rather than on trajectory amplitude.

Wrist-twist gestures (Blue, Green, Purple, Yellow, Hurt, Play and all weekdays except Thursday and Sunday) use `requiresRotation` with `minRotationAngle` ≤ 30 to capture the characteristic lateral wrist movement without the strict directional constraint that would penalise slight execution variations.

Circular gestures (Please, Monday–Wednesday, Friday–Saturday, Sunday) use `requiresCircularMotion` with a deliberately low `minCircularityScore` (0.25–0.40) to account for the natural elliptical path produced by wrist rotation on the Quest 3.

Direction-change gestures (Hello, No, Yes, Gray, Pink, Red, Z, Draw, Drink, Eat, Read, Tap, Write, Blue, Green, Purple, Yellow) require between one and two direction changes above the 30° threshold. Most are waving or back-and-forth movements; Tap and Z require two changes to enforce the full zig-zag trajectory.

Sleep is a special case with `minSpeed = 0`, `minDistance = 0`, and `directionTolerance = 180°`, which effectively disables all trajectory constraints. Recognition relies entirely on the pose transition from handshape 5 to handshape White as the fingers close toward the face.

Appendix F: User Study Protocol

This appendix contains the complete materials used in the user study described in Section 6.5, including the session information sheet, pre-session questionnaire, observer task guide, System Usability Scale, and qualitative questions. One copy of this protocol was used per participant.

F.1 Session Information

Participant ID: _____

Date: _____

Session start time: _____

Observer: _____

Estimated duration: ∼45 minutes

F.2 Pre-Session Questionnaire

F.2.1 Demographic Data

Age: _____

Gender: _____

Occupation: _____

F.2.2 Prior Experience

Have you ever used a VR headset before? / ¿Has usado alguna vez un casco de realidad virtual?

☐ Yes / Sí ☐ No

If yes, how often? / En caso afirmativo, ¿con qué frecuencia?

☐ Once or twice / Una o dos veces ☐ Occasionally / Ocasionalmente

☐ Regularly / Regularmente

Do you have any knowledge of American Sign Language (ASL)? / ¿Tienes algún conocimiento de la Lengua de Signos Americana?

☐ None / Ninguno ☐ Basic / Básico
☐ Intermediate / Intermedio ☐ Advanced / Avanzado

Do you have any hearing impairment? / ¿Tienes alguna discapacidad auditiva?

☐ Yes / Sí ☐ No ☐ Prefer not to say / Prefiero no decir

F.3 Task Session

F.3.1 Task Overview

The observer guides the participant through the following tasks. / El observador guía al participante a través de las siguientes tareas.

#	**Task / Tarea**	**Time**	**Done**
1	Put on the headset and explore the main screen freely. *Ponerse el casco y explorar la pantalla principal libremente.*	∼2 min	☐
2	Navigate to Basic → Alphabet. Practice A, B, C, G, Q, Y. *Navegar a Basic → Alphabet. Practicar A, B, C, G, Q, Y.*	∼10 min	☐
3	Practice dynamic sign Z (alphabet). *Practicar el signo dinámico Z (alfabeto).*	∼5 min	☐
4	Navigate to Basic Communication (Hello, Bye, Thank You, Please). *Navegar a Basic Communication (Hello, Bye, Thank You, Please).*	∼8 min	☐
5	Practice one sequential sign (month abbreviation: DEC). *Practicar un signo secuencial (abreviatura de mes: DEC).*	∼3 min	☐
6	Access Self-Assessment and attempt all practiced signs. *Acceder a Autoevaluación e intentar todos los signos practicados.*	∼7 min	☐

F.3.2 Sign Recognition Data

Record the number of attempts and time in seconds until first correct recognition. / Registrar el número de intentos y el tiempo en segundos hasta el primer reconocimiento correcto.

Sign / Signo	Attempts	Time (s)	Observer notes / Notas
A *(static)*			
B *(static)*			
C *(static)*			
G *(static, orient.)*			
Q *(static, orient.)*			
Y *(static)*			
Z *(dynamic)*			
Hello *(dynamic)*			
Bye *(dynamic)*			
Thank You *(dynamic)*			
Please *(dynamic)*			
DEC *(sequential)*			

F.3.3 General Observations During Session

Notable behaviors, difficulties, or spontaneous comments. / Comportamientos relevantes, dificultades o comentarios espontáneos.

F.4 System Usability Scale (SUS)

Please rate each statement from 1 (totally disagree) to 5 (totally agree). / Por favor, puntúa cada afirmación del 1 (totalmente en desacuerdo) al 5 (totalmente de acuerdo).

Statement / Afirmación	**1**	**2**	**3**	**4**	**5**
1. I think that I would like to use this system frequently. *Creo que me gustaría usar este sistema con frecuencia.*	◯ 1	◯ 2	◯ 3	◯ 4	◯ 5
2. I found the system unnecessarily complex. *Encontré el sistema innecesariamente complejo.*	◯ 1	◯ 2	◯ 3	◯ 4	◯ 5
3. I thought the system was easy to use. *Pensé que el sistema era fácil de usar.*	◯ 1	◯ 2	◯ 3	◯ 4	◯ 5
4. I think that I would need the support of a technical person to use this system. *Creo que necesitaría el apoyo de una persona técnica para poder usar este sistema.*	◯ 1	◯ 2	◯ 3	◯ 4	◯ 5
5. I found the various functions in this system were well integrated. *Encontré que las distintas funciones del sistema estaban bien integradas.*	◯ 1	◯ 2	◯ 3	◯ 4	◯ 5
6. I thought there was too much inconsistency in this system. *Pensé que había demasiada inconsistencia en este sistema.*	◯ 1	◯ 2	◯ 3	◯ 4	◯ 5
7. I would imagine that most people would learn to use this system very quickly. *Imagino que la mayoría de las personas aprenderían a usar este sistema muy rápidamente.*	◯ 1	◯ 2	◯ 3	◯ 4	◯ 5
8. I found the system very cumbersome to use. *Encontré el sistema muy incómodo de usar.*	◯ 1	◯ 2	◯ 3	◯ 4	◯ 5
9. I felt very confident using the system. *Me sentí muy seguro/a usando el sistema.*	◯ 1	◯ 2	◯ 3	◯ 4	◯ 5
10. I needed to learn a lot of things before I could get going with this system. *Necesité aprender muchas cosas antes de poder empezar a usar este sistema.*	◯ 1	◯ 2	◯ 3	◯ 4	◯ 5

Overall, how would you rate your experience with ASL LearnVR? / ¿Cómo valorarías tu experiencia general con ASL LearnVR?

◯ 1 *(Very poor)* ◯ 2 ◯ 3 ◯ 4 ◯ 5 *(Excellent)*

F.5 Qualitative Questions

Open questions — record participant's answers as precisely as possible. / Preguntas abiertas — registrar las respuestas del participante con la mayor precisión posible.

Q1. Was the finger-level visual feedback easy to understand and act on? / ¿Fue fácil entender y seguir el feedback visual por dedo?

Q2. Did the guide hands help you understand how to perform the sign correctly? / ¿Las manos guía te ayudaron a entender cómo realizar el signo correctamente?

Q3. What aspect of the system did you find most difficult to use? / ¿Qué aspecto del sistema te resultó más difícil de usar?

Q4. Would you use this application to learn sign language independently? / ¿Usarías esta aplicación para aprender lengua de signos de forma independiente?

Additional comments / Comentarios adicionales: