



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Implementación de técnicas de visión artificial para la
detección de objetos en contextos de realidad mixta*

Grado en Ingeniería Mecánica

ALUMNO: Daniel Redondo Contreras

DIRECTORES: Francisco Troncoso Pastoriza
Xavier Núñez Nieto

CURSO ACADÉMICO: 2025-2026

Universida_{de}Vigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Implementación de técnicas de visión artificial para la
detección de objetos en contextos de realidad mixta*

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

UniversidadeVigo

RESUMEN

Este Trabajo Fin de Grado aborda la estimación de pose 6D de objetos físicos en entornos de realidad mixta, utilizando exclusivamente las cámaras pasivas del visor Meta Quest 3. El objetivo es implementar y evaluar un *pipeline* de visión artificial capaz de determinar la posición y orientación de un controlador Touch Plus a partir de pares de imágenes estéreo, sin recurrir a sensores adicionales. El sistema desarrollado se organiza en una secuencia de etapas encadenadas que comprenden la adquisición de datos mediante la *Passthrough Camera API* del dispositivo, la reconstrucción estereoscópica 3D (rectificación, estimación de disparidad y generación de nubes de puntos), la segmentación geométrica del objeto de interés y la estimación de pose mediante *Point Pair Features* (PPF) con refinamiento *Iterative Closest Point* (ICP). Este proceso se complementa con una evaluación cuantitativa basada en la comparación de las poses estimadas con la referencia proporcionada por el sistema de seguimiento nativo del visor. Los resultados obtenidos ponen de manifiesto la viabilidad del enfoque propuesto, evidenciando un comportamiento robusto y preciso en condiciones experimentales representativas, y respaldan el uso de técnicas estereoscópicas para la estimación de pose en contextos de realidad mixta, estableciendo una base sólida para futuras líneas de investigación.

PALABRAS CLAVE

Visión artificial, realidad mixta, reconstrucción estéreo, segmentación geométrica, estimación de pose

AGRADECIMIENTOS

En primer lugar, agradezco a mis tutores, Don Francisco Troncoso Pastoriza y Don Xavier Núñez Nieto, por su excelente labor de guía y asesoramiento a lo largo de todo el proyecto, propiciando un ambiente de trabajo basado en la confianza y el buen hacer.

A mis compañeros y amigos más cercanos, por haber sido durante estos meses una fuente constante de apoyo y motivación.

Finalmente, a mi familia, por su ayuda incondicional, por brindarme apoyo emocional ante las dificultades encontradas en el camino y por animarme siempre a dar lo mejor de mí.

CONTENIDO

Contenido	1
Índice de Figuras	4
Índice de Tablas.....	6
1 Introducción y objetivos	7
1.1 Contexto	7
1.2 Motivación	7
1.3 Objetivos	8
1.4 Organización de la memoria	8
2 Estado del arte	11
2.1 Introducción	11
2.1.1 Conceptos de Visión Artificial y Realidad Mixta.....	11
2.1.2 Motivación técnica.....	12
2.1.3 Alcance del Estado del Arte	13
2.2 Marco tecnológico de la Realidad Mixta	13
2.2.1 Línea evolutiva en arquitectura de sensores	13
2.2.2 Dispositivos representativos actuales	14
2.2.3 Motores de desarrollo	15
2.3 Adquisición de datos	16
2.3.1 Modalidades de captura	16
2.3.2 Calibración y sincronización	17
2.3.3 Sistemas de referencia	18
2.4 Reconstrucción estereoscópica	18
2.4.1 Geometría epipolar	19
2.4.2 Rectificación estereoscópica.....	20
2.4.3 Disparidad.....	20
2.4.4 Profundidad.....	22
2.4.5 Nube de puntos 3D	22
2.5 Estimación de pose.....	22
2.5.1 Segmentación geométrica	23
2.5.2 PPF.....	23
2.5.3 ICP	24
2.5.4 PPF+ICP	25
2.6 Limitaciones en MR.....	26

2.6.1 Baseline. Degradación en profundidad	27
2.6.2 Disparidad. Oclusiones y baja textura	27
2.6.3 Profundidad. Sensibilidad al error	27
2.6.4 Calibración. Intrínsecos y extrínsecos.	28
2.6.5 Restricciones de tiempo real	28
2.6.6 Deriva de pose	28
2.7 Métricas evaluables.....	28
2.7.1 Concepto y criterios de selección	28
2.7.2 Métricas de profundidad	29
2.7.3 Métricas de pose 6D	29
2.7.4 Métricas de detección	30
2.7.5 Estabilidad temporal	30
2.7.6 Coste computacional.....	30
3 Desarrollo	31
3.1 Contextualización.....	31
3.1.1 Herramientas y entorno de desarrollo	31
3.1.2 Estructura completa del sistema	35
3.2 Fase 1: Adquisición de datos	40
3.2.1 Meta Quest 3 y Passthrough Camera API	40
3.2.2 Entorno Unity y configuración del proyecto	42
3.2.3 Escena, jerarquía y sistema de referencia	44
3.2.4 Captura y exportación de fotogramas	46
3.3 Fase 2: Exportación, carga y validación de datos	50
3.3.1 Transferencia de datos desde el dispositivo al PC	50
3.3.2 Estructura del repositorio de postprocesado	51
3.3.3 Carga de datos y metadatos	57
3.4 Fase 3: Reconstrucción estereoscópica 3D	58
3.4.1 Rectificación estéreo.....	58
3.4.2 Cálculo de disparidad	60
3.4.3 Estimación de profundidad	62
3.4.4 Generación de nube de puntos 3D	62
3.5 Fase 4: Estimación de pose	64
3.5.1 Preparación del modelo CAD del controlador.....	64
3.5.2 Entrenamiento del descriptor PPF	66
3.5.3 Segmentación del objeto.....	66

3.5.4 Matching PPF+ICP	67
3.5.5 Evaluación por fotograma y gate pre-matching.....	71
3.6 Pipeline de ejecución	72
3.6.1 Orquestador del pipeline.....	73
3.6.2 Parámetros configurables.....	74
3.6.3 Entrada de usuario y ejecución práctica	74
4 Resultados	77
4.1 Fase 5: Evaluación de métricas	78
4.1.1 Evaluación por fotograma.....	79
4.1.2 Guía de métricas del sistema	80
4.1.3 Evaluación de sesión y trayectoria	81
4.1.4 Reporting y visualización interactiva	82
4.2 Resultados	82
4.2.1 Descripción de la sesión analizada	82
4.2.2 Resumen cuantitativo de la sesión	83
4.2.3 Análisis de la trayectoria estimada	85
4.2.4 Evolución temporal de las métricas	86
4.2.5 Datos por fotograma	88
4.2.6 Discusión de los resultados.....	90
4.2.7 Valoración global.....	91
5 Conclusiones y líneas futuras	93
5.1 Conclusiones	93
5.2 Valoración personal	94
5.3 Líneas Futuras	94
5.3.1 Prioridad	94
5.3.2 Mejora.....	95
5.3.3 Diversificación.....	95
6 Bibliografía.....	97
Anexo I: Implicaciones Sociales, y/o Económicas, y/o Ambientales	101
Anexo II: Reflexiones Éticas y Sociales	103
Anexo III: GitHub – Repositorios	105
Anexo IV: Declaración responsable sobre el uso de inteligencia artificial generativa	107

ÍNDICE DE FIGURAS

Figura 2-1: Diagrama del entorno RM [2]	12
Figura 2-2: Evolución cronológica de los dispositivos XR (generada con IAG).....	14
Figura 2-3: Dispositivo Meta Quest3 [8].....	15
Figura 2-4: Plataformas software utilizadas para aplicaciones XR [4]	15
Figura 2-5: Diagrama de flujo para la elección del motor de desarrollo [12]	16
Figura 2-6: Diagrama sobre geometría epipolar [20]	19
Figura 2-7: Diagrama para comprender la disparidad [24]	20
Figura 2-8: Apreciación visual del filtro WLS [26]	21
Figura 2-9: Point Pair Feature (PPF) [28]	24
Figura 2-10: Variantes de ICP [29]	25
Figura 2-11: Comparación de convergencia: ICP punto a punto e ICP punto a plano [29].....	25
Figura 2-12: Algoritmo <i>Surface Matching</i> [31]	26
Figura 3-1: Dispositivo Meta Quest 3 [8].....	32
Figura 3-2: Interfaz de Unity	32
Figura 3-3: Interfaz de VSCode	33
Figura 3-4: Interfaz de CloudCompare	34
Figura 3-5: Interfaz de Blender	35
Figura 3-6: Diagrama de flujo. Bloques funcionales del sistema (Autoría propia).....	35
Figura 3-7: Diagrama de flujo. Estructura completa del sistema (Autoría propia)	36
Figura 3-8: Fase 1 (Autoría propia).....	37
Figura 3-9: Fase 2 (Autoría propia).....	37
Figura 3-10: Fase 3 (Autoría propia).....	38
Figura 3-11: Fase 4 (Autoría propia).....	39
Figura 3-12: Fase 5 (Autoría propia).....	39
Figura 3-13: Interfaz de la aplicación Meta Horizon Link.....	41
Figura 3-14: Interfaz de la aplicación MQDH	41
Figura 3-15: Versión del editor instalada en Unity Hub	42
Figura 3-16: Proyecto Unity- <i>CameraPassthroughApiSamples</i>	42
Figura 3-17: Acceso a <i>XR Plug-in-Management</i> desde “Project Settings”	43
Figura 3-18: Herramienta <i>Project Setup Tool</i>	43
Figura 3-19: Escena <i>CaptureAndExport</i> (Autoría propia)	44
Figura 3-20: <i>OVRCameraRig</i>	45

Figura 3-21: <i>PassthroughCameraAccess</i>	45
Figura 3-22: Sistema de referencia <i>TrackingSpace</i> (Generado con IAG).....	46
Figura 3-23: <i>Scripts</i> asociados a la escena (Autoría propia)	46
Figura 3-24: Elemento CAPTURE (Autoría propia)	47
Figura 3-25: Elemento SYSTEM (Autoría propia)	48
Figura 3-26: Elemento UI (Autoría propia)	48
Figura 3-27: Visualización del <i>canvas</i> (Autoría propia)	49
Figura 3-28: Archivos obtenidos en un fotograma cualquiera (Autoría propia).....	49
Figura 3-29: Estructura de primer nivel del repositorio <i>PythonProcessing</i> (Autoría propia).....	51
Figura 3-30: Estructura de la carpeta <i>data</i> (Autoría propia)	52
Figura 3-31: Estructura de la carpeta “raw/” (Autoría propia).....	52
Figura 3-32: Estructura de la carpeta “ <i>processed/</i> ” (Autoría propia).....	54
Figura 3-33: Mapa de código del repositorio (Autoría propia)	56
Figura 3-34: Imágenes estereoscópicas rectificadas (Autoría propia)	59
Figura 3-35: Mapa de disparidad con y sin filtro WLS (Autoría propia).....	61
Figura 3-36: Mapa de profundidad (Autoría propia).....	62
Figura 3-37: Nube de puntos 3D de la escena completa (Autoría propia)	63
Figura 3-38: Modelo CAD de los controladores Touch Plus en Blender (Autoría Propia)	64
Figura 3-39: Detalle del controlador derecho limpio en Blender (Autoría propia).....	65
Figura 3-40: Modelo CAD como malla y nube de puntos en CloudCompare (Autoría Propia).....	65
Figura 3-41: Objeto segmentado aislado en CloudCompare (Autoría Propia)	68
Figura 3-42: Modelo CAD alineado sobre la nube segmentada (Autoría propia)	71
Figura 3-43: Estimación de pose de otro fotograma de la misma sesión (Autoría propia).....	71
Figura 3-44: Flujo de procesamiento del <i>pipeline</i> (Autoría propia).....	73
Figura 3-45: Marcos de referencia y cadena SE(3) (Autoría propia).....	73
Figura 3-46: Zona de configuración de usuario de “run_pipeline.ps1” (Autoría Propia)	75
Figura 4-1: Generación del <i>dashboard</i> interactivo en “run_pipeline.ps1” (Autoría Propia)	77
Figura 4-2: Código QR para acceder al <i>dashboard</i> interactivo.....	78
Figura 4-3: Flujo de archivos por etapa del <i>pipeline</i> (Autoría propia).....	79
Figura 4-4: Resumen cuantitativo de la sesión (Autoría propia).....	83
Figura 4-5: Trayectoria 3D estimada del controlador (Autoría propia)	85
Figura 4-6: Componentes de traslación por fotograma (Autoría propia)	86
Figura 4-7: Evolución del <i>fitness</i> por fotograma (Autoría propia).....	87
Figura 4-8: Evolución del RMSE por fotograma (Autoría propia)	87
Figura 4-9: Distancia al centroide por fotograma (Autoría propia)	88

ÍNDICE DE TABLAS

Tabla 2-1: Comparación de dispositivos MR [7]	14
Tabla 2-2: Modalidades para adquisición de datos	17
Tabla 2-3: Métricas del <i>benchmark BOP</i>	29
Tabla 2-4: Fenómenos temporales relevantes	30
Tabla 3-1: Contenido de la carpeta “processed/” por fotograma	53
Tabla 3-2: Módulos de “src/core/”	55
Tabla 3-3: <i>Scripts</i> de “src/pipeline/”	55
Tabla 3-4: Elementos auxiliares del repositorio	56
Tabla 3-5: Campos principales del archivo “metadata.json”	57
Tabla 3-6: Parámetros del algoritmo SGBM.....	61
Tabla 3-7: Parámetros de filtrado de profundidad.....	62
Tabla 3-8: Etapas de la segmentación y parámetros clave	67
Tabla 3-9: Parámetros del <i>matching</i> PPF+ICP.....	68
Tabla 3-10: Clasificación de fotogramas.....	72
Tabla 3-11: Parámetros configurables del <i>pipeline</i>	74
Tabla 4-1: Guía de métricas del sistema	80
Tabla 4-2: Resumen estadístico de la sesión analizada	84
Tabla 4-3: Trayectoria detallada. Subconjunto representativo de 10 fotogramas	89
Tabla 5-1: Propuesta de TFGs para cursos académicos futuros.....	96

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Contexto

La realidad mixta (MR) representa una de las fronteras más activas en la convergencia entre percepción computacional e interacción humana. En un entorno de realidad mixta, los objetos físicos y los elementos virtuales coexisten en el mismo espacio, de modo que el usuario puede interactuar simultáneamente con ambos mundos de forma coherente. Para que esta fusión resulte creíble, el sistema debe conocer en todo momento dónde se encuentran los objetos reales y cuál es su orientación: es decir, necesita estimar su pose 6D (tres grados de libertad en traslación y tres en rotación).

La visión artificial (VA) proporciona las herramientas para resolver este problema. A través de técnicas de calibración de cámaras, reconstrucción 3D a partir de imágenes estéreo, segmentación geométrica y algoritmos de registro como *Point Pair Features* (PPF) e *Iterative Closest Point* (ICP), es posible determinar la posición y orientación de un objeto en el espacio tridimensional utilizando únicamente cámaras. Cuando estas técnicas se integran en un visor de realidad mixta, el resultado es un sistema capaz de detectar objetos reales y superponer contenido virtual sobre ellos con precisión espacial.

En los últimos años, el acceso a dispositivos de realidad mixta de prestaciones avanzadas se ha democratizado notablemente. El lanzamiento de Meta Quest 3 en 2023 puso a disposición del mercado un visor autónomo con cámaras estéreo RGB, seguimiento nativo de controladores y una API que expone los datos de las cámaras al desarrollador. Esta combinación convierte al dispositivo en una plataforma idónea para investigar técnicas de visión artificial aplicadas a la estimación de pose, ya que reúne en un único hardware tanto el sistema de captura como la referencia (*ground truth*) necesaria para evaluar los resultados.

1.2 Motivación

La motivación de este proyecto nace de una necesidad concreta: explorar hasta qué punto es posible estimar la pose de un objeto físico utilizando únicamente las cámaras pasivas de un visor de consumo, sin recurrir a sensores adicionales.

Los sistemas de seguimiento nativos de los visores MR, como el de Meta Quest 3, alcanzan niveles de precisión muy elevados, pero están limitados a un conjunto cerrado de objetos predefinidos (los

propios controladores y las manos del usuario). Si se desea detectar y localizar objetos arbitrarios en la escena, es necesario desarrollar soluciones propias de visión artificial. Este proyecto aborda precisamente ese desafío: implementar desde cero un sistema completo que, a partir de pares de imágenes estéreo, sea capaz de reconstruir la escena en 3D, identificar el objeto de interés y estimar su pose con precisión medible.

Más allá del interés académico, este trabajo se enmarca en una línea de investigación con proyección a largo plazo. El Centro Universitario de la Defensa en la Escuela Naval Militar contempla la continuidad de este proyecto en sucesivos Trabajos Fin de Grado, con el objetivo último de desplegar un algoritmo robusto de detección de objetos en escenarios de simulación de combate mediante dispositivos de realidad mixta. El presente TFG constituye el primer eslabón de esa cadena: establecer la base técnica, implementar el *pipeline* completo y generar métricas cuantitativas que permitan evaluar la viabilidad del enfoque.

1.3 Objetivos

El objetivo general de este Trabajo Fin de Grado es implementar, evaluar y documentar un sistema de visión artificial capaz de estimar la pose 6D de un objeto físico en un entorno de realidad mixta, utilizando exclusivamente las cámaras estéreo del visor Meta Quest 3. Este objetivo general se descompone en los siguientes objetivos específicos:

- Diseñar e implementar una aplicación en el motor Unity que capture pares de imágenes estéreo desde el visor Meta Quest 3, junto con los metadatos geométricos asociados (parámetros intrínsecos, poses de cámaras y poses de los controladores como referencia).
- Desarrollar un *pipeline* de postprocesado en Python que ejecute de forma secuencial las etapas de rectificación estéreo, cálculo de disparidad, estimación de profundidad y generación de nubes de puntos 3D a partir de los datos capturados.
- Implementar un módulo de segmentación geométrica capaz de aislar el objeto de interés (controlador Touch Plus) de la nube de puntos de la escena.
- Implementar la estimación de pose del objeto mediante el algoritmo *Point Pair Features* (PPF) con refinamiento *Iterative Closest Point* (ICP), registrando un modelo CAD del controlador contra la nube segmentada.
- Definir un marco de evaluación cuantitativa que compare las poses estimadas con el *ground truth* proporcionado por el sistema de seguimiento nativo del visor, calculando errores de traslación, rotación y métricas de calidad del alineamiento.
- Analizar los resultados obtenidos, identificar las limitaciones del sistema y proponer líneas de mejora para futuros trabajos.

1.4 Organización de la memoria

La presente memoria se estructura en cinco capítulos, cada uno con un propósito específico dentro del relato del proyecto:

- **Capítulo 1. Introducción y objetivos.** Presenta el contexto del proyecto, la motivación que lo impulsa, los objetivos que persigue y la organización del documento.
- **Capítulo 2. Estado del arte.** Revisa los fundamentos teóricos de visión artificial y realidad mixta que sustentan el proyecto: reconstrucción estereoscópica, estimación de pose (PPF, ICP), limitaciones

conocidas en entornos MR y métricas de evaluación. Establece el marco conceptual sobre el que se apoya el desarrollo.

- **Capítulo 3. Desarrollo.** Describe en detalle las fases del sistema implementado: adquisición de datos con Meta Quest 3 y Unity, exportación y validación de datos, reconstrucción estereoscópica 3D, estimación de pose y *pipeline* de ejecución. Cada fase se explica con sus fundamentos algorítmicos, decisiones de diseño y resultados intermedios.

- **Capítulo 4. Evaluación y resultados.** Presenta el marco de evaluación de métricas (que atiende a la última fase del *pipeline*) y los resultados cuantitativos obtenidos en una sesión experimental real, incluyendo el análisis de la trayectoria estimada, la evolución temporal de las métricas y la discusión de fortalezas y limitaciones.

- **Capítulo 5. Conclusiones y líneas futuras.** Sintetiza las conclusiones técnicas del proyecto, valora el grado de cumplimiento de los objetivos y propone líneas de continuidad para futuros Trabajos Fin de Grado.

2 ESTADO DEL ARTE

2.1 Introducción

La convergencia entre la visión artificial (VA) y las tecnologías de realidad extendida (XR) redefine cómo los sistemas interactivos perciben, modelan e integran el entorno físico. En este contexto, la realidad mixta (MR) se define como el entorno donde elementos reales y virtuales coexisten e interactúan en tiempo real.

Para ello, los visores de MR incorporan sensores y técnicas avanzadas de procesamiento geométrico. Esta relación posibilita aplicaciones donde la detección de objetos, la reconstrucción 3D y la estimación de pose 6D dejan de ser procesos independientes y pasan a formar parte de un flujo continuo que enriquece la experiencia inmersiva.

Para abordar este panorama se revisarán los conceptos fundamentales de visión artificial y realidad mixta.

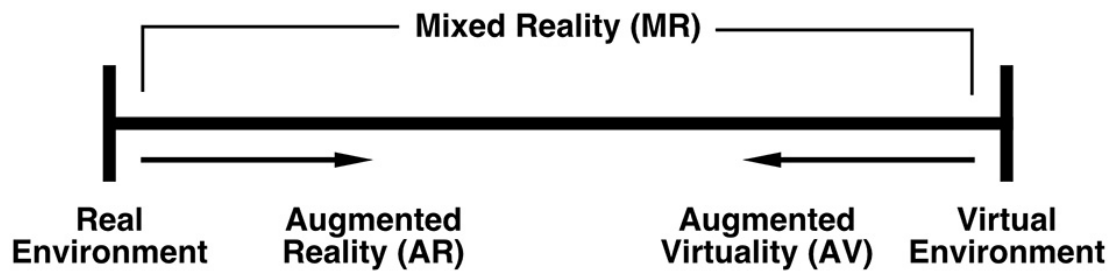
2.1.1 Conceptos de Visión Artificial y Realidad Mixta

La visión artificial comprende un conjunto de métodos computacionales para adquirir, procesar y analizar imágenes con el fin de extraer la estructura del entorno [1]. Tradicionalmente, este campo incluye tareas básicas como calibración de cámaras, reconstrucción 3D a partir de varias imágenes, segmentación geométrica y estimación de pose. Estos son los pilares de cualquier sistema de percepción en robótica, realidad aumentada (AR) o realidad mixta (MR) [1].

En el caso de la visión estéreo, obtener un mapa de disparidad y convertirlo en profundidades métricas provee la base geométrica para reconstruir modelos tridimensionales del entorno [2]. Sobre estas representaciones se aplican algoritmos de registro (estimación de pose) y seguimiento (*tracking*), formando un flujo de procesamiento encadenado que alimenta el sistema.

Por su parte, la realidad mixta (MR) se ubica entre el mundo completamente real y el completamente virtual (Véase la Figura 2-1). En ella, los elementos reales y sintéticos se fusionan explícitamente: el usuario interactúa simultáneamente con objetos físicos y virtuales de forma coherente en espacio y tiempo. En la práctica, esto implica combinar técnicas de AR (superponer información digital en la escena real) y VR (entornos sintéticos inmersivos), apoyándose en dispositivos que registran la posición

y orientación del usuario, así como la geometría del entorno circundante. El concepto más amplio de realidad extendida (XR) actúa como paraguas que integra AR, VR y MR.



Reality-Virtuality (RV) Continuum

Figura 2-1: Diagrama del entorno RM [2]

Todas estas modalidades requieren modelos 3D consistentes y un alineamiento robusto entre coordenadas físicas y virtuales. En MR, esta exigencia es especialmente crítica: los errores en la reconstrucción, la estimación de pose o la detección de objetos se traducen en inconsistencias perceptivas que degradan la sensación de presencia y la usabilidad del sistema [3].

2.1.2 Motivación técnica

Desde un punto de vista técnico, se busca evaluar en qué medida las técnicas clásicas de visión artificial pueden integrarse con los sistemas de seguimiento nativos en dispositivos *standalone* actuales de MR.

Actualmente existen dispositivos comerciales que alcanzan gran precisión de reconstrucción estereoscópica en entornos controlados, pero su rendimiento varía según la textura, iluminación, oclusiones y configuración de la escena. Esto plantea la cuestión de si algoritmos explícitos de visión, aplicados sobre el flujo de cámara disponible, podrían mejorar la precisión o servir como referencia rigurosa.

Las aplicaciones de MR con alta exigencia geométrica (por ejemplo, mantenimiento industrial o manipulación precisa de objetos virtuales sobre objetos reales [4]) requieren estimaciones de pose estables tanto del usuario como de los objetos de interés.

Métodos bien establecidos como *Iterative Closest Point* (ICP), *Point Pair Feature* (PPF) o variantes de registro basado en nubes de puntos siguen siendo referentes cuando se dispone de datos de profundidad fiables. Estos métodos permiten controlar explícitamente los compromisos entre precisión, robustez y coste computacional. Por tanto, evaluar dichos algoritmos en el contexto de visores MR independientes es relevante para entender sus límites prácticos en hardware con restricciones de potencia y memoria [5].

Dada la rápida expansión de la MR y la gran cantidad de trabajos existentes, es difícil justificar las decisiones de diseño en un proyecto aplicado sin un estado del arte estructurado que distinga entre resultados metodológicamente sólidos y nuevas propuestas.

2.1.3 Alcance del Estado del Arte

El estado del arte de esta memoria se circunscribe a técnicas de visión artificial clásicas aplicables a entornos de realidad mixta, prestando especial atención al *pipeline* (secuencia de etapas) que va desde la adquisición sensorial hasta la estimación de pose 6D.

En concreto, se abordan los siguientes bloques:

- Sensores y estrategias de adquisición en dispositivos MR autónomos (*standalone*).
- Fundamentos de geometría epipolar, rectificación estereoscópica, estimación de disparidad, conversión de disparidad a profundidad y construcción de nubes de puntos 3D.
- Segmentación geométrica y métodos de estimación de pose 6D (incluyendo ICP y PPF entre otros).
- Métricas de evaluación; y limitaciones actuales en entornos MR.

El foco se sitúa deliberadamente en enfoques geométricos y basados en visión clásica. Así, queda de forma general fuera del alcance de esta memoria técnicas basadas en aprendizaje profundo (*deep learning*), trabajos centrados exclusivamente en aspectos de interacción o experiencia de usuario y aplicaciones educativas que no contemplen problemas de detección de objetos, reconstrucción 3D o estimación de pose. Asimismo, no se abordan en detalle las técnicas de *Simultaneous Localization and Mapping* (SLAM) puramente basadas en sensores inerciales o en fusión de sensores.

Este acotamiento garantiza coherencia y profundidad en el análisis, evitando la dispersión temática. El resultado esperado es un mapa conceptual y metodológico que facilite tanto la implementación experimental del trabajo como la interpretación crítica de los resultados obtenidos en el dispositivo de realidad mixta seleccionado.

2.2 Marco tecnológico de la Realidad Mixta

2.2.1 Línea evolutiva en arquitectura de sensores

La evolución de los sensores en MR refleja un avance hacia la independencia del usuario. Inicialmente, los sistemas dependían de sensores externos y conexiones cableadas a bases estacionarias, lo que limitaba la movilidad, aunque permitía un seguimiento muy estable mediante marcadores ópticos fijos.

La irrupción de dispositivos autónomos [6] marcó un punto de inflexión: al incorporar cámaras pasivas RGB y unidades de medición inercial (IMU) se habilitaron algoritmos de odometría visual-inercial (VIO) y SLAM sin infraestructura externa. Este cambio trasladó la carga computacional al propio dispositivo, imponiendo restricciones de potencia y memoria que afectan directamente al rendimiento de los algoritmos de visión.

Posteriormente, la integración de sensores activos como *Time-of-Flight* (ToF) y LiDAR permitió obtener mediciones de profundidad directas [7]. Esto mejoró la robustez frente a oclusiones y variaciones lumínicas al complementar los datos de cámaras pasivas con mapas de disparidad de mayor precisión.

En etapas recientes, se han logrado hitos como el seguimiento ocular y manual mediante iluminación infrarroja (IR), optimizando la detección de gestos sin mandos físicos. Esta trayectoria no solo impulsa y fomenta el interés por la MR, sino que también sienta las bases para aplicar visión artificial en contextos reales con hardware que permite combinar múltiples fuentes de datos sensoriales.

A continuación, se presenta la Figura 2-2, que representa de forma visual la evolución tecnológica y los hitos más relevantes relativos a los dispositivos de VR, AR y RM desde su irrupción hace una década hasta la actualidad (imagen que refleja la información proporcionada en la referencia [6]).

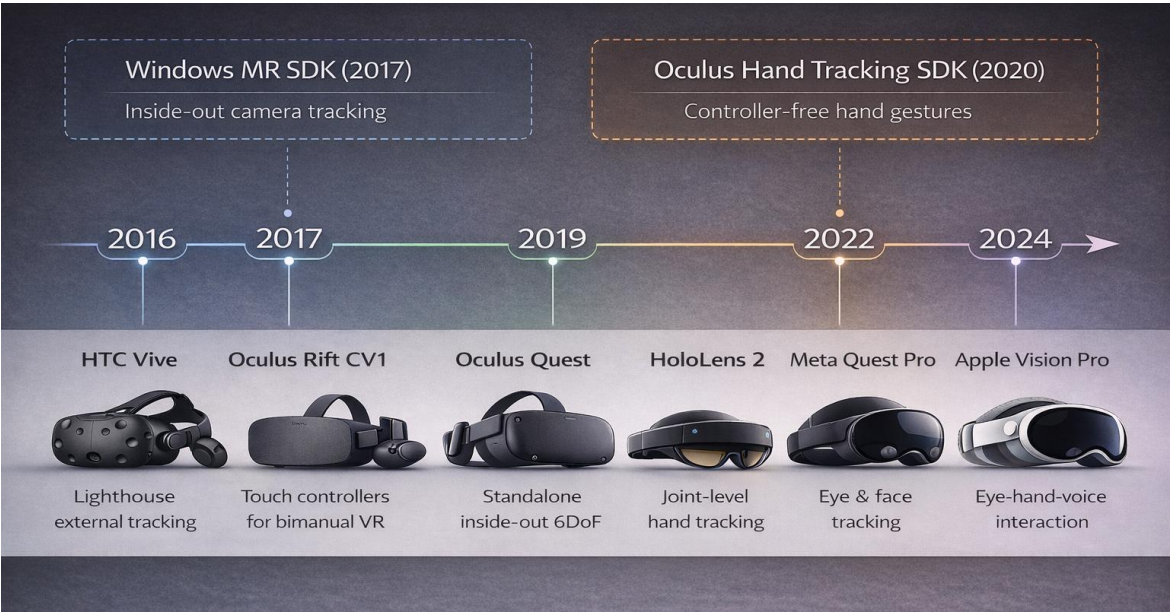


Figura 2-2: Evolución cronológica de los dispositivos XR (generada con IAG)

2.2.2 Dispositivos representativos actuales

Para ilustrar las implicaciones prácticas de las diferentes arquitecturas sensoriales existentes, la Tabla 2-1 resume los dispositivos autónomos actuales más relevantes, centrándose en sus sensores principales y en las ventajas y limitaciones teóricas relativas.

Dispositivo	Sensores	Ventajas	Desventajas
Quest 3 (Meta) [8]	Cámara RGB Passthrough IMU	Bajo coste Field Of View (FOV) amplio Versatilidad en diversos entornos	Sensibilidad a luz tenue Precisión media en profundidad
HoloLens 2 (Microsoft) [9]	RGB/IR ToF activo IMU	Profundidad robusta en entornos controlados	Mayor consumo energético FOV limitado
Vision Pro (Apple) [10]	RGB/IR LiDAR activo IMU	Alta precisión en <i>mapping</i> 3D dinámico	Alto coste de fabricación Peso elevado
Vive XR Elite (HTC) [11]	RGB <i>tracking</i> IMU	Modularidad <i>hardware</i>	Ecosistema limitado

Tabla 2-1: Comparación de dispositivos MR [7]

Los sensores pasivos destacan por su bajo coste y amplio campo de visión (FOV), aunque resultan vulnerables a condiciones lumínicas adversas. Los sensores activos ofrecen mayor precisión en profundidad, pero a expensas de complejidad y consumo energético.

Se destaca especialmente el dispositivo Meta Quest 3, mostrado en la Figura 2-3, por ser competitivo en prestaciones y coste [7]; de hecho, es el hardware de referencia elegido para este proyecto.



Figura 2-3: Dispositivo Meta Quest3 [8]

2.2.3 Motores de desarrollo

Unity representa un motor gráfico versátil y ampliamente adoptado en el ámbito MR, diseñado para el desarrollo en diversas plataformas para dispositivos móviles y visores autónomos. Su arquitectura y sus herramientas (como *XR Interaction Toolkit*) permiten la creación intuitiva de interacciones espaciales, apoyadas en *scripting C#* y con acceso a un extenso *Asset Store*.

Unreal Engine es referente en gráficos de alta fidelidad para MR. Incorpora tecnologías como *Nanite* (para geometrías complejas) y *Lumen* (iluminación global dinámica) [12], posibilitando experiencias inmersivas fotorrealistas. Además, su sistema de *blueprints* visuales facilita el desarrollo sin necesidad de conocimientos extensos en programación, lo que resulta muy conveniente.

Estas dos herramientas compiten directamente en diferentes entornos, siendo uno de ellos la realidad mixta y a la visión artificial. Debido a esto, la elección de un motor específico puede resultar motivo de estudio. Así, a modo ilustrativo se muestra en la Figura 2-4 los porcentajes de uso de ambos motores en materia de XR, y en la Figura 2-5 un ejemplo con prioridades de elección en función de las necesidades que cubre cada motor.

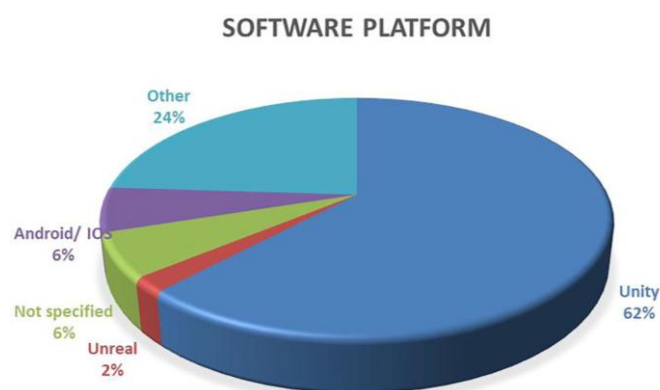


Figura 2-4: Plataformas software utilizadas para aplicaciones XR [4]

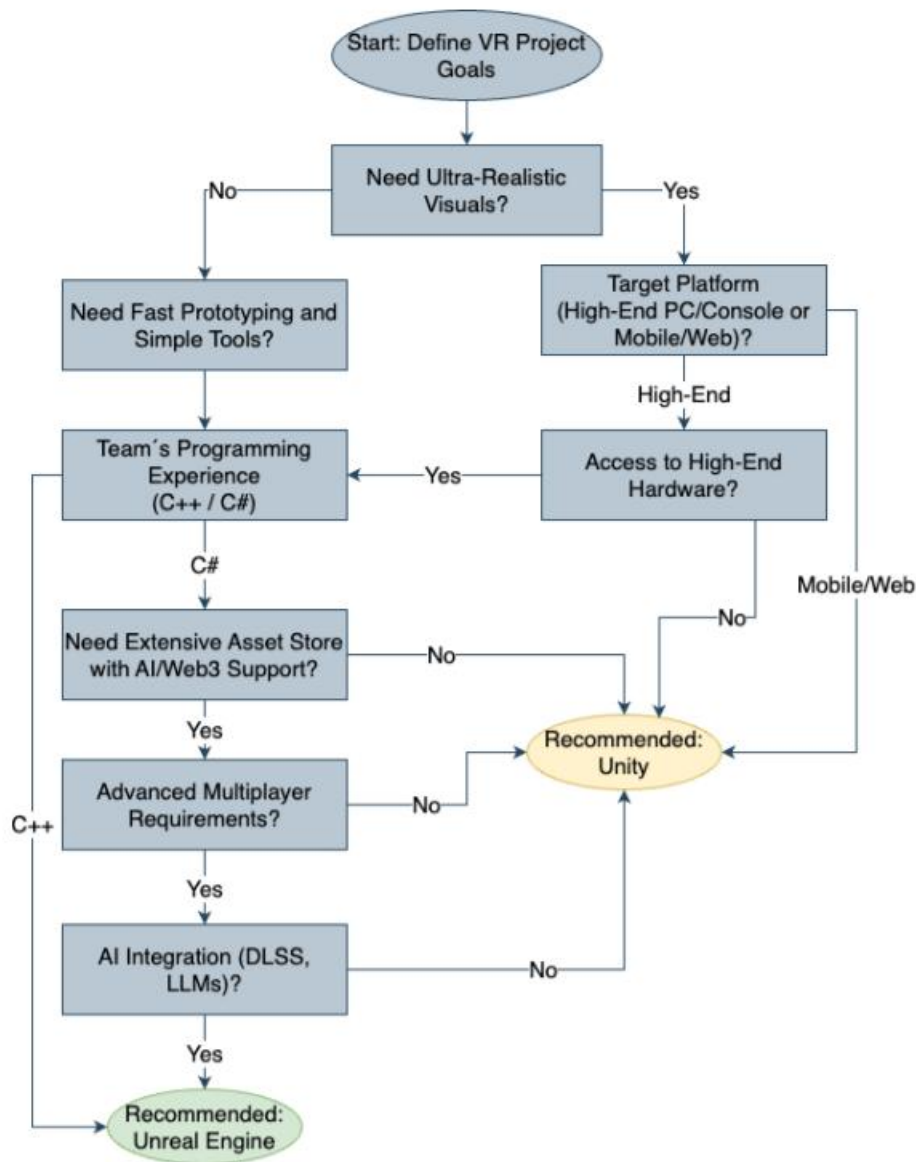


Figura 2-5: Diagrama de flujo para la elección del motor de desarrollo [12]

Ambos motores aprovechan OpenXR, una API abierta y estandarizada que abstrae el hardware subyacente. Esto permite desplegar la misma aplicación en plataformas como Quest, HoloLens o Vision Pro sin depender de SDKs propietarios. Esta interoperabilidad es crucial para el presente proyecto, pues asegura robustez y escalabilidad en la implementación de la estimación de pose 6D sobre la Quest 3 [13].

2.3 Adquisición de datos

2.3.1 Modalidades de captura

La captura de datos en MR emplea modalidades adaptadas a la fusión espacial y temporal. La elección de la modalidad condiciona la calidad del mapa de profundidad y, por tanto, la robustez de la segmentación y la estimación de pose. Las principales se muestran en la Tabla 2-2.

Modalidades	Descripción
RGB monocular [14]	Cámara única pasiva. Provee imágenes de alta resolución a bajo coste, útiles para la segmentación. Sin embargo, no ofrece profundidad directa, dependiendo de estimaciones geométricas indirectas.
RGB estéreo [14]	Par de cámaras calibradas en paralelo. Genera mapas de disparidad (profundidad densa) sin emitir luz activa, pero su precisión decae en zonas de baja textura u oclusiones.
RGB-D [14]	Dispositivos con sensor de profundidad integrado. Fusionan el canal RGB con un mapa de profundidad, facilitando reconstrucciones densas en tiempo real sin cálculos de disparidad. Son sensibles a condiciones lumínicas extremas o superficies reflectivas
<i>Time-of-Flight</i> (ToF) [15]	Sensores activos infrarrojos modulados. Miden distancias directas a pocos metros y son robustos en interiores, pero su precisión se ve afectada por la reflectividad de los materiales.
LiDAR [16]	Escáner láser activo. Genera mapas de profundidad precisos a larga distancia, con poca densidad de puntos y coste elevado.

Tabla 2-2: Modalidades para adquisición de datos

Cada modalidad presenta diversas ventajas y limitaciones. En MR es común combinar varias fuentes (RGB, IMU, ToF, etc.) para obtener información más rica y compensar las debilidades individuales de estas tecnologías.

2.3.2 Calibración y sincronización

La calibración garantiza la fidelidad geométrica de los datos y se puede dividir en intrínseca y extrínseca [17].

Abordando la calibración intrínseca, esta ocupa la determinación de la matriz interna K de la cámara [18]. Esta matriz, que se muestra en (2-1), modela la proyección perspectiva mediante las distancias focales (f_x, f_y) y los puntos principales (c_x, c_y).

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad (2-1)$$

Estos parámetros, estimados típicamente mediante tableros de calibración, corrigen las distorsiones ópticas y alinean los píxeles reales utilizando coordenadas normalizadas.

Por su parte, la calibración extrínseca resuelve la transformación rígida (rotación y traslación) entre múltiples vistas o sensores. Las siguientes ecuaciones pretenden ilustrar cómo se resuelve matemáticamente una transformación, la cual establece la relación entre dos puntos a través de las matrices de rotación y traslación (conocida como matriz de transformación). En (2-2) se presenta la transformación de un punto cualquiera P en otro mediante la matriz de transformación, la cual se

desarrolla en (2-3). Además, se aclara explícitamente con (2-4) cómo funciona esa transformación con las coordenadas 3D de un punto cualquiera [19].

$$P_c = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \cdot P_w \quad (2-2)$$

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \tau_{11} & \tau_{12} & \tau_{13} & t_x \\ \tau_{21} & \tau_{22} & \tau_{23} & t_y \\ \tau_{31} & \tau_{32} & \tau_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-3)$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = [R|t] \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad (2-4)$$

Los errores en estos parámetros extrínsecos generan desalineaciones especialmente en regiones lejanas (donde la disparidad disminuye mucho), lo que degrada significativamente la precisión de la reconstrucción de profundidad. Así, una calibración precisa (especialmente extrínseca) es esencial para evitar errores acumulativos en todo el *pipeline*.

También se debe considerar la sincronización temporal para preservar la integridad de escenas dinámicas, ya que un desfase temporal equivale a un error de pose (que es proporcional a la velocidad con la que se mueve el visor), rompiendo el supuesto de rigidez de la escena y falseando correspondencias epipolares que afectan directamente a la disparidad y, por consiguiente, al resto del *pipeline*.

2.3.3 Sistemas de referencia

Los sistemas de referencia organizan las coordenadas mediante una jerarquía de transformaciones desde los sensores hasta el espacio global. Por ejemplo, el marco de la cámara ancla los puntos 3D en el centro óptico del sensor; el marco del visor incorpora la IMU para definir la pose de la cabeza relativa al mundo; el marco *world* sitúa al usuario en un sistema global mediante SLAM o anclajes espaciales; y el marco del objeto define la pose 6D local para el seguimiento de cada objeto.

En un *pipeline* MR típico, se transforma la pose del objeto al sistema global combinando las poses de visor y cámara [19]. Matemáticamente, si $T_{\text{obj}}^{(\text{world})}$ es la transformación del objeto al mundo, $T_{\text{visor}}^{(\text{world})}$ la del visor, y $T_{\text{cam}}^{(\text{visor})}$ la de la cámara en el visor, entonces se cumple lo establecido en (2-5) [18].

$$T_{\text{obj}}^{(\text{world})} = T_{\text{visor}}^{(\text{world})} \cdot T_{\text{cam}}^{(\text{visor})} \quad (2-5)$$

Esta cadena evita la acumulación de deriva, estabiliza las superposiciones virtuales y habilita métricas absolutas. En cambio, inconsistencias en cualquiera de estos marcos generan *jitter* o escalas erróneas que comprometen la detección de objetos y la coherencia perceptiva.

2.4 Reconstrucción estereoscópica

La reconstrucción estereoscópica 3D [20] sigue un *pipeline* clásico y robusto que convierte pares de imágenes sincronizadas y calibradas en representaciones tridimensionales densas.

El proceso encadena varias etapas: primero se aplican las restricciones de geometría epipolar, luego se realiza la rectificación estereoscópica, después se calcula la imagen de disparidad píxel a píxel, y finalmente se triangula esa disparidad para obtener profundidades y generar una nube de puntos 3D. A continuación, se abordarán dichas etapas.

2.4.1 Geometría epipolar

La geometría epipolar [21] surge de la proyección central de dos cámaras no coincidentes. Para cada punto en la imagen izquierda, se define una línea epipolar en la imagen derecha donde debe encontrarse su correspondiente (véase la Figura 2-6).

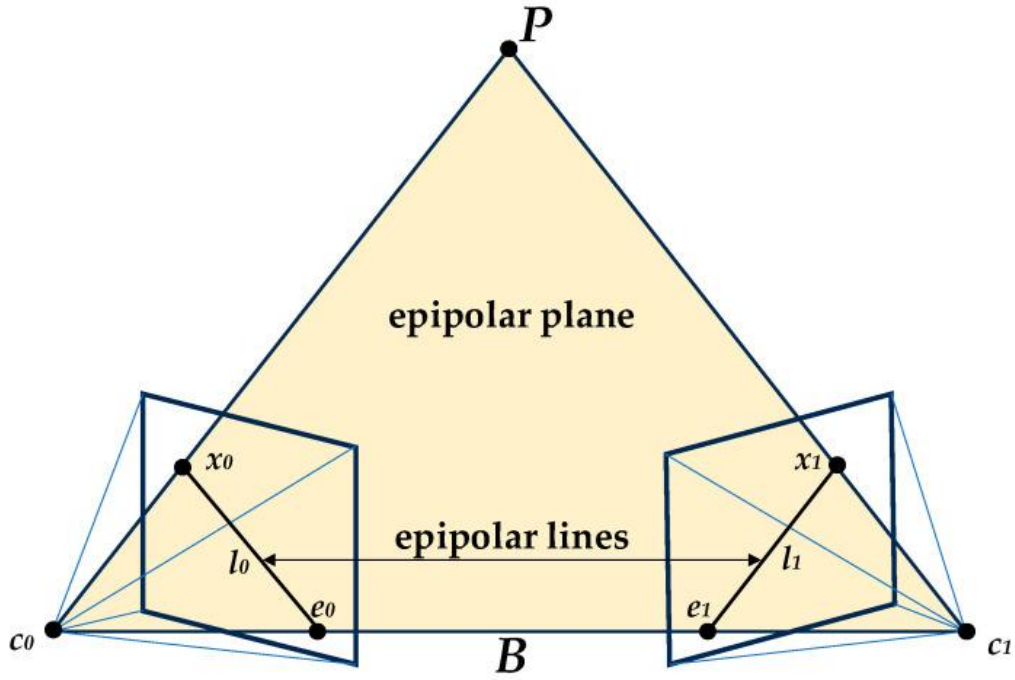


Figura 2-6: Diagrama sobre geometría epipolar [20]

Esta restricción se formaliza mediante la estimación de la matriz fundamental F [22]. Esta matriz encapsula la rotación relativa, la traslación y las calibraciones intrínsecas de ambas cámaras. Si x_1 es un punto en la imagen izquierda, su línea epipolar en la derecha viene dada por:

$$l_1 = F \cdot x_0 \quad (2-6)$$

Gracias a esta restricción de búsqueda de coincidencias se reduce drásticamente la ambigüedad de correspondencias, pues en lugar de buscar en un espacio bidimensional completo (toda la imagen), el *match* se limita a una línea unidimensional [23], acelerando los algoritmos densos y mitigando falsos positivos en regiones ambiguas como superficies reflectantes, comunes en MR. A esto le sigue *RANdom SAmple Consensus* (RANSAC), con lo que se consigue rechazar correspondencias erróneas (*outliers*).

La precisión del resultado depende de las correspondencias, extraídas habitualmente mediante descriptores como *Scale-Invariant Feature Transform* (SIFT) u *Oriented FAST and Rotated BRIEF* (ORB) [20].

La relación entre la matriz fundamental y la calibración intrínseca da lugar a la matriz esencial E (véase (2-7)), que opera sobre coordenadas normalizadas y permite la descomposición en la rotación R

y la traslación t entre las dos cámaras. Esta descomposición constituye el punto de partida para el proceso de rectificación.

$$E = [R|t] \quad (2-7)$$

2.4.2 Rectificación estereoscópica

La rectificación estereoscópica transforma el par estéreo para alinear todas las líneas epipolares horizontalmente [23]. Esto se consigue aplicando homografías H y H' derivadas de la descomposición de la matriz esencial E , de modo que los epipolos queden en el infinito y las filas de ambas imágenes rectificadas sean paralelas entre sí.

Esta transformación simplifica la búsqueda de disparidad a escaneos horizontales por fila, eliminando la componente vertical. Esto permite la aplicación eficiente de algoritmos posteriores como *Semi-Global Matching* (SGM).

2.4.3 Disparidad

Una vez rectificadas las imágenes, la disparidad d mide el desplazamiento horizontal en píxeles entre un punto en la imagen izquierda y su homólogo en la derecha. En un sistema calibrado esto se relaciona con la profundidad Z mediante

$$d = x - x' = \frac{B \cdot f}{Z} \quad (2-8)$$

donde f es la distancia focal en píxeles y B el *baseline*, es decir, la separación entre cámaras (Véase la Figura 2-7) [24].

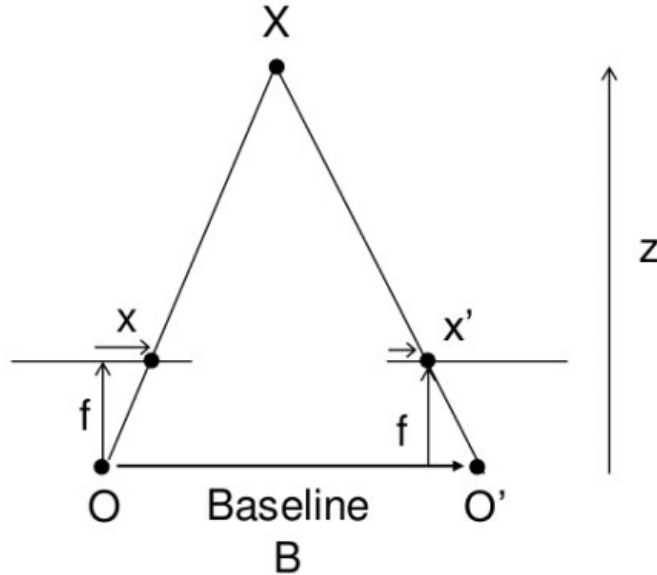


Figura 2-7: Diagrama para comprender la disparidad [24]

A partir de esta relación geométrica, el problema fundamental pasa a ser estimar con precisión la disparidad para cada píxel. Esto se realiza mediante algoritmos de correspondencia estéreo aplicados a imágenes previamente rectificadas, lo que reduce la búsqueda del punto homólogo a la misma línea horizontal en ambas imágenes.

Una primera aproximación corresponde a los métodos locales, que calculan el costo de correspondencia comparando ventanas de píxeles entre las dos imágenes. Para evaluar la similitud se utilizan métricas como la suma de diferencias al cuadrado (SSD) o la transformada *Census*, esta última especialmente útil porque se adapta a cambios de iluminación.

No obstante, al depender únicamente de información local, estos métodos presentan limitaciones en regiones con baja textura o patrones repetitivos, donde la correspondencia es ambigua. Como resultado, los mapas de disparidad generados suelen contener ruido granular conocido como *speckle*.

Para mejorar la coherencia espacial del resultado se emplean métodos globales, siendo uno de los más utilizados SGM, una variante modificada y acelerada de SBGM, ampliamente utilizada en visión artificial [25]. Este algoritmo busca un equilibrio entre precisión y eficiencia, analizando el contexto alrededor de cada píxel (mediante “ventanas” locales) y aplicando penalizaciones de suavidad (mediante los términos $P1$, para penalizar pequeños cambios de disparidad entre píxeles vecinos, y $P2$, que penaliza discontinuidades más abruptas). Así SBGM combina información local con consistencia espacial para elegir la disparidad con menor costo acumulado y así producir mapas de disparidad más densos y menos ruidosos que con métodos puramente locales.

Una vez estimada la disparidad, es habitual aplicar filtros de postprocesado para refinar el resultado. Un ejemplo ampliamente utilizado es *Weighted Least Squares* (WLS) [26], que suaviza el mapa de disparidad evitando mezclar objetos distintos. Esto lo consigue asignando diferentes pesos a píxeles de regiones diferentes (véase la Figura 2-8).



Figura 2-8: Apreciación visual del filtro WLS [26]

Este refinamiento resulta especialmente relevante en aplicaciones MR, donde la correcta alineación entre elementos virtuales y la geometría de la escena real depende de mapas de profundidad estables y bien definidos.

2.4.4 Profundidad

Con el modelo rectificado y la disparidad calculada, la profundidad Z surge de la relación inversamente proporcional a la disparidad, como se muestra en la ecuación (2-9).

$$Z = \frac{(f \cdot b)}{d} \quad (2-9)$$

donde f es la distancia focal efectiva en píxeles y b el *baseline* efectivo. Dada la naturaleza de esta fórmula, los errores de disparidad se amplifican con la distancia [20], según se establece en (2-10):

$$\delta Z \approx \left(\frac{Z^2}{f \cdot b} \right) \cdot \delta d \quad (2-10)$$

Esto significa que objetos lejanos (de disparidad baja, pues esta corresponde a distancias grandes) y dispositivos con *baseline* reducida (como los propios visores MR) agravan la inexactitud en la profundidad. En la práctica, la precisión de Z cae bruscamente con la distancia y conocer este comportamiento es esencial para ajustar las expectativas de precisión según el rango operativo deseado.

2.4.5 Nube de puntos 3D

Finalmente, concepto que cierra la reconstrucción estereoscópica 3D es la creación de una nube de puntos. Mediante la retroproyección se triangula cada píxel válido (u, v) proveniente del mapa de disparidad al espacio 3D en el marco de la cámara izquierda mediante las relaciones expuestas en (2-11).

$$\begin{aligned} X &= (u - c_x) \cdot Z \cdot f_x \\ Y &= (v - c_y) \cdot Z \cdot f_y \end{aligned} \quad (2-11)$$

Este proceso genera nubes de puntos densas que constituyen la entrada directa de los algoritmos de registro y estimación de pose [20].

El filtrado de la nube es un paso previo imprescindible. Mediante el filtrado estadístico de *outliers* se eliminan los puntos cuya distancia media a sus vecinos supera un cierto umbral, reduciendo así el *speckle*.

2.5 Estimación de pose

La estimación de pose 6D (6 grados de libertad) es el núcleo analítico de la detección de objetos en MR. Esta consiste en determinar la transformación rígida $T \in SE(3)$ (que contiene parámetros de rotación, $R \in SO(3)$, y traslación, $t \in \mathbb{R}^3$) para alinear una nube de puntos observada con un modelo conocido M . Es decir, el objetivo es determinar dónde se encuentra y cómo está orientado un objeto en el espacio. Formalmente, se busca

$$T = (R, t) \quad (2-12)$$

$$T' = \operatorname{argmin}_T E(T, P, M) \quad (2-13)$$

donde T es la matriz de transformación (compuesta por R y t) y donde E mide discrepancias geométricas entre la escena observada P y el modelo M . Esta ecuación representa la búsqueda de coincidencias para asegurar que el modelo coincida lo mejor posible con la escena observada.

En entornos MR, los métodos geométricos clásicos siguen siendo relevantes frente a enfoques de aprendizaje profundo debido a su eficiencia en hardware limitado, su interpretabilidad y su capacidad de alcanzar precisión milimétrica cuando se dispone de una inicialización adecuada.

2.5.1 Segmentación geométrica

La segmentación geométrica existe como etapa previa a la propia estimación de pose y se aplica para extraer regiones candidatas de la nube 3D completa que correspondan a objetos o superficies coherentes. En lugar de trabajar con toda la escena, se agrupan puntos coherentes según cercanía euclidiana, continuidad de normales y ajuste a primitivas geométricas para determinar qué grupos de puntos tienen mayor probabilidad de pertenecer al mismo objeto.

En MR, este preprocesamiento es esencial para separar los objetos de interés de fondos estructurados (típicos en interiores), reduciendo el espacio de búsqueda y evitando alineaciones erróneas con elementos no relevantes.

El procedimiento estándar combina RANSAC con el algoritmo de segmentación *región growing*. Mientras RANSAC detecta y elimina planos dominantes de la nube de puntos, *region growing* agrupa puntos de una nube 3D en función de su similitud geométrica. A partir de puntos “semilla” expande una región de búsqueda para encontrar vecinos cercanos con normales y posiciones similares. De este modo se forman regiones que corresponden a superficies coherentes de la escena [27]. En entornos interiores, este método produce segmentaciones precisas, medidas con valores de *Intersection over Union* (IoU) que indica una si existe coincidencia entre la región segmentada y el objeto real.

Una limitación clave es la sensibilidad al ruido en los bordes: el ruido de profundidad típico (aproximadamente 1-2 mm en cámaras estéreo) puede propagarse a las fronteras de las regiones segmentadas. Por ello a menudo se aplican filtros previos para suavizar el ruido sin perder los bordes principales.

2.5.2 PPF

PPF (*Point Pair Feature*) es un método para estimar la pose 3D de un objeto comparando pares de puntos entre un modelo conocido y la nube de puntos observada en la escena. La idea principal subyacente consiste en lograr describir la relación geométrica entre dos puntos del objeto de forma que esa descripción no cambie, aunque el objeto rote o se desplace (véase la Figura 2-9).

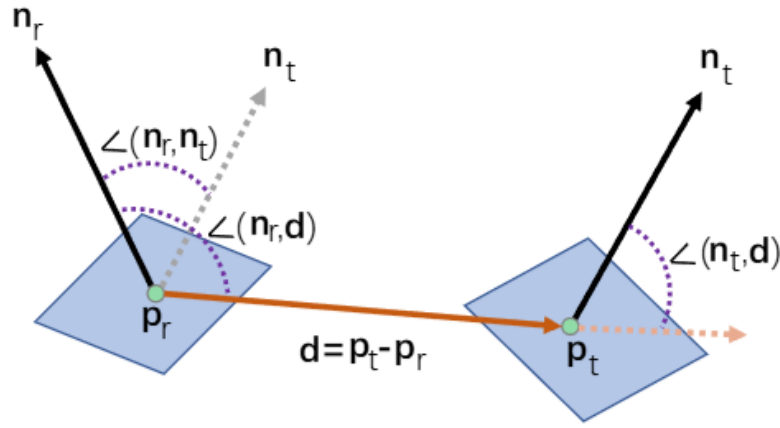


Figura 2-9: Point Pair Feature (PPF) [28]

Para cada par de puntos del modelo (p_r, p_t) , junto con sus normales de superficie (n_r, n_t) , se construye un descriptor geométrico (véase la ecuación (2-14)) que incluye la distancia entre los puntos y tres ángulos que describen cómo se orientan las normales respecto a la línea que une ambos puntos. Este descriptor captura la geometría local del objeto y es invariante a transformaciones rígidas, es decir, no cambia si el objeto se rota o se traslada [28].

$$PPF(p_r, p_t) = (\|d\|, \angle(n_r, d), \angle(n_t, d), \angle(n_r, n_t)) \quad (2-14)$$

Los descriptores de todos los pares del modelo se almacenan previamente. Cuando se analiza la escena, se calculan descriptores equivalentes para pares de puntos observados y se buscan coincidencias con los del modelo. Cada coincidencia genera un voto para una posible pose del objeto dentro de un “espacio de Hough” discretizado y la pose que acumula más votos se considera la hipótesis más probable.

Una ventaja importante de PPF es que tolera oclusiones parciales: no es necesario observar todo el objeto para reconocerlo. En muchos casos basta con que aproximadamente un pequeño porcentaje de los pares de puntos coincidan, lo que resulta útil en entornos de MR, donde los objetos suelen estar parcialmente ocluidos.

La robustez de PPF [27] depende de la calidad de las normales estimadas: en nubes estéreo ruidosas, las normales calculadas presentan mayor varianza, lo que incrementa los falsos positivos en la votación. Este efecto se mitiga mediante *clustering* de los votos y umbrales adaptativos de soporte que descartan hipótesis con soporte insuficiente.

2.5.3 ICP

ICP (*Iterative Closest Point*) [29] es un método clásico que itera entre asignar correspondencias “punto-a-punto” (o “punto-a-plano”; véase la Figura 2-10) y resolver la transformación rígida que minimiza el error cuadrático medio [18]. Formalmente, en cada iteración se minimiza

$$E = \sum \|R \cdot p_i + t - q_i\|^2 \quad (2-15)$$

donde p_i son puntos del modelo y q_i sus correspondientes más cercanos en la escena observada. La solución óptima en cada paso se obtiene mediante descomposición en valores singulares (SVD), garantizando convergencia a un mínimo local.

Existen varias extensiones de ICP que mejoran su rendimiento en distintos escenarios. La variante “punto-a-plano” (véase Figura 2-10) mide el error proyectándolo sobre la normal de la superficie del punto objetivo, en lugar de usar solo la distancia entre puntos. Esto aprovecha la orientación local de la superficie y permite convergencias más rápidas en superficies suaves.

En contextos MR, ICP suele emplearse como refinamiento fino cuando ya se dispone de una inicialización cercana (Véase la Figura 2-11), dado que es muy preciso localmente, pero puede atascarse en mínimos subóptimos sin buena aproximación inicial.

Colored ICP [30] añade además información de color (RGB) al proceso de correspondencia. Esto ayuda a resolver ambigüedades geométricas, por ejemplo, para objetos con formas simétricas o con poca textura geométrica.

Por su parte, *Generalized ICP* (GICP) [30] modela la distribución local de los puntos mediante matrices de covarianza, teniendo en cuenta la incertidumbre de las mediciones. Este enfoque resulta más robusto cuando las nubes de puntos están degradadas o contienen ruido (como ocurre frecuentemente con cámaras estéreo), mejorando la convergencia y reduciendo el error de estimación respecto al ICP estándar.

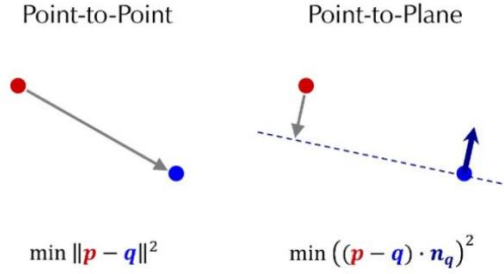


Figura 2-10: Variantes de ICP [29]

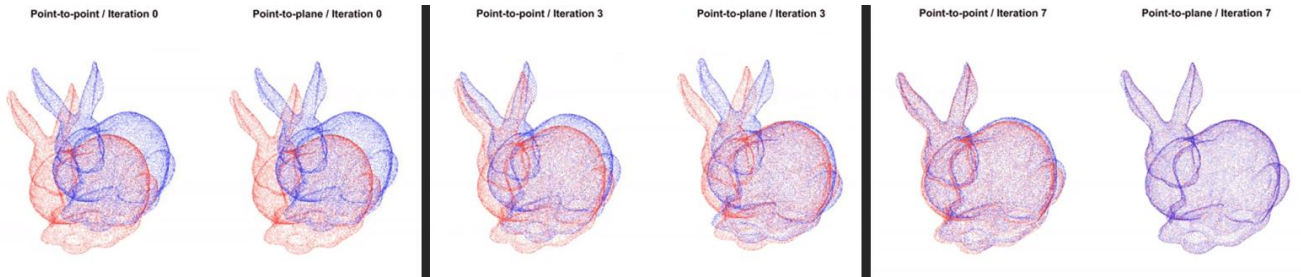


Figura 2-11: Comparación de convergencia: ICP punto a punto e ICP punto a plano [29]

2.5.4 PPF+ICP

La estrategia más empleada en la práctica combina en cascada PPF para la generación de hipótesis globales con ICP o GICP para el refinamiento local. Esta hibridación aprovecha las fortalezas de ambos: PPF proporciona robustez ante oclusiones parciales y explora exhaustivamente el espacio de posibles

poses, mientras que ICP otorga alineación precisa y rápida convergencia cuando parte de una inicialización adecuada.

El *pipeline* opera en dos fases complementarias. En la primera, PPF genera múltiples hipótesis de pose mediante votación geométrica en el espacio de Hough que posteriormente se filtran mediante *clustering* (agrupar datos por similitud) para conservar únicamente las hipótesis más consistentes. En la segunda fase, ICP o GICP refinan cada una de estas hipótesis mediante un número reducido de iteraciones hasta alcanzar la convergencia y obtener una alineación precisa entre el modelo y la nube de puntos observada.

Este algoritmo, subdividido en los carriles “Training” y “Runtime”, se adecúa perfectamente a la descripción del método *Surface Matching* que proporciona OpenCV (Véase la Figura 2-12).

Esta estrategia en cascada mejora la robustez frente a oclusiones parciales y alcanza una alta precisión medida mediante la métrica *Average Distance of model points* (ADD), relevante en aplicaciones de MR dinámica.

En dispositivos autónomos como Quest 3, esta hibridación también contribuye a reducir la latencia, ya que el refinamiento con ICP se limita a las regiones de interés (ROIs) propuestas por PPF, permitiendo tiempos de procesamiento del orden de 30–50 ms por fotograma.

No obstante, el método presenta limitaciones en objetos con simetrías rotacionales, donde diferentes orientaciones pueden generar hipótesis equivalentes, y en escenarios con calibración estéreo imprecisa, que puede degradar la calidad de la nube de puntos y afectar al proceso de alineación.



OUTLINE OF THE ALGORITHM

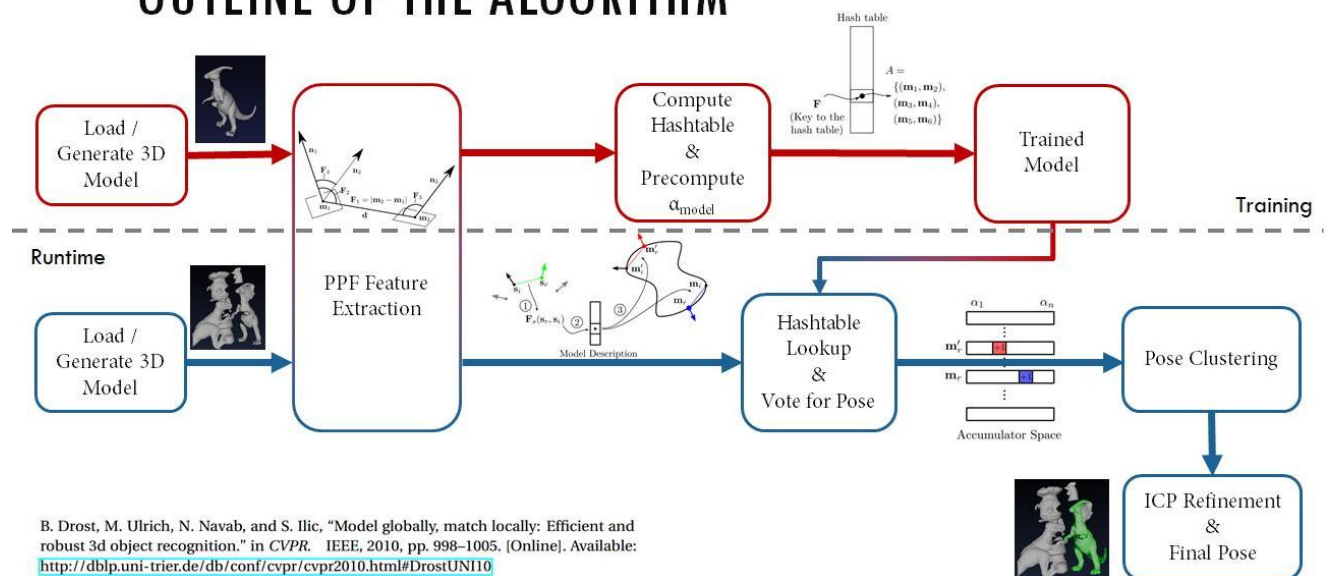


Figura 2-12: Algoritmo *Surface Matching* [31]

2.6 Limitaciones en MR

Los *pipelines* de visión artificial 3D en sistemas MR operan bajo restricciones estrictas (por ejemplo, con exigencias de baja latencia para evitar mareos).

Estas condiciones introducen errores sistemáticos que degradan la precisión de profundidad, la robustez de segmentación y la fiabilidad de la estimación de pose 6D. A su vez, limitan el rango operativo efectivo (prefiriendo interacciones cercanas sobre escenas lejanas) y pueden romper la coherencia entre mundo real y virtual. La relación entre limitaciones sensoriales (por ejemplo, una *baseline* reducida) y efectos perceptivos (como el *jitter*) subraya la necesidad de *pipelines* adaptativos.

2.6.1 Baseline. Degradación en profundidad

La *baseline* estéreo en visores MR es inherentemente corta (por ejemplo, 6-7 cm en la Quest 3). Esta limitación física reduce la disparidad máxima detectable y, con ella, la sensibilidad de la triangulación. La ecuación fundamental de la profundidad implica que el error de triangulación crece cuadráticamente con la distancia, como se ve en la ecuación (2-10) [32].

En sistemas tradicionales se pueden aplicar soluciones como *baseline* variable, permitiendo mantener precisión constante para distintos rangos. En MR, el diseño fijo de los lentes impide estas adaptaciones, confinando la reconstrucción métrica precisa (error inferior a 1 cm) a distancias cortas (menores de 2-3 metros) y generando degradaciones altas en rangos medios-lejanos (3-10 m).

Esta limitación explica muchas observaciones empíricas obtenidas a través del *passthrough* en MR, donde la profundidad aparenta aplastarse en los fondos, impactando directamente sobre las aplicaciones que requieren la alineación virtual de objetos extendidos en el espacio.

2.6.2 Disparidad. Oclusiones y baja textura

Las oclusiones (regiones visibles en una cámara pero ocultas en la otra) [33] y las zonas de baja textura (superficies uniformes sin gradientes salientes) provocan fallos estructurales: disparidades indefinidas (huecos) o ambiguas (múltiples *matches* igualmente válidos). Estos problemas persisten incluso en algoritmos como SGBM.

En entornos MR, estos problemas se amplifican por la prevalencia de manos que intervienen en la escena, objetos próximos que generan oclusiones frecuentes, y escenas interiores con paredes lisas y superficies monocromas. Como consecuencia práctica, se generan mapas de profundidad incompletos y ruidosos, reduciendo la robustez de la segmentación y la detección de objetos.

2.6.3 Profundidad. Sensibilidad al error

La conversión de disparidad a profundidad exhibe alta sensibilidad al ruido en d . La derivada de Z respecto a d evidencia que pequeños errores en disparidad amplifican dramáticamente la profundidad en entornos de disparidades bajas (véase (2-16)).

$$\frac{\partial Z}{\partial d} = \frac{-(f \cdot b)}{d^2} \quad (2-16)$$

Este comportamiento produce ruido temporal (*jitter*) observable durante procesos de seguimiento continuo. En MR, el efecto no uniforme desestabiliza el registro de *overlays*, donde fluctuaciones locales inducen parpadeos en los elementos virtuales pese a que el error medio global sea bajo.

2.6.4 Calibración. Intrínsecos y extrínsecos.

La calibración precisa de los parámetros intrínsecos y extrínsecos es un prerequisite para la rectificación epipolar y la triangulación. Errores en este ámbito pueden inducir líneas epipolares desalineadas, errores en la rectificación y sesgos sistemáticos en la profundidad [34].

Para ilustrar la sensibilidad de estos parámetros, un error de reproyección de 1 píxel puede suponer hasta 10 cm de error con respecto a objetos que se encuentren a 5 metros de distancia. Sin acceso a recalibración dinámica en el hardware comercial, esta limitación enfatiza la importancia de métricas de validación (como el error de reproyección que se acaba de mencionar) para detectar degradaciones de la calibración.

2.6.5 Restricciones de tiempo real

Las plataformas MR exigen procesamiento en tiempo real (por ejemplo, 60 FPS) para una experiencia fluida. Esto obliga a diversos algoritmos a sacrificar parte de la precisión por velocidad. Los recursos limitados imponen compromisos en la complejidad de los métodos. En la práctica, los *pipelines* deben equilibrar precisión y latencia, a menudo reduciendo la densidad de procesamiento para cumplir plazos estrictos.

2.6.6 Deriva de pose

En sesiones prolongadas de MR, la estimación de pose acumulada puede derivar debido a errores pequeños constantes. Sin un bucle de cierre activo (*loop closure*), la pose del visor y los objetos pueden alejarse progresivamente de la realidad. Esta deriva acumulativa deteriora la consistencia visual: los objetos virtuales comienzan a “flotar” o desalinearse. Por ello, los sistemas MR suelen implementar estrategias de reinicio o corrección periódica de la pose para mitigar la deriva en el tiempo.

2.7 Métricas evaluables

Las métricas evaluables constituyen el pilar cuantitativo para validar los *pipelines* de visión artificial en MR. Permiten medir precisión, robustez y orientar optimizaciones en dispositivos con recursos limitados. Su diseño considera los desafíos propios de MR: el crecimiento cuadrático de errores con la distancia, las ambigüedades por simetrías en la pose, y las exigencias de tiempo real.

2.7.1 Concepto y criterios de selección

Una métrica de evaluación en visión artificial 3D transforma una salida cualitativa (mapa de profundidad, nube de puntos, estimación de pose) en un indicador numérico estandarizado [35].

Este indicador refleja la fidelidad geométrica, la estabilidad perceptual o el coste computacional del sistema. La elección de métricas adecuadas es tan crítica como el diseño del algoritmo: una métrica inapropiada puede ocultar fallos relevantes o penalizar comportamientos aceptables en el contexto de uso. Por ejemplo, en MR un desplazamiento milimétrico en la pose puede generar disparidades binoculares molestas, por lo que las métricas deben ser sensibles a esos pequeños errores.

Los criterios de selección deben alinearse con las necesidades del usuario: las discrepancias de profundidad no uniformes (errores menores en corto alcance y mayores en alcance medio) exigen métricas que ponderen el rango operativo real. Asimismo, la latencia se debe medir de extremo a extremo (captura de sensores hasta emisión en pantalla), no solo teniendo en cuenta el tiempo de cómputo del algoritmo aislado.

Finalmente, las métricas estándar (KITTI para reconstrucción, BOP para pose [36]) deben adaptarse al hardware MR, pues la línea base, el FOV y el ruido pueden diferir de las plataformas de referencia.

2.7.2 Métricas de profundidad

Para evaluar mapas de profundidad se emplean métricas complementarias que cubren distintas dimensiones del error [37]. Cada métrica resalta un aspecto del comportamiento del estimador:

- *Mean Absolute Error* (MAE): El error absoluto medio calcula la media aritmética de las diferencias absolutas entre las profundidades predichas y las reales (*ground truth*), ofreciendo una visión lineal de la precisión global, sin penalizar especialmente los valores atípicos (*outliers*).
- *Root Mean Square Error* (RMSE): La raíz del error cuadrático medio penaliza fuertemente los errores grandes típicos en regiones de baja textura o bordes (véase (2-17)).

$$RMSE = \sqrt{\left(\frac{1}{N} \cdot \left(\sum |Z_i - \hat{Z}_i|^2\right)\right)} \quad (2-17)$$

- *Absolute Relative Error* (AbsRel): El error relativo absoluto consiste en la media de los errores absolutos normalizados por la profundidad real. Esta normalización es útil en MR, donde los errores tienden a crecer con la distancia. Esta métrica permite comparar de forma uniforme sensores pasivos (estéreo) y activos (ToF/LiDAR) (Véase (2-18)).

$$AbsRel = \frac{1}{N} \cdot \sum \frac{|Z_i - \hat{Z}_i|}{Z_i} \quad (2-18)$$

2.7.3 Métricas de pose 6D

El *benchmark* BOP (*Benchmark for 6D Object Pose Estimation*) [36] estandariza la evaluación de pose 6D mediante tres funciones de error principales, las cuales se presentan en la Tabla 2-3.

Métricas	Descripción
<i>Visible Surface Discrepancy</i> (VSD)	VSD compara las siluetas proyectadas del objeto bajo la pose estimada y la real, considerando solo la superficie visible y permitiendo pequeñas discrepancias. Es adecuada para objetos parcialmente ocluidos.
<i>Maximum Symmetry-Aware Surface Distance</i> (MSSD)	MSSD mide la máxima distancia 3D entre superficies correspondientes, tomando la mejor transformación por simetría del objeto. Es ideal para objetos simétricos en aplicaciones de manipulación en MR.
<i>Maximum Symmetry-Aware Projection Distance</i> (MSPD)	MSPD mide las distancias máximas en la imagen (reproyección) entre la pose estimada y la real, teniendo en cuenta simetrías. Es relevante en MR donde la percepción visual prioriza la coherencia 2D.

Tabla 2-3: Métricas del *benchmark* BOP

2.7.4 Métricas de detección

La calidad de una detección 3D se evalúa combinando criterios volumétricos y de pose. Se considera una detección correcta si el IoU 3D (intersección sobre unión de volúmenes del *bounding box* detectado frente a *ground truth*) supera 0.5.

Como complemento, se emplea la métrica ADD (*Average Distance*) [38]. Esta métrica mide la distancia promedio entre los puntos del modelo 3D transformados en comparación con la pose estimada y la pose real.

2.7.5 Estabilidad temporal

La estabilidad temporal cuantifica la variabilidad de la estimación de pose a lo largo de una secuencia. En MR son críticos los fenómenos presentados en la Tabla 2-4.

Métricas	Descripción
<i>Jitter</i>	Consiste en oscilaciones de alta frecuencia en la posición u orientación estimadas en una escena estática. Se mide como la desviación estándar de los parámetros de pose entre fotogramas consecutivos.
<i>Drift</i>	La deriva es el error acumulado en trayectorias largas, evaluado con métricas como ATE y RPE [39]. La deriva indica cómo las estimaciones se alejan progresivamente del <i>ground truth</i> .
<i>Swim</i>	Este efecto consiste en oscilaciones de baja frecuencia que hacen “nadar” visualmente los objetos virtuales sobre la escena real. También llamado “ <i>drift</i> perceptual”, resulta especialmente perturbador para el usuario, aunque el error global sea pequeño.

Tabla 2-4: Fenómenos temporales relevantes

2.7.6 Coste computacional

El rendimiento en MR se mide no solo en FPS, sino en la latencia *motion-to-photon*: el tiempo total desde la captura sensorial hasta la emisión en pantalla [38].

Para evitar mareos se apunta a latencias <20 ms en VR y <7 ms en AR, con frecuencias objetivo de 72–120 Hz según el visor. También se monitoriza el consumo energético (normalmente como porcentaje de GPU/CPU), dado que en dispositivos autónomos la batería es crítica. Además, en la práctica, los *pipelines* buscan un balance entre densidad de puntos y latencia.

3 DESARROLLO

3.1 Contextualización

El Estado del Arte tiene por objetivo contextualizar el proyecto, describiendo el estado de algunas técnicas de visión artificial aplicables a entornos de realidad mixta, identificando sus fundamentos geométricos, documentando algunas de sus limitaciones conocidas y estableciendo un breve marco de métricas bajo el cual pueden evaluarse sus resultados. Sin embargo, entre el marco teórico que recoge el Estado del Arte y el sistema funcional que se presenta a lo largo de este capítulo existen tomas de decisiones de diseño orientadas a la búsqueda de buenos resultados.

Para comenzar el desarrollo del proyecto, el presente apartado tiene el cometido de introducir los entornos de trabajo y presentar la estructura (*pipeline*) completa del sistema, que sirve como mapa de navegación para las secciones que siguen.

3.1.1 Herramientas y entorno de desarrollo

El sistema implementado para este proyecto se articula en dos capas funcionales diferenciadas: preprocesado y postprocesado. El preprocesado se nutre del dispositivo Meta Quest 3 y se desarrolla mediante la aplicación Unity. Por su parte, el postprocesado se ejecuta en un PC local y se desarrolla en VSCode, trabajando con lenguaje Python.

Cada capa emplea un conjunto específico de herramientas seleccionadas en función de los requisitos técnicos de las tareas que resuelve. A continuación, se describen las herramientas principales empleadas a lo largo de todo el proyecto.

3.1.1.1 Meta Quest 3

Meta Quest 3 es el visor autónomo de realidad mixta de Meta lanzado en 2023 (véase la Figura 3-1). Se trata del dispositivo de mayor accesibilidad económica entre los visores MR de prestaciones avanzadas disponibles en el mercado en el momento de desarrollo de este proyecto.

Estas gafas de realidad mixta disponen de dos cámaras RGB pasivas orientadas hacia el exterior con resolución suficiente para la reconstrucción estereoscópica, proporcionando un campo de visión amplio. Además, incorpora una Unidad de Medida Inercial (IMU) y un sistema de seguimiento nativo que estima en todo momento la posición (pose) de los visores y de los controladores con alta precisión. Cabe recalcar

que, gracias a esta capacidad, los controladores Touch Plus serán utilizados como objetos de referencia a lo largo de todo el proyecto. Además, el dispositivo tiene capacidad para almacenar datos en su memoria interna. Una vez obtenidos los datos deseados, estos se transferirán desde la memoria interna del dispositivo hacia un PC de procesamiento.



Figura 3-1: Dispositivo Meta Quest 3 [8]

3.1.1.2 Unity

Unity [40] es el motor de desarrollo empleado en la capa de preprocesado, la cual se introducirá en apartados posteriores (véase la Figura 3-2). Es una de las plataformas de desarrollo más maduras y adoptadas (como se muestra en la Figura 2-5 del apartado 2.2.3) para la realidad mixta. Además, dispone de soporte nativo para OpenXR (un estándar de código abierto que permite que una aplicación funcione en diversos dispositivos XR) y un ecosistema de *Software Development Kits* (SDKs) específicos para Meta, los cuales permiten desarrollar software para las Quest 3.

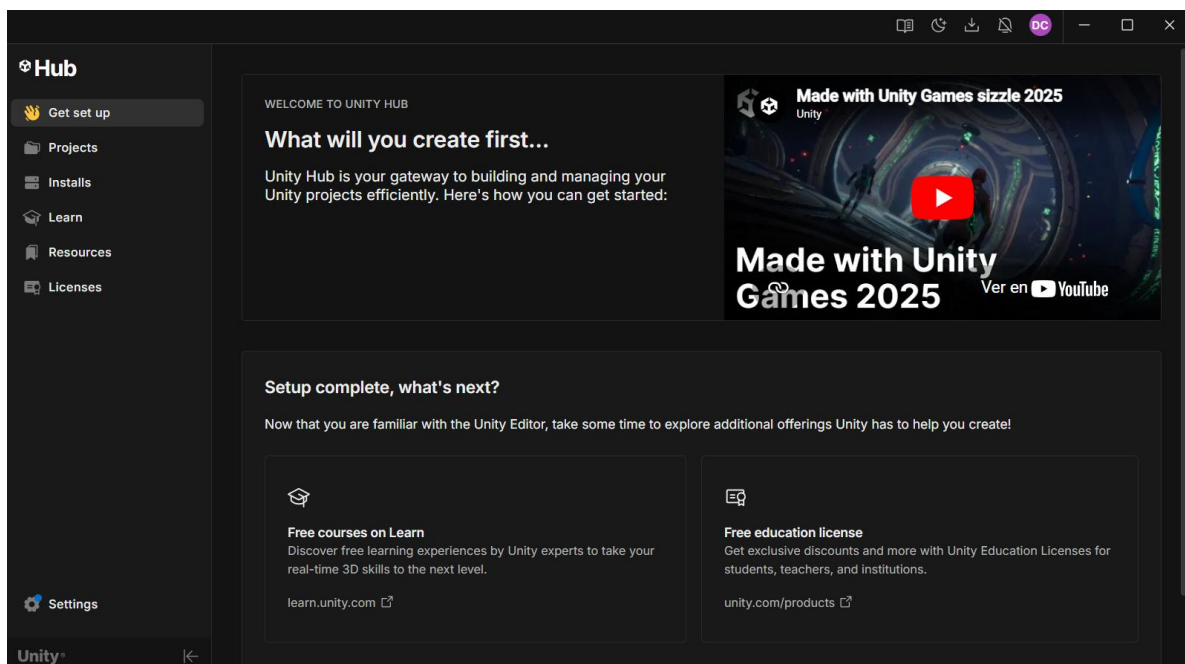


Figura 3-2: Interfaz de Unity

3.1.1.3 Lenguajes de programación

A continuación, se introducen los dos lenguajes de programación utilizados en este proyecto.

C# es el lenguaje nativo de Unity. En el contexto de este proyecto, los *scripts* C# implementan la lógica de captura de imágenes y su uso se focaliza íntegramente en la capa de adquisición de datos. Toda la cadena de procesamiento posterior se implementa en Python.

Python es el lenguaje empleado en la totalidad de la capa de postprocesado. Su elección se fundamenta en la madurez y amplitud del ecosistema de bibliotecas de visión artificial y computación científica disponibles. Las principales librerías utilizadas son:

- **OpenCV** (*Open Source Computer Vision Library*) para toda la cadena estereoscópica del sistema, empleándose en las fases de rectificación, estimación de disparidad y conversión a profundidad.
- **Open3D** para el procesamiento de nubes de puntos y para la estimación de pose.
- **NumPy** para el cómputo numérico matricial.

Las dependencias a estas librerías, y sus versiones utilizadas, vienen declaradas claramente en el documento “requirements.txt”.

Además, el entorno de desarrollo empleado es *Visual Studio Code* (VSCode). Esta herramienta permite organizar y modificar código de diferentes *scripts* de forma sencilla (véase la Figura 3-3). Además, para este proyecto se crea un entorno virtual gestionado mediante la carpeta “.venv”, el cual permite aislar la versión de Python utilizada y sus dependencias para un proyecto concreto.

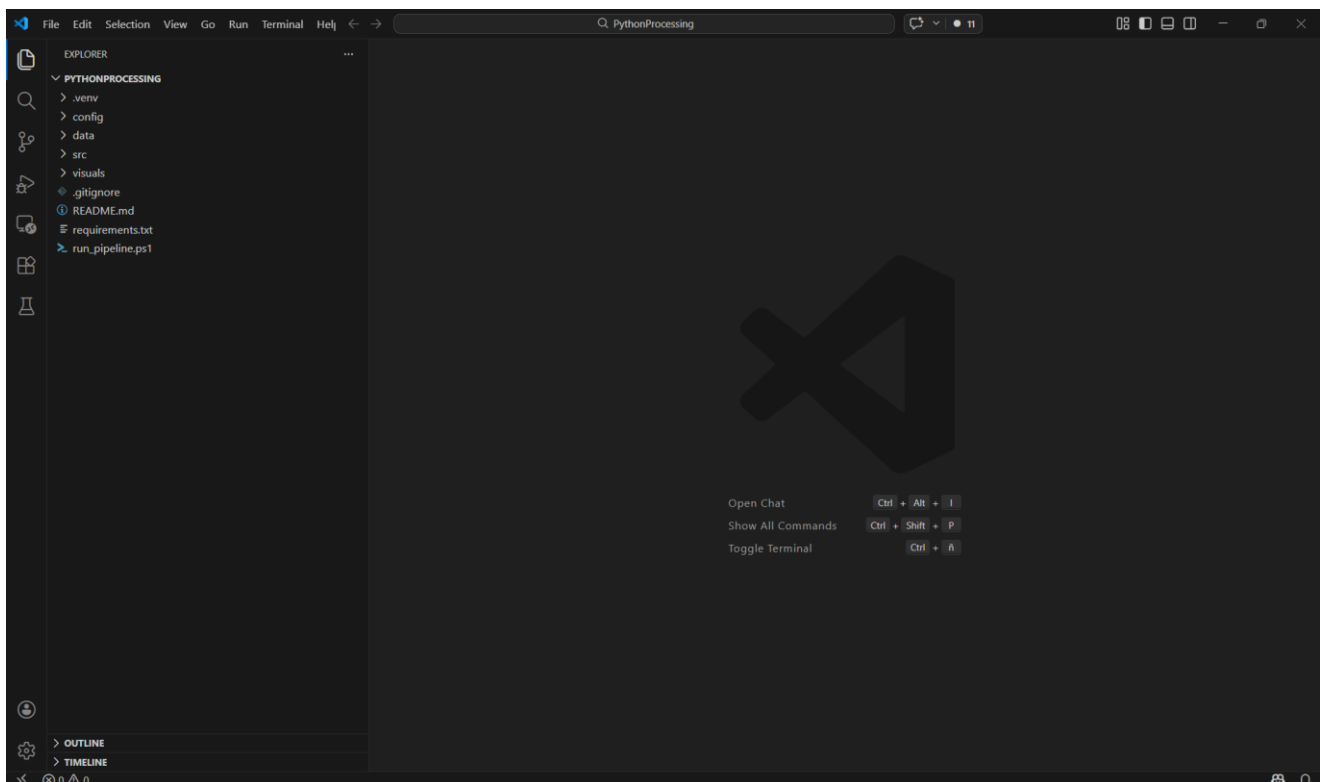


Figura 3-3: Interfaz de VSCode

3.1.1.4 CloudCompare

CloudCompare es un software de código abierto especializado en la visualización e inspección de nubes de puntos tridimensionales y mallas poligonales (véase la Figura 3-4). En el contexto de este proyecto, se emplea como herramienta de validación visual cualitativa a lo largo de la cadena de postprocesado, ya que permite inspeccionar las nubes de puntos generadas y verificar visualmente la coherencia de los resultados antes de proceder a la evaluación cuantitativa.

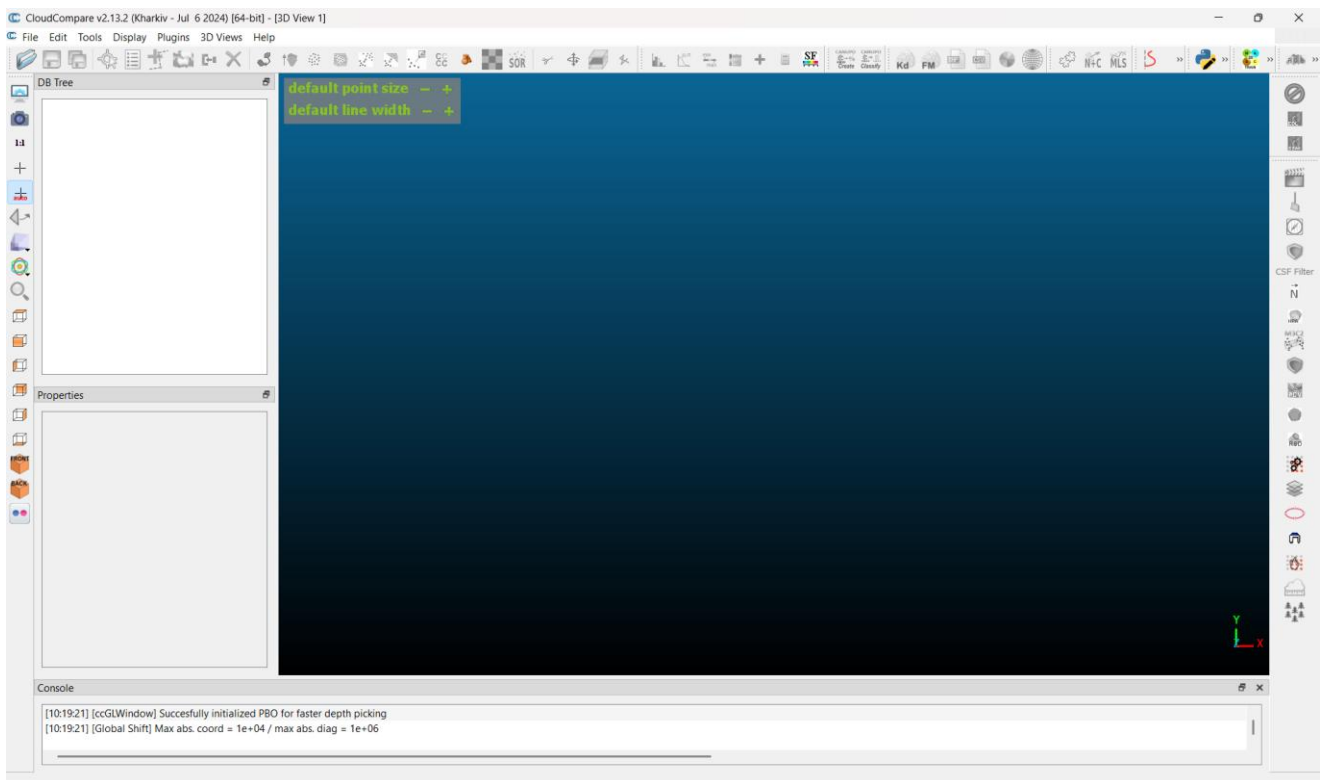


Figura 3-4: Interfaz de CloudCompare

Cabe aclarar que esta aplicación no forma parte del *pipeline* automatizado, sino que se utiliza de forma interactiva durante el desarrollo y la depuración del sistema, siendo especialmente útil para detectar artefactos de segmentación incorrectos o desalineamientos en la estimación de pose. Esta aplicación brinda apoyo directo para la evaluación de los resultados y complementa los indicadores numéricos de los archivos JSON generados automáticamente en el repositorio.

3.1.1.5 Blender

Blender es el software de modelado 3D empleado para la preparación del modelo CAD (véase la Figura 3-5). Esta plataforma permite obtener el modelo tridimensional de los controladores Touch Plus del dispositivo Meta Quest 3. Estos modelos se obtienen a partir de recursos encontrados en fuentes abiertas, los cuales requieren un proceso de acondicionamiento previo a su uso. Este acondicionamiento consiste en la eliminación de diversos elementos geométricos irrelevantes, la limpieza de la malla CAD generada y exportación en formato PLY compatible con Open3D. El archivo resultante constituye el modelo de referencia sobre el que se entrena el detector PPF y con el que se realiza el alineamiento ICP en la fase de estimación de pose. Estos conceptos serán abordados con mayor profundidad en apartados futuros.

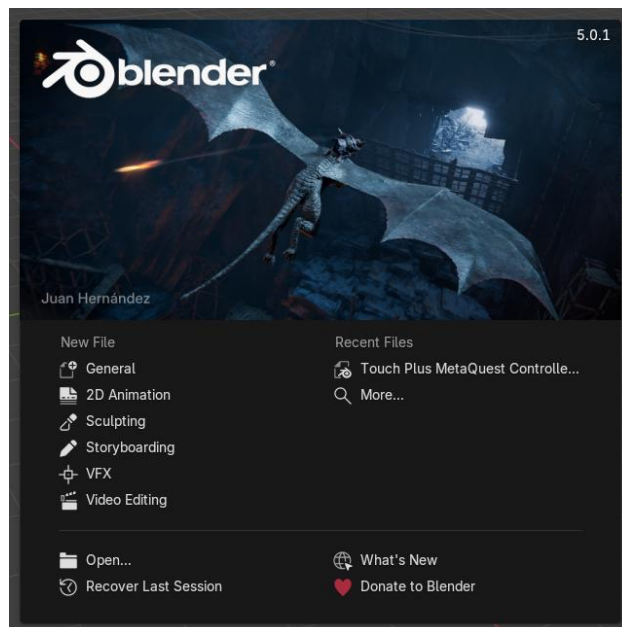


Figura 3-5: Interfaz de Blender

3.1.2 Estructura completa del sistema

El sistema desarrollado para este trabajo se organiza mediante una estructura (pipeline) secuencial de procesamiento que transforma pares de imágenes estéreo capturadas con la Meta Quest 3 en una estimación de pose 6D del controlador Touch Plus derecho, evaluando cuantitativamente la calidad de la estimación de pose respecto al *ground truth* proporcionado por el sistema de seguimiento nativo del dispositivo.

El *pipeline* se divide en dos grandes bloques funcionales: el preprocesado, que se ejecuta sobre el propio dispositivo de realidad mixta y genera los datos de entrada al sistema, y el postprocesado, que se ejecuta en un PC local y lleva a cabo la totalidad de la cadena de análisis. El punto de transferencia entre ambos bloques es la exportación del resultado generado en el preprocesado al repositorio de postprocesado. Este paso desacopla físicamente ambos bloques y permite operar sobre los datos de cualquier sesión de forma reproducible e independiente del dispositivo. Esta estructura se dibuja claramente en la Figura 3-6, la cual se presenta a continuación.



Figura 3-6: Diagrama de flujo. Bloques funcionales del sistema (Autoría propia)

Dentro del bloque de postprocesado, el *pipeline* sigue una arquitectura secuencial en la que la salida de cada etapa es la entrada de la siguiente. La trazabilidad de los datos se mantiene a nivel de sesión y de fotograma; es decir, cada sesión genera su propia carpeta de resultados y dentro de ella cada fotograma genera el conjunto completo de archivos intermedios correspondientes a las diferentes etapas de procesado. Esta secuenciación permite diagnosticar fallos en etapas concretas y proporciona la base de datos que alimenta la fase de evaluación.

La Figura 3-7 ilustra el sistema completo, diferenciando claramente entre las etapas de preprocesado y postprocesado, e indicando en cada bloque sus etapas y los archivos principales generados en cada una de ellas.

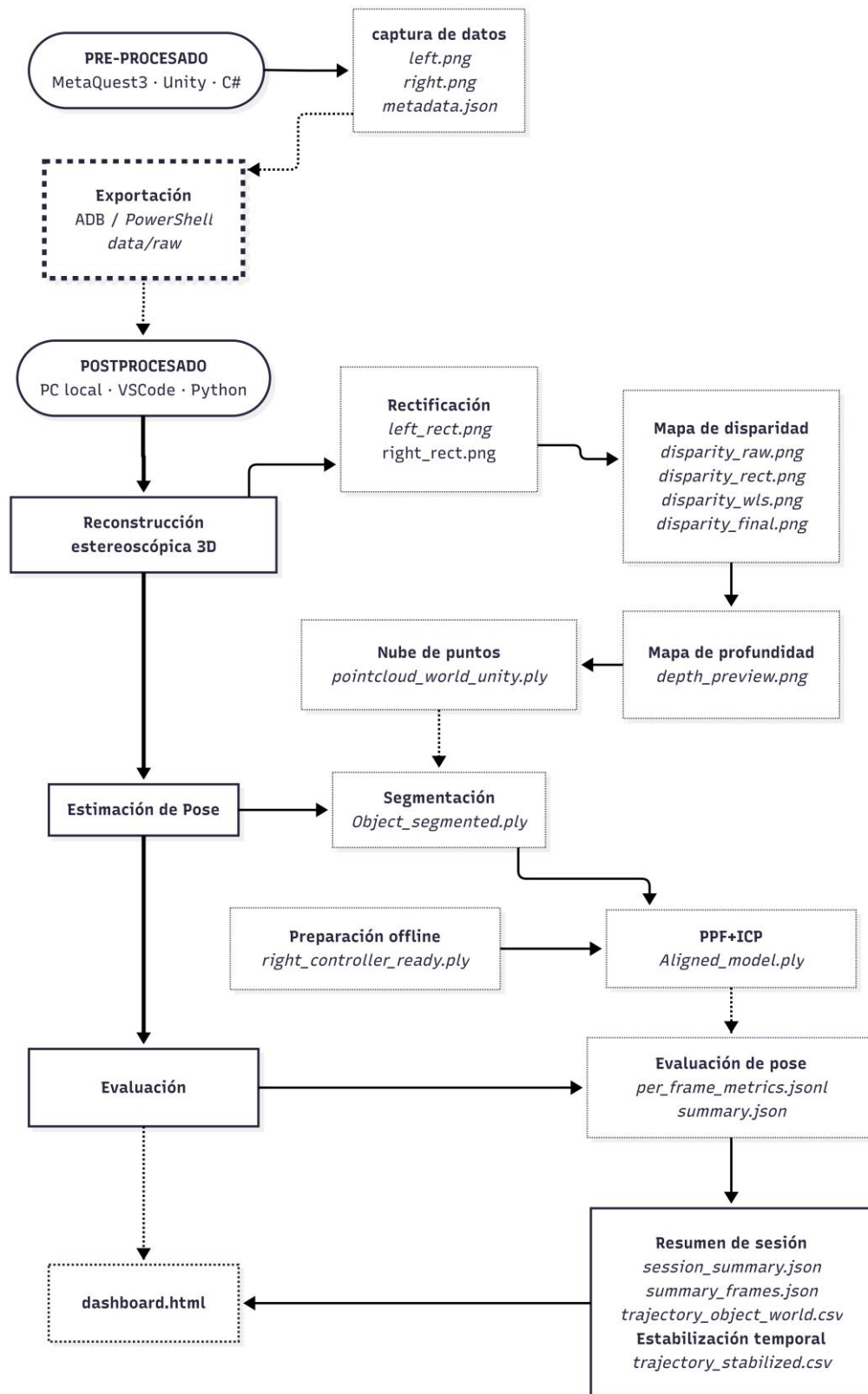


Figura 3-7: Diagrama de flujo. Estructura completa del sistema (Autoría propia)

A continuación, se ofrece una descripción funcional de cada etapa del sistema, que anticipa el contenido de las secciones siguientes del capítulo.

3.1.2.1 Fase 1: Adquisición de datos

El bloque de preprocesado, encuadrado en el proyecto *Unity_Quest3_PassthroughCamera* (véase el Anexo III: GitHub – Repositorios) se lleva a cabo de forma paralela a través de la aplicación Unity y el dispositivo Meta Quest 3. Para la adquisición de datos se accede a las cámaras del dispositivo a través de la Passthrough Camera API (una interfaz que permite acceder al vídeo que generan las cámaras externas de un dispositivo MR cualquiera) y se captura, de forma sincronizada, los fotogramas de los visores izquierdo y derecho junto con sus metadatos, organizados en una estructura de carpetas por sesión y por fotograma (véase la Figura 3-8).

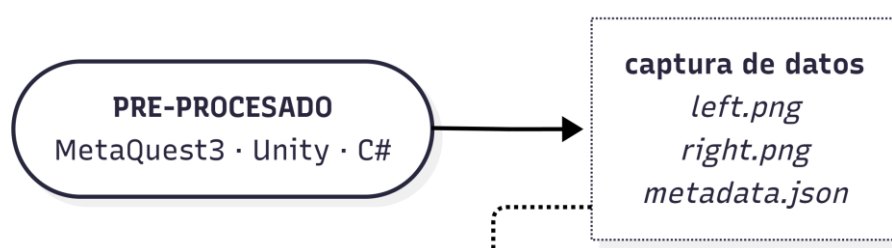


Figura 3-8: Fase 1 (Autoría propia)

3.1.2.2 Fase 2: Exportación, carga y validación de datos

Las carpetas de sesión se transfieren desde la memoria interna del dispositivo MR a un PC local mediante ADB (una herramienta de comandos que permite comunicarse con un dispositivo Android desde un ordenador) sobre PowerShell (el terminal por defecto de Windows), integrando los datos obtenidos al repositorio *PythonProcessing* (véase la Figura 3-9).



Figura 3-9: Fase 2 (Autoría propia)

3.1.2.3 Fase 3: Reconstrucción estereoscópica 3D.

En la fase de reconstrucción estereoscópica 3D se abordan 4 elementos claves:

- Proceso de rectificación estereoscópica
- Generación de mapas de disparidad
- Conversión de disparidad a profundidad
- Reconstrucción estereoscópica mediante nubes de puntos 3D

En primer lugar, a partir de los parámetros de calibración disponibles, se obtienen las imágenes rectificadas junto con diferentes métricas de validación epipolar.

A continuación, se aplica el algoritmo SGBM y el filtro WLS (referenciados en el apartado 2.4.2) sobre el resultado de la rectificación, generando una progresión de mapas intermedios en formato PNG que ilustran el progreso en materia de disparidad y que se complementan con métricas de densidad y distribución en diversos archivos JSON.

Después, se procede a convertir el mapa de disparidad filtrado en un mapa de profundidad métrico, acompañado de un archivo con indicadores de la calidad del mapa resultante.

Finalmente, se convierten los píxeles válidos del mapa de profundidad en una nube de puntos 3D, obteniendo como resultado un archivo que manifiesta la reconstrucción estereoscópica lograda, al cual lo acompañan sus respectivas métricas evaluables (véase la Figura 3-10).

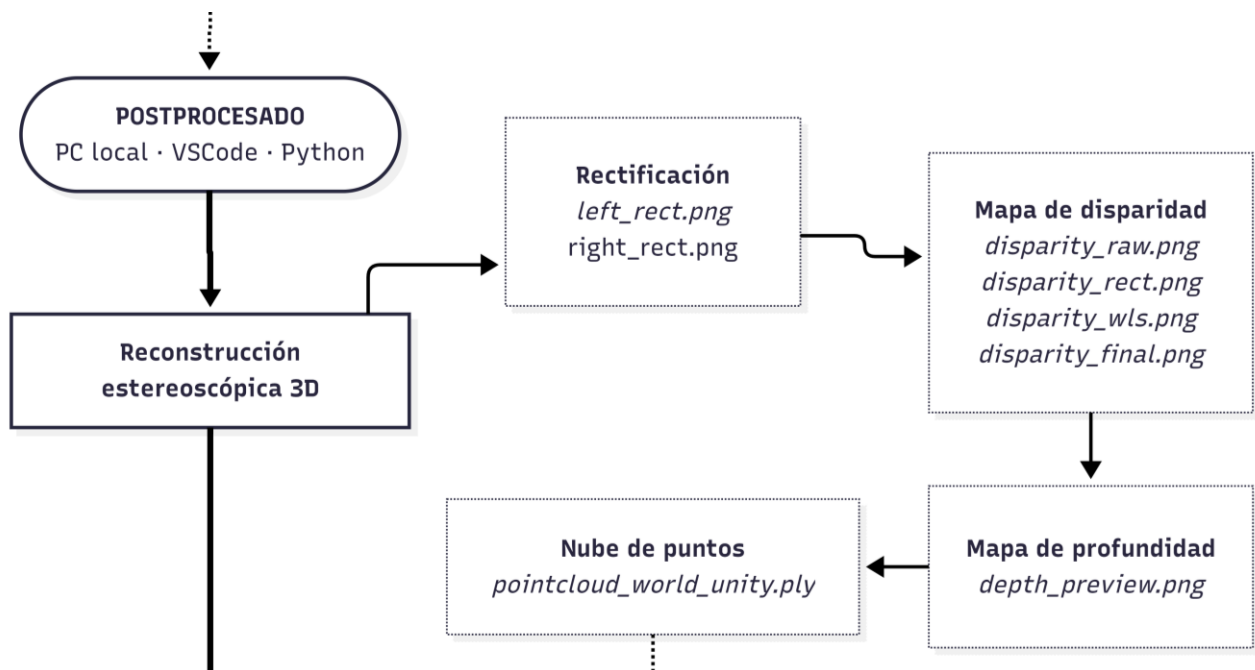


Figura 3-10: Fase 3 (Autoría propia)

3.1.2.4 Fase 4: Estimación de pose

En esta fase se atiende específicamente al proceso de estimación de pose y se abordan 3 elementos esenciales:

- Generación del modelo CAD del controlador derecho TouchPlus (objeto del que se desea estimar su pose)
- Llevar a cabo la segmentación del objeto a partir de la nube de puntos 3D
- Combinación de PPF e ICP para estimar la pose del objeto

En primer lugar, se convierte el modelo CAD del controlador derecho TouchPlus (obtenido de fuentes abiertas) en un archivo PLY, el cual supone la base para la estimación de pose.

A continuación, se extrae el objeto de interés de la nube de puntos de la escena, generando una nube de puntos única en la que se localiza el objeto. La segmentación del objeto viene acompañada de un archivo que contiene diversos indicadores de calidad. Finalmente, se implementa la estimación de pose del objeto.

De forma resumida, se generan hipótesis globales de pose, las cuales se refinan para obtener la mejor hipótesis hasta que esta converja en el objeto buscado con la mayor precisión posible. El modelo alineado se guarda en un archivo PLY y los resultados completos de la sesión se almacenan en un archivo JSON (véase la Figura 3-11).

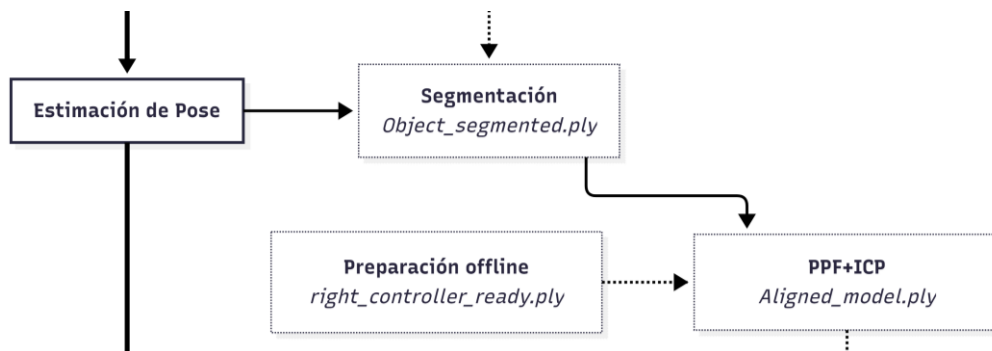


Figura 3-11: Fase 4 (Autoría propia)

3.1.2.5 Fase 5: Evaluación

La última fase del sistema, centrada en la evaluación de los resultados, obtiene principalmente 3 tipos de resultados:

- Resultados específicos en materia de estimación de pose
- Resultados genéricos de la sesión
- “dashboard.html” interactivo para visualización de la arquitectura del proyecto y de los resultados obtenidos en la sesión objetivo

La pose estimada del objeto durante la sesión genera una trayectoria que se recoge en el archivo CSV. En primer lugar, se compara la pose estimada del objeto con el *ground truth* que aporta el dispositivo Quest 3 y se calculan los errores de traslación y rotación en cada fotograma. Los resultados se guardan en diversos archivos JSON y JSONL, generándose además para el conjunto de la sesión otros archivos que constituyen el soporte cuantitativo del análisis de resultados (véase la Figura 3-12).

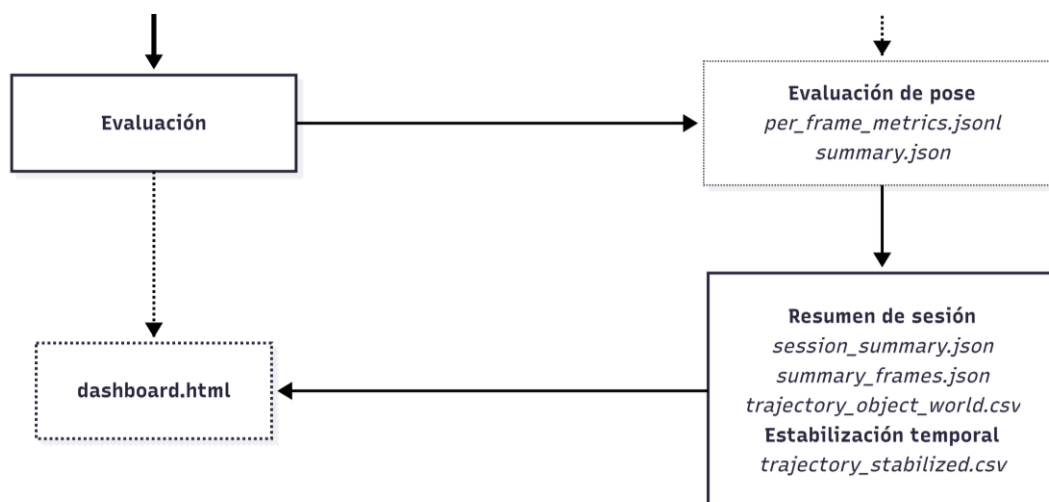


Figura 3-12: Fase 5 (Autoría propia)

Tras haber abordado de manera conceptual y esquemática las diferentes fases del sistema, las siguientes secciones de este capítulo desarrollan en detalle cada una de estas fases recientemente introducidas. Para abordarlas, se seguirá estrictamente el camino que recorren los datos a lo largo del sistema y se describirá para cada una de las etapas existentes sus fundamentos algorítmicos inherentes, las decisiones de parametrización adoptadas (cuando corresponda) y los resultados obtenidos que sean imprescindibles para comprender el desarrollo del proyecto.

3.2 Fase 1: Adquisición de datos

El objetivo de esta fase es la obtención de imágenes estéreo de ambas cámaras del dispositivo Meta Quest 3 junto con sus metadatos geométricos asociados, de forma estructurada y reproducible.

El resultado de esta fase constituye la entrada del *pipeline* de postprocesado: una carpeta de sesión que contiene, para cada fotograma capturado, las imágenes “left.png” y “right.png” y un archivo “metadata.json” con los parámetros intrínsecos de ambas cámaras, las poses de las cámaras y de los controladores, y los parámetros de tiempo correspondientes.

3.2.1 Meta Quest 3 y Passthrough Camera API

El dispositivo Meta Quest 3, introducido en el apartado 3.1.1.1, tiene la funcionalidad Passthrough Camera API (*Application Programming Interface*), la cual se obtiene de componente XR SDK (*Software Development Kit*) de Meta.

Para entenderlo de forma sencilla, Passthrough Camera API garantiza el acceso a las cámaras físicas del visor desde Unity. Al mismo tiempo, el SDK [41] proporciona en tiempo real la posición real de los controladores físicos a través del sistema de seguimiento nativo del visor. Así, para cada par de imágenes capturadas, la API expone:

- La textura de las cámaras, obtenidas a través de “PassthroughCameraAccess.GetTexture()”
- La pose de las cámaras, obtenidas mediante “GetCameraPose()”
- La pose de los controladores, obtenidas mediante “GetControllerPose()”
- Los parámetros intrínsecos del sensor, obtenidos mediante “CameraIntrinsics”. Estos parámetros incluyen las longitudes focales (f_x, f_y), los puntos principales (c_x, c_y) y la resolución de cada uno de los visores.

La documentación de referencia de *PassthroughCameraAccess* y de la API están disponibles en el portal de desarrolladores de Meta [42]. Además, cabe recalcar que la disponibilidad simultánea de imágenes estéreo calibradas y la posición de los controladores en el mismo dispositivo hace de las Quest 3 una plataforma adecuada para el diseño experimental de este proyecto, ya que permite obtener los datos de entrada del *pipeline* sin necesidad de instrumentación adicional y habilita directamente la evaluación de los resultados obtenidos.

Se introducen a continuación las aplicaciones Meta Horizon Link y Meta Quest Developer Hub (MQDH), necesarias para trabajar con el dispositivo Meta Quest 3. Las interfaces de estas aplicaciones se muestran en la Figura 3-13 y en la Figura 3-14.

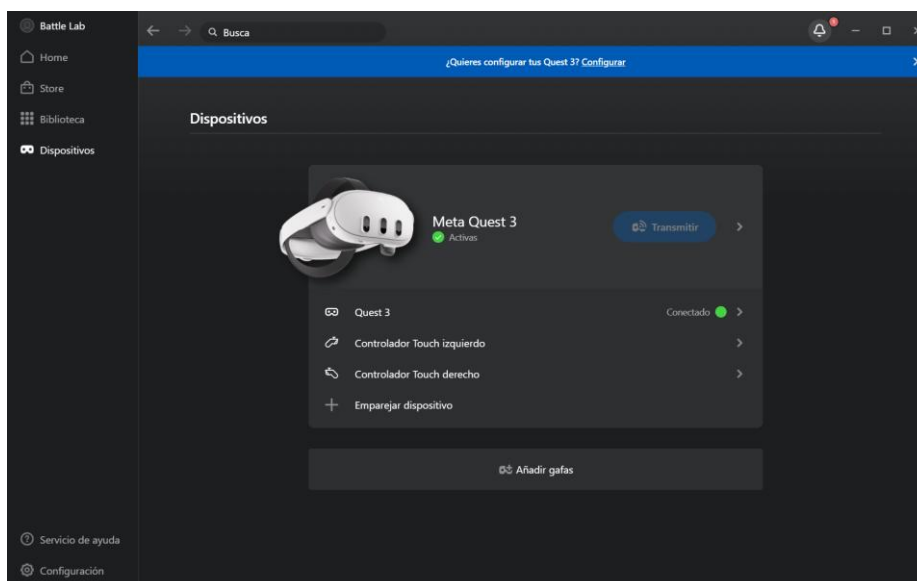


Figura 3-13: Interfaz de la aplicación Meta Horizon Link

Meta Horizon Link es la aplicación de escritorio que establece la conexión entre el PC de desarrollo y el visor Quest 3 mediante cable USB o red Wi-Fi. Permite ejecutar y depurar aplicaciones Unity directamente en el dispositivo sin necesidad de compilar y desplegar un APK (*Android Package Kit*) completo, lo que agiliza el trabajo durante el desarrollo.

MQDH, por su parte, complementa este entorno ofreciendo una interfaz gráfica para la gestión del dispositivo (véase la Figura 3-14) mediante acceso al sistema de archivos internos, permitiendo visualizar el proyecto creado, la extracción de datos almacenados en la memoria del dispositivo y la visualización de capturas de pantalla.

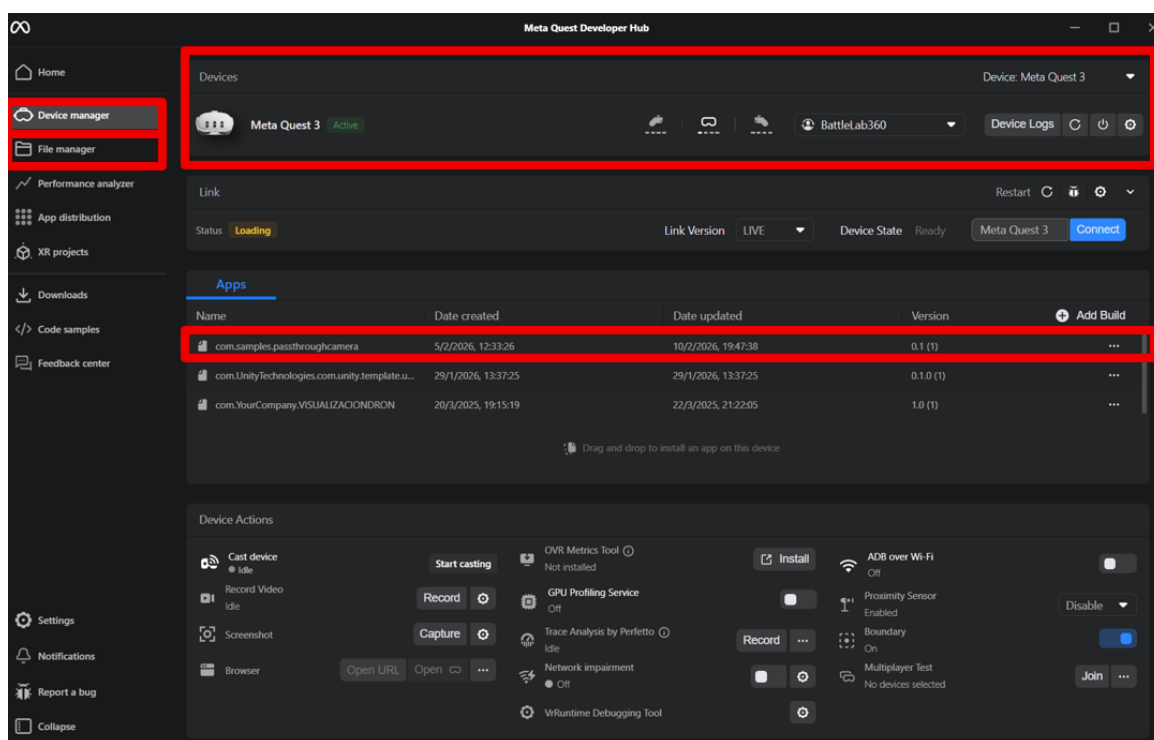


Figura 3-14: Interfaz de la aplicación MQDH

3.2.2 Entorno Unity y configuración del proyecto

La puesta a punto del entorno de desarrollo se realiza siguiendo paso a paso la guía oficial de Meta para proyectos de realidad mixta en Unity [41].

Para este proyecto, el flujo de configuración parte de Unity Hub (aplicación que gestiona todo lo relacionado con el motor de desarrollo Unity), desde donde se selecciona la versión del editor a emplear (véase la Figura 3-15) y desde donde se crea el proyecto (véase la Figura 3-16).

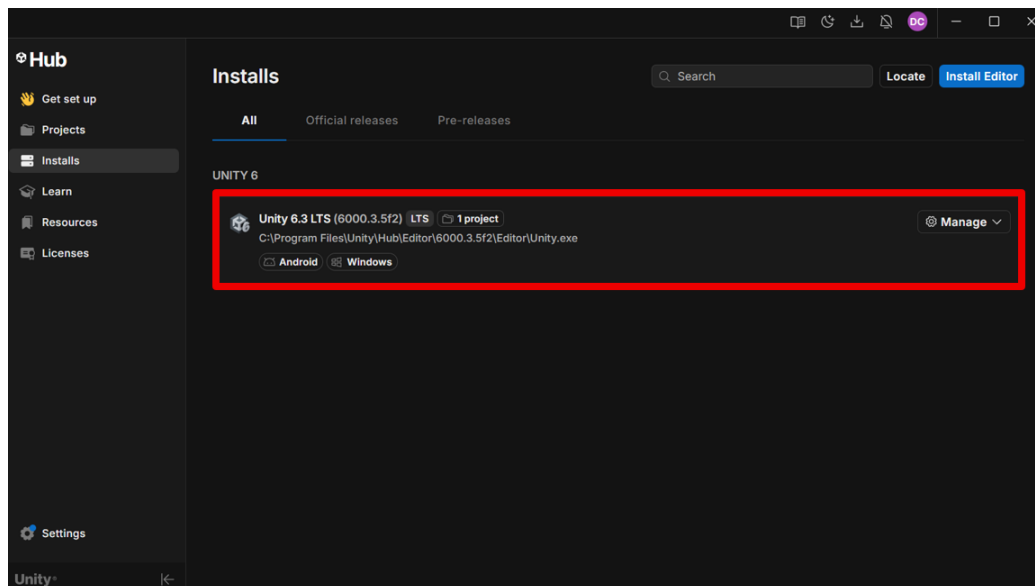


Figura 3-15: Versión del editor instalada en Unity Hub

El proyecto, denominado localmente *Unity_Quest3_PassthroughCamera* (véase el Anexo III: GitHub – Repositorios), tiene su origen en el repositorio oficial *Unity-PassthroughCameraAPISamples* de *oculus-samples* [43]. Se elige este repositorio como base y punto de partida pues proporciona una implementación funcional y validada de la Passthrough Camera API para el motor de Unity. Recordemos que este elemento proporciona los fotogramas de las cámaras izquierda y derecha junto con sus metadatos de calibración.

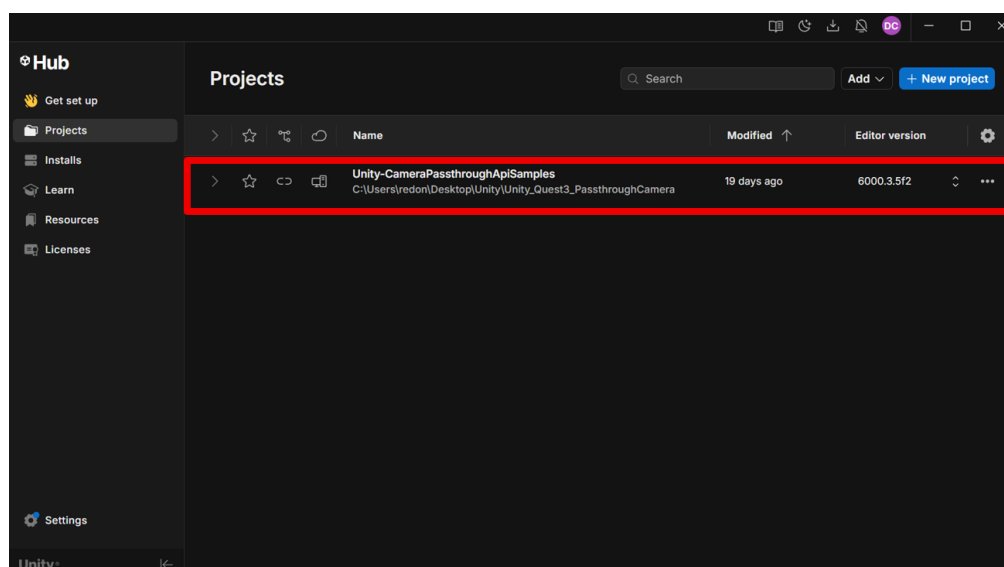


Figura 3-16: Proyecto Unity-CameraPassthroughApiSamples

Además, la guía de Meta señala todos los elementos y configuraciones de OpenXR necesarias, así como la integración del Meta XR SDK, lo que facilita enormemente la creación del proyecto propio. Así, en “Project Settings” se debe configurar “XR Plug-in Management” (véase la Figura 3-17), habilitando OpenXR para la plataforma en cuestión.

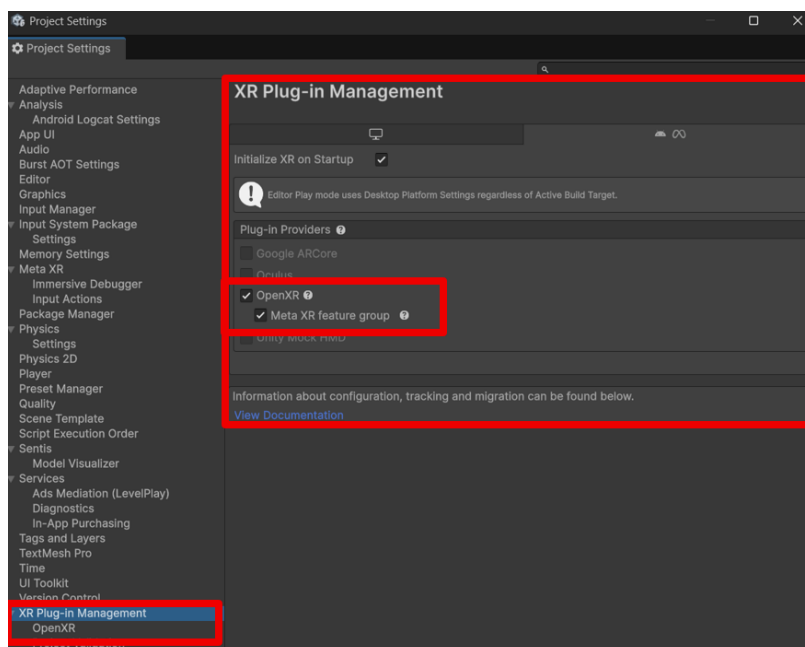


Figura 3-17: Acceso a *XR Plug-in-Management* desde “Project Settings”

También se debe habilitar en la pestaña OpenXR el conjunto de *features* (extensiones o paquetes) de Meta XR deseados. Esto permite, junto con lo reseñado anteriormente, el acceso a las cámaras mediante el *passthrough*. Finalmente, se integran los paquetes del Meta XR SDK a través del “Package Manager”, y la herramienta “Project Setup Tool” de Meta (véase la Figura 3-18) automatiza la detección y corrección de los ajustes de proyecto requeridos.

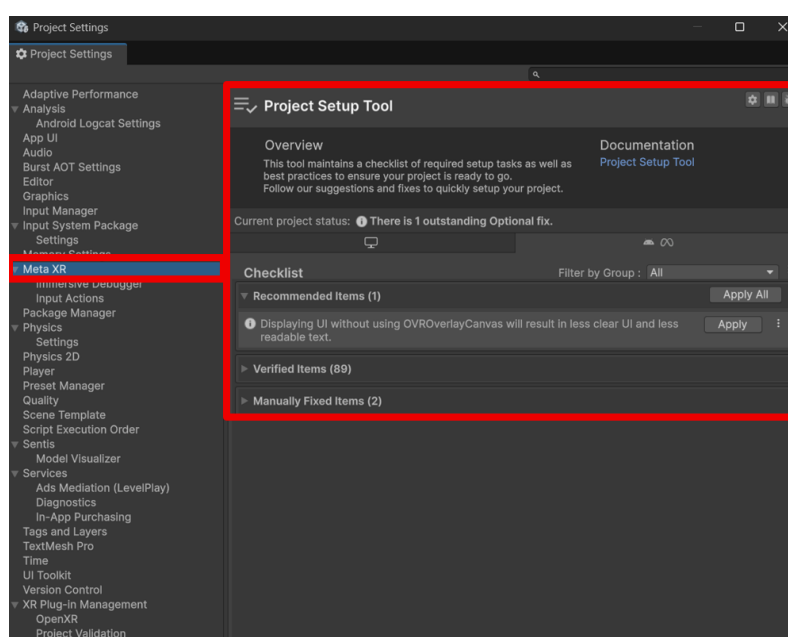


Figura 3-18: Herramienta *Project Setup Tool*

3.2.3 Escena, jerarquía y sistema de referencia

Tras verificar el funcionamiento del repositorio base, se eliminan las muestras no pertinentes para el proyecto (“CameraToWorld”, “BrightnessEstimation”, “MultiObjectDetection” y “ShaderSample”) y se crea la escena “CaptureAndExport”, aprovechando la infraestructura de acceso a cámara ya presente en el proyecto.

Así, la escena creada (“CaptureAndExport”) constituye el entorno de ejecución de la aplicación en el dispositivo, la cual se puede apreciar en la Figura 3-19. En dicha imagen encontramos diferentes paneles:

- *Hierarchy*, desde donde se gestionan los diversos *GameObjects* (elementos) de la escena.
- *Scene*, que permite la visualización virtual de la escena.
- *Project*, donde se encuentran las carpetas principales “Assets” y “Packages”, que a su vez contienen sus propias subcarpetas con diversos archivos necesarios para completar el proyecto.
- *Console*, que funciona como elemento para la detección de errores.
- *Inspector*, que permite analizar la composición de cada elemento.

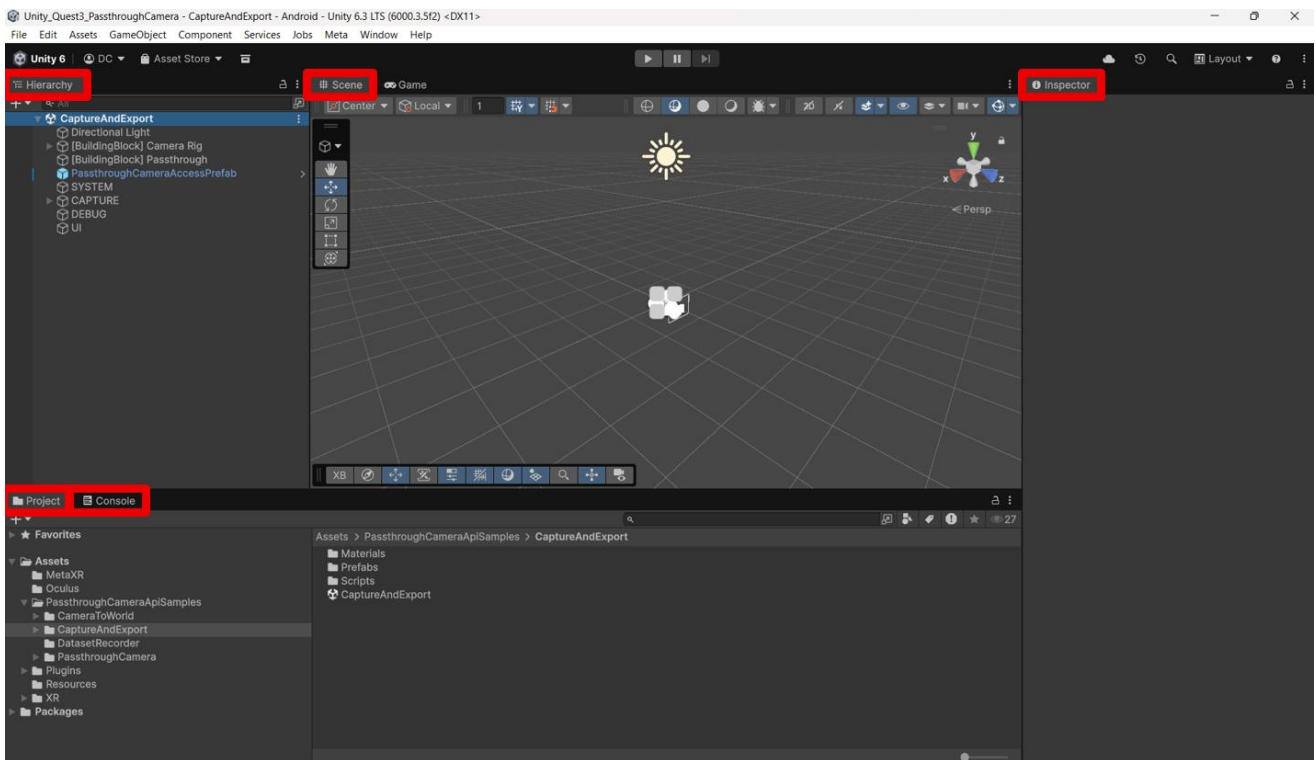


Figura 3-19: Escena *CaptureAndExport* (Autoría propia)

Abordando ahora la propia escena, esta presenta una jerarquía que se articula en torno al “OVRCameraRig” (véase la Figura 3-20), la plantilla reutilizable (*prefab*) central del Meta XR SDK, la cual actúa como raíz del sistema en Unity. De ello depende el *TrackingSpace*, que define el origen del espacio de seguimiento del dispositivo, y de éste cuelgan los puntos de referencia (*anchors*) de las cámaras (“LeftEyeAnchor”, “RightEyeAnchor”, “CenterEyeAnchor”) y de los controladores (“LeftControllerAnchor”, “RightControllerAnchor”).

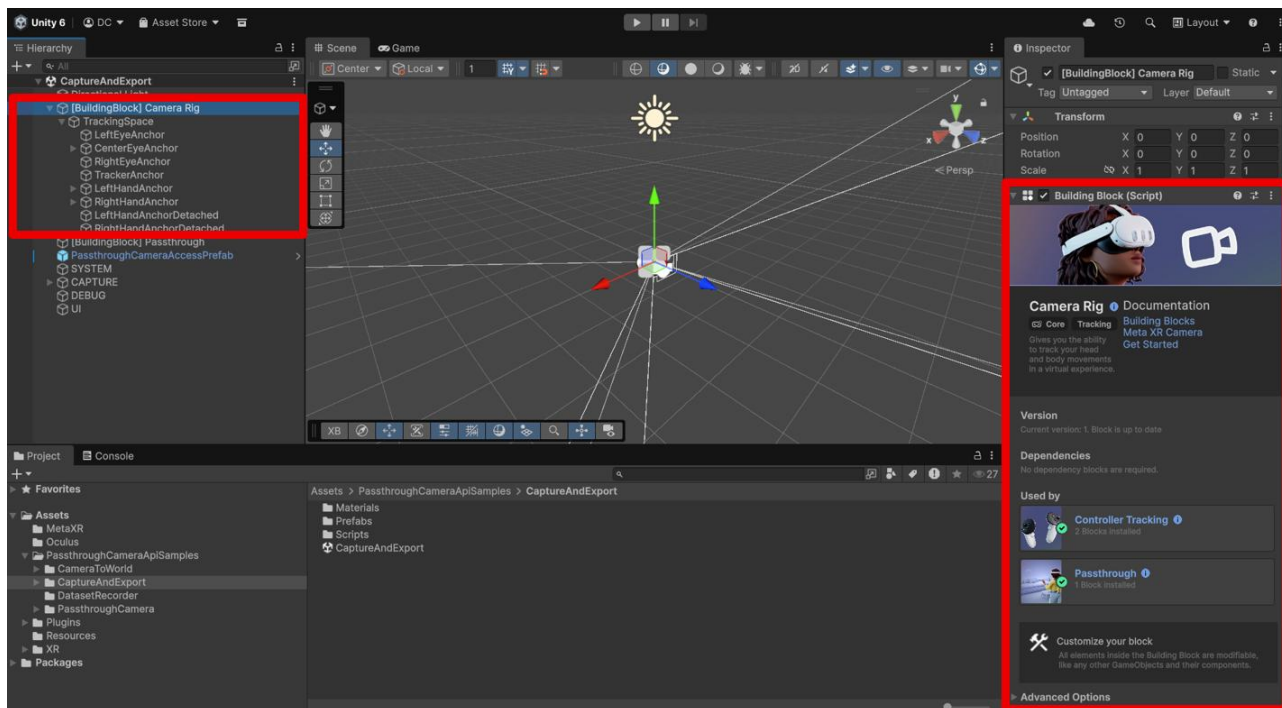


Figura 3-20: *OVRCameraRig*

Los componentes del *passthrough* se asocian a los *anchors* de las cámaras izquierda y derecha respectivamente, y los *scripts* de captura y exportación se alojan en *GameObjects* propios conectados entre sí a través del inspector (véase la Figura 3-21).

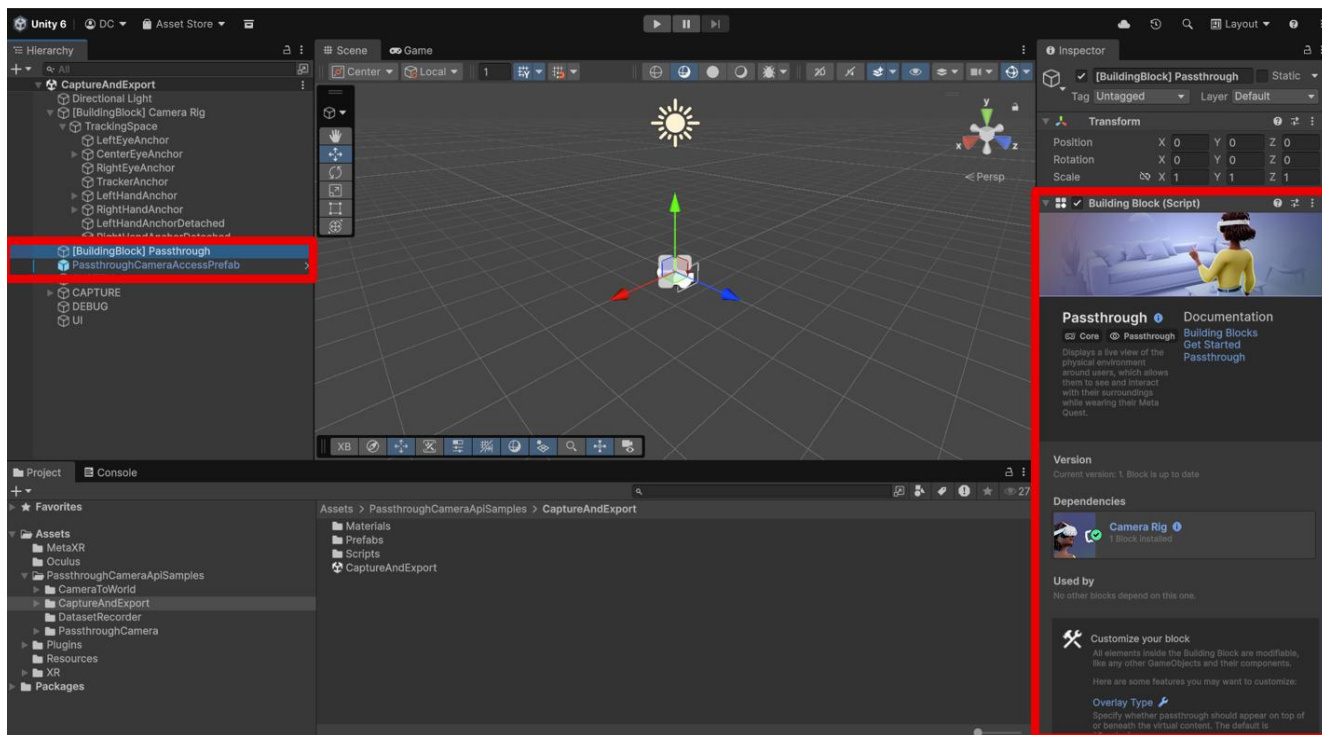


Figura 3-21: *PassthroughCameraAccess*

En resumen, la escena desarrollada gestiona mediante sus diferentes elementos la captura sincronizada de ambas cámaras, la construcción de la estructura de carpetas de sesión en la memoria

interna del dispositivo y la exportación de los archivos “left.png”, “right.png” y “metadata.json” por cada fotograma capturado.

Llegados a este punto y antes de avanzar con los *scripts* que logran tal efecto (materia del próximo subapartado), se debe destacar el aspecto geométricamente más relevante de la escena: la definición del sistema de referencia. Todas las poses exportadas en el archivo “metadata.json” de cada fotograma (la pose de la cámara izquierda, la pose de la cámara derecha, la pose del controlador izquierdo y la pose del controlador derecho) se expresan en un mismo sistema de coordenadas. Este sistema de referencia se conoce como *TrackingSpace* y se puede definir como el punto $(0,0,0)$ en el mundo real que corresponde al origen $(0,0,0)$ en el mundo de Unity (marco *World*). Esta coherencia la garantiza “FileDataExporter.cs” mediante la función “ToLocalPose()”, la cual transforma cada pose desde el espacio mundo al espacio local del *TrackingSpace*. El resultado es que todas las entidades del sistema (cámaras y mandos) comparten un origen común, el cual se establece en el centro del propio dispositivo al iniciar la captura de datos de una sesión determinada (véase la Figura 3-22).



Figura 3-22: Sistema de referencia *TrackingSpace* (Generado con IAG)

3.2.4 Captura y exportación de fotogramas

La lógica de adquisición de datos se distribuye entre cinco scripts C# (véase la Figura 3-23) organizados según el principio de separación de responsabilidades. Es decir, cada uno de ellos ejerce una función determinada para conseguir el objetivo del proyecto.

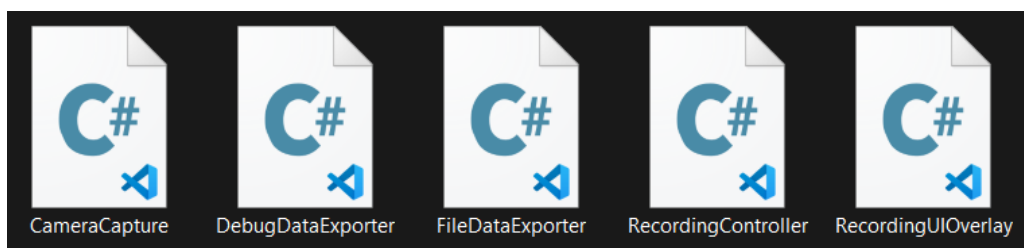


Figura 3-23: *Scripts* asociados a la escena (Autoría propia)

“CameraCapture.cs” es el componente central de captura, el cual se encuentra referenciado dentro del *GameObject* “CAPTURE” (Véase la Figura 3-24). En cada ciclo “Update()” se verifica que ambas

instancias de “*PassthroughCameraAccess*” estén activas para obtener las texturas de cada cámara (mediante “*GetTexture()*”), su pose (mediante “*GetCameraPose()*”) y los intrínsecos del sensor. Las texturas se exportan a PNG y los intrínsecos se obtienen una sola vez, guardándolos como válidos para todos los fotogramas. Además, se obtienen los *timestamps* por cámara (parámetro de tiempo desde que inicia una sesión determinada) mediante “*GetFrameTimestamp*”.

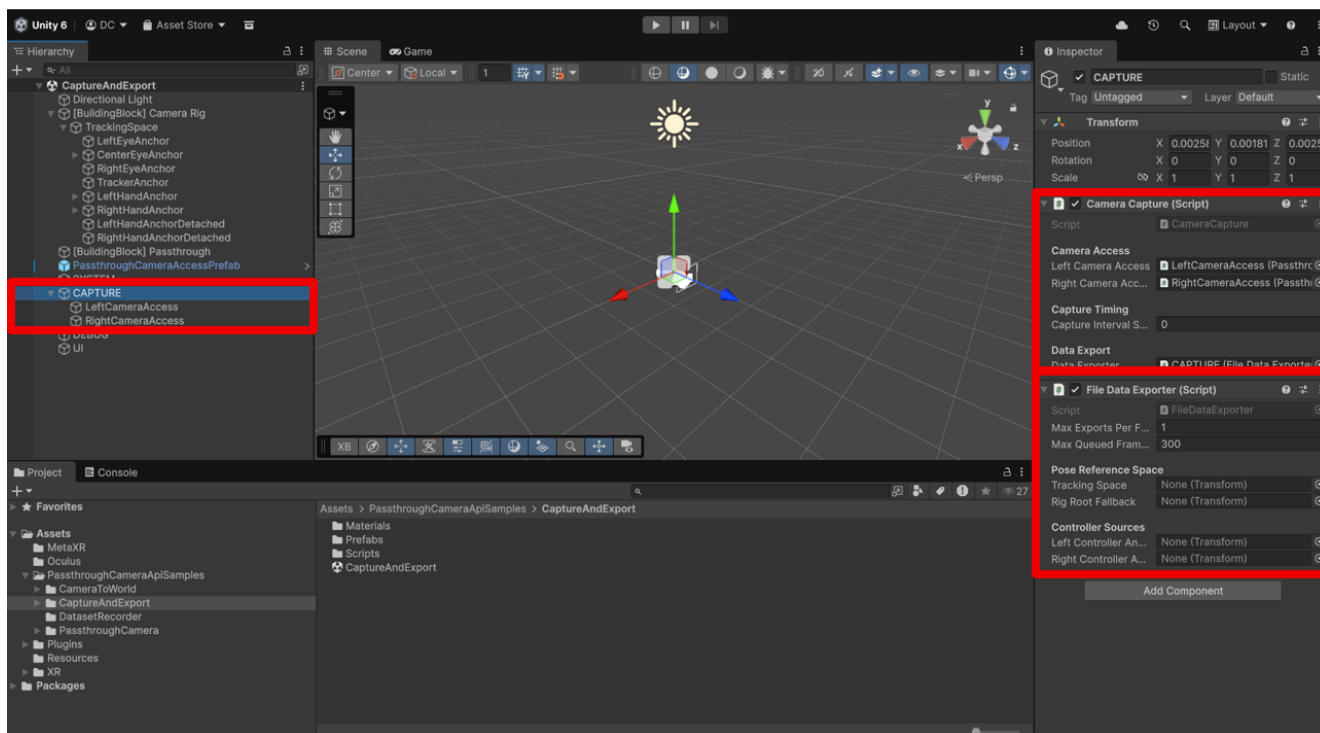


Figura 3-24: Elemento CAPTURE (Autoría propia)

“*FileDataExporter.cs*”, *referenciado también* en el elemento “CAPTURE”, gestiona la escritura en disco. Al inicializarse, crea una carpeta de sesión con nombre “*session_YYYYMMDD_HHMMSS/*” en el directorio “*Application.persistentDataPath/Exporteddata/*”. Los fotogramas se reciben en una cola interna (“*Queue<Queuedframe>*”) y se escriben en dicha carpeta mediante “*Update()*” (hasta “*m_maxExportsPerframe*” por ciclo, para evitar bloqueos por sesiones muy largas). Así, se genera una subcarpeta “*frame_NNNNNN*” para cada uno de los instantes capturados. Finalmente, este *script* guarda dentro de cada una de estas carpetas los archivos “*left.png*” y “*right.png*” y “*metadata.json*”. Este último documento recoge los siguientes parámetros:

- Índice de fotograma
- *Timestamp* global y *timestamps* individuales de cada cámara
- Dimensiones de textura
- Parámetros intrínsecos de ambas cámaras (*fx*, *fy*, *cx*, *cy*, resolución)
- Poses de cámara izquierda, cámara derecha, controlador izquierdo y controlador derecho

Para gestionar e independizar las sesiones existe el *GameObject* “SYSTEM” (véase la Figura 3-25) que controla, mediante “*RecordingController.cs*”, el inicio y la parada de la grabación a través de los botones del mando físico (con “*OVRInput.Button.One*” para iniciar y “*OVRInput.Button.Two*” para detener la grabación), activando o desactivando *EnableExport* de *FileDataExporter*.

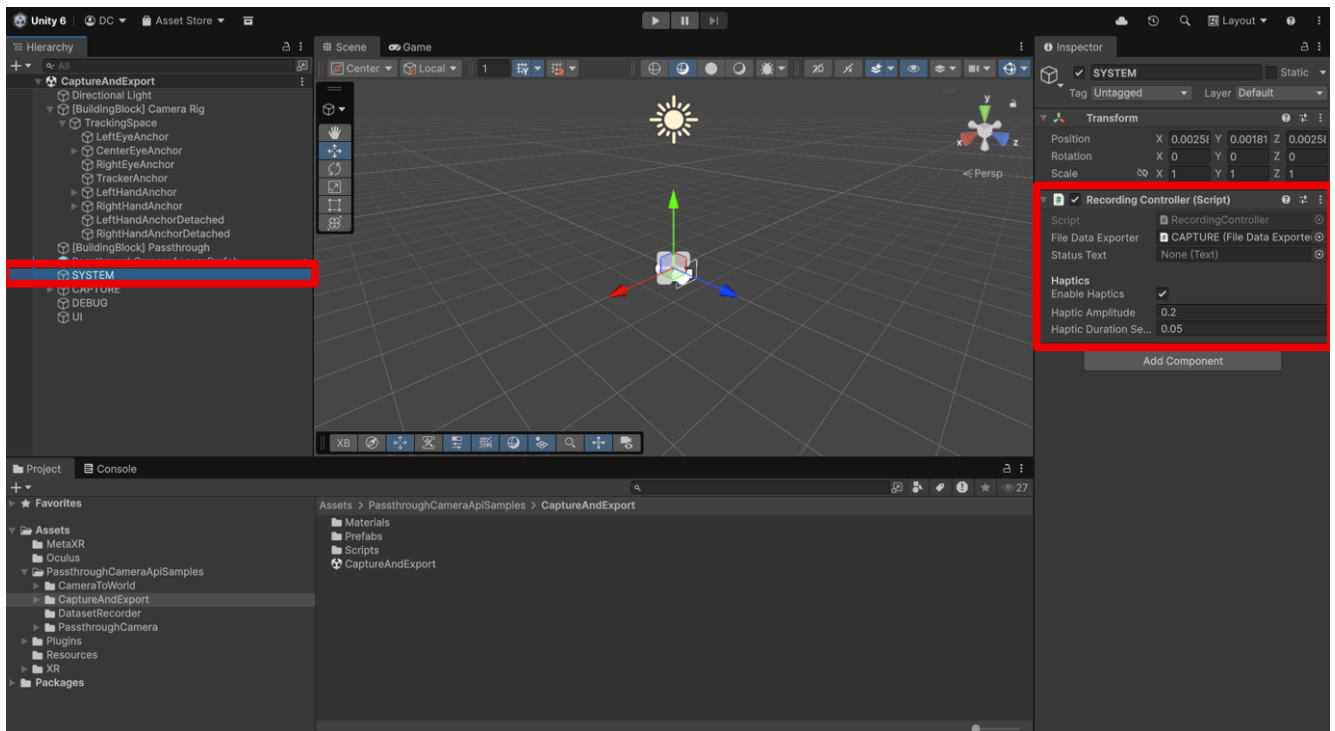


Figura 3-25: Elemento SYSTEM (Autoría propia)

Además, el elemento “UI” mediante “RecordingUIOverlay.cs” (véase la Figura 3-26) presenta un elemento gráfico (*canvas*) visible durante la sesión de captura. Mediante este elemento gráfico se introduce el proyecto y se muestran las instrucciones básicas de uso de la aplicación para grabar una sesión de forma exitosa (véase Figura 3-27).

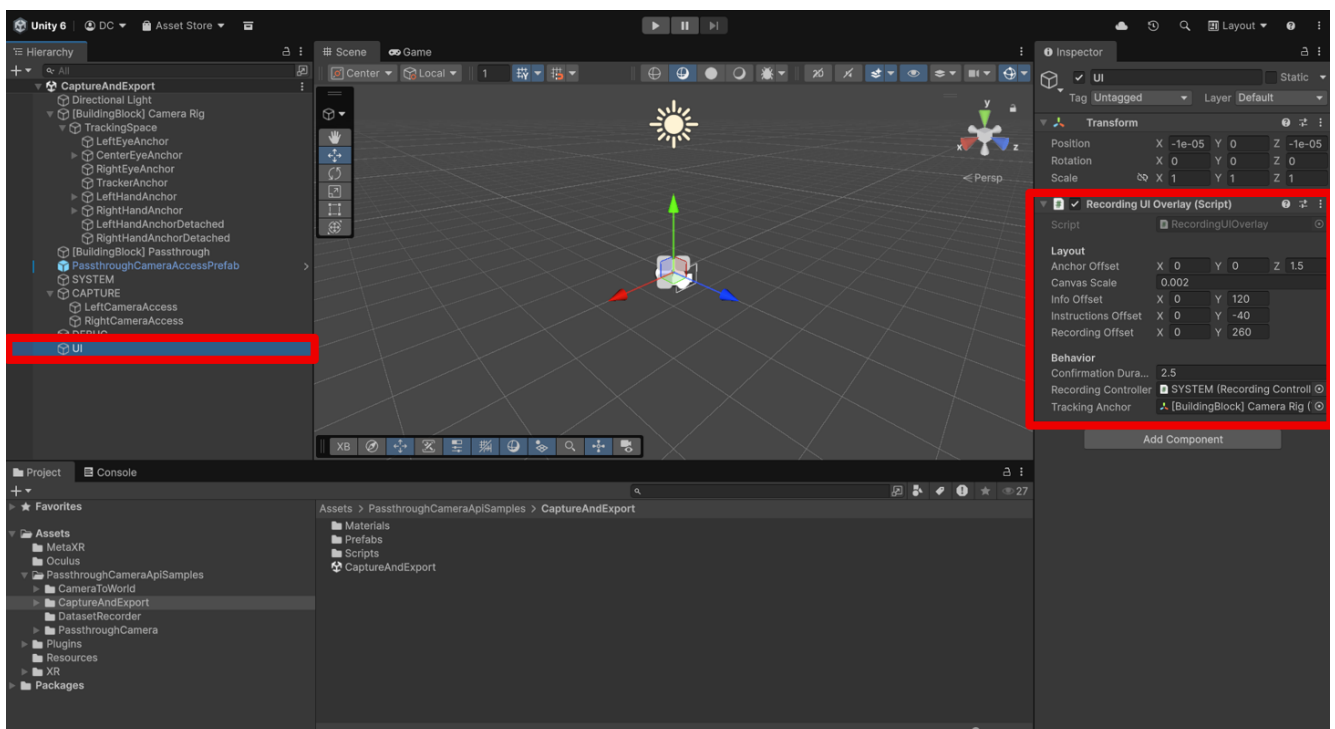


Figura 3-26: Elemento UI (Autoría propia)



Figura 3-27: Visualización del *canvas* (Autoría propia)

Finalmente, el elemento “DEBUG” que emplea el *script* “DebugDataExporter.cs” sirve para verificar que la API esté entregando datos válidos. Este es un paso previo a la propia activación de la escritura en disco y evita errores de guardado en la memoria interna del dispositivo.

Dándole la profundidad que se merece, el resultado de esta fase es una carpeta de sesión almacenada en la memoria interna del dispositivo, organizada de la siguiente forma:

- “*session_YYYYMMDD_HHMMSS/*”
- “*frame_000000*” → “left.png”, “right.png”, “metadata.json”
- “*frame_000001*” → “left.png”, “right.png”, “metadata.json”
- ...
- “*frame_NNNNNN*” → “left.png”, “right.png”, “metadata.json” (estos tres archivos se muestran en la Figura 3-28)

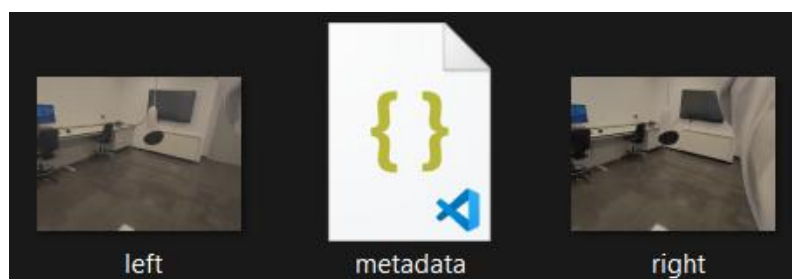


Figura 3-28: Archivos obtenidos en un fotograma cualquiera (Autoría propia)

Esta estructura es el contrato de datos entre los bloques existentes en este sistema (preprocesado y postprocesado). Así, todo el código Python de las fases siguientes asume exactamente esta organización para localizar y cargar cada fotograma de sesión.

3.3 Fase 2: Exportación, carga y validación de datos

Una vez finalizada la fase de adquisición de datos, el siguiente paso consiste en transferir las sesiones capturadas al entorno de procesamiento offline en el PC local e integrarlas en el repositorio de postprocesado introduciendo una validación previa al inicio del *pipeline*.

Esta fase actúa como bisagra entre el bloque de preprocesado (C# en Unity) y el bloque de postprocesado (Python en VSCode), garantizando que los datos que entran al repositorio son coherentes y están completos antes de ejecutar algoritmos de visión artificial.

3.3.1 Transferencia de datos desde el dispositivo al PC

Meta Quest 3 ejecuta un sistema operativo basado en Android. Los datos de sesión generados por Unity se almacenan en la memoria interna del dispositivo, en la ruta (1):

```
/sdcard/Android/data/com.samples.PassthroughCamera/files/Exporteddata/.
```

 (1)

La transferencia al PC se realiza mediante ADB (*Android Debug Bridge*), la herramienta estándar del ecosistema Android para comunicación entre dispositivo y ordenador a través de USB. El proceso se ejecuta desde PowerShell en el PC local siguiendo la secuencia de comandos que se presenta a continuación.

3.3.1.1 Verificación de la conexión ADB

Se comienza verificando la conexión del dispositivo Quest3 mediante ADB escribiendo en PowerShell el comando (2):

```
>> adb devices
```

 (2)

Este comando lista los dispositivos Android reconocidos. Debe aparecer el identificador del Quest 3 con estado “device” para que la transferencia sea posible.

3.3.1.2 Listado de sesiones disponibles en el dispositivo

Para comprobar las sesiones disponibles en el dispositivo se ejecutan en orden los comandos (3), (4) y (5):

```
>> adb shell
```

 (3)

```
>> "ls /sdcard/Android/data/com.samples.PassthroughCamera/files/  
ExportedData"
```

 (4)

```
>> exit
```

 (5)

Con esto se accede al intérprete (*shell*), el cual proporciona una lista de las sesiones existentes, seleccionando la sesión de interés (formato: “*session_YYYYMMDD_HHMMSS*”).

3.3.1.3 Extracción de la sesión al PC

Una vez comprobadas las sesiones disponibles, se procede a extraer la sesión de interés mediante el comando (6):

```
>> adb pull "/sdcard/Android/data/com.samples.PassthroughCamera/files/  
Exporteddata/session_YYYYMMDD_HHMMSS" "C:\QuestExports\  
session_YYYYMMDD_HHMMSS"
```

 (6)

El comando *adb pull* copia la carpeta de sesión completa desde el dispositivo hasta la carpeta “C:\QuestExports\”, la cual debe ser creada previamente en el PC local. Esta carpeta actúa como zona intermedia entre dispositivo y repositorio.

3.3.1.4 Verificación de la copia

Es posible verificar desde el propio terminal que la estructura de carpetas y archivos se ha transferido correctamente al PC local antes de proceder a la integración de los archivos en el repositorio. En ese caso, se debe proceder mediante el comando (7):

```
>> dir "C:\QuestExports\session_YYYYMMDD_HHMMSS" (7)
```

3.3.1.5 Integración en el repositorio

Finalmente, mediante *robocopy* se copia la sesión (incluyendo todas las subcarpetas y archivos) hasta su ubicación definitiva en “data/raw/” dentro del repositorio de postprocesado, la cual se detalla en el apartado 3.3.2. Así, el comando a ejecutar es (8):

```
>> robocopy "C:\QuestExports\session_YYYYMMDD_HHMMSS" "<PROJECT_DIR>\data\  
raw\session_YYYYMMDD_HHMMSS" /E (8)
```

3.3.2 Estructura del repositorio de postprocesado

El repositorio de postprocesado, denominado *PythonProcessing* (véase el Anexo III: GitHub – Repositorios), contiene la totalidad del código Python del proyecto, los datos de entrada y salida de cada uno de los scripts y los archivos de configuración necesarios para ejecutar el *pipeline*.

Su organización interna responde a un principio fundamental: la separación estricta entre datos y código. Esta decisión permite modificar los algoritmos sin alterar los datos originales y viceversa, permitiendo incorporar nuevas sesiones al repositorio sin que esto afecte al código fuente ni los archivos previamente existentes.

La estructura de primer nivel del repositorio se compone de los siguientes elementos (véase la Figura 3-29).

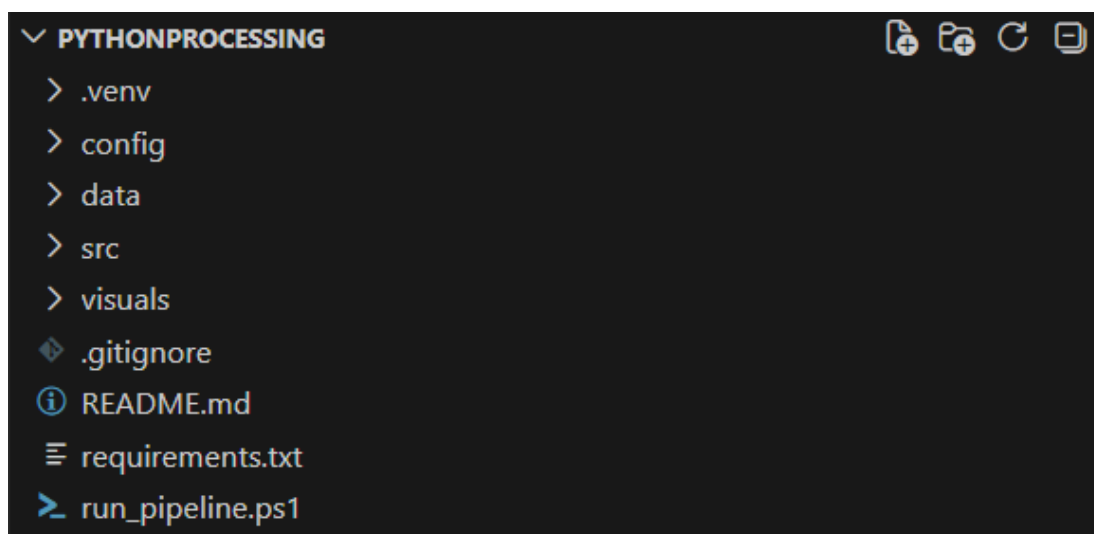


Figura 3-29: Estructura de primer nivel del repositorio *PythonProcessing* (Autoría propia)

3.3.2.1 La carpeta “data/”

La carpeta “data/” centraliza todos los datos del proyecto. Dentro de ella existen tres subcarpetas, mostradas en la Figura 3-30, las cuales se abordan a continuación

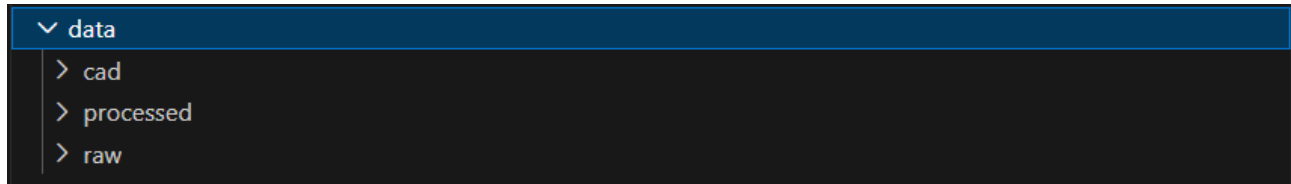


Figura 3-30: Estructura de la carpeta *data* (Autoría propia)

La carpeta “cad/” contiene el modelo CAD del controlador *Meta Quest Touch Plus* en formato Blender (BLEND). En esta carpeta se generan también los archivos PLY del modelo acondicionado, los descriptores PPF precomputados y todas las métricas asociadas a estos en formato JSON.

La carpeta “raw/” almacena las sesiones tal y como llegan del dispositivo, sin modificación alguna. Cada sesión ocupa una subcarpeta con la nomenclatura “*session_YYYYMMDD_HHMMSS*” y dentro de ella existe una carpeta por cada fotograma capturado (“*frame_NNNNNN*”), como muestra la Figura 3-31. Cada carpeta de fotograma dispone, a su vez, de los archivos “*left.png*”, “*right.png*” y “*metadata.json*” (como muestra la Figura 3-28). Esta carpeta es, por tanto, el destino final de la transferencia descrita en el apartado anterior.

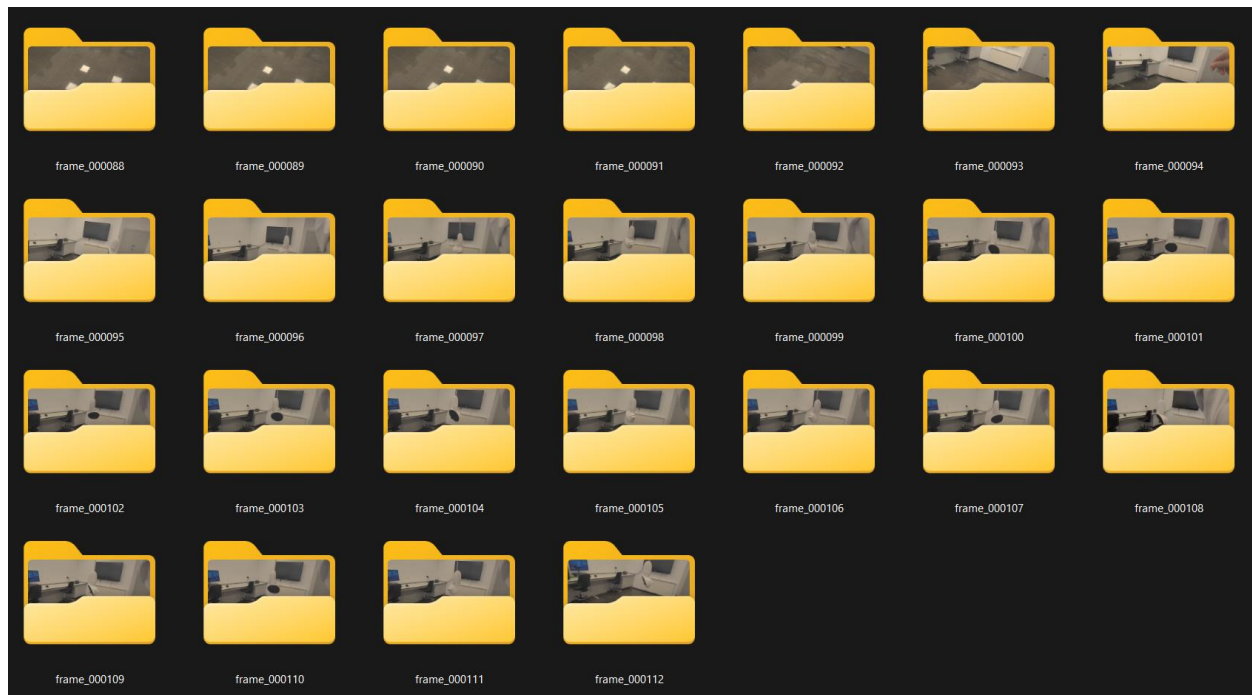


Figura 3-31: Estructura de la carpeta “raw/” (Autoría propia)

La carpeta “processed/” replica de forma idéntica la estructura de sesiones y fotogramas de la carpeta “raw/”. En esta carpeta se guardan todos los archivos generados por el *pipeline* una vez se ejecuta sobre una sesión cualquiera. Así, para cada fotograma procesado, se genera un conjunto extenso de resultados intermedios y finales, incluyendo archivos PNG, JSON y PLY, los cuales se resumen en la Tabla 3-1.

Categoría	Archivos	Descripción
Imágenes rectificadas	left_rect.png right_rect.png	Par estéreo rectificado
Mapas de disparidad	disparity_raw.png disparity_rect.png disparity_rect_wls.png disparity_final.png	Progresión del mapa de disparidad
Mapa de profundidad	depth_preview.png	Profundidad métrica y previsualización
Nubes de puntos	pointcloud_world_unity.ply pointcloud_camera_rect.ply	Escena completa en coordenadas mundo y cámara
Segmentación	object_roi_raw.ply object_segmented.ply object_segmented_normals.ply	Etapas de aislamiento del controlador
Estimación de pose	ppf_match/aligned_model.ply	Modelo alineado y transformación estimada
Métricas JSON	disparity_metrics.json depth_metrics.json pointcloud_metrics.json segmentation_metrics.json ppf_match/pose_best.json pose_eval.json	Métricas cuantitativas de cada etapa
Configuración	sgbm_preset.json rectification_debug.json	Parámetros y datos de depuración

Tabla 3-1: Contenido de la carpeta “processed/” por fotograma

Esta separación entre “raw/” y “processed/” permite conservar intactos los datos originales del dispositivo y reproducir el procesado completo en cualquier momento, garantizando la trazabilidad del *pipeline*.

Para facilitar la comprensión de esta estructura, la Figura 3-32 muestra el mapa de datos del repositorio. Dicho mapa recoge, mediante un evidente código de colores, los diferentes tipos de archivos generados en las carpetas “cad/”, “raw/” y “processed/”. Este mapa se autogenera en un *dashboard* interactivo, el cual se presenta en la sección 4 de esta memoria. Para un fácil acceso a dicho documento, se proporciona también un código QR que permite visualizar este mapa si se accede a la pestaña “Arquitectura”.

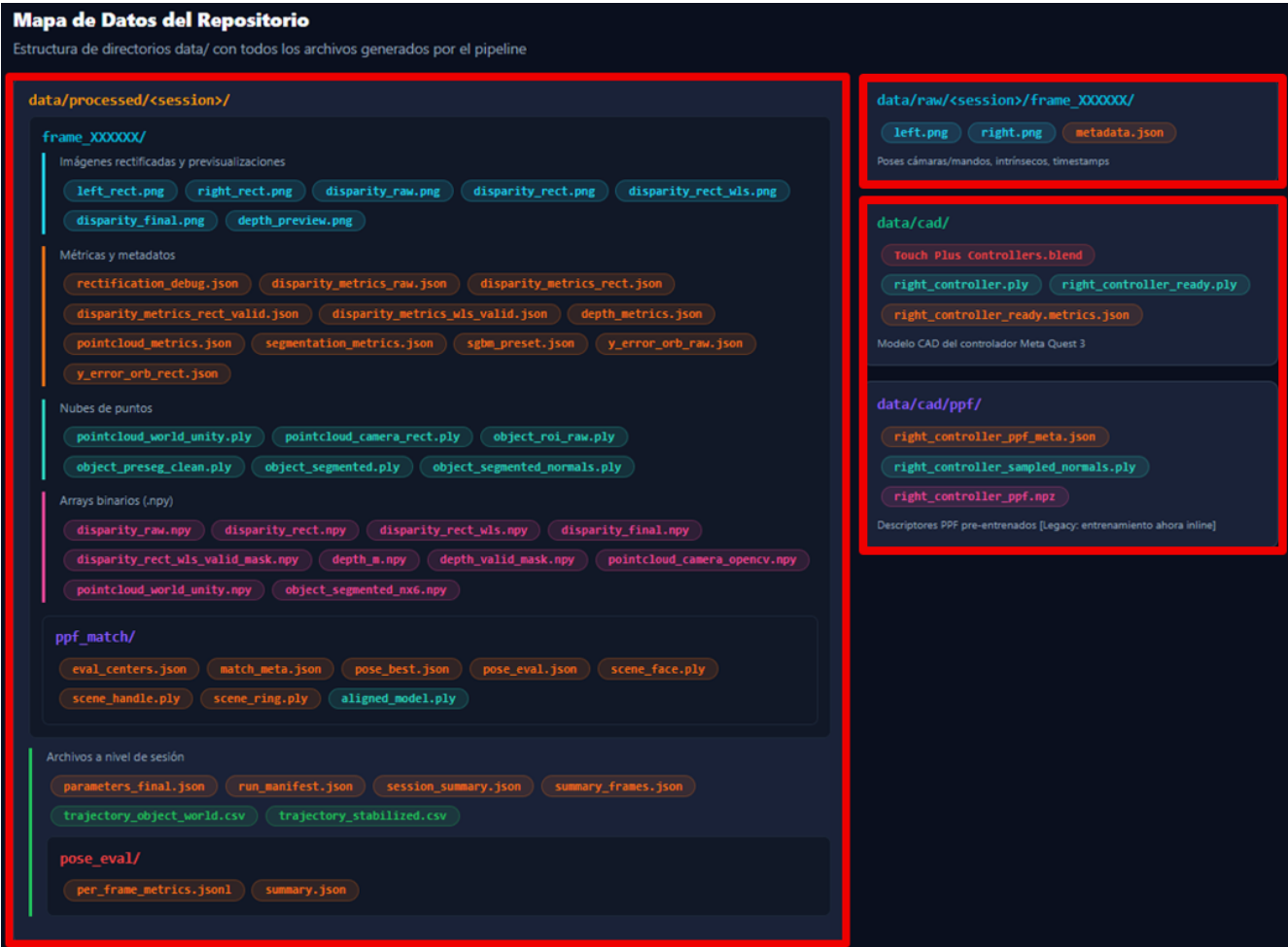


Figura 3-32: Estructura de la carpeta “processed/” (Autoría propia)

3.3.2.2 La carpeta “src/”

La carpeta “src/” contiene el código fuente organizado en dos subcarpetas: “core/” y “pipeline/”. Esta organización refleja una separación deliberada entre la lógica algorítmica y la orquestación de la ejecución, lo que facilita tanto el mantenimiento como la comprensión del repositorio.

La subcarpeta “core/” agrupa los módulos que implementan la lógica de cada etapa del *pipeline*. La Tabla 3-2 recoge estos módulos junto con una breve descripción de la funcionalidad que implementan.

Módulo	Descripción
<code>data_loader.py</code>	Carga y validación de datos de entrada (imágenes, metadatos e intrínsecos)
<code>transforms.py</code>	Matemática SE(3)/SO(3) centralizada (conversión de poses y composición de transformaciones)
<code>io_utils.py</code>	Lectura y escritura unificada de archivos (JSON, NPY, PLY)
<code>reporting.py</code>	Exportación de resultados en formatos JSON, JSONL y CSV
<code>rectification.py</code>	Rectificación estereoscópica del par de imágenes

disparity.py	Cálculo del mapa de disparidad mediante SGBM y filtro WLS
depth.py	Estimación de profundidad métrica a partir de la disparidad
pointcloud.py	Generación de nubes de puntos 3D por retroproyección
cad_model.py	Preparación del modelo CAD
segmentation.py	Segmentación geométrica de la nube de puntos 3D para aislar el objeto
ppf_match.py	<i>Matching</i> PPF+ICP para estimación de pose 6D (<i>Surface Matching</i>)
pose_eval.py	Evaluación de la estimación de pose frente al <i>ground truth</i>
frame_status.py	Determinación de la validez de cada fotograma según umbrales configurables
temporal_stabilizer.py	Estabilización temporal para rechazo de saltos implausibles entre fotogramas

Tabla 3-2: Módulos de “src/core/”

La subcarpeta “*pipeline/*” contiene los *scripts* de ejecución de cada fase, que importan los módulos de “core/” y orquestan el procesado fotograma a fotograma para una sesión completa. La Tabla 3-3 describe estos *scripts*.

<i>Script</i>	Descripción
run_session_pipeline.py	Orquestador principal. Ejecuta el <i>pipeline</i> completo para una sesión.
run_disparity.py	Ejecuta el cálculo de disparidad para un fotograma.
run_depth.py	Ejecuta la conversión de disparidad a profundidad para un fotograma.
run_pointcloud.py	Ejecuta la generación de la nube de puntos 3D para un fotograma.
run_cad_model.py	Ejecuta la preparación del modelo CAD (una sola vez por modelo).
run_segmentation.py	Ejecuta la segmentación del objeto en la nube de puntos de un fotograma.
run_ppf_match.py	Ejecuta el <i>matching</i> PPF+ICP para estimar la pose en un fotograma.
run_pose_eval.py	Ejecuta la evaluación de la pose estimada frente al <i>ground truth</i> .
run_session_summary.py	Genera los archivos de resumen, trayectoria y estadísticas de sesión.

Tabla 3-3: Scripts de “src/pipeline/”

Esta separación entre lógica algorítmica (“core/”) y orquestación (“*pipeline/*”) permite, por ejemplo, modificar cualquier algoritmo sin alterar la secuencia de ejecución, o ejecutar cualquier etapa de forma aislada para su depuración y mejora. La Figura 3-33 muestra visualmente esta organización por fases

funcionales. Dicha figura pertenece también al *dashboard* interactivo previamente mencionado y que se puede consultar en la sección 4 mediante el código QR previamente mencionado.

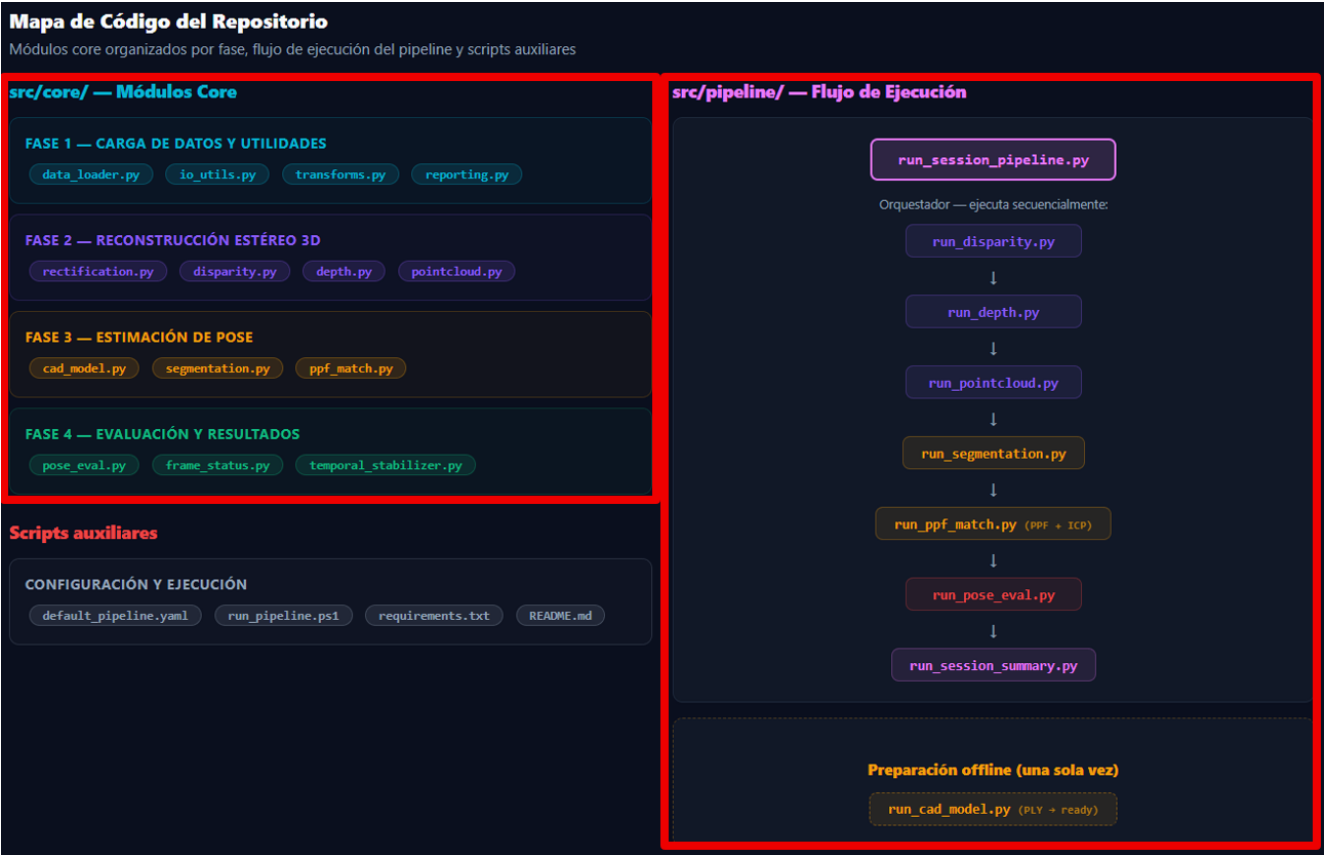


Figura 3-33: Mapa de código del repositorio (Autoría propia)

3.3.2.3 Archivos auxiliares del repositorio

Finalmente, en el primer nivel del repositorio se encuentran archivos auxiliares orientados a la configuración y ejecución del proyecto. La funcionalidad de cada uno de estos archivos se describe en la Tabla 3-4.

Archivo	Descripción
.gitignore	Establece los elementos a ignorar para la subida del repositorio a GitHub (mediante Git).
README.md	Contiene una descripción detallada del repositorio, instrucciones de instalación y uso.
requirements.txt	Declara las dependencias Python del proyecto (OpenCV, Open3D, NumPy, etc.) para poder reproducir el proyecto.
run_pipeline.ps1	Zona de configuración de usuario para la ejecución del <i>pipeline</i> desde PowerShell.

Tabla 3-4: Elementos auxiliares del repositorio

3.3.3 Carga de datos y metadatos

El módulo “data_loader.py” constituye la capa de acceso a datos del *pipeline*. Su función principal, “load_and_validate_frame()”, carga y valida un fotograma concreto de una sesión y devuelve un diccionario con todos los datos necesarios para las fases de procesamiento posteriores. El proceso de carga se articula en los pasos que se exponen a continuación.

3.3.3.1 Carga de imágenes

En primer lugar, se realiza la carga de las imágenes estéreo. Se leen “left.png” y “right.png” desde la carpeta “raw/” de la sesión. Inmediatamente después se verifica que ambas imágenes tienen la misma resolución, son bidimensionales y tienen tipo *uint8* (es decir, que ambas imágenes son compatibles para poder procesarlas). Si alguna de estas condiciones falla, se lanza una excepción que detiene el *pipeline*.

3.3.3.2 Carga de metadatos

En segundo lugar, se procede a la carga de metadatos desde el archivo “metadata.json”. Este archivo, generado por la aplicación Unity durante la captura, contiene toda la información geométrica del fotograma: los parámetros intrínsecos de ambas cámaras, las poses de las cámaras en el espacio mundo y las poses de los controladores. La Tabla 3-5 describe los campos principales de este archivo.

Campo	Descripción
FrameIndex	Índice secuencial del fotograma dentro de la sesión
TimestampSeconds	Marca temporal en segundos desde el inicio de la sesión
LeftTextureWidth/Height RightTextureWidth/Height	Resolución real de las imágenes izquierda y derecha (1280×960)
LeftIntrinsics RightIntrinsics	Intrínsecos de la cámara izquierda y derecha: f_x, f_y, c_x, c_y , “sensorWidth” y “sensorHeight”
CameraPoseLeft CameraPoseRight	Pose de las cámaras izquierda y derecha: Posición $[x, y, z]$ y Rotación $[x, y, z, w]$
ControllerPoseLeft ControllerPoseRight	Pose del controlador izquierdo y derecho en coordenadas mundo Unity. El <i>ground truth</i> del controlador derecho se emplea para la evaluación de la estimación de pose

Tabla 3-5: Campos principales del archivo “metadata.json”

3.3.3.3 Ajuste de intrínsecos

En tercer lugar, se ejecuta el ajuste de intrínsecos al tamaño real de la imagen. La Passthrough Camera API del SDK de Meta define los intrínsecos para un sensor de 1280×1280 píxeles, mientras que las imágenes PNG capturadas tienen una resolución de 1280×960. Esta discrepancia indica que el dispositivo realiza un recorte vertical de la imagen antes de entregarla. La función “adjust_intrinsics_to_image()” resuelve esta diferencia asumiendo un recorte vertical simétrico (mediante *crop_y*) y corrigiendo el punto principal c_y en consecuencia.

3.3.3.4 Validación de intrínsecos y construcción de matrices K

En cuarto lugar, se lleva a cabo la validación de los intrínsecos y la construcción de las matrices de proyección K . Se comprueba que las distancias focales f_x y f_y sean positivas y que el punto principal cx esté dentro de un margen respecto al centro horizontal de la imagen. Con los intrínsecos validados se construyen las matrices de proyección K_left y K_right en formato NumPy (3×3). La formulación de estas matrices se expone en (2-1) en el apartado 2.3.2.

3.3.3.5 Transformaciones cámara-mundo

En quinto lugar, se obtienen las transformaciones cámara-mundo. A partir de las poses proporcionadas por el archivo “metadata.json”, se construyen las matrices homogéneas 4×4 que representan la transformación del mundo a cada cámara ($T_W^{(left)}$ y $T_W^{(right)}$). El módulo “transforms.py” proporciona la función “pose_dict_to_T44()”, que convierte el formato Unity (posición $[x, y, z]$ y cuaternión $[x, y, z, w]$) a una matriz $SE(3)$ estándar.

3.3.3.6 Cálculo de la transformación relativa estéreo

Finalmente, se calcula la transformación relativa entre las cámaras izquierda y derecha. La función “compute_relative_transform()” obtiene la rotación y traslación relativa mediante (3-1).

$$T_R^{(left)} = \left(T_W^{(right)}\right)^{-1} \cdot T_W^{(left)} \quad (3-1)$$

Sobre este resultado se verifican tres condiciones geométricas que actúan como salvaguarda ante metadatos corruptos: que el ángulo de rotación relativa sea inferior a 5°, que el *baseline* tenga una magnitud coherente (entre 0.05 m y 0.08 m), y que la traslación sea predominantemente horizontal. Si alguna de estas comprobaciones falla, el fotograma se marca como inválido antes de que entre al *pipeline* de reconstrucción.

3.4 Fase 3: Reconstrucción estereoscópica 3D

Si la Fase 1 provee los datos brutos y la Fase 2 los integra en el entorno de procesado, la Fase 3 es donde comienza la visión artificial propiamente dicha. Aquí se produce la transición fundamental del proyecto: se pasa de dos imágenes planas a una representación tridimensional de la escena.

Esta transformación requiere cuatro etapas encadenadas, en las que un error en cualquiera de ellas puede degradar silenciosamente toda la cadena posterior. Las cuatro etapas son: rectificación estereoscópica, cálculo de disparidad, estimación de profundidad y generación de la nube de puntos 3D.

3.4.1 Rectificación estéreo

La rectificación estereoscópica constituye el primer paso imprescindible del *pipeline* de reconstrucción. Su objetivo es transformar el par de imágenes original para que todas las líneas epipolares queden perfectamente horizontales, de modo que la búsqueda de correspondencias se reduzca a un barrido unidimensional por fila. Sin este paso, los algoritmos de disparidad posteriores no pueden operar correctamente.

3.4.1.1 Cálculo de la transformación relativa

La implementación se articula en torno al módulo “rectification.py”. El proceso comienza con la función “Rt_left_to_right_from_Twc”, que calcula la transformación relativa entre las cámaras

izquierda y derecha. Este cálculo parte de las poses de ambas cámaras expresadas en el sistema mundo de Unity y consta de dos pasos.

En primer lugar, se compone la transformación relativa (3-1), obteniendo la rotación y traslación entre cámaras en coordenadas Unity. En segundo lugar, se aplica un cambio de base de Unity a OpenCV, necesario porque Unity emplea un sistema de coordenadas con el eje Y hacia arriba, mientras que OpenCV utiliza Y hacia abajo. Este cambio se resuelve mediante una matriz diagonal (3-2) aplicada tanto a la rotación como a la traslación relativas.

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (3-2)$$

3.4.1.2 Rectificación y generación de mapas

Una vez obtenidas R y t en coordenadas OpenCV, la función “rectify_pair” llama a “cv2.stereoRectify” con los intrínsecos de ambas cámaras y su transformación relativa. Esta operación calcula las matrices de rotación R_1 y R_2 , que alinean las imágenes haciendo que las líneas epipolares sean horizontales, y las matrices de proyección P_1 y P_2 , que definen la nueva proyección en las imágenes rectificadas. A partir de ellas se obtienen los intrínsecos rectificados y el *baseline* efectivo, necesarios para estimar la profundidad a partir de la disparidad.

Además, se obtiene la matriz Q , que permite recuperar coordenadas 3D a partir de la disparidad, y las regiones de interés válidas (ROI) de cada imagen rectificada, que delimitan el área útil tras la transformación.

A continuación, mediante “cv2.remap”, se obtienen las imágenes rectificadas “left_rect.png” y “right_rect.png”, que se muestran en la Figura 3-34.



Figura 3-34: Imágenes estereoscópicas rectificadas (Autoría propia)

3.4.1.3 Validación de la rectificación

Para cuantificar la calidad de la rectificación, se implementa la función “y_error_orb”, que estima el error epipolar vertical residual. Esta función detecta puntos característicos en ambas imágenes mediante el descriptor ORB (*Oriented FAST and Rotated BRIEF*, introducido en el apartado 2.4.1), que localiza esquinas y bordes relevantes y genera para cada uno un vector que describe su entorno visual.

A continuación, se emparejan los descriptores de ambas imágenes buscando los más similares. Para filtrar coincidencias espurias se aplica la ratio de Lowe. Este método consiste en que para cada punto de la imagen izquierda se buscan sus dos mejores candidatos en la derecha. En caso de que la distancia al mejor candidato sea inferior al 75% de la distancia al segundo, se acepta la correspondencia. En caso contrario, se descarta al considerarse ambigua.

Tras este filtrado, se calcula la matriz fundamental con RANSAC (véase el apartado 2.4.1), que retiene las correspondencias geoméricamente consistentes, denominadas *inliers* (emparejamientos que satisfacen la restricción epipolar), y descarta las inconsistentes, denominadas *outliers* (emparejamientos que no satisfacen la restricción anterior). Sobre las correspondencias *inlier* se mide la diferencia vertical en píxeles entre los puntos emparejados: en una rectificación perfecta ambos puntos deben situarse en la misma fila, por lo que este error debería ser cero.

Los resultados se guardan en “y_error_orb_raw.json” (antes de rectificar) e “y_error_orb_rect.json” (tras la rectificación), los cuales se evalúan para determinar si la rectificación es adecuada. Además, se genera “rectification_debug.json”, que contiene todas las matrices de rectificación, las ROI (las regiones de interés válidas de cada imagen; es decir, el área útil donde no aparecen zonas negras tras la transformación), el *baseline* efectivo y los intrínsecos rectificadas. Este archivo resulta esencial para las etapas posteriores, pues la matriz de intrínsecos rectificadas (K_{rect} , extraída de $P1$) y el *baseline* son los parámetros fundamentales para la conversión de disparidad a profundidad, tal como establece la ecuación (2-9).

3.4.2 Cálculo de disparidad

Una vez obtenidas las imágenes rectificadas, la siguiente etapa consiste en calcular el mapa de disparidad, que asigna a cada píxel de la imagen izquierda el desplazamiento horizontal de su punto homólogo en la imagen derecha. Esta información codifica implícitamente la profundidad de cada punto de la escena.

La implementación del módulo “disparity.py” se estructura en tres etapas encadenadas: el cálculo mediante SGBM, la comparación entre disparidad bruta y rectificadas, y el refinamiento con el filtro WLS.

3.4.2.1 Algoritmo SGBM

El algoritmo SGBM (*Semi-Global Block Matching*), introducido en el apartado 2.4.3, busca para cada píxel de la imagen izquierda su correspondiente en la derecha. Para ello compara pequeñas ventanas cuadradas de píxeles entre ambas imágenes, midiendo su similitud.

Lo que diferencia a SGBM de los métodos puramente locales es que no evalúa cada píxel de forma aislada, sino que recorre la imagen en múltiples direcciones acumulando un coste que penaliza los cambios de disparidad entre píxeles vecinos. Así, el algoritmo elige la disparidad que minimiza el coste global, produciendo mapas más suaves y coherentes.

Los parámetros $P1$ y $P2$ controlan la intensidad de estas penalizaciones: mientras el parámetro $P1$ se aplica para cambios pequeños, el parámetro $P2$ atiende a saltos bruscos (por ejemplo, bordes de objetos). La función “compute_disparity_sgbm” configura el algoritmo con los parámetros que recoge la Tabla 3-6.

Parámetro	Efecto
<i>block_size</i>	Tamaño de la ventana de comparación (píxeles)
<i>P1</i>	Penalización por cambios suaves de disparidad
<i>P2</i>	Penalización por cambios bruscos (debe ser $> P1$)
<i>uniqueness_ratio</i>	Margen (%) para filtrar correspondencias ambiguas
<i>displ2MaxDiff</i>	Diferencia máxima en comprobación izquierda-derecha
<i>speckleWindowSize</i>	Tamaño mínimo de región (las de menor tamaño se eliminan)
<i>speckleRange</i>	Variación máxima de disparidad dentro de una región

Tabla 3-6: Parámetros del algoritmo SGBM

Se calculan dos versiones del mapa de disparidad: una sobre las imágenes originales sin rectificar (“disparity_raw”) y otra sobre las imágenes rectificadas (“disparity_rect”). La primera sirve exclusivamente como referencia para cuantificar el efecto de la rectificación; la segunda es la que alimenta el resto del *pipeline*.

3.4.2.2 Refinamiento con filtro WLS

Para mejorar la calidad del mapa de disparidad rectificado, se aplica el filtro WLS (*Weighted Least Squares*), implementado en la función “compute_disparity_wls”. Este filtro utiliza la imagen guía (la imagen izquierda rectificada) para propagar información de bordes al mapa de disparidad, rellenando huecos y suavizando transiciones ruidosas. El resultado se almacena como “disparity_rect_wls.png”, generándose métricas de densidad y distribución en archivos JSON.

La Figura 3-35 muestra la progresión visual del mapa de disparidad a lo largo de estas etapas, lo que permite apreciar la mejora que aporta cada paso del proceso.

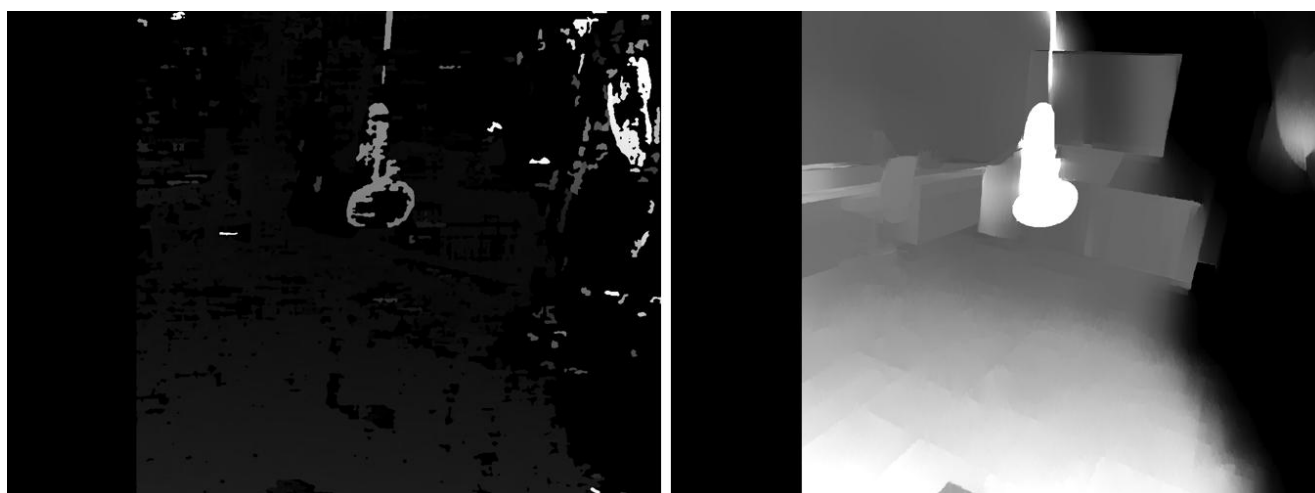


Figura 3-35: Mapa de disparidad con y sin filtro WLS (Autoría propia)

3.4.3 Estimación de profundidad

La conversión de disparidad a profundidad se basa en la ecuación de proyección estéreo (2-9), expuesta en el apartado 2.4.4, de la que se puede intuir que la precisión en la profundidad varía enormemente en función de la distancia del objeto al dispositivo.

3.4.3.1 Filtrado previo

La función “compute_depth_from_disparity” del módulo “depth.py” implementa esta conversión de disparidad a profundidad aplicando dos capas de filtrado previas, cuyos parámetros vienen recogidos en la Tabla 3-7.

Parámetro	Valor	Efecto
<i>min_disp_px</i>	1.0 px	Invaldar disparidades muy próximas a cero (profundidad prácticamente infinita)
<i>z_min</i>	0.20 m	Descartar estimaciones de objetos demasiado próximos
<i>z_max</i>	2.0 m	Descartar errores generados por disparidades pequeñas (objetos lejanos)

Tabla 3-7: Parámetros de filtrado de profundidad

El resultado es un mapa de profundidad métrico (“depth_m.npy”) acompañado de una máscara de validez (“depth_valid_mask.npy”) que indica para cada píxel si su profundidad es fiable. Además, la función *depth_preview_uint8* genera una imagen de previsualización (“depth_preview.png”) donde los objetos cercanos se representan en tonos claros y los lejanos en tonos oscuros. La Figura 3-36 muestra esta previsualización.

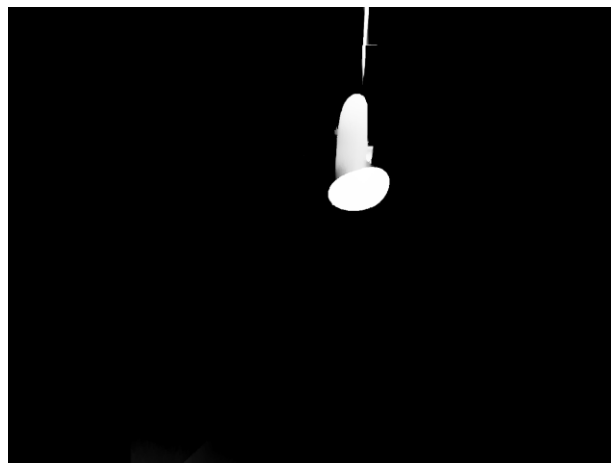


Figura 3-36: Mapa de profundidad (Autoría propia)

3.4.4 Generación de nube de puntos 3D

El mapa de profundidad es una representación 2D donde cada píxel tiene asociado un valor escalar de distancia. Para obtener una representación tridimensional real de la escena, es necesario invertir la operación de proyección de cámara. Este proceso, denominado *backprojection*, constituye el último paso de la Fase 3.

3.4.4.1 Retroproyección

La función “`build_pointcloud_camera_opencv`” del módulo “`pointcloud.py`” implementa esta operación, calculando las coordenadas 3D mediante las fórmulas de la ecuación (2-11) del apartado 2.4.5. La nube resultante se expresa inicialmente en coordenadas de la cámara OpenCV.

3.4.4.2 Cambio de sistema de referencia

Para integrar la nube en el sistema de referencia del dispositivo, se aplican dos transformaciones sucesivas. Primero, la función “`points_OpenCV_cam_to_Unity_cam`” invierte el eje Y para pasar de la convención OpenCV (Y hacia abajo) a la convención Unity (Y hacia arriba). Después, la función “`transform_points`” aplica la transformación T_{W_L} (mundo a cámara izquierda) para expresar todos los puntos en coordenadas mundo Unity.

Finalmente, se aplica un *voxel downsample* configurable (por defecto 2 mm). Esta operación divide el espacio 3D en pequeños cubos (*voxels*) del tamaño indicado y sustituye todos los puntos de cada cubo por uno solo situado en su centro. El resultado es una nube con densidad uniforme que conserva la estructura geométrica de la escena, reduce el volumen de datos y acelera las operaciones posteriores.

3.4.4.3 Archivos generados

El resultado se guarda en cuatro archivos: “`pointcloud_camera_opencv.npy`” y “`pointcloud_world_unity.npy`” (formato NumPy), y “`pointcloud_camera_rect.ply`” y “`pointcloud_world_unity.ply`” (formato PLY para inspección directa en CloudCompare). El archivo “`pointcloud_metrics.json`” registra el número de puntos, los rangos de profundidad y las coordenadas del *bounding box* (la caja envolvente alineada con los ejes que delimita el volumen mínimo que contiene toda la nube) en ambos sistemas de referencia.

La Figura 3-37 muestra la nube de puntos resultante visualizada en CloudCompare, donde se aprecia la geometría completa de la escena reconstruida. Se puede identificar claramente el controlador derecho desde ambas perspectivas, a pesar de ir acompañado de ruido (*speckle*, descrito en el apartado 2.4.3). El ruido provocado por regiones con disparidades incorrectas se ve enormemente reducido gracias a la acción del filtro WLS, pero no se elimina por completo, subrayando la importancia de la segmentación posterior para aislar correctamente el objeto de interés.

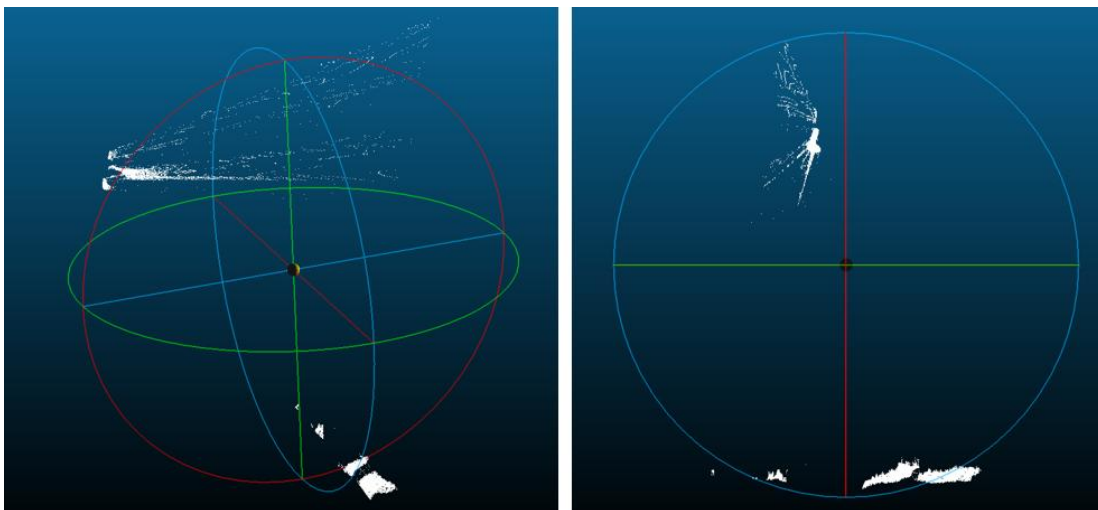


Figura 3-37: Nube de puntos 3D de la escena completa (Autoría propia)

Con estos archivos, la Fase 3 cumple su objetivo: transformar un par de imágenes 2D en una representación 3D de la escena geoméricamente consistente con el espacio del dispositivo. Todo el trabajo de rectificación, disparidad y profundidad que precede a este punto tiene como única finalidad garantizar que esta nube sea lo más fiel posible a la geometría real. Es sobre esta base donde la Fase 4 construirá la estimación de pose del controlador.

3.5 Fase 4: Estimación de pose

Si la fase anterior tenía por objetivo entregar, para cada fotograma de la sesión, una nube de puntos densa de la escena capturada, esta fase aborda un reto radicalmente distinto: localizar un objeto concreto dentro de esa nube y determinar su posición y orientación en el espacio. Esto es lo que en visión artificial se denomina estimación de pose 6-DoF (*Degrees of Freedom*), y constituye el objetivo central del *pipeline* de postprocesado.

Como ya se ha mencionado, el objeto elegido para esta tarea es el controlador derecho Touch Plus. La elección no es arbitraria: el propio SDK del dispositivo proporciona, fotograma a fotograma, la pose del controlador con alta precisión a través de su sistema de seguimiento interno. Esto convierte al controlador en el candidato ideal, pues permite comparar la pose estimada mediante visión artificial con la pose que el SDK considera real, obteniendo así una referencia (*ground truth*) para validar los resultados.

La fase se organiza en torno a dos carriles conceptualmente diferenciados, tal y como se describe en la documentación oficial de OpenCV para el módulo *Surface Matching*: el carril de entrenamiento (*Training*), que se realiza una única vez antes de procesar cualquier sesión, y el carril de ejecución (*Runtime*), que opera fotograma a fotograma.

3.5.1 Preparación del modelo CAD del controlador

El primer paso del carril de entrenamiento consiste en disponer de un modelo 3D del objeto cuya pose se desea estimar. En este proyecto se utiliza el modelo CAD del controlador *Meta Quest Touch Plus*, obtenido de la plataforma de modelos 3D Sketchfab y convertido a formato BLEND (Blender). El modelo original incluía, además de ambos controladores, la geometría del propio dispositivo Meta Quest 3, que se eliminó durante el acondicionamiento al no ser un objeto de interés para la estimación de pose. La Figura 3-38 muestra los controladores cargados en Blender.

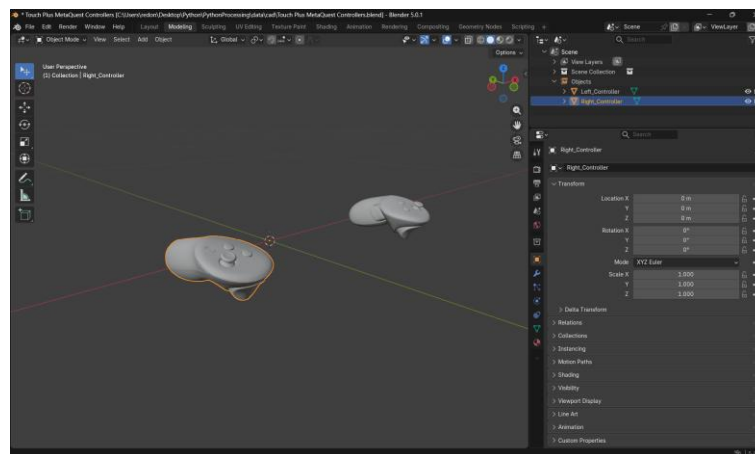


Figura 3-38: Modelo CAD de los controladores Touch Plus en Blender (Autoría Propia)

3.5.1.1 Acondicionamiento del modelo

La conversión del modelo a un formato compatible la realiza el *script* “run_cad_model.py”, invocando las funciones del módulo “cad_model.py”. El proceso consta de tres pasos.

En primer lugar, se exporta la malla del controlador derecho desde Blender a formato PLY, eliminando los elementos que no pertenecen a la geometría del mando. La Figura 3-39 muestra el controlador derecho aislado.



Figura 3-39: Detalle del controlador derecho limpio en Blender (Autoría propia)

En segundo lugar, se aplica un *voxel downsample* (por defecto 5 mm) para reducir la densidad de la nube manteniendo su estructura geométrica.

En tercer lugar, se estiman las normales de superficie mediante una búsqueda híbrida radio-vecinos. Este método selecciona, para cada punto, los vecinos que se encuentren dentro de un radio máximo de 15 mm y hasta un límite de 50 vecinos. Con ellos se ajusta un plano local, cuyo vector perpendicular define la normal de superficie en ese punto. Combinar ambos criterios evita mezclar superficies distintas (límite de radio) y mantiene el coste computacional controlado (límite de vecinos). El resultado es el archivo “right_controller_ready.ply”, que contiene la nube de puntos del modelo con normales orientadas y constituye la entrada para todas las operaciones de correspondencia posteriores.

El archivo “right_controller_ready_metrics.json” registra el número de puntos del modelo y la extensión de su *axis-aligned bounding box* (AABB), es decir, la caja envolvente cuyos lados son paralelos a los ejes X, Y y Z, y la posición de su centro geométrico. La Figura 3-40 muestra el modelo convertido a malla y nube de puntos.

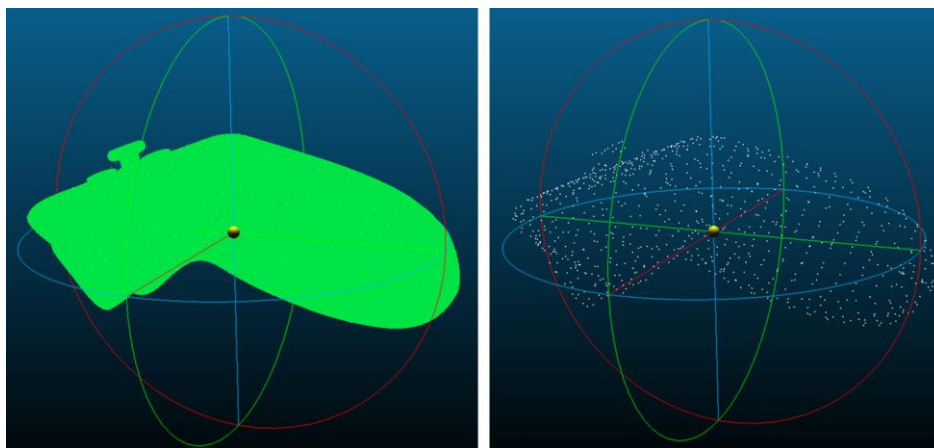


Figura 3-40: Modelo CAD como malla y nube de puntos en CloudCompare (Autoría Propia)

3.5.2 Entrenamiento del descriptor PPF

El segundo paso del carril de entrenamiento consiste en construir los descriptores *Point Pair Feature* (PPF) del modelo CAD. Estos descriptores, introducidos en el apartado 2.5.2, codifican relaciones geométricas entre pares de puntos del modelo en un vector de cuatro dimensiones que es invariante a la pose del objeto.

3.5.2.1 Construcción de la tabla hash

El entrenamiento se implementa mediante la clase “cv2.ppf_match_3d_PPF3DDetector” de OpenCV, instanciada en el módulo “ppf_match.py” con dos parámetros relativos al diámetro del modelo: *relativeSamplingStep* (0.05), que controla la densidad del submuestreo (se muestrea aproximadamente un punto por cada 5% del diámetro), y *relativeDistanceStep* (0.05), que define la resolución con la que se discretizan las distancias entre pares de puntos. Valores menores producen descriptores más densos y correspondencias más precisas, a costa de mayor tiempo de cómputo.

El método “trainModel()” del detector recibe la nube del modelo como una matriz Nx6 en formato *float32*: un arreglo numérico donde N es el número de puntos y cada fila contiene las coordenadas espaciales (x , y , z) y las componentes de la normal (n_x , n_y , n_z) de un punto. Internamente, el detector calcula el descriptor PPF (véase la ecuación (2-14)) para cada par de puntos posible y lo almacena en una tabla *hash*. Una tabla *hash* es una estructura de datos que funciona como un índice: asocia cada descriptor (la “clave”) con la lista de pares del modelo que lo generan (el “valor”), permitiendo búsquedas casi instantáneas. Cuando en la fase de ejecución se calcula el descriptor de un par de puntos de la escena, basta con consultar esta tabla para encontrar todos los pares del modelo con geometría equivalente, sin recorrer la base de datos completa.

3.5.3 Segmentación del objeto

La segmentación constituye la primera etapa del carril de ejecución y tiene por objetivo aislar la región de la nube de puntos que corresponde al controlador, descartando el fondo y cualquier otro elemento de la escena. La calidad de esta segmentación condiciona directamente la calidad del *matching* posterior: una nube demasiado ruidosa produce hipótesis de pose erróneas, mientras que una nube excesivamente recortada puede eliminar partes del controlador necesarias para un alineamiento correcto.

La implementación reside en el módulo “segmentation.py” y su *script* de ejecución “run_segmentation.py”. El proceso se organiza en una secuencia de cinco pasos, precedidos por un recorte opcional. La Tabla 3-8 resume las etapas y sus parámetros clave.

3.5.3.1 Paso 0: Recorte por ROI orientada

El recorte por ROI es un paso previo opcional que aprovecha la pose del controlador proporcionada por el SDK. Cuando se activa el parámetro *use_roi_gt*, se define una *Oriented Bounding Box* (OBB) en coordenadas mundo, con dimensiones configurables (por defecto 0.30 m en cada eje).

La función “crop_obb_world” transforma los puntos al sistema local del controlador y retiene únicamente aquellos que caen dentro de la caja, reduciendo drásticamente el volumen de datos.

3.5.3.2 Pasos 1 y 2: Filtrado y regularización

El primer paso aplica un filtrado por banda de profundidad que descarta puntos fuera del rango $[z_{min}, z_{max}]$ configurable.

Paso	Operación	Parámetros Clave	Archivo de salida
0	Recorte por ROI-GT	$use_roi_gt = true$ $roi_s = 0.30 \text{ m}$	object_roi_raw.ply
1	Filtrado por banda Z	$z_min = 0.15 \text{ m}$ $z_max = 2.0 \text{ m}$	-
2	<i>Voxel downsample</i>	$voxel = 1.5 \text{ mm}$	-
3	Eliminación planos y <i>outliers</i>	RANSAC, filtro estadístico	object_preseg_clean.ply
4	<i>Clustering</i> DBSCAN	$min_cluster_size = 200$	object_segmented.ply
5	Estimación de normales	Búsqueda híbrida radio- vecinos	object_segmented_normals.ply

Tabla 3-8: Etapas de la segmentación y parámetros clave

El segundo paso aplica un *voxel downsample* con un tamaño de *voxel* de 1.5 mm, que regulariza la distribución espacial de los puntos y acelera los pasos posteriores.

3.5.3.3 Paso 3: Eliminación de planos y outliers

La función “remove_dominant_planes_o3d” utiliza RANSAC iterativo para detectar y eliminar los planos más grandes de la nube (típicamente el suelo o las paredes). Por su parte, “remove_outliers_o3d” aplica un filtro estadístico que elimina los puntos cuya distancia media a sus vecinos supera un umbral, reduciendo el ruido de medición.

3.5.3.4 Paso 4: Clustering DBSCAN

DBSCAN agrupa los puntos restantes en *clusters* según cercanía euclidiana. La función de selección escoge el *cluster* más relevante según la estrategia configurada: por cercanía al centro de referencia (*ground truth*) o por tamaño. Un filtro de tamaño mínimo ($min_cluster_size = 200$ puntos) descarta *clusters* demasiado pequeños.

3.5.3.5 Paso 5: Estimación de normales y archivos generados

Tras la segmentación, se estiman las normales de superficie. El resultado final se guarda en “object_segmented_normals.ply”, y la matriz Nx6 lista para el *matching* PPF se almacena en “object_segmented_nx6.npy”.

Las métricas completas del proceso se registran en “segmentation_metrics.json”. El resultado final de la segmentación geométrica se muestra a través de la Figura 3-41, mediante la que se puede observar la reducción efectiva y centrada en el objeto de interés de la nube de puntos reconstruida.

3.5.4 Matching PPF+ICP

Una vez segmentado el objeto, se procede a la estimación de pose propiamente dicha. Esta etapa implementa la estrategia en cascada PPF + ICP descrita en el apartado 2.5.4 del Estado del Arte,

siguiendo la arquitectura del módulo *Surface Matching* de OpenCV. La Tabla 3-9 recoge los parámetros más relevantes.

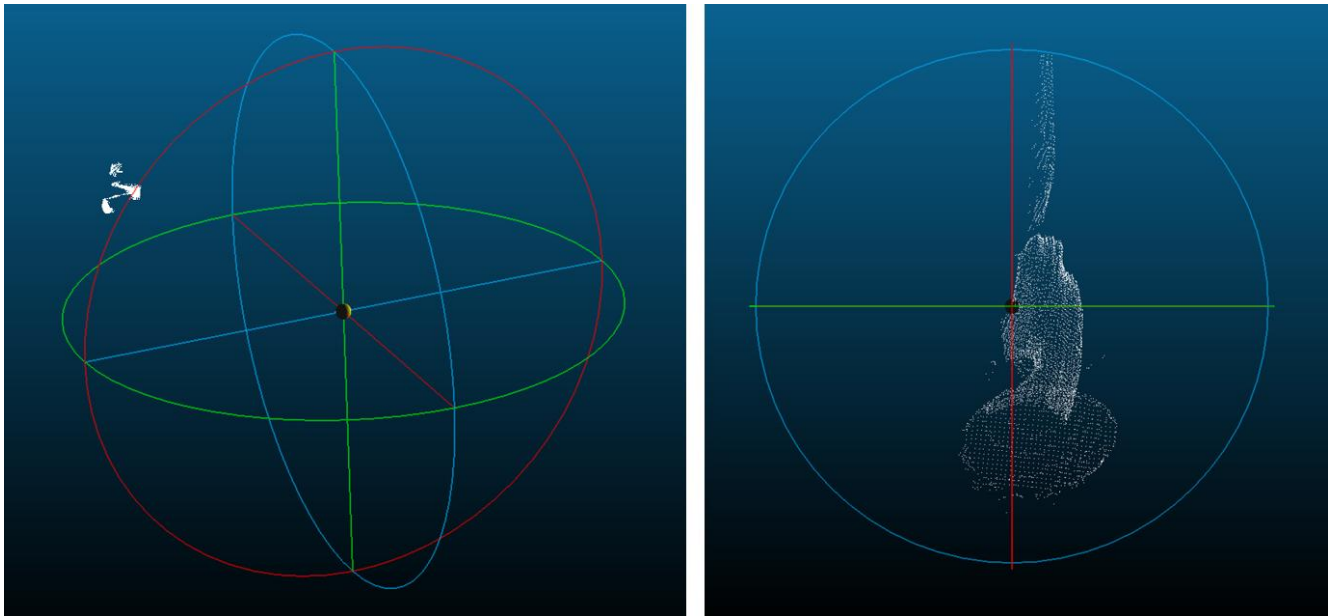


Figura 3-41: Objeto segmentado aislado en CloudCompare (Autoría Propia)

Parámetro	Valor	Descripción
<i>PPF_rel_sampling</i>	0.05	Submuestreo del modelo para entrenamiento PPF
<i>PPF_rel_distance</i>	0.05	Discretización de distancias en tabla hash
<i>PPF_scene_sample_step</i>	0.025	Fracción de puntos de escena usados como referencia
<i>PPF_top_n</i>	5	Hipótesis PPF refinadas con ICP
<i>ICP_iterations</i>	100	Iteraciones máximas del refinamiento ICP
<i>ICP_tolerance</i>	0.005 m	Convergencia por desplazamiento mínimo
<i>ICP_rejection_scale</i>	2.5	Factor de rechazo de <i>outliers</i> (desviación estándar)
<i>ICP_num_levels</i>	4	Niveles de resolución (multirresolución)
<i>crop_radius</i>	0.14 m	Radio de recorte esférico para prefiltrado

Tabla 3-9: Parámetros del *matching* PPF+ICP

Dos parámetros de la tabla anterior merecen aclaración. El parámetro *ICP_num_levels* (4 niveles) activa el modo multirresolución: en lugar de procesar las nubes directamente a su densidad completa, el algoritmo comienza con versiones muy simplificadas y va aumentando la resolución nivel a nivel hasta llegar a la original. Esta estrategia “de grueso a fino” evita que ICP se atasque en soluciones erróneas y acelera la convergencia.

Por su parte, *ICP_rejection_scale* (2.5) controla el rechazo de *outliers* durante el alineamiento. En cada iteración, se calcula la distancia entre cada par de puntos correspondientes y se obtiene la desviación estándar (σ) de esas distancias. Los pares cuya distancia supera $2.5 \times \sigma$ se descartan como correspondencias erróneas, mejorando la robustez del ajuste frente a puntos que no pertenecen realmente al objeto.

3.5.4.1 Prefiltrado de la escena segmentada

Aunque la segmentación ya aísla la región del controlador, pueden persistir puntos espurios. El prefiltrado aplica tres operaciones: un recorte esférico por radio (*crop_radius* = 0.14 m), un recorte (*clamp*) por profundidad en el eje de la cámara (que descarta los puntos cuya distancia a la cámara quede fuera del intervalo de distancia estimado, reteniendo solo la franja donde es razonable encontrar el controlador) y un filtro de densidad local que elimina puntos con densidad anómalamente alta o baja.

3.5.4.2 Matching PPF

El detector recibe la nube de escena filtrada en el mismo formato que el modelo entrenado (cada fila contiene las coordenadas y la normal de un punto, como se describe en el apartado 3.5.2). Para cada punto de referencia en la escena, calcula el descriptor PPF con respecto a una fracción de los demás puntos (controlada por *PPF_scene_sample_step* = 0.025; es decir, el 2.5% de los puntos de la escena) y lo busca en la tabla *hash* del modelo. Cada coincidencia genera un “voto” a favor de una posible pose del objeto.

Estos votos se acumulan en un espacio de Hough discretizado, que funciona como un tablero de votación: el conjunto de todas las posibles posiciones y orientaciones del objeto se divide en celdas, y cada voto incrementa el contador de la celda que corresponde a la pose sugerida por esa coincidencia. Al finalizar, las celdas más votadas representan las hipótesis de pose más probables. Se seleccionan las 5 mejores (*PPF_top_n* = 5) para su refinamiento posterior con ICP.

3.5.4.3 Refinamiento ICP

Las 5 mejores hipótesis generadas por PPF se refinan mediante ICP (*Iterative Closest Point*), el algoritmo de registro iterativo descrito en el apartado 2.5.3. Se emplea la implementación “cv2.ppf_match_3d_ICP” de OpenCV en modo punto-a-plano. A diferencia del ICP clásico (que minimiza la distancia directa entre puntos), esta variante proyecta el error sobre la normal de la superficie del punto objetivo (véase la Figura 2-10), lo que permite una convergencia más rápida en superficies suaves como las del controlador.

El refinamiento opera en modo multirresolución con 4 niveles, comenzando con versiones simplificadas de las nubes y aumentando progresivamente la densidad. En cada nivel se ejecutan hasta 100 iteraciones, deteniéndose si el desplazamiento entre iteraciones consecutivas cae por debajo de la tolerancia (0.005 m) establecida. En cada paso se descartan los pares de correspondencia cuya distancia supera 2.5 veces la desviación estándar. El resultado es un alineamiento preciso entre el modelo CAD y la nube segmentada, partiendo de la hipótesis inicial proporcionada por PPF.

3.5.4.4 Proyección a $SO(3)$

Tras el refinamiento ICP, se aplica una corrección sobre la matriz de rotación de cada candidato. Toda matriz de rotación válida debe ser ortogonal (columnas perpendiculares entre sí y de longitud

unitaria) y tener determinante exactamente +1. Sin embargo, las sucesivas iteraciones del ICP pueden acumular pequeños errores numéricos que desvíen ligeramente la matriz de estas condiciones.

Para corregirlo, se proyecta al espacio $SO(3)$ (el conjunto de todas las rotaciones 3D válidas) mediante una descomposición en valores singulares (SVD), que reconstruye la rotación más cercana que sí cumple ambas condiciones. Aunque pueda parecer un detalle menor, esta operación es importante para evitar que las transformaciones estimadas incluyan escalados o reflexiones espurias, lo que se verifica posteriormente con la métrica $\det(R)$ de la Tabla 4-1.

3.5.4.5 Selección del mejor resultado

La selección final evalúa cada una de las 5 hipótesis refinadas con la función “evaluate_registration” de Open3D, que calcula dos métricas complementarias.

El *fitness* mide la fracción de puntos del modelo CAD que encuentran al menos un punto correspondiente en la nube segmentada dentro de un radio umbral (un *fitness* de 0.5 indica que el 50% de los puntos del modelo están correctamente alineados con la escena).

El RMSE cuantifica el error cuadrático medio de esas correspondencias, expresado en metros. Se selecciona el candidato con mayor *fitness*, usando el RMSE como criterio de desempate.

Conviene aclarar por qué se prioriza el *fitness* y qué puede esperarse de esta métrica. En este sistema, alcanzar un valor cercano a 1.0 es prácticamente imposible, por tres razones.

- En primer lugar, las cámaras captan el controlador desde un único ángulo, de modo que las caras opuestas son siempre invisibles.
- En segundo lugar, la reconstrucción estereoscópica genera nubes parciales e incompletas en zonas de baja textura.
- En tercer lugar, en el cálculo de la disparidad se producen huecos que la segmentación no elimina por completo.

En contraste, el sistema de *tracking* nativo de la Quest 3 conoce de antemano la geometría del controlador y emplea sensores infrarrojos dedicados, lo que le permite alcanzar precisiones muy superiores. Por todo ello, un *fitness* superior a 0.5 se considera excelente en este contexto: significa que más de la mitad del modelo (incluyendo caras no visibles) se ha alineado correctamente, lo que en la práctica se traduce en estimaciones de posición y orientación coherentes con la realidad.

3.5.4.6 Archivos generados y resultados visuales

Los resultados se almacenan en la subcarpeta “ppf_match/” de cada fotograma. En dicha subcarpeta se encuentran los archivos “pose_best.json”, que contiene la transformación 4×4 estimada, y “aligned_model.ply”, que permite inspeccionar visualmente el alineamiento en CloudCompare.

Dicho alineamiento se muestra mediante la Figura 3-42 y la Figura 3-43. En dichas imágenes se puede observar de manera sencilla cómo las partes más representativas del controlador “encajan” en la nube de puntos reconstruida. Este encaje visual es síntoma de la obtención de resultados de estimación de pose competentes.

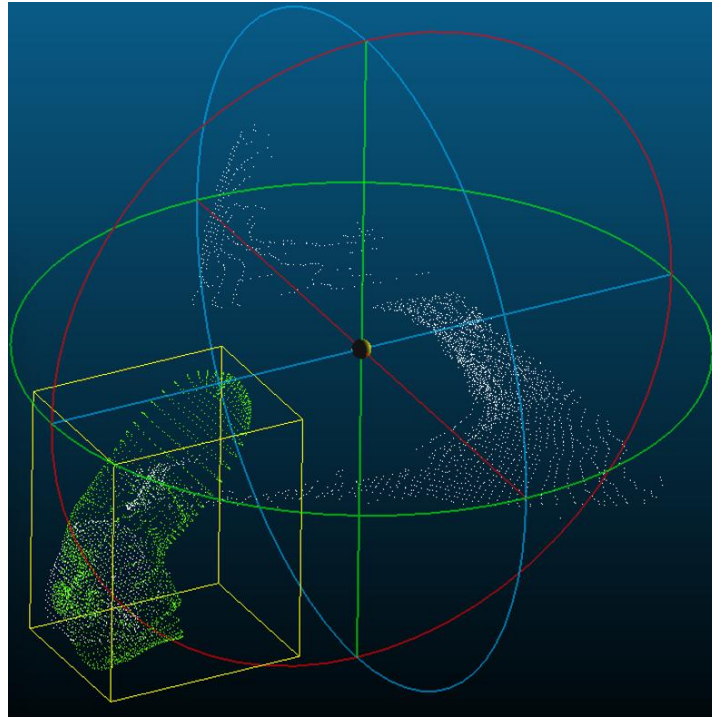


Figura 3-42: Modelo CAD alineado sobre la nube segmentada (Autoría propia)

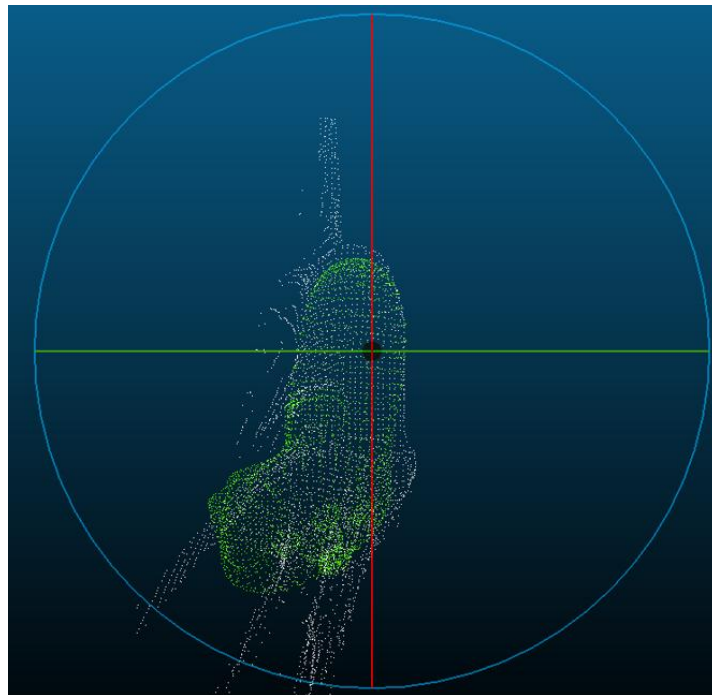


Figura 3-43: Estimación de pose de otro fotograma de la misma sesión (Autoría propia)

3.5.5 Evaluación por fotograma y gate pre-matching

Antes de ejecutar el *matching* PPF+ICP, el *pipeline* implementa un mecanismo de control de calidad denominado *gate pre-matching*. Tras la segmentación, el orquestador lee “segmentation_metrics.json” y verifica dos condiciones: que el campo *ok* sea verdadero y que el número de puntos de salida (n_{out}) sea ≥ 300 . Si alguna condición no se cumple, el fotograma se registra como fallido, evitando ejecutar un *matching* costoso sobre nubes inválidas.

3.5.5.1 Evaluación de la pose estimada

Para cada fotograma procesado, se carga la transformación estimada (almacenada en “pose_best.json”) y la transformación de referencia proporcionada por el SDK del dispositivo (extraída de “metadata.json”).

A partir de ambas transformaciones se calculan el error de traslación (distancia euclidiana entre las posiciones estimada y real) y el error de rotación (ángulo geodésico entre ambas matrices de rotación).

Adicionalmente, se verifican comprobaciones de integridad (*sanity checks*) sobre la transformación estimada. Estas comprobaciones confirman que la matriz de rotación tiene determinante +1 (descartando reflexiones o escalados espurios) y que es ortogonal (véase la ecuación (3-3))

$$R \cdot R^t \approx I \quad (3-3)$$

Si alguna condición falla, la estimación se marca como degenerada y se clasifica como fallida, independientemente de los valores de *fitness* o RMSE.

3.5.5.2 Clasificación de fotogramas

La clasificación de cada fotograma se realiza mediante la función “classify_frame”, que asigna una etiqueta según los umbrales configurables. La Tabla 3-10 describe las etiquetas posibles.

Etiqueta	Condición	Significado
OK	Todos los criterios cumplidos	Estimación válida y dentro de tolerancia
FAIL_NO_EST	Sin “pose_best.json”	El <i>matching</i> no produjo estimación
FAIL_NO_GT	Sin <i>ground truth</i>	No es posible evaluar (falta referencia)
FAIL_NO_DATA	Segmentación < 300 puntos	Datos insuficientes para <i>matching</i> fiable
FAIL_BAD_ICP	$\text{fitness} < 0.15$ o $\text{RMSE} > 0.02 \text{ m}$	Alineamiento ICP no convergió correctamente
FAIL_POSE	Error traslación > 0.05 m o rotación > 10°	Estimación fuera de tolerancia respecto al GT

Tabla 3-10: Clasificación de fotogramas

3.6 Pipeline de ejecución

Hasta este punto se han descrito, una a una, las etapas que componen el *pipeline* de postprocesado: desde la carga de datos hasta la evaluación de la pose estimada. Esta sección aborda cómo se orquestan y ejecutan estas etapas de forma integrada para procesar una sesión completa.

La Figura 3-44, obtenida de la pestaña “Flujo de datos” del *dashboard* interactivo, muestra el flujo de procesamiento completo del *pipeline*, que comprende desde la captura estéreo hasta la evaluación de datos, incluyendo además una última etapa de estabilización temporal para rechazar saltos implausibles entre fotogramas consecutivos. Dicha etapa se aborda en el apartado 4.1.3.



Figura 3-44: Flujo de procesamiento del *pipeline* (Autoría propia)

3.6.1 Orquestador del *pipeline*

El *script* “run_session_pipeline.py” constituye el núcleo de orquestación del repositorio. Su arquitectura sigue un patrón de ejecución por subprocesos: cada etapa se ejecuta como un módulo Python independiente invocado mediante “subprocess.run”. Esta decisión aporta tres ventajas: aislamiento de fallos (un error no corrompe el proceso principal), independencia de memoria (cada subproceso libera recursos al finalizar) y posibilidad de ejecutar cualquier etapa de forma aislada para depuración.

3.6.1.1 Modos de ejecución

El orquestador acepta dos modos controlados por el *flag* “--robust”. En modo robusto (activo por defecto), un fallo en una etapa se registra como advertencia y el *pipeline* continúa con el siguiente fotograma. En modo estricto, cualquier fallo detiene la ejecución. El modo robusto es esencial para sesiones largas donde algunos fotogramas pueden presentar datos degradados.

3.6.1.2 Trazabilidad y reproducibilidad

Al inicio de cada ejecución, el orquestador genera “parameters_final.json” con la configuración completa que proporciona el *hash* SHA-256 del archivo YAML (una huella que identifica de forma única el contenido exacto del archivo, de modo que cualquier cambio, por mínimo que sea, produce una huella completamente distinta), declarando además las versiones de las librerías principales empleadas. Al finalizar, genera “run_manifest.json” con la marca temporal y el comando exacto. Estos archivos garantizan la reproducibilidad exacta de cualquier resultado.

3.6.1.3 Marcos de referencia del sistema

Un aspecto transversal al *pipeline* es la cadena de marcos de referencia que intervienen en las transformaciones. El sistema opera con cinco marcos: *World* (sistema mundo Unity), *C_left* (cámara izquierda Unity), *C_left_rect* (cámara izquierda rectificada OpenCV), *Model* (sistema local del modelo CAD) y *Controller_GT* (sistema del controlador según el SDK). La Figura 3-45 ilustra esta cadena.



Figura 3-45: Marcos de referencia y cadena SE(3) (Autoría propia)

3.6.2 Parámetros configurables

La configuración del *pipeline* se centraliza en el archivo “config/default_pipeline.yaml”, que actúa como fuente única de verdad para la experimentación reproducible. Cada parámetro puede ser sobrescrito desde la línea de comandos (CLI, *Command-Line Interface*; es decir, la interfaz de texto del terminal), siguiendo una jerarquía de prioridad: CLI explícito > YAML > *defaults* del código. La Tabla 3-11 recoge los parámetros con mayor impacto en la calidad de los resultados.

<i>Categoría</i>	<i>Parámetro</i>	<i>Valor</i>	<i>Efecto</i>
Reconstrucción	<i>z_min</i>	0.20 m	Profundidad mínima del volumen de trabajo
	<i>z_max</i>	2.0 m	Profundidad máxima del volumen de trabajo
	<i>voxel (pointcloud)</i>	0.002 m	Resolución de la nube de puntos
Segmentación	<i>use_roi_gt</i>	true	Habilitar recorte por ROI con <i>ground truth</i>
	<i>roi_sx</i> , <i>roi_sy</i> , <i>roi_sz</i>	0.30 m	Dimensiones de la OBB de recorte
	<i>min_cluster_size</i>	200	Tamaño mínimo de <i>cluster</i> válido
Matching	<i>ppf_rel_sampling</i>	0.05	Submuestreo PPF (relativo al diámetro)
	<i>icp_iterations</i>	100	Iteraciones máximas ICP
	<i>ppf_top_n</i>	5	Hipótesis refinadas con ICP
	<i>crop_radius</i>	0.14 m	Radio de recorte esférico
	<i>ok_min_fitness</i>	0.15	Umbral mínimo de <i>fitness</i>
	<i>ok_max_rmse</i>	0.02 m	Umbral máximo de RMSE
	<i>ok_trans_m</i>	0.05 m	Error de traslación máximo tolerable
	<i>ok_rot_deg</i>	10°	Error de rotación máximo tolerable

Tabla 3-11: Parámetros configurables del *pipeline*

La posibilidad de modificar estos parámetros sin alterar el código fuente es una característica de diseño deliberada que permite variar parámetros y comparar diferentes configuraciones.

3.6.3 Entrada de usuario y ejecución práctica

Desde la perspectiva del usuario, la ejecución del *pipeline* se reduce a editar tres variables en el archivo “run_pipeline.ps1” y ejecutar el *script* desde PowerShell. Estas variables son: la sesión que se pretende procesar, el rango de fotogramas y el controlador objetivo (véase la Figura 3-46).

```
> run_pipeline.ps1 > ...
1 # =====
2 # PIPELINE COMPLETO - Vision Artificial MR
3 # Ejecutar: .\run_pipeline.ps1
4 # =====
5 #
6 # +-----+
7 # |          ZONA DE CONFIGURACION DE USUARIO          |
8 # |          (Modificar según la sesión)                |
9 # +-----+
10
11 $SESSION = "session_20260223_160956" # <-- NOMBRE DE LA SESION
12 $FRAMES  = "88:115"                 # <-- RANGO DE FRAMES (p.ej. "0:115", "0,5,10")
13 $SIDE    = "right"                  # <-- CONTROLADOR ("right" o "left")
14
15 # =====
```

Figura 3-46: Zona de configuración de usuario de "run_pipeline.ps1" (Autoría Propia)

Internamente, este *script* activa el entorno virtual de Python, invoca “run_session_pipeline.py” con la configuración YAML y, tras completar el procesado, genera un *dashboard* HTML interactivo mediante “generate_dashboard.py”. Esta interfaz visual, abordada explícitamente en la sección 4, permite inspeccionar los resultados de la sesión complementando los archivos individuales del procesado, facilitando tanto la validación rápida de resultados como la preparación de material visual para la memoria.

4 RESULTADOS

Una vez descrito en detalle el funcionamiento del sistema completo, este capítulo aborda dos preguntas fundamentales: *cómo se mide la calidad de las estimaciones y qué resultados concretos se han obtenido*.

El capítulo se estructura en dos secciones complementarias. En primer lugar, el apartado 4.1 presenta el marco de evaluación de métricas, que define qué se mide, cómo se mide y qué umbrales determinan el éxito o el fracaso de la estimación de pose. En segundo lugar, el apartado 4.2 aplica ese marco a una sesión real capturada, presentando los resultados cuantitativos obtenidos y su análisis.

La evaluación de los resultados se complementa con un *dashboard* interactivo. Este documento, disponible como archivo HTML autocontenido, se genera al ejecutar el procesamiento de una sesión cualquiera (véase la Figura 4-1) y se incluye explícitamente en el repositorio *PythonProcessing* de GitHub.

```
# =====
# DASHBOARD INTERACTIVO
#
# Genera un archivo HTML autocontenido con visualización interactiva de la sesión (arquitectura, flujo, métricas, 3D).
#
# Requiere:
# - visuals/dashboard_app.js (plantilla JavaScript)
# - visuals/generate_dashboard.py (generador)
# - data/processed/<session>/trajectory_object_world.csv
#
# Salida:
# - visuals/dashboard_<session>.html
#
# Para generar manualmente el dashboard de una sesión concreta:
# python visuals/generate_dashboard.py --session session_20260223_160956
# =====

Write-Host "`n[DASHBOARD] Generando dashboard interactivo..."
python visuals/generate_dashboard.py --session $SESSION
```

Figura 4-1: Generación del *dashboard* interactivo en "run_pipeline.ps1" (Autoría Propia)

El *dashboard* puede abrirse directamente en cualquier navegador web sin necesidad de conexión a internet ni de un servidor web local. Además, el documento se organiza en cuatro pestañas, las cuales tienen propósitos diferentes. A continuación se exponen dichas pestañas, reseñando la función que cumple cada una de ellas.

- **Arquitectura:** Genera un mapa de datos y un mapa de código del repositorio, además de describir los *scripts* principales que componen el algoritmo de visión artificial
- **Flujo de Datos:** Muestra el diagrama del *pipeline* de procesamiento, así como el flujo de archivos principal por etapa y los marcos de referencia SE(3).
- **Métricas:** Proporciona un resumen estadístico de la sesión, acompañado de una guía de métricas y los criterios de clasificación de los fotogramas.
- **Visualización 3D:** Detalla la trayectoria 3D seguida por el controlador durante la sesión de forma interactiva. Además, muestra diversas gráficas que atienden a los componentes de traslación del controlador, diversas métricas de calidad (*fitness*, RMSE, *center distance*, *score*). Finalmente presenta una tabla de trayectoria detallada.

La Figura 4-2 presenta el código QR que proporciona acceso directo al *dashboard* interactivo desde un dispositivo móvil.



Figura 4-2: Código QR para acceder al *dashboard* interactivo

4.1 Fase 5: Evaluación de métricas

La quinta y última fase del *pipeline* tiene por objetivo cuantificar rigurosamente la calidad de las estimaciones de pose obtenidas. A diferencia de las fases anteriores, que transforman datos progresivamente, esta fase no modifica ningún resultado: se limita a medirlo, clasificarlo y agregarlo para facilitar su interpretación.

La evaluación es posible gracias a la existencia de una referencia verdadera y fiable: la pose del controlador proporcionada por el sistema de seguimiento nativo del SDK de Meta Quest 3. Esta referencia (*ground truth*) permite calcular errores absolutos de traslación y rotación, lo que constituye la base del análisis presentado en la siguiente sección.

Para facilitar la comprensión del marco de evaluación, la Figura 4-3 presenta el flujo de archivos principal por etapa del *pipeline*, mostrando qué produce y consume cada fase. Esta visión de conjunto permite situar las métricas en su contexto dentro de la cadena de procesamiento.

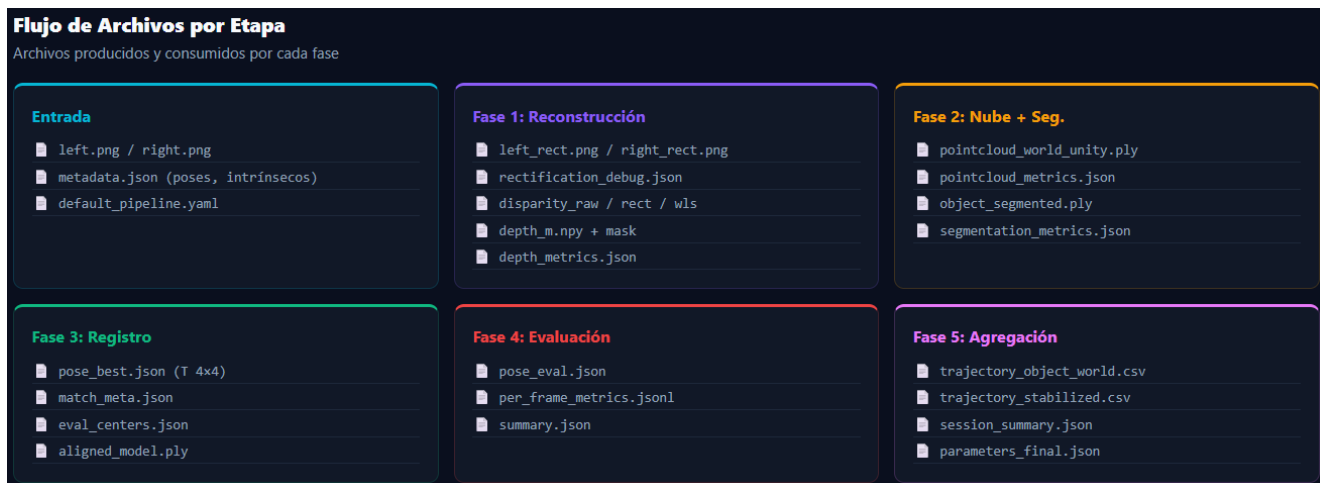


Figura 4-3: Flujo de archivos por etapa del *pipeline* (Autoría propia)

4.1.1 Evaluación por fotograma

La evaluación por fotograma se ejecuta mediante el *script* “run_pose_eval.py”, que compara la transformación estimada (almacenada en “pose_best.json”) con el *ground truth* extraído de “metadata.json”. De esta comparación se obtienen dos métricas fundamentales.

La primera es el error de traslación, calculado como la distancia euclidiana entre los vectores de posición estimado y real. Se expresa en metros y proporciona una medida directa de cuánto se desvía la posición estimada del controlador respecto a su posición real. La segunda es el error de rotación, calculado como el ángulo geodésico entre las matrices de rotación estimada y real. Se expresa en grados y cuantifica la desviación angular de la orientación estimada. Ambos cálculos se implementan en la función “pose_error” del módulo “transforms.py”.

Adicionalmente, se evalúan dos métricas de calidad geométrica del alineamiento, independientes del *ground truth*: *fitness* y RMSE.

- *Fitness*: mide la fracción de puntos del modelo CAD que encuentran correspondencia en la nube segmentada dentro de un radio umbral. Por ejemplo, un *fitness* de 0.5 indica que el 50% de los puntos del modelo están correctamente alineados con la escena. Valores altos reflejan una buena cobertura geométrica.
- RMSE (*Root Mean Square Error*): cuantifica el error cuadrático medio de las correspondencias *inlier*. Se expresa en metros y proporciona una estimación de la precisión puntual del alineamiento. Ambas métricas son calculadas por la función “evaluate_registration” de Open3D.

Un *inlier*, en este contexto, es un punto del modelo CAD que se encuentra suficientemente cerca de al menos un punto de la nube segmentada, dentro del radio umbral configurado. Los puntos que no cumplen esta condición se denominan *outliers* y se excluyen del cálculo del RMSE. El *fitness* es, por tanto, la proporción de *inliers* respecto al total de puntos del modelo.

Los resultados de cada fotograma se almacenan en “pose_eval.json” y se clasifican automáticamente según los umbrales configurados, generando una etiqueta que facilita el análisis estadístico posterior. Los resultados de todos los fotogramas de la sesión se consolidan en el archivo “per_frame_metrics.jsonl”, mediante una línea JSON por cada fotograma.

4.1.2 Guía de métricas del sistema

El sistema calcula nueve métricas distintas a lo largo del *pipeline*. Cada una de ellas tiene un rango, un archivo fuente y unos umbrales que clasifican el resultado como excelente, aceptable o problemático. La Tabla 4-1 recoge esta información de forma compacta para servir como referencia rápida durante el análisis de resultados.

Métrica	Archivo fuente	Descripción	Rango	Excelente	Aceptable	Problema
<i>fitness</i>	pose_best.json	Fracción de correspondencias ICP	[0, 1]	> 0.5	0.15 – 0.5	< 0.15
<i>RMSE</i>	pose_best.json	Error cuadrático medio ICP (m)	[0, ∞)	< 0.004	0.004 – 0.02	> 0.02
<i>score</i>	pose_best.json	Votos PPF del mejor candidato	[0, ∞)	> 100	50 – 100	< 50
<i>center_distance</i>	eval_centers.json	Distancia entre el centroide alineado y el elemento segmentado	[0, ∞)	< 0.02 m	0.02 – 0.05 m	> 0.05 m
<i>trans_error</i>	pose_eval.json	Error de traslación	[0, ∞)	< 0.02 m	0.02 – 0.05 m	> 0.05 m
<i>rot_error</i>	pose_eval.json	Error de rotación geodésico	[0°, 180°]	< 10°	10° – 30°	> 30°
<i>det_R</i>	trajectory.csv	Ortogonalidad de las transformaciones estimadas	{1.0}	$ \det-1 < 1e^{-6}$	$ \det-1 < 0.01$	$ \det-1 > 0.01$
<i>n_out</i>	seg_metrics	Puntos segmentados finales	[0, ∞)	> 1000	300 – 1000	< 300
<i>ok_rate</i>	session_summary	Tasa de éxito de la sesión	[0, 1]	> 0.8	0.5 – 0.8	< 0.5

Tabla 4-1: Guía de métricas del sistema

A modo de aclaración y para una mejor comprensión de la Tabla 4.1, conviene definir las métricas que todavía no se han abordado en profundidad:

- *score*: es el número de votos que acumuló la mejor hipótesis durante la votación PPF (véase el apartado 3.5.4.2). Un valor alto indica que muchos pares de la escena coinciden geoméricamente con el modelo, reforzando la confianza en la hipótesis seleccionada.
- *center_distance*: mide la distancia euclidiana entre el centroide del modelo alineado y el de la nube segmentada. Complementa al RMSE, que evalúa la precisión local punto a punto, añadiendo una medida de la posición global del objeto.

- *det_R*: es el determinante de la submatriz de rotación 3×3 (como se explicó en el apartado 3.5.4.4, toda rotación válida tiene determinante +1, por lo que esta métrica actúa como verificación de integridad).
- *ok_rate*: es la proporción de fotogramas clasificados como “OK” respecto al total procesado, es decir, la tasa de éxito de la sesión.

Los umbrales de la columna «Excelente» representan el rendimiento deseable del sistema. Los umbrales «Aceptable» delimitan la zona operativa en la que el sistema produce resultados utilizables, aunque con margen de mejora. La columna «Problema» señala valores que indican un fallo en alguna etapa del *pipeline*.

Estos umbrales no se escogen de forma arbitraria buscando la validación expresa de los resultados obtenidos, si no que atienden a los propios límites físicos que es capaz de percibir el ser humano al comprobar la superposición de un objeto virtual sobre uno real al emplear un dispositivo de realidad mixta.

4.1.3 Evaluación de sesión y trayectoria

Al finalizar el procesamiento de todos los fotogramas, el *script* “run_session_summary.py” genera los archivos de resumen a nivel de sesión.

El archivo “trajectory_object_world.csv” recoge, para cada fotograma, la pose estimada del controlador en coordenadas mundo Unity, junto con los errores de traslación y rotación, el *fitness*, el RMSE y la clasificación asignada. Este archivo constituye la serie temporal principal sobre la que se analizan los resultados.

Sobre esta trayectoria se aplica un módulo de estabilización temporal, implementado en “temporal_stabilizer.py”. La estabilización utiliza un filtro de media móvil exponencial (EMA), que calcula la pose estabilizada de cada fotograma como una combinación ponderada entre la estimación actual y la pose estabilizada del fotograma anterior. El factor *alpha* (configurado en 0.4) controla ese balance: con un valor de 0.4, cada pose estabilizada incorpora un 40% de la estimación nueva y un 60% de la anterior, produciendo un suavizado que amortigua oscilaciones sin introducir un retardo excesivo.

Además, se aplican umbrales de rechazo de saltos bruscos: si entre dos fotogramas consecutivos la traslación varía más de 0.15 m o la rotación más de 45° , se considera que la estimación es anómala y se mantiene la pose anterior. Esto evita que un único fotograma con error grande contamine la trayectoria suavizada. El resultado se guarda en “trajectory_stabilized.csv”.

El archivo “session_summary.json” resume la sesión completa con estadísticas agregadas: tasa de fotogramas con estimación exitosa, percentiles del error de traslación y rotación (P50 como mediana, reflejando el error típico que cometen la mitad de los fotogramas; y P90 como indicador del peor comportamiento en el conjunto de los fotogramas analizados), *fitness* medio y distribución de causas de fallo.

El archivo “summary_frames.json” proporciona un resumen por fotograma que permite identificar rápidamente los fotogramas problemáticos y sus razones de fallo.

4.1.4 Reporting y visualización interactiva

El módulo `reporting.py` centraliza las funciones de escritura en tres formatos. Los archivos JSON (JavaScript Object Notation) se escriben con indentación de 2 espacios y codificación UTF-8, el estándar universal de representación de caracteres. Los archivos JSONL (JSON Lines) emplean una variante donde cada línea es un registro independiente, lo que permite procesar archivos extensos línea a línea; este es el formato de `per_frame_metrics.jsonl`, donde cada línea corresponde a un fotograma. Los archivos CSV (*Comma-Separated Values*) utilizan el formato tabular estándar, compatible con hojas de cálculo como Microsoft Excel. Todos los archivos de métricas del *pipeline* utilizan este módulo para garantizar un formato consistente que facilita tanto la lectura humana como el procesamiento automático.

Como complemento visual, el *script* `generate_dashboard.py` previamente mencionado genera un archivo HTML autocontenido que integra una interfaz interactiva desarrollada con React (biblioteca de JavaScript para componentes visuales dinámicos), cuyo código fuente reside en `dashboard_app.jsx`. Este *dashboard* permite visualizar la arquitectura del repositorio, el flujo del *pipeline*, las métricas de cada fotograma, la trayectoria 3D estimada y los gráficos de evolución temporal. Al ser un archivo único con las dependencias embebidas (React, Three.js y estilos CSS incluidos directamente, sin necesidad de descarga externa), puede compartirse y abrirse en cualquier navegador moderno sin instalar software adicional. Esta decisión de diseño facilita la revisión de resultados para futuros desarrolladores del proyecto.

Esta arquitectura de evaluación (desde las métricas por fotograma hasta la visualización interactiva de sesión completa) proporciona el soporte cuantitativo necesario para analizar los resultados con rigor y fundamentar las conclusiones del proyecto.

Establecido el marco de evaluación, el siguiente apartado aplica estas métricas a una sesión experimental real capturada con el sistema descrito en el capítulo 3. El objetivo es doble: demostrar la viabilidad operativa del *pipeline* y cuantificar su precisión frente al *ground truth* del sistema de seguimiento nativo del visor.

4.2 Resultados

Este apartado presenta los resultados obtenidos por el sistema sobre una sesión de captura real. El objetivo es doble: por un lado, demostrar que el sistema es capaz de estimar la pose de un objeto a partir de imágenes estereoscópicas; por otro, cuantificar la precisión alcanzada e identificar las limitaciones del enfoque.

4.2.1 Descripción de la sesión analizada

La sesión seleccionada para la evaluación es `session_20260223_160956`, capturada en el entorno interior del laboratorio.

El controlador se sitúa, durante la sesión, a una distancia de entre 0.5 y 1.5 m del con respecto a los visores, realizando movimientos deliberados de traslación y rotación para cubrir un rango representativo con diferentes posiciones y orientaciones.

La sesión completa comprende 116 fotogramas (del 0 al 115). El controlador derecho aparece de forma visible y continua en el campo de visión (FOV) de las cámaras entre los fotogramas 88 y 111. En el resto (0-87 y 113-115) se encuentra fuera del alcance visual o es apenas perceptible, impidiendo su

segmentación. La decisión de acotar el análisis a dichos fotogramas (88-112) es consciente y justificada, pues se busca evaluar el rendimiento del *pipeline* exclusivamente cuando objeto de interés está presente. Incluir fotogramas sin controlador contaminaría las métricas con fallos que no reflejan deficiencias del algoritmo, sino la mera ausencia del objeto.

Para ilustrarlo, se incorpora deliberadamente el fotograma 112, en el que el controlador se sitúa en el borde extremo del campo de visión. En este caso la segmentación no produce una nube válida, evidenciando esta limitación física y confirmando que la ventana temporal útil termina en el fotograma 111.

Así, la sesión procesada comprende 25 fotogramas consecutivos (del 88 al 112), ejecutados íntegramente con la configuración por defecto de *default_pipeline.yaml*. De ellos, 24 completan satisfactoriamente todas las etapas y reciben la clasificación “OK”. El único fotograma restante (el 112) falla en segmentación al no generarse *object_segmented.ply*, recibiendo la etiqueta *no_segment*. En este marco, se obtiene una tasa de éxito del 96%. Si bien 25 fotogramas constituyen un conjunto reducido para extraer conclusiones estadísticas definitivas, la coherencia interna de los resultados y la ausencia de anomalías sistemáticas permiten validar el funcionamiento del sistema. Además, durante la validación del sistema logrado, se probaron sesiones en condiciones similares, obteniendo resultados igualmente válidos, lo que fortalece la validación de la sesión objetivo aquí presentada. La ampliación a múltiples sesiones y condiciones se contempla explícitamente como línea futura en el Capítulo 5.

4.2.2 Resumen cuantitativo de la sesión

Los resultados de la sesión confirman la viabilidad del enfoque propuesto. De los 25 fotogramas procesados, 24 producen estimaciones de pose válidas, lo que sitúa la tasa de éxito en el 96%. Las métricas de calidad del alineamiento se mantienen estables a lo largo de toda la sesión, logrando precisión milimétrica en el RMSE y valores de *fitness* coherentes con las limitaciones inherentes a la captura estereoscópica desde un único punto de vista.

La Figura 4-4 y la Tabla 4-2 presentan una visión global de los resultados de la sesión. Las seis tarjetas de resumen del *dashboard* condensan los indicadores clave de rendimiento.



Figura 4-4: Resumen cuantitativo de la sesión (Autoría propia)

Los resultados ponen de manifiesto varios puntos clave. El *fitness* promedio, situado en 0.467, se encuadra dentro del rango “aceptable” (0.15–0.5), lo que implica que cerca de la mitad de los puntos del modelo CAD logran correspondencias válidas en la nube segmentada. Este comportamiento es coherente con la propia naturaleza del problema: se trata de alinear un modelo completo con reconstrucciones

estereoscópicas que, por definición, son incompletas, ya que el controlador solo se observa desde un único punto de vista en cada instante.

Métrica	Valor
Fotogramas totales	25
Fotogramas validos (OK)	24
Fotogramas fallidos	1 (fotograma 112, <i>no_segment</i>)
Tasa de éxito	96%
<i>Fitness</i> medio $\pm \sigma$	0.469 ± 0.124
RMSE medio $\pm \sigma$	4.84 ± 0.27 mm
<i>Center distance</i> medio $\pm \sigma$	4.32 ± 2.16 cm
<i>Score</i> medio	113
<i>SE(3)</i> válido	100% (24/24)

Tabla 4-2: Resumen estadístico de la sesión analizada

Este contexto explica por qué no es realista aspirar a un *fitness* de 1.0. La captura estéreo limita la visibilidad a una sola perspectiva, dejando siempre ocultas ciertas superficies. Además, la reconstrucción presenta carencias en regiones con poca textura y puede incluir huecos derivados de errores de disparidad. Frente a ello, sistemas como el *tracking* nativo de Meta Quest 3 disponen de conocimiento previo de la geometría del objeto, utilizan sensores infrarrojos específicos y aplican técnicas avanzadas de fusión sensorial, lo que les permite alcanzar niveles de precisión significativamente superiores. En consecuencia, considerar valores de *fitness* superiores a 0.5 como “excelentes” tiene en cuenta esta diferencia de condiciones, ya que estimar la pose únicamente a partir de cámaras pasivas y sin información previa supone una dificultad mucho mayor.

En cuanto al RMSE, el valor establecido como “excelente” (4 mm) se traduce en un nivel de precisión que resulta prácticamente imperceptible en entornos de realidad mixta. El RMSE medio obtenido, de 4.84 mm, también se sitúa en el rango “aceptable” (4–20 mm), lo que indica que las correspondencias válidas presentan una precisión milimétrica consistente. Además, la desviación estándar (0.27 mm) evidencia una estabilidad elevada en el comportamiento del algoritmo ICP a lo largo de toda la sesión.

Por su parte, el *center distance*, con un límite de 2 cm, representa la tolerancia en la posición global, donde desviaciones inferiores no generan discrepancias visuales apreciables al superponer contenido virtual. Los criterios de traslación (< 2 cm) y rotación ($< 10^\circ$) siguen las referencias del *Benchmark BOP* descrito en el apartado 2.7.3. El *center distance* medio de 4.27 cm evidencia una mayor dispersión ($\sigma = 2.16$ cm), indicando que, aunque el alineamiento punto a punto sea preciso, la posición global del modelo alineado respecto al centroide de la nube segmentada varía entre fotogramas. Esta métrica es sensible a la completitud de la segmentación y a la simetría de la nube visible.

Finalmente, el 100% de las transformaciones estimadas son matrices $SE(3)$ válidas (ortogonalidad correcta, $\det(R) \approx 1.0$), lo que confirma que el *pipeline* no produce poses geoméricamente degeneradas.

4.2.3 Análisis de la trayectoria estimada

La Figura 4-5 muestra la trayectoria 3D estimada del controlador en coordenadas mundo Unity. Cada punto de la trayectoria se colorea según el valor de *fitness* del fotograma correspondiente: verde ($fitness > 0.5$), amarillo ($0.3 - 0.5$) o rojo (< 0.3). Esta codificación permite identificar de un vistazo las zonas de la trayectoria donde la estimación es más fiable.

La representación interactiva de la trayectoria 3D tiene un propósito concreto: ofrecer una visión espacial intuitiva de la estimación de pose a lo largo de la sesión, de modo que se pueda visualizar rápidamente si las posiciones estimadas forman un recorrido coherente o si existen saltos anómalos.

El *fitness* se elige como métrica relevante pues es el indicador que mejor resume la confianza global de cada estimación, integra cobertura geométrica y calidad del alineamiento. Además, su valor es independiente del *ground truth*, lo que lo convierte en un indicador intrínseco del *pipeline*.

En la visualización, el fotograma 112 (el único clasificado como FAIL) se muestra con una marca en forma de cruz en lugar de un punto circular, señalando que es un fotograma no válido dentro de la trayectoria.

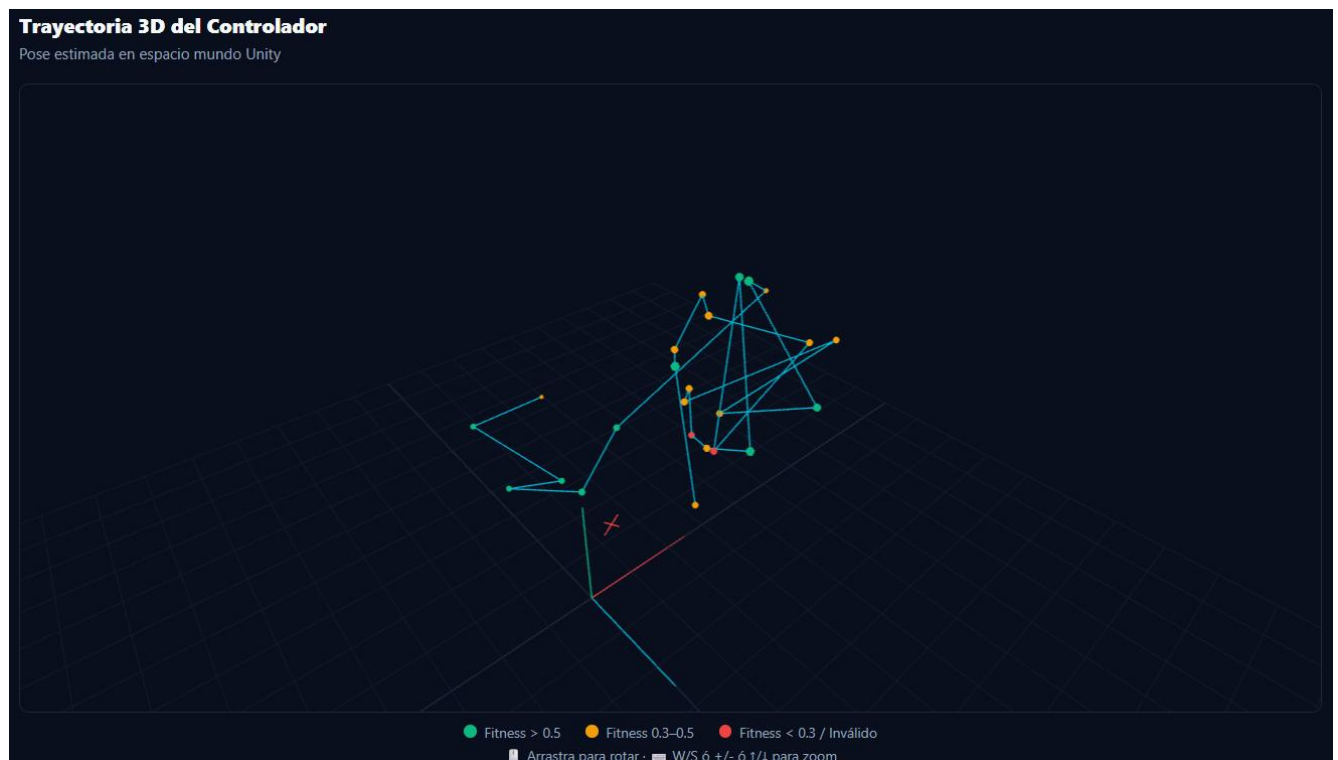


Figura 4-5: Trayectoria 3D estimada del controlador (Autoría propia)

La trayectoria describe un recorrido coherente en el espacio, sin saltos bruscos ni discontinuidades evidentes. El controlador se desplaza desde una posición inicial cercana al centro de la escena hacia posiciones más alejadas.

La Figura 4-6 descompone el movimiento del controlador en sus tres componentes de traslación (t_x , t_y , t_z) a lo largo del tiempo. La componente t_z (profundidad) se mantiene relativamente estable, coherente con la postura del usuario sosteniendo el controlador con el brazo extendido. Las componentes t_x (desplazamiento horizontal) y t_y (altura) presentan mayor variabilidad, reflejando los movimientos deliberados del usuario durante la captura.

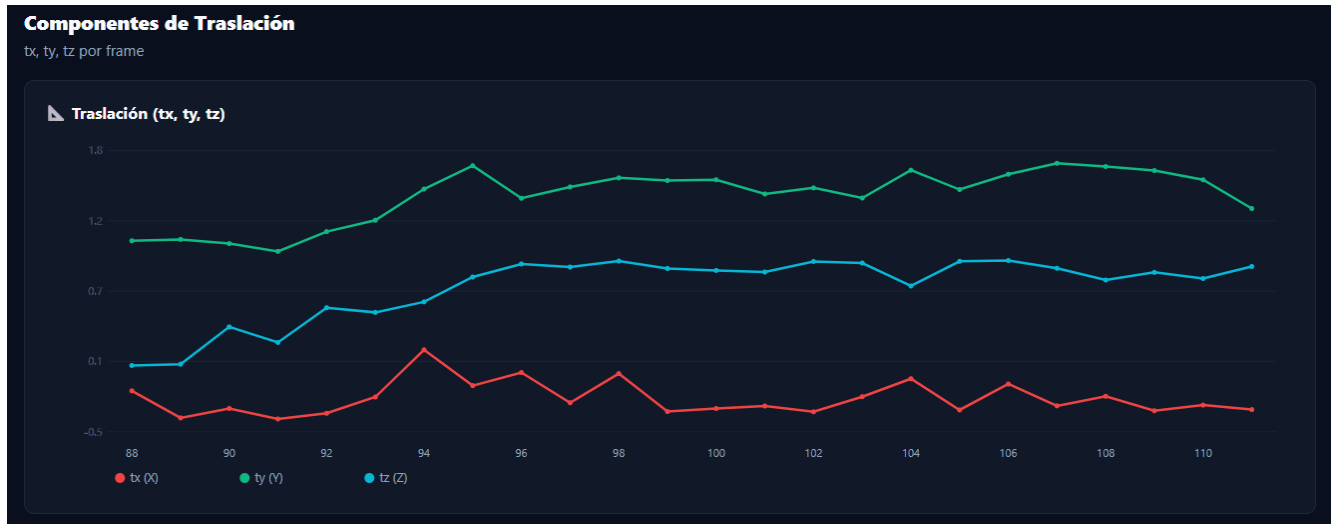


Figura 4-6: Componentes de traslación por fotograma (Autoría propia)

Un aspecto notable es la suavidad de las curvas: no se observan saltos repentinos entre fotogramas consecutivos, lo que confirma que tanto el *pipeline* de estimación como el filtro EMA de estabilización temporal (descrito en el apartado 4.1.3) funcionan correctamente.

Los fotogramas con *fitness* más bajo (101 con 0.237 y 105 con 0.241) se sitúan en zonas donde el controlador muestra una vista lateral estrecha a las cámaras, lo que reduce la superficie visible y, en consecuencia, el número de puntos disponibles para el alineamiento.

4.2.4 Evolución temporal de las métricas

Las siguientes figuras presentan la evolución por fotograma de las tres métricas de calidad principales: *fitness*, RMSE y *center distance*. El análisis conjunto de estas tres curvas permite identificar patrones y correlaciones que, considerados de forma aislada, pasarían inadvertidos. En particular, se busca determinar si existe una relación sistemática entre la orientación del controlador y el rendimiento global del sistema.

4.2.4.1 Fitness

La Figura 4-7 muestra la evolución del *fitness* por fotograma. Los valores oscilan entre 0.237 (fotograma 101) y 0.764 (fotograma 93), con una media de 0.467, pero se observa una tendencia general: hay saltos constantes entre valores altos y bajos, demostrando mucha variabilidad.

Esto se explica perfectamente mediante la orientación del controlador respecto a las cámaras. El objeto rota sobre su propio eje a la vez que se desplaza sobre los visores. Así, cuando el controlador muestra su cara frontal (más amplia), el *fitness* sube y cuando se gira mostrando una vista lateral más estrecha, descende.

Estos resultados son perfectamente coherentes, demostrando que la estimación de pose depende en gran medida de la reconstrucción estereoscópica, la cual depende a su vez de la vista del objeto que pueden captar los visores. Las asimetrías del controlador justifican que ante objetos simétricos (por ejemplo, figuras simples como esferas o prismas) estos resultados deberían ser casi totalmente homogéneos.

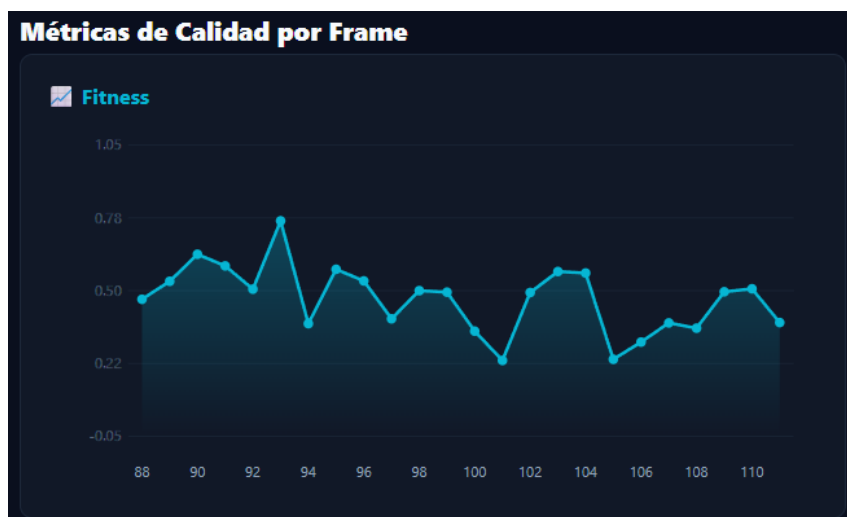


Figura 4-7: Evolución del *fitness* por fotograma (Autoría propia)

4.2.4.2 RMSE

La Figura 4-8 muestra la evolución del RMSE por fotograma. A primera vista, la gráfica presenta picos y valles que sugieren una variabilidad significativa. Sin embargo, es imprescindible observar la escala del eje vertical: los valores oscilan entre 4.41 mm (fotograma 90) y 5.50 mm (fotograma 101), un rango total entre extremos de poco más de un milímetro.

La apariencia de variabilidad pronunciada se debe al autoajuste de la escala, que amplifica visualmente diferencias milimétricas. En la práctica, esta estabilidad indica que la precisión puntual del alineamiento ICP es consistente con independencia de la posición u orientación del controlador.

Los fotogramas con menor *fitness* tienden a presentar un RMSE ligeramente superior, lo que resulta esperable: con menos correspondencias válidas, el ajuste pierde algo de robustez.

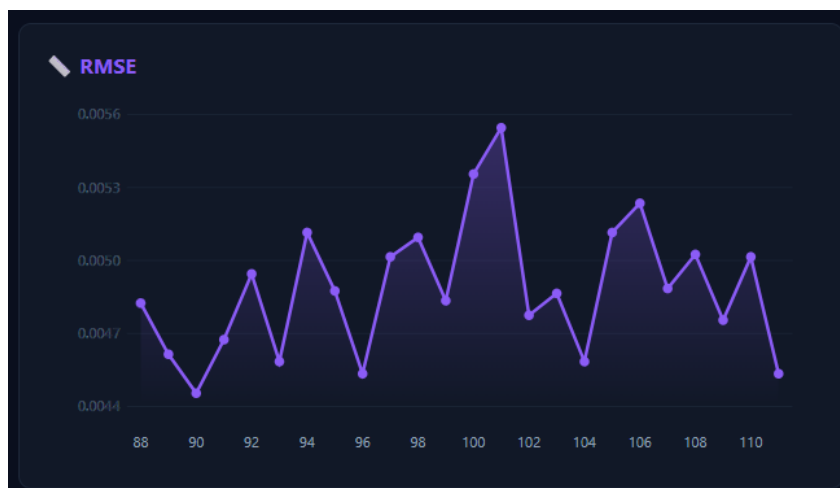


Figura 4-8: Evolución del RMSE por fotograma (Autoría propia)

4.2.4.3 Center distance

La Figura 4-9 muestra la distancia entre el centroide del modelo alineado y el centroide de la nube segmentada, métrica denominada *center distance*.

Mientras el RMSE evalúa la calidad del ajuste punto a punto (precisión local), el *center distance* mide si el modelo está globalmente bien posicionado respecto al objeto real (precisión global). Los valores varían entre 1.37 cm (fotograma 104) y 9.15 cm (fotograma 100), con una media de 4.27 cm.

Esta dispersión, la mayor de las tres métricas principales, refleja la sensibilidad del centroide segmentado a la fracción visible del controlador: cuando la segmentación captura la cara frontal completa, el centroide se aproxima al centro real del objeto y la métrica mejora; pero cuando la vista es parcial, el centroide se desplaza hacia la zona visible y la distancia crece.

A pesar de ello, los valores se mantienen mayoritariamente por debajo de 5 cm, lo que indica un alineamiento global razonablemente bueno.

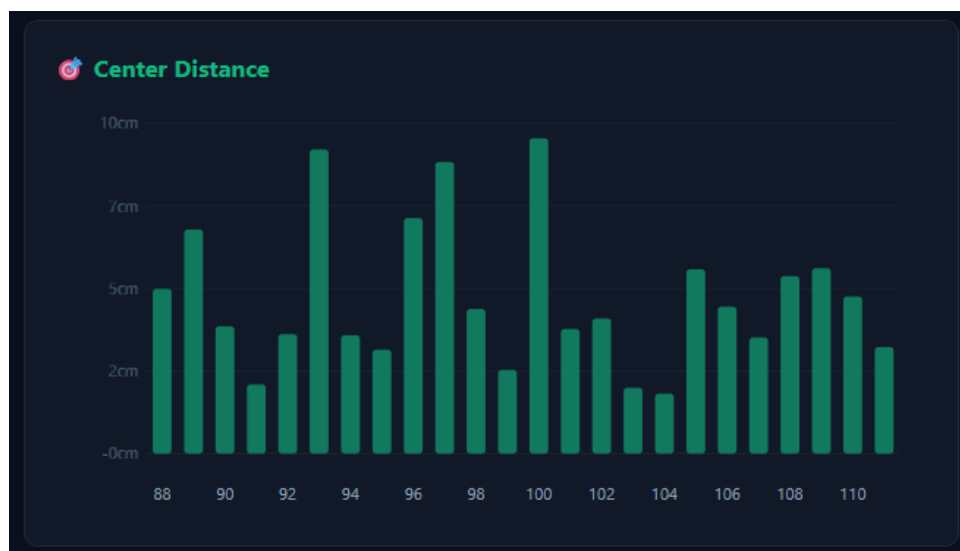


Figura 4-9: Distancia al centroide por fotograma (Autoría propia)

El análisis conjunto de las tres métricas de calidad revela un patrón coherente: los fotogramas donde el controlador presenta su cara frontal amplia obtienen simultáneamente mejor *fitness* (mayor cobertura), mejor RMSE (mayor precisión local) y menor *center distance* (mejor posicionamiento global).

Esta correlación confirma que la orientación del objeto respecto a las cámaras es el factor determinante del rendimiento del sistema, lo que resulta coherente con las limitaciones inherentes a la captura estereoscópica desde un punto de vista único. En consecuencia, las líneas de mejora prioritarias (desambiguación rotacional y fusión multivista, descritas en el Capítulo 5) se orientan precisamente a mitigar esta dependencia.

4.2.5 Datos por fotograma

La Tabla 4-3 recoge un subconjunto de fotogramas seleccionados por su representatividad: el mejor fotograma global (90, con métricas excelentes en todas las categorías), el de mayor *fitness* y *score* (93), dos fotogramas con *fitness* inferior al umbral de excelencia, pero clasificados como “OK” (101, 105), dos con *center distance* elevado (100, 108), un fotograma representativo del rendimiento medio (96) y

el único fallido (112). El conjunto completo puede consultarse en la trayectoria detallada del *dashboard* interactivo.

<i>Frame</i>	<i>Estado</i>	t_x, t_y, t_z (m)	<i>Fitness</i>	RMSE (mm)	<i>Score</i>	<i>Center distance</i> (cm)
88	OK	-0.136, 1.077, 0.069	0.467	4.78	112	4.56
90	OK+	-0.278, 1.054, 0.381	0.637	4.41	153	3.42
93	OK+	-0.186, 1.243, 0.498	0.764	4.54	183	8.81
96	OK+	0.012, 1.421, 0.890	0.537	4.49	129	6.72
100	OK	-0.279, 1.571, 0.837	0.347	5.31	83	9.15
101	WARN	-0.259, 1.456, 0.825	0.237	5.50	57	3.34
104	OK+	-0.038, 1.648, 0.712	0.567	4.54	136	1.37
105	WARN	-0.291, 1.492, 0.911	0.241	5.07	58	5.16
109	OK	-0.297, 1.645, 0.822	0.496	4.71	119	5.19
112	FAIL	-0.306, 1.112, 0.626	-	-	-	-

Tabla 4-3: Trayectoria detallada. Subconjunto representativo de 10 fotogramas

Cabe aclarar un aspecto que podría generar dudas: fotogramas como el 101 o el 105 presentan un *fitness* bajo (0.237 y 0.241) y, sin embargo, reciben la clasificación “OK”. Esto se debe a que la etiqueta “OK” se asigna exclusivamente en función de los errores de traslación y rotación respecto al *ground truth* (< 0.05 m y $< 10^\circ$ respectivamente), no del *fitness*.

El *fitness* refleja la cobertura geométrica del alineamiento entre modelo y nube, mientras que la clasificación “OK” evalúa la precisión de la pose estimada respecto a la posición real. Un fotograma puede tener *fitness* bajo (poca cobertura) y aun así producir una pose correcta, porque el algoritmo PPF+ICP es capaz de converger hacia la posición adecuada incluso con correspondencias parciales, siempre que la fracción visible del controlador sea suficientemente informativa. Dicho de otro modo, el *fitness* indica la dificultad del alineamiento, pero no determina su resultado final en términos de precisión de pose.

Con respecto a la Tabla 4.1, esta pone de manifiesto varios patrones. Los fotogramas 90 y 93 destacan como los mejores de la sesión. El fotograma 90 cumple con todas las métricas de la mejor forma, si se tiene en cuenta el conjunto completo de ellas. El fotograma 93 tiene el mayor *fitness* y el mejor *score*. En estos casos, el algoritmo encontró una cobertura geométrica excepcionalmente buena, porque el controlador presentaba su cara más ancha a las cámaras.

En el extremo opuesto, el fotograma 105 es válido, pero se encuentra fuera de los umbrales de *fitness* y *center distance* y el fotograma 101 registra el *fitness* y *score* más bajos, aunque aún cumplen los umbrales mínimos para su clasificación como “OK”.

El fotograma 112, como ya se ha comentado, es especialmente ilustrativo de los límites del sistema. En él, el controlador se encuentra prácticamente fuera del campo de visión, por lo que la segmentación no logró producir una nube de puntos válida. Al no generarse el archivo “object_segmented.ply”, el *matching* PPF+ICP no llegó a ejecutarse, y en consecuencia no existen valores de *fitness* ni de *score* para este fotograma. Su clasificación como *no_segment* demuestra que el sistema de control de calidad (el *gate pre-matching* descrito en el apartado 3.5.5) funciona correctamente, detectando los fotogramas con datos insuficientes antes de ejecutar un *matching* que sería inútil y evitando tanto el coste computacional como la generación de estimaciones potencialmente corruptas.

4.2.6 Discusión de los resultados

Los resultados de la sesión analizada permiten extraer varias conclusiones operativas sobre el rendimiento del sistema.

4.2.6.1 Viabilidad del enfoque

La tasa de éxito del 96% demuestra que el *pipeline* estereoscópico basado en PPF+ICP es capaz de estimar la pose 6D del controlador de forma fiable en condiciones de interior controladas. De los 25 fotogramas procesados, únicamente se registra un fallo (por encontrarse el controlador fuera del FoV de los visores), lo que sugiere que el *matching* PPF+ICP es robusto cuando recibe datos de entrada válidos.

4.2.6.2 Precisión del alineamiento

El RMSE medio de 4.84 mm indica que el alineamiento punto a punto entre el modelo CAD y la nube segmentada alcanza una precisión milimétrica consistente. Este nivel de precisión es comparable al reportado en el apartado 2.5.4 para métodos PPF+ICP en condiciones de laboratorio, y es más que suficiente para aplicaciones de realidad mixta donde el usuario no suele percibir errores de posicionamiento inferiores a 1 cm.

4.2.6.3 Cobertura geométrica

El *fitness* medio de 0.467 refleja una limitación inherente al enfoque estereoscópico, ya que al capturar el controlador desde un único punto de vista, solo una fracción de su superficie es visible en cada fotograma.

Esto explica consistentemente el ruido generado en la reconstrucción 3D mediante nubes de puntos e implica que el algoritmo trabaja consistentemente con nubes parciales, lo que dificulta alcanzar valores de *fitness* superiores a 0.7. Aun así, los resultados demuestran que PPF+ICP es capaz de converger correctamente con coberturas competentes en materia de estimación de pose.

4.2.6.4 Estabilidad temporal

La continuidad de la trayectoria estimada y la ausencia de saltos bruscos entre fotogramas consecutivos confirman que la combinación del *pipeline* de estimación con el filtro EMA de estabilización produce resultados temporalmente coherentes.

Esta propiedad es esencial para aplicaciones de realidad mixta, donde las discontinuidades de pose se perciben como saltos visuales disruptivos que molestan al usuario, propiciando la pérdida parcial de la capacidad inmersiva inherente de estos dispositivos.

4.2.6.5 Limitaciones identificadas

El análisis revela tres limitaciones principales: la dependencia de la segmentación, la sensibilidad a la orientación del controlador y la variabilidad del *center distance*.

La primera limitación es la dependencia de la segmentación. El único fallo de la sesión analizada se produjo porque la etapa de segmentación no generó una nube de puntos válida, situación extensible a todos los fotogramas donde el controlador no aparece en el campo de visión (0-87 y 113-115). Esto pone de manifiesto que, si el objeto de interés no se aísla correctamente, las etapas posteriores no pueden ejecutarse.

Ahora bien, es importante matizar que esta dependencia no constituye un fallo del algoritmo, pues la segmentación cumple su función correctamente al no intentar estimar la pose de un objeto que no está presente. La limitación reside, más bien, en que el sistema actual carece de un mecanismo que detecte automáticamente si el objeto es visible antes de iniciar el procesamiento, lo cual podría abordarse en futuros desarrollos.

En segundo lugar, se establece la sensibilidad a la orientación del controlador como uno de los mayores cuellos de botella del *pipeline* durante el desarrollo del proyecto. Aunque la traslación (posición) es la magnitud más intuitiva y la primera que se busca estimar, la orientación (rotación) ha supuesto un reto de gran complejidad técnica.

La raíz del problema es la desambiguación. El controlador Touch Plus presenta regiones de geometría similar vistas desde distintos ángulos, especialmente en su perfil lateral estrecho, lo que puede generar hipótesis de orientación incompatibles durante la votación PPF. En estos casos, el refinamiento ICP puede converger hacia una orientación incorrecta que, sin embargo, produce métricas locales aparentemente razonables.

Los fotogramas donde el controlador presenta su cara frontal amplia obtienen *fitness* superiores a 0.5 y errores de orientación menores, porque la geometría visible resulta más informativa y reduce la ambigüedad. Las vistas laterales estrechas, en cambio, degradan tanto la cobertura como la fiabilidad de la orientación estimada.

Este hallazgo sugiere que, para objetos asimétricos, la estimación de pose mejora sustancialmente cuando el objeto muestra su cara geoméricamente más rica, y que futuros desarrollos deberían incorporar mecanismos de detección de ambigüedad rotacional.

En tercer y último lugar, la variabilidad del *center distance*, que oscila entre 1.37 y 9.15 cm. Esta dispersión indica que, aunque el alineamiento local (RMSE) es preciso, la posición global estimada del objeto puede variar varios centímetros entre fotogramas. En aplicaciones donde se requiera posicionamiento absoluto de alta precisión, esta variabilidad podría ser limitante.

4.2.7 Valoración global

Los resultados experimentales validan la hipótesis central del proyecto: es posible estimar la pose 6D de un objeto físico utilizando exclusivamente las cámaras pasivas de un visor de consumo, sin sensores adicionales. El sistema alcanza una tasa de éxito del 96%, con precisión milimétrica en el alineamiento y estabilidad temporal suficiente para aplicaciones de realidad mixta.

Las limitaciones identificadas (dependencia de la segmentación, sensibilidad a la orientación y variabilidad del *center distance*) no invalidan el enfoque, sino que definen el espacio de mejora para los trabajos futuros descritos en el Capítulo 5. Considerando el contexto del proyecto (un TFG de carácter exploratorio que implementa un *pipeline* completo desde cero utilizando únicamente las cámaras pasivas de un casco de consumo), los resultados obtenidos constituyen una base cuantitativa sólida y reproducible sobre la que las sucesivas iteraciones del proyecto podrán construir.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

El objetivo de este Trabajo Fin de Grado era implementar, evaluar y documentar un sistema de visión artificial capaz de estimar la pose 6D de un objeto físico en un entorno de realidad mixta, utilizando las cámaras estéreo del visor Meta Quest 3. Tras el desarrollo completo del *pipeline* y el análisis de los resultados experimentales, se concluye que todos los objetivos planteados en el Capítulo 1 se han alcanzado satisfactoriamente.

En lo que respecta al primer objetivo específico, se ha diseñado e implementado una aplicación en Unity que captura pares de imágenes estéreo desde el visor Meta Quest 3, junto con los metadatos geométricos asociados (parámetros intrínsecos, poses de cámaras y poses de los controladores como referencia). La aplicación genera una estructura de datos organizada por sesión y fotograma.

El segundo objetivo, relativo al desarrollo de un *pipeline* de postprocesado en Python, se ha cumplido mediante la implementación secuencial de diferentes etapas para la reconstrucción estereoscópica 3D. El *pipeline* es configurable, reproducible y modular, lo que facilita tanto su mantenimiento como su extensión futura.

El tercer y cuarto objetivos se han materializado en la implementación del módulo de segmentación geométrica y de la estimación de pose mediante el algoritmo *Point Pair Features* (PPF) con refinamiento *Iterative Closest Point* (ICP). La combinación de ambas técnicas ha demostrado ser capaz de estimar la pose del controlador Touch Plus con una tasa de éxito del 96% y precisión milimétrica en el alineamiento, validando la viabilidad técnica del enfoque.

El quinto objetivo, la definición de un marco de evaluación cuantitativa, se ha implementado mediante nueve métricas complementarias que comparan las poses estimadas con el *ground truth* proporcionado por el sistema de seguimiento nativo del visor. El marco de evaluación incluye umbrales calibrados, clasificación automática de fotogramas y un *dashboard* interactivo para la visualización de resultados.

Finalmente, el sexto objetivo se ha cubierto con el análisis detallado de los resultados (Capítulo 4), la identificación de tres limitaciones principales y la formulación de líneas de mejora concretas para futuros trabajos.

La aportación principal de este TFG al campo de estudio es la demostración de que las cámaras pasivas de un visor de consumo son suficientes para la estimación de pose 6D. Esta conclusión tiene implicaciones directas para futuras investigaciones en detección de objetos en realidad mixta con *hardware* accesible y económico, y abre la puerta a la integración de estos sistemas en escenarios de entrenamiento y simulación.

Las tres limitaciones identificadas (dependencia de la calidad de la segmentación, sensibilidad a la orientación del objeto y variabilidad del posicionamiento global) delimitan el alcance del sistema actual y definen los ejes de mejora para la continuación del proyecto, detallados en el apartado 0.

5.2 Valoración personal

Este proyecto ha supuesto un recorrido formativo de gran amplitud. Se han abarcado conceptos previamente desconocidos, nuevos entornos de desarrollo y campos ingenieriles completamente nuevos. El manejo de entornos de realidad mixta con Unity y la naturaleza multidisciplinar del trabajo, combinando fundamentos matemáticos, ingeniería de software y evaluación experimental, han proporcionado una visión integral de lo que implica desarrollar un sistema tecnológico completo desde cero.

Entre los retos específicos afrontados, cabe destacar la complejidad de integrar hardware de realidad mixta con un *pipeline* de visión artificial. La documentación de la Passthrough Camera API de Meta Quest 3 presentaba errores significativos relacionados con la compilación del proyecto, lo que exigió un esfuerzo considerable de experimentación y depuración. Asimismo, la resolución de la desambiguación rotacional y la discrepancia de parámetros intrínsecos (documentada en el Capítulo 3) constituyeron problemas técnicos que requirieron un análisis riguroso para su resolución. La curva de aprendizaje en visión artificial tridimensional, partiendo de una formación base en ingeniería mecánica, ha sido pronunciada pero enormemente enriquecedora.

Resulta especialmente gratificante haber implementado un sistema funcional completo que abarca desde la captura de datos en el visor hasta la evaluación cuantitativa automatizada, incluyendo la generación de un *dashboard* interactivo. La experiencia de diseñar un proyecto con visión de continuidad a largo plazo, articulado como el primer eslabón de una cadena de TFGs sucesivos, ha aportado una perspectiva de planificación y visión estratégica que trasciende el ámbito puramente técnico del trabajo.

5.3 Líneas Futuras

Los resultados y limitaciones identificados en este trabajo abren varias líneas de mejora y ampliación que podrían abordarse en futuros desarrollos. A continuación se describen las principales, organizadas según su vinculación con las limitaciones identificadas en el Capítulo 4.

5.3.1 Prioridad

Las tres líneas siguientes abordan directamente las limitaciones identificadas en los resultados y constituyen los requisitos previos para la migración a tiempo real.

- Detección automática de visibilidad del objeto: actualmente, el *pipeline* procesa todos los fotogramas de la sesión con independencia de si el objeto está presente en la escena. Incorporar un mecanismo de detección previo evitaría el procesamiento inútil de fotogramas vacíos y mejoraría la eficiencia global del sistema. Además, esta línea se puede extender a la detección inequívoca de diversos

objetos de interés y se vincula directamente con la primera limitación identificada (dependencia de la segmentación).

- Desambiguación rotacional: la sensibilidad a la orientación del controlador es una de las limitaciones más complejas de resolver. Futuros trabajos podrían investigar técnicas de desambiguación, como la incorporación de información temporal entre fotogramas consecutivos (seguimiento de hipótesis a lo largo del tiempo) o el uso de descriptores geométricos más discriminativos que reduzcan las ambigüedades en la votación PPF para vistas parciales. Esta línea responde a la segunda limitación identificada.

- Análisis de eficiencia computacional y migración a tiempo real: el *pipeline* actual opera en modo *offline*. Es decir, los datos se capturan, se transfieren a un PC y se procesan secuencialmente. El objetivo sería trasladar el sistema a un entorno de ejecución en tiempo real, ya sea dentro del propio visor (aprovechando el procesador del dispositivo) o en un PC conectado, lo que permitiría superponer contenido virtual sobre el objeto detectado de forma interactiva.

5.3.2 Mejora

Estas líneas amplían el alcance experimental del sistema una vez resueltas las limitaciones prioritarias.

- Comparación con otros algoritmos de estimación de pose: durante el desarrollo de este proyecto se exploraron y probaron diversos algoritmos de visión artificial antes de adoptar la combinación PPF+ICP como solución principal. Una extensión natural del trabajo sería realizar una comparación sistemática y cuantitativa entre distintas técnicas de estimación de pose (por ejemplo, enfoques basados en aprendizaje profundo), utilizando un mismo conjunto de datos y un mismo marco de evaluación. Esto permitiría identificar qué métodos ofrecen el mejor equilibrio entre precisión y robustez para este contexto específico.

- Validación en condiciones variadas: los resultados presentados corresponden a una sesión capturada en condiciones de interior controladas. Ampliar la validación a múltiples sesiones con diferentes condiciones de iluminación, presencia de oclusiones parciales del objeto y distintas distancias objeto-cámara, proporcionaría una evaluación más robusta de las capacidades del sistema. Este tipo de pruebas permitiría cuantificar la degradación del rendimiento bajo condiciones adversas y guiar el diseño de mejoras específicas para cada escenario.

5.3.3 Diversificación

Estas líneas requieren que las mejoras prioritarias estén implementadas y suponen un salto cualitativo en la capacidad del sistema.

- Generalización a múltiples objetos: el sistema actual está diseñado para un único objeto (el controlador Touch Plus). Extender el *pipeline* para aceptar varios modelos CAD simultáneos y segmentar múltiples objetos en la misma escena ampliaría significativamente su aplicabilidad, especialmente en escenarios de simulación donde intervienen distintos elementos físicos.
- Mejora de la reconstrucción estereoscópica: la calidad de la nube de puntos condiciona directamente todas las etapas posteriores. Incorporar técnicas de fusión multivista (las cuales se basan en acumular nubes de puntos de varios fotogramas para obtener una reconstrucción más completa) o algoritmos de estimación de disparidad basados en aprendizaje profundo podría mejorar sustancialmente tanto el

fitness como la estabilidad del *center distance*. Esta línea aborda directamente la tercera limitación identificada en los resultados.

Estas líneas de mejora se integran dentro de una visión a medio-largo plazo. La Tabla 5-1 traduce estas líneas de mejora en una hoja de ruta concreta a tres años vista, donde cada TFG futuro aborda un subconjunto de los desafíos identificados, construyendo mejoras sustanciales sobre los cimientos técnicos establecidos en este proyecto.

Curso académico	Título TFG	Objetivo
2025-2026	Implementación de técnicas de visión artificial para la detección de objetos en contextos de realidad mixta	Implementar, comparar y evaluar técnicas de visión artificial aplicadas a estimación de pose, analizando métricas de error para determinar la precisión espacial del algoritmo creado.
2026-2027	Desarrollo de un algoritmo robusto y en tiempo real para la detección de objetos mediante dispositivos de realidad mixta.	Optimización de un sistema de estimación de pose 6D para que opere en tiempo real, ampliar su capacidad a múltiples objetos y condiciones adversas (iluminación variable, oclusiones), y desplegar el algoritmo resultante en Unity sobre el visor Meta Quest 3.
2027-2028	Diseño e implementación de un escenario de simulación de combate con detección de objetos en realidad mixta.	Integración del algoritmo de detección de objetos validado en un escenario funcional de simulación de combate en Unity, desarrollando la interacción entre objetos físicos y virtuales y evaluando su viabilidad como herramienta de entrenamiento con usuarios reales.

Tabla 5-1: Propuesta de TFGs para cursos académicos futuros

El presente Trabajo Fin de Grado constituye el primer eslabón de esta cadena y demuestra, en definitiva, que la estimación de pose 6D mediante técnicas estereoscópicas es viable y cuantitativamente evaluable en entornos de realidad mixta con hardware de consumo.

El sistema desarrollado, junto con el marco de evaluación, la arquitectura de software y los datos experimentales generados, constituyen una base técnica sólida y reproducible sobre la que las sucesivas iteraciones del proyecto podrán construir hacia el objetivo final: la detección robusta de objetos en escenarios de simulación de combate mediante realidad mixta.

6 BIBLIOGRAFÍA

- [1] U. Mujika Torrontegi, «Reconstrucción densa de modelos tridimensionales utilizando visión artificial», Máster Universitario en Ingeniería Computacional y Sistemas Inteligentes, Univ. País Vasco / Euskal Herriko Unibertsitatea, Donostia, Spain, 2010. Accedido: 9 de marzo de 2026. [En línea]. Disponible en: <https://www.ehu.eus/documents/1545039/1570316/10umujika.pdf>
- [2] R. Skarbez, M. Smith, y M. C. Whitton, «Revisiting Milgram and Kishino's Reality-Virtuality Continuum», *Front. Virtual Real.*, vol. 2, mar. 2021, doi: 10.3389/frvir.2021.647997.
- [3] T. Q. Tran, T. Langlotz, and H. Regenbrecht, «A Survey On Measuring Presence in Mixed Reality». Accedido: 9 de marzo de 2026. [En línea]. Disponible en: <https://dl.acm.org/doi/epdf/10.1145/3613904.3642383>
- [4] N. Alam, N. Saha, V. Gadow, R. Harik, y J. Ryu, «Role of extended reality (XR) technologies in maintenance operations: Trends, challenges, and integration in industry 4.0», *Manufacturing Letters*, vol. 44, pp. 1545-1557, ago. 2025, doi: 10.1016/j.mfglet.2025.06.174.
- [5] K. Toczé, J. Lindqvist, y S. Nadjm-Tehrani, «Performance Study of Mixed Reality for Edge Computing», en *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, en UCC'19. New York, NY, USA: Association for Computing Machinery, dic. 2019, pp. 285-294. doi: 10.1145/3344341.3368816.
- [6] «From Controllers to Multimodal Input: A Chronological Review of XR Interaction Across Device Generations», PubMed Central (PMC). Accedido: 9 de marzo de 2026. [En línea]. Disponible en: <https://pmc.ncbi.nlm.nih.gov/articles/PMC12788019/>
- [7] M. Frąckiewicz, «VR Headset Showdown 2025: Meta Quest 3 & Quest Pro vs Apple Vision Pro vs HTC Vive XR Elite – Which XR Device Reigns Supreme?», TechStock². Accedido: 9 de marzo de 2026. [En línea]. Disponible en: <https://ts2.tech/en/vr-headset-showdown-2025-meta-quest-3-quest-pro-vs-apple-vision-pro-vs-htc-vive-xr-elite-which-xr-device-reigns-supreme/>
- [8] «Meta Quest 3: la nueva generación de gafas de realidad virtual». Accedido: 9 de marzo de 2026. [En línea]. Disponible en: <https://www.meta.com/es/quest/quest-3/>
- [9] MSBryan, «Hardware HoloLens 2». Accedido: 9 de marzo de 2026. [En línea]. Disponible en: <https://learn.microsoft.com/es-es/hololens/hololens2-hardware>

- [10] H. Azizbekyan, P. J. Cobb, y E. Cobb, «The Apple Vision Pro: Useful Mixed/Augmented Reality (MR/AR) Headset for Archaeology or Not Quite There Yet?», *Advances in Archaeological Practice*, vol. 13, n.º 2, pp. 346-355, may 2025, doi: 10.1017/aap.2025.7.
- [11] «VIVE XR Elite - Base Station-Free PC VR in a Standalone Headset | VIVE European Union». Accedido: 13 de marzo de 2026. [En línea]. Disponible en: <https://www.vive.com/eu/product/vive-xr-elite/overview/>
- [12] S. Berrezueta-Guzman y S. Wagner, «Choosing the Right Engine in the Virtual Reality Landscape», 18 de agosto de 2025, *arXiv*: arXiv:2508.13116. doi: 10.48550/arXiv.2508.13116.
- [13] P. N. Patil, «OpenXR Usage in Gaming: A Comparative Study on Cross-Platform XR Development», *International Journal of Innovative Research in Technology*, vol. 12, n.º 4, 2025.
- [14] U. Rajapaksha, F. Soheli, H. Laga, D. Diepeveen, y M. Bennamoun, «Deep Learning-based Depth Estimation Methods from Monocular Image and Videos: A Comprehensive Survey», *ACM Comput. Surv.*, vol. 56, n.º 12, pp. 1-51, dic. 2024, doi: 10.1145/3677327.
- [15] X. Qiao, M. Poggi, P. Deng, H. Wei, C. Ge, y S. Mattoccia, «RGB Guided ToF Imaging System: A Survey of Deep Learning-based Methods», 16 de mayo de 2024, *arXiv*: arXiv:2405.10357. doi: 10.48550/arXiv.2405.10357.
- [16] A. Scicluna, C. L. Gentil, S. Sutjipto, y G. Paul, «Towards Robust Perception for Assistive Robotics: An RGB-Event-LiDAR Dataset and Multi-Modal Detection Pipeline», 23 de agosto de 2024, *arXiv*: arXiv:2408.13394. doi: 10.48550/arXiv.2408.13394.
- [17] J. J. Alcalde Sanz, «Diseño de un Protocolo de Calibración de Cámaras Estéreo», Proyecto Fin de Carrera, Dept. de Ingeniería de Sistemas y Automática, Escuela Politécnica Superior, Univ. Carlos III Madrid, 2014.
- [18] «OpenCV: Camera Calibration». Accedido: 9 de marzo de 2026. [En línea]. Disponible en: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html
- [19] «OpenCV: Camera Calibration and 3D Reconstruction». Accedido: 9 de marzo de 2026. [En línea]. Disponible en: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html
- [20] L. Zhou, G. Wu, Y. Zuo, X. Chen, y H. Hu, «A Comprehensive Review of Vision-Based 3D Reconstruction Methods», *Sensors (Basel)*, vol. 24, n.º 7, p. 2314, abr. 2024, doi: 10.3390/s24072314.
- [21] «OpenCV: Epipolar Geometry». Accedido: 9 de marzo de 2026. [En línea]. Disponible en: https://docs.opencv.org/4.x/da/de9/tutorial_py_epipolar_geometry.html
- [22] F. Darmon y P. Monasse, «The Polar Epipolar Rectification», *Image Processing On Line*, vol. 11, pp. 56-75, mar. 2021, doi: 10.5201/ipol.2021.328.
- [23] M. Jeon, J. Park, J.-W. Kim, y S. Woo, «Triple-Camera Rectification for Depth Estimation Sensor», *Sensors (Basel)*, vol. 24, n.º 18, p. 6100, sep. 2024, doi: 10.3390/s24186100.
- [24] «OpenCV: Depth Map from Stereo Images». Accedido: 9 de marzo de 2026. [En línea]. Disponible en: https://docs.opencv.org/4.x/dd/d53/tutorial_py_depthmap.html
- [25] P. V. Phan, «Stereo Matching with Semi-Global Matching (SGM)», Paul Vinh Phan. Accedido: 9 de marzo de 2026. [En línea]. Disponible en: <https://pvphan.com/gatech-cs6476-stereo>
- [26] «OpenCV: Disparity map post-filtering». Accedido: 9 de marzo de 2026. [En línea]. Disponible en: https://docs.opencv.org/4.x/d3/d14/tutorial_ximgproc_disparity_filtering.html
- [27] D. Liu, S. Arai, J. Miao, J. Kinugawa, Z. Wang, y K. Kosuge, «Point Pair Feature-Based Pose Estimation with Multiple Edge Appearance Models (PPF-MEAM) for Robotic Bin Picking», *Sensors (Basel)*, vol. 18, n.º 8, p. 2719, ago. 2018, doi: 10.3390/s18082719.

- [28] J. Guo *et al.*, «Efficient Center Voting for Object Detection and 6D Pose Estimation in 3D Point Cloud», *IEEE Trans. on Image Process.*, vol. 30, pp. 5072-5084, 2021, doi: 10.1109/TIP.2021.3078109.
- [29] Jaykumaran, «Iterative Closest Point (ICP) for 3D Explained with Code». Accedido: 9 de marzo de 2026. [En línea]. Disponible en: <https://learnopencv.com/iterative-closest-point-icp-explained/>
- [30] J. Servos y S. L. Waslander, «Multi-Channel Generalized-ICP: A robust framework for multi-channel scan registration», *Robotics and Autonomous Systems*, vol. 87, pp. 247-257, ene. 2017, doi: 10.1016/j.robot.2016.10.016.
- [31] «OpenCV: Surface Matching». Accedido: 9 de marzo de 2026. [En línea]. Disponible en: https://docs.opencv.org/4.x/d9/d25/group__surface__matching.html
- [32] D. Gallup, J.-M. Frahm, P. Mordohai, y M. Pollefeys, «Variable baseline/resolution stereo», en *2008 IEEE Conference on Computer Vision and Pattern Recognition*, Anchorage, AK, USA: IEEE, jun. 2008, pp. 1-8. doi: 10.1109/CVPR.2008.4587671.
- [33] D. Scharstein, R. Szeliski, y R. Zabih, «A taxonomy and evaluation of dense two-frame stereo correspondence algorithms», en *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*, Kauai, HI, USA: IEEE Comput. Soc, 2001, pp. 131-140. doi: 10.1109/SMBV.2001.988771.
- [34] «Depth accuracy», Luxonis Documentation, Luxonis, 2026. Accedido: 9 de marzo de 2026. [En línea]. Disponible en: <https://docs.luxonis.com>
- [35] B. Meden, A. Brazi, S. Bourgeois, F. M. de Chamisso, y V. Lepetit, «BOP-D: Revisiting 6D Pose Estimation Benchmark for Better Evaluation under Visual Ambiguities», 30 de agosto de 2024, *arXiv*: arXiv:2408.17297. doi: 10.48550/arXiv.2408.17297.
- [36] «BOP: Benchmark for 6D Object Pose Estimation». Accedido: 9 de marzo de 2026. [En línea]. Disponible en: <https://bop.felk.cvut.cz/home/>
- [37] A. Gurram y A. M. Lopez, «On the Metrics for Evaluating Monocular Depth Estimation», 20 de febrero de 2023, *arXiv*: arXiv:2302.10007. doi: 10.48550/arXiv.2302.10007.
- [38] Y. Deng, J. Xue, T. Cao, Y. Zhang, L. Wen, y Y. Chen, «SuperPose: Improved 6D Pose Estimation with Robust Tracking and Mask-Free Initialization», 20 de octubre de 2024, *arXiv*: arXiv:2409.19986. doi: 10.48550/arXiv.2409.19986.
- [39] J. Morris, Y. Wang, y V. Ila, «The Importance of Coordinate Frames in Dynamic SLAM», 30 de septiembre de 2024, *arXiv*: arXiv:2312.04031. doi: 10.48550/arXiv.2312.04031.
- [40] «Actualización de los Términos de software del Editor de Unity: Cancelación del costo de Runtime | Blog de Unity», Unity. Accedido: 15 de marzo de 2026. [En línea]. Disponible en: <https://unity.com/blog/terms-update-runtime-fee-cancellation>
- [41] «Building with Unity». Accedido: 13 de marzo de 2026. [En línea]. Disponible en: <https://developers.meta.com/horizon/develop/unity/>
- [42] «Passthrough Camera API overview». Accedido: 13 de marzo de 2026. [En línea]. Disponible en: <https://developers.meta.com/horizon/documentation/spatial-sdk/spatial-sdk-pca-overview/>
- [43] *oculus-samples*, «Unity-PassthroughCameraApiSamples». Accedido: 14 de marzo de 2026. [En línea]. Disponible en: <https://github.com/oculus-samples/Unity-PassthroughCameraApiSamples>

ANEXO I: IMPLICACIONES SOCIALES, Y/O ECONÓMICAS, Y/O AMBIENTALES

En cumplimiento del Sello Internacional de Calidad EUR-ACE, el presente anexo recoge las consideraciones pertinentes y las principales implicaciones del proyecto.

Implicaciones sociales

El sistema emplea cámaras estéreo de un visor de realidad mixta para capturar el entorno físico. Aunque las capturas se realizaron en el laboratorio y sin presencia de terceros, su uso en entornos abiertos requeriría garantizar el cumplimiento de medidas como anonimización de rostros o limitación del campo de visión procesado. En términos de salud y seguridad, los sistemas de realidad mixta deben considerar efectos como la fatiga visual y la distracción al superponer elementos virtuales. Aunque el procesamiento actual es *offline*, una futura ejecución en tiempo real deberá asegurar latencias suficientemente bajas para evitar dichos efectos. Como impacto positivo, las técnicas desarrolladas pueden tener aplicación en entornos de formación simulados (en lo que respecta al centro docente, relativos al ámbito militar).

Implicaciones económicas

El proyecto se basa en hardware accesible (Meta Quest 3) y software de código abierto (OpenCV, Open3D, NumPy), eliminando costes de licencia y facilitando su adopción en entornos académicos o con recursos limitados. La modularidad del *pipeline* (separación entre lógica algorítmica y orquestación) favorece la reutilización de componentes en futuros desarrollos, reduciendo tiempos y costes.

Implicaciones ambientales

El impacto ambiental directo es reducido, al emplear un PC convencional y un visor de bajo consumo, sin recurrir a computación en la nube ni hardware especializado.

ANEXO II: REFLEXIONES ÉTICAS Y SOCIALES

En cumplimiento del Sello Internacional de Calidad EUR-ACE, el presente anexo recoge las reflexiones éticas derivadas del proyecto.

Privacidad y captura de datos visuales

El sistema captura imágenes del entorno mediante cámaras. Aunque las pruebas se realizaron en laboratorio controlado, cualquier despliegue fuera de este requerirá informar a las personas presentes y obtener su consentimiento, conforme al derecho a la intimidad.

Uso dual de la tecnología

El proyecto se enmarca en una línea de investigación orientada a simulación de combate mediante realidad mixta. La detección y localización de objetos presenta naturaleza dual, con aplicaciones civiles (formación, industria, asistencia) y de defensa. En este contexto, la simulación permite entrenar en condiciones realistas sin riesgos físicos ni elevados costes logísticos. Este trabajo contribuye a dicha línea desde una base técnica.

Transparencia y reproducibilidad

Se ha adoptado una política de transparencia: el código se aloja en repositorios públicos, los datos se documentan para su reproducción y el *dashboard* interactivo permite inspección abierta de resultados, favoreciendo la integridad académica.

Sesgo y limitaciones de los datos

Los resultados están condicionados por un conjunto de datos limitado (una sesión de 25 fotogramas en entorno controlado), por lo que no deben extrapolarse sin validación adicional. El uso responsable exige reconocer explícitamente estos límites.

ANEXO III: GITHUB – REPOSITORIOS

Los dos repositorios del presente proyecto se encuentran alojados en GitHub y son de acceso público. A continuación, se proporcionan los enlaces directos:

Repositorio [Unity Quest3 PassthroughCamera](#)

Contiene el proyecto Unity completo utilizado para la captura de datos desde el visor Meta Quest 3. Incluye los *scripts* C# de la escena “CaptureAndExport”, la configuración de OpenXR y Meta XR SDK, y todos los elementos necesarios para reproducir la adquisición de datos.

Repositorio [PythonProcessing](#)

Contiene la totalidad del *pipeline* de postprocesado desarrollado en Python. Incluye los módulos algorítmicos (“src/core/”), los orquestadores de ejecución (“src/pipeline/”), los archivos de configuración, el generador del *dashboard* HTML y la documentación del proyecto.

Ambos repositorios incluyen archivos README.md con instrucciones de instalación, dependencias y guías de uso. La política del “.gitignore” del repositorio *PythonProcessing* excluye los datos binarios (imágenes, nubes de puntos, archivos JSON y NumPy) y conserva únicamente el código fuente, los archivos de configuración y la estructura de carpetas.

ANEXO IV: DECLARACIÓN RESPONSABLE SOBRE EL USO DE INTELIGENCIA ARTIFICIAL GENERATIVA

En cumplimiento de los principios de honestidad académica y transparencia, y de acuerdo con la decisión adoptada conjuntamente con los directores del presente Trabajo Fin de Grado, se incluye esta declaración responsable sobre el uso de herramientas de inteligencia artificial generativa (IAG) durante la elaboración del proyecto.

Alcance y naturaleza del uso

Durante el desarrollo de este trabajo se ha empleado inteligencia artificial generativa como herramienta auxiliar, circunscrita en todo momento a las siguientes funciones:

- **Redacción y homogeneización estilística de la memoria.** Se ha utilizado IA generativa como herramienta de apoyo a la redacción, con el objetivo de encuadrar el texto en un estilo y forma homogéneos a lo largo de todo el documento. En concreto, la herramienta ha contribuido a proporcionar un tono riguroso, coherente y uniforme entre las distintas secciones de la memoria, así como a mejorar la claridad expositiva y la corrección gramatical y ortográfica del texto. El contenido técnico, la estructura argumental y las decisiones sobre qué información incluir han sido en todo momento de elaboración propia.

- **Análisis y corrección de errores en el código fuente.** Se ha recurrido a IA generativa como apoyo en la identificación y corrección de errores durante el desarrollo del software, tanto en los scripts C# del proyecto Unity como en los módulos Python del pipeline de postprocesado. En todos los casos, se ha supervisado y validado cada resultado proporcionado por la herramienta, revisando y comprendiendo cada solución antes de incorporarla al código. La arquitectura del sistema, el diseño algorítmico y la toma de decisiones técnicas son de elaboración propia.

- **Generación de material gráfico específico.** Se ha empleado IA generativa para la creación de dos imágenes concretas incluidas en la memoria, debidamente identificadas en sus respectivos pies de figura con la anotación “Generado con IAG”. Dichas imágenes fueron generadas a partir de indicaciones específicas (*prompts*) y revisadas para garantizar su adecuación al contenido técnico del documento.

Principios de uso

El empleo de inteligencia artificial generativa en este proyecto se ha regido por los siguientes principios:

- **Complementariedad:** la IAG se ha utilizado exclusivamente como herramienta de apoyo, nunca como sustituta del criterio técnico ni de la labor intelectual propia del trabajo.

- **Supervisión permanente:** todo resultado proporcionado por la herramienta ha sido evaluado, comprendido y, en su caso, modificado antes de su incorporación al trabajo.

- **Transparencia:** se declaran de forma explícita los ámbitos en los que se ha empleado IA generativa, así como las figuras específicas generadas mediante esta tecnología.

- **Autoría y responsabilidad:** el diseño conceptual del sistema, la fundamentación teórica, el desarrollo experimental, la obtención e interpretación de resultados y las conclusiones del trabajo son de elaboración propia, bajo la supervisión de los directores del proyecto.

Declaración final

El uso de herramientas de inteligencia artificial generativa se ha realizado de forma responsable y ética, como complemento al trabajo personal y siempre bajo supervisión directa. La autoría intelectual, el criterio técnico y la responsabilidad final sobre la totalidad del contenido de este Trabajo Fin de Grado recaen íntegramente en quien lo suscribe, en conformidad con los principios de integridad académica del Centro Universitario de la Defensa en la Escuela Naval Militar.