

MÁSTER UNIVERSITARIO EN INVESTIGACIÓN
EN INGENIERÍA DE SOFTWARE Y SISTEMAS
INFORMÁTICOS



Telepresencia Monocular: Del 2D al
3D en tiempo real mediante modelos
de IA sin sensores dedicados

31105151 Trabajo Fin de Máster

Curso 2025/2026 Convocatoria Febrero

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

31105151 Trabajo Fin de Máster

Telepresencia Monocular: Del 2D al 3D en Tiempo Real mediante modelos de IA sin Sensores Dedicados. Trabajo Tipo B

Estudiante: Diego Cangas Moldes

Directores: Pedro Javier Herrera Caro / José Antonio Piedra Fernández

Resumen

El presente trabajo fin de máster aborda el diseño, implementación y evaluación de un sistema de reconstrucción 3D monocular optimizado para el tiempo real, orientado a aplicaciones de comunicación volumétrica inmersiva. A pesar de que las videollamadas son hoy en día una herramienta esencial, las soluciones actuales bidimensionales fallan a la hora de replicar la sensación de presencia física y pueden llegar a fatigar al usuario. La comunicación volumétrica surge como una evolución necesaria, pero su implementación enfrenta desafíos, como el alto coste del hardware especializado que requiere (Sensores LiDAR o sistemas multicámara) además de la inestabilidad en los modelos de reconstrucción monocular actuales. La solución propuesta se basa en un pipeline desacoplado mediante una arquitectura de microservicio que separa el procesamiento e inferencia (Backend) de la visualización de los datos (Frontend). El núcleo del Backend, diseñado con Python y Flask, emplea YOLO para segmentar y aislar al usuario del entorno, optimizando el cómputo y la calidad resultante, por otro lado, el modelo Intel MiDaS destaca como el corazón del sistema de estimación de profundidad monocular.

Para superar las limitaciones de baja resolución y falta de detalle fino propios de la estimación monocular, se ha implementado una etapa de superresolución mediante una red neuronal FSRCNN personalizada, que a través de un proceso de fine-tuning con pares de mapas de profundidad reales y estimados, aprende a reconstruir detalles geométricos precisos, como rasgos faciales. Así mismo, se mitiga el fenómeno de flickering o parpadeo mediante un algoritmo de estabilización temporal basado en medias móviles ponderadas exponenciales, que ofrece una respuesta fluida ante movimientos rápidos al mismo tiempo que mantiene un bajo consumo de memoria.

La información de color y profundidad se encapsula de forma sincronizada en el canal Alpha de una imagen PNG, evitando de esta manera desincronizaciones y el uso de protocolos pesados. El Frontend, desarrollado en Unity 3D recibe la información y renderiza una nube de puntos dinámica mediante un VFX Graph Personalizado que desplaza las partículas directamente en la GPU, logrando un rendimiento hasta diez veces superior a los métodos tradicionales basados en CPU. Los resultados de la evaluación técnica demuestran la viabilidad del sistema en hardware de consumo actual, manteniendo tasas de rendimiento estables entre 30 y 60 FPS con latencias aceptables para la interacción humana natural. En conclusión, este trabajo demuestra la viabilidad de la comunicación volumétrica basada en software para equipos de consumo, ofreciendo así una experiencia inmersiva de comunicación accesible y eficiente.

Palabras Clave

Reconstrucción 3D, Sistemas en tiempo real, Estimación de Profundidad Monocular, Comunicaciones en tiempo Real, Aprendizaje Automático, Entornos Inmersivos, Intel MiDaS, Superresolución (FSRCNN), Arquitectura de Microservicios

Índice

Resumen.....	V
Palabras Clave	V
Lista de Figuras	XI
Lista de Tablas	XIII
1. Introducción	1
1.1. Contextualización	1
1.2. Motivación	2
1.3. Planteamiento del problema	2
1.4. Objetivos	3
1.4.1 Objetivo general.....	3
1.4.2 Objetivos Específicos	3
1.5. Metodología general.....	4
1.5.1 Herramientas y Tecnologías	4
1.5.2 Adquisición de datos	4
1.5.3 Procesamiento de Datos	5
1.5.4 Evaluación del sistema	5
1.6. Estructura del trabajo	6
2. Marco Teórico y Estado del Arte	9
2.1. Fundamentos teóricos	9
2.1.1 Reconstrucción 3D a partir de imágenes	9
2.1.2 Enfoques Clásicos de reconstrucción	11
2.1.3 Estimación de profundidad monocular.....	13
2.1.4 Mapa de Profundidad.....	15
2.1.5 Redes neuronales: modelos convolucionales frente a modelos Transformers.....	16
2.1.6 Detección de Objetos y Regresión de cajas	20
2.1.7 Superresolución mediante Aprendizaje Profundo	22
2.1.8 Estabilización temporal en video.....	23
2.2. Revisión bibliográfica	24
2.2.1 Aprendizaje Profundo para estimar la profundidad	25
2.2.2 Reconstrucción y transmisión 3D en tiempo real	26

2.2.3 Optimización de Pipelines mediante Regiones de Interés	27
2.2.4 Optimización de Pipelines mediante Superresolución	29
2.3. Análisis crítico de las soluciones y tecnologías existentes.....	30
2.3.1 Microsoft Holoportation.....	31
2.3.2 Meta Codec Avatars.....	32
2.3.3 Google Beam.....	32
2.3.4 Z-Buffer 3D (Z3D) de VorpX y similares.....	33
2.3.5 Soapbox	34
2.4 Conclusiones	35
3. Solución Aportada.....	39
3.1 Descripción de la Solución	39
3.1.1 Objetivos del sistema	40
3.2 Arquitectura del sistema y Flujo de datos del Backend	41
3.2.1 Arquitectura del sistema	41
3.2.2 Flujo de datos de los procesos del Backend	43
3.3. Tecnologías, Metodologías y Herramientas aplicadas	45
3.3.1 Python	45
3.3.2 YOLO.....	45
3.3.3 Intel MiDaS	46
3.3.4 Flask	46
3.3.5 Unity.....	46
3.3.6 Docker.....	47
3.3.7 Evaluación tecnológica y “Tech Radar” del proyecto	47
3.4 Diseño y configuración de los Modelos Neuronales.....	51
3.4.1 Configuración del Detector (YOLO)	51
3.4.2 Modelo de Estimación de Profundidad (MiDaS)	52
3.4.3 Arquitectura de Superresolución (FSRCNN)	52
3.5 Desarrollo	55
3.5.1 Diseño detallado de los componentes clave del sistema	56
3.5.2 Estrategias de Implementación	56
3.5.3 Desafíos Técnicos y Soluciones Aportadas	58
3.5.4 Pruebas y validación durante el desarrollo	64

3.6 Implementación del sistema	65
3.6.1 Entorno de ejecución y despliegue	65
3.6.2 Interfaz y flujo de usuario	65
3.6.3 Gestión de problemas en la implementación	68
3.6.4 Control de versiones y mantenimiento	68
3.7 Resumen del Capítulo	69
4. Medición y Evaluación	71
4.1 Objetivos de la evaluación	71
4.2 Métricas utilizadas	72
4.2.1 Rendimiento y Fluidez	72
4.2.2 Eficiencia de Recursos	72
4.2.3 Precisión de Detección y Segmentación	73
4.2.4 Estabilidad y Calidad Visual	74
4.2.5 Calidad Percibida por el usuario	74
4.3 Metodología de evaluación	75
4.3.1 Escenarios de prueba y entorno técnico	76
4.3.2 Protocolo de recolección de los datos	76
4.3.3 Ejecución del test subjetivo de usuario	76
4.4 Resultados obtenidos	77
4.4.1 Validación de los Modelos de IA	78
4.4.2 Rendimiento y Latencia del Sistema	81
4.4.3 Estabilidad y Calidad de Reconstrucción	85
4.4.4 Evaluación Subjetiva (Escala Likert)	87
4.5 Análisis e interpretación de resultados	88
4.5.1 Equilibrio entre latencia y Calidad (El compromiso para conseguir el Tiempo Real)	88
4.5.2 Independencia del hardware	89
4.5.3 Comparación con el estado del arte	90
4.5.4 Interpretación de la Percepción del Usuario	91
4.5.5 Viabilidad y Cumplimiento de Hipótesis	92
5. Conclusiones y Trabajo Futuro	93
5.1 Conclusiones y Resumen del Trabajo	93

5.2 Aportaciones.....	95
5.2.1 Contribuciones Técnicas y Metodológicas.....	96
5.2.2 Impacto y Valor Añadido	96
5.3 Limitaciones del trabajo	97
5.3.1 Falta de información en las zonas ocluidas	97
5.3.2 Dependencia del hardware en el Backend	97
5.3.3 Estimación de la Profundidad	97
5.3.4 Latencia Acumulada	97
5.3.5 Limitaciones de Iluminación	98
5.3.6 Limitaciones del proceso de fine-tuning.....	98
5.4 Trabajo Futuro	98
5.4.1 Uso de modelos generativos para la reconstrucción 360°.....	98
5.4.2 Migración a protocolos de baja latencia como WebRTC	98
5.4.3 Adopción de técnicas de representación visual más eficientes como 3D Gaussian Splatting	99
5.4.4 Optimización para dispositivos “Standalone”	99
5.4.5 Aceleración mediante el uso de Hardware Emergente	99
5.4.6 Retroalimentación de superresolución a partir de información RGB....	99
Bibliografía	101
Siglas y Acrónimos	107
Notas Adicionales.....	107
Anexo I. Código Fuente y diagramas de componentes críticos	109
Anexo II. Cuestionario de evaluación	111
Anexo III. Tabla de Respuestas de Usuarios	113
Anexo IV. Hoja de especificaciones de Prueba y Entorno experimental	115
Anexo V. Manual de Usuario	117

Lista de Figuras

<i>Figura 2.1 Ejemplo de modelo 3D obtenido mediante SfM, (a) el modelo junto con los recuadros azules que representan las posiciones de las cámaras desde donde se han capturado las imágenes, (b) imagen del modelo 3D resultante Fuente: smns-games.com</i>	<i>10</i>
<i>Figura 2.2 Ejemplo de imagen estereoscópica, Como se puede observar ambas imágenes son similares, pero no idénticas, a partir de la diferencia de posición de cada elemento en ambas imágenes se puede determinar la distancia a la cámara de cada punto y por tanto la geometría 3D de la imagen. Fuente: artesvisualessemimara.blogspot.com</i>	<i>12</i>
<i>Figura 2.3 Ejemplo de uso del escaneo 3D mediante técnicas fotométricas. Se toman imágenes del objeto bajo distintas iluminaciones (a) y a partir de ellas, se reconstruye su mapa de normales (b). Fuente: dcmsistemas.com</i>	<i>13</i>
<i>Figura 2.4 Un ejemplo de habitación de Ames. (a) fotografía en la que parece que ambas personas tienen tamaños completamente distintos. Fuente: Pinterest (b) esquema de cómo está diseñada esta ilusión. Fuente: elconfidencial.com</i>	<i>14</i>
<i>Figura 2.5 Ejemplo de mapa de profundidad. Fuente: researchgate.net</i>	<i>16</i>
<i>Figura 2.6 Esquema de una red neuronal, las bolas representan las neuronas y las flechas los flujos de datos que las conectan. Fuente: IBM</i>	<i>17</i>
<i>Figura 2.7 Arquitectura de una CNN. Fuente: aprendemachinelearning.com</i>	<i>18</i>
<i>Figura 2.8 Ejemplo de análisis de flujo óptico, a: figura original, b: figura movida, c: flujo óptico representado por flechas del objeto de (a) a (b). Fuente: uiowa.edu ..</i>	<i>24</i>
<i>Figura 2.9 Ejemplo de resultados de Intel MiDaS, (a) la imagen de entrada, (b) el mapa de profundidad obtenido por la red neuronal. Fuente: pyimagesearch.com</i>	<i>26</i>
<i>Figura 2.10 Microsoft Holoportation en funcionamiento, la persona de la izquierda es un modelo 3D retransmitido en tiempo real que la persona de la derecha puede ver a través de su dispositivo HoloLens. Fuente: Microsoft</i>	<i>31</i>
<i>Figura 2.11 (a) Proceso de escaneo del usuario. (b) Codec Avatar Generado por Meta. Fuente Meta Platforms. Inc.....</i>	<i>32</i>
<i>Figura 2.12 Google Beam. (a) sistema HP Dimension, (b) sistema usándose con Google Beam, donde la persona de rojo es un modelo 3D fotorrealista transmitido en tiempo real. Fuente: HP</i>	<i>33</i>
<i>Figura 2.13 Imagen 3D generada haciendo uso de un algoritmo de Z-Buffer. Pueden verse algunos defectos en la imagen generada (a) respecto a la original (b) entre los dos personajes debido a la falta de información de color a la hora de reconstruir la imagen. Fuente Paradise Decay</i>	<i>34</i>
<i>Figura 2.14 Soapbox. Un cantante virtual canta y se mueve dentro del espacio físico real del usuario, funcionando en unas Meta Quest 3/3S. Fuente: Horizon Store ..</i>	<i>35</i>

Figura 3.1 Diagrama C4 de nivel 2 (Contenedores) del funcionamiento del sistema con un solo usuario	42
Figura 3.2 Diagrama C4 de nivel 1 (Contexto) del funcionamiento del sistema con dos usuarios interconectados.....	43
Figura 3.3 Diagrama de Flujo de Datos entre los módulos del Backend	44
Figura 3.4 Logo de Python. Fuente: www.python.org	45
Figura 3.5 Logo de Flask. Fuente: flask.palletsproject.com	46
Figura 3.6 Logo de Unity. Fuente: Unity	47
Figura 3.7 Diagrama "Tech Radar" del proyecto	49
Figura 3.8 a) Imagen de referencia b) Mapa de profundidad real del modelo c) Mapa de profundidad generado por MiDaS	60
Figura 3.9 Interfaz del Frontend. Conexión del Backend.....	66
Figura 3.10 Interfaz del Frontend. Imagen de pruebas.....	66
Figura 3.11 Frontend. Modos de visualización Alternativos. (a) Half SBS (para dispositivos VR). (b) Full SBS (para Proyector 3D). (c) Anaglifo	67
Figura 3.12 Interfaz del Backend. Para iniciar el Backend basta con iniciar el contenedor (botón de play) e iniciar la retransmisión desde la fuente de vídeo (OBS Studio)	67
<i>Figura 4.1 Ejemplo de lo que los usuarios veían durante las pruebas.....</i>	<i>77</i>
<i>Figura 4.2 Matriz de confusión YOLOv8 Clase Persona.....</i>	<i>79</i>
<i>Figura 4.3 Gráfica de Precisión-Recall de YOLOv8 Nano</i>	<i>79</i>
<i>Figura 4.4 Gráfica de Depth Loss del fine-tuning de FSRCNN</i>	<i>80</i>
<i>Figura 4.5 Comparativa visual. (a) Imagen RGB de partida. (b) MiDaS + FSRCNN Genérico. (c) MiDaS + FSRCNN Personalizado. (d) y (e) MiDaS + FSRCNN Personalizado hasta Primera y Segunda etapa respectivamente.</i>	<i>81</i>
<i>Figura 4.6 Comparación entre Ground Truth contra MiDaS y contra el modelo propuesto.....</i>	<i>86</i>
<i>Figura 4.7 Comparación más detallada de MiDaS Base (MiDaS + Bicúbico) con el modelo propuesto</i>	<i>87</i>
<i>Figura 4.8 Comparación Fine-Tuning. (a) FSRCNN genérico. (b) FSRCNN con Fine-Tuning</i>	<i>87</i>

Lista de Tablas

Tabla 2.1 Comparación de las tecnologías analizadas	36
Tabla 2.2 Adecuación de las tecnologías al presente trabajo	36
Tabla 3.1 Tecnologías descartadas del cuadrante de visión por computador.....	49
Tabla 3.2 Tecnologías descartadas del cuadrante de visualización 3D	50
Tabla 3.3 Tecnologías descartadas del cuadrante de infraestructura de comunicación	50
Tabla 3.4 Tecnologías descartadas del cuadrante de Optimización.....	51
Tabla 3.5 Comparación entre formatos de imagen valorados	63
Tabla 3.6 Mejoras de rendimiento del sistema	64
<i>Tabla 4.1 Dimensiones de evaluación de la escala de Likert. Las preguntas L5, L6 y L7 han sido extraídas del Igroup Presence Questionnaire</i>	<i>75</i>
<i>Tabla 4.2 Métricas del Subsistema de Detección</i>	<i>78</i>
<i>Tabla 4.3 Comparativa de métricas de similitud entre el modelo base y el modelo con fine-tuning</i>	<i>80</i>
<i>Tabla 4.4 Tabla de rendimiento (FPS) del Backend en el equipo de pruebas principal</i>	<i>82</i>
<i>Tabla 4.5 Uso de recursos del Backend en equipo de pruebas principal</i>	<i>82</i>
<i>Tabla 4.6 Comparación de latencias a distintas resoluciones y compresiones</i>	<i>83</i>
<i>Tabla 4.7 Resultados obtenidos en los escenarios de conectividad y latencia a 480P Nivel 0</i>	<i>83</i>
<i>Tabla 4.8 Rendimiento del Frontend</i>	<i>84</i>
<i>Tabla 4.9 Métricas temporales. Resultados</i>	<i>86</i>
<i>Tabla 4.10 Resultados obtenidos de la escala Likert</i>	<i>88</i>

1. Introducción

1.1. Contextualización

Las videollamadas se han convertido en una herramienta esencial para la comunicación remota, especialmente en el ámbito personal, educativo y profesional. Según (Fernández, 2024) en 2023 casi el 70% de la población europea hizo uso de esta tecnología. Plataformas como Skype, Zoom o Google Meet han dominado este campo durante más de dos décadas, permitiendo interacciones "cara a cara" en un entorno bidimensional. Sin embargo, estas soluciones, aunque eficaces, no logran replicar la sensación de cercanía y presencia que se experimenta en una conversación física y, en sesiones prolongadas incluso son capaces de generar cierto nivel de fatiga en el usuario (Webb, 2021)

Con el desarrollo de tecnologías como la realidad virtual o los nuevos "wearables" inteligentes con pantalla, surge una oportunidad para evolucionar hacia un modelo de comunicación más inmersivo y "cercano": las videollamadas volumétricas.

Las videollamadas volumétricas buscan superar las limitaciones de las plataformas bidimensionales tradicionales al representar a los interlocutores en 3D, facilitando una interacción más natural y cercana. Recientemente, se han desarrollado sistemas que generan avatares tridimensionales en tiempo real a partir de cámaras web monoculares, combinando nubes de puntos con representaciones faciales de alta resolución para mejorar la expresividad y la interacción en entornos virtuales (Deng y otros, 2023)

Este enfoque permite representar a los interlocutores de manera tridimensional mediante nubes de puntos o mallas 3D generadas a partir de imágenes capturadas por una o varias cámaras, y visualizarlas dentro de un entorno inmersivo (Realidad

Virtual) o integrarlas dentro del mundo real a través de pantallas holográficas o dispositivos de Realidad Mixta

1.2. Motivación

La comunicación volumétrica representa un avance significativo respecto a las videollamadas tradicionales, ya que ofrece una mayor sensación de realismo y presencia. No obstante, si salimos del marco teórico y buscamos un enfoque más pragmático, podremos observar que la implementación práctica de estos sistemas enfrenta varios desafíos, especialmente en el campo de la generación en tiempo real de mallas o nubes de puntos 3D a partir de la información de una sola cámara (reconstrucción monocular).

Los modelos actuales que tratan de llevar a cabo esta tarea, suelen presentar limitaciones derivadas del ruido en los datos de origen, falta de estabilidad temporal entre fotogramas de un mismo vídeo o en el caso de los modelos más grandes y precisos, un alto coste computacional, lo que lo aleja del uso en tiempo real, así como de su uso en dispositivos con una capacidad de computación más humilde, como suelen ser los dispositivos móviles, ya sean teléfonos, tabletas u ordenadores portátiles. A pesar de estos limitantes, trabajos como (Rovinelli, 2021) demuestran que es posible llevar a cabo estimaciones en tiempo real en dispositivos móviles haciendo uso de redes neuronales ligeras y técnicas de optimización a costa de cierto grado de calidad en el resultado final

Estos obstáculos, que serán los que se intentarán abordar en este proyecto, limitan la viabilidad de la comunicación volumétrica en tiempo real como alternativa a las aplicaciones cotidianas.

1.3. Planteamiento del problema

Aunque el 3D no es algo nuevo y lleva abordándose en sistemas informáticos ya desde el siglo pasado, su enfoque siempre ha estado ligado al uso de sensores especiales para este ámbito, como lidares, cámaras de profundidad, o sistemas estereoscópicos. Con el auge de las Redes Neuronales, o como se les conoce comercialmente, “La Inteligencia Artificial” se nos abre una nueva vía, ¿Podríamos usar sistemas basados en una sola cámara, como una webcam o la cámara de un teléfono para generar imágenes 3D a través de algoritmos? Trabajos como (Zollhöfer y otros, 2018) demuestran que el enfoque monocular, aunque enfrenta desafíos como los que se detallarán más adelante, representa una línea de investigación prometedora que podría democratizar el acceso al 3D, permitiendo su uso en millones de dispositivos sin hardware especializado.

Por supuesto, este enfoque no es completamente novedoso; actualmente, métodos de reconstrucción 3D monocular como MiDaS de Intel ofrecen resultados prometedores en la reconstrucción de imágenes. Sin embargo, estos métodos enfrentan desafíos importantes, como la falta de optimización para video, especialmente en tiempo real, lo que se traduce en inestabilidad visual entre fotogramas, aparición de artefactos en la reconstrucción y un alto requerimiento computacional.

Este Trabajo de Fin de Máster plantea como problema central la necesidad de mejorar la calidad, estabilidad y eficiencia de las reconstrucciones 3D monoculares en tiempo real, con el fin de integrarlas de forma práctica en una experiencia de videollamada inmersiva.

1.4. Objetivos

Los objetivos a tratar en el presente trabajo se dividen en dos grupos, por un lado, el objetivo general y consecuentemente una serie de objetivos específicos que ayudarán al lector a comprender el alcance del presente trabajo

1.4.1 Objetivo general

El objetivo general de este proyecto consiste en diseñar y evaluar un sistema de reconstrucción 3D monocular optimizado para su uso en tiempo real o cuasi real. Orientando su uso a un entorno de videollamadas (uso centrado en imágenes de personas) volumétricas para entornos inmersivos como realidad virtual, realidad extendida e incluso computación espacial. El fin del proyecto es mejorar la calidad, estabilidad y eficiencia del proceso de reconstrucción para su viabilidad en dispositivos de uso común.

1.4.2 Objetivos Específicos

Los objetivos específicos de este trabajo son los que siguen:

1. Implementar un sistema funcional basado en MiDaS u otros modelos de estimación de profundidad monocular con el fin de generar nubes de puntos 3D en tiempo real a partir de un sistema monocular (empleando una única cámara).
2. Diseñar e implementar un algoritmo de estabilización que permita “suavizar” las fluctuaciones de profundidad entre fotogramas consecutivos que presentan los algoritmos actuales, mejorando así la coherencia visual del sistema.
3. Evaluar el rendimiento del sistema antes y después de aplicar las optimizaciones propuestas, así como comparar esta propuesta con otras existentes, haciendo uso de métricas clave como la tasa de fotogramas por

segundo obtenidas (FPS), la latencia promedio del sistema o, la calidad visual percibida.

4. Proponer un método eficiente de representación y transmisión de nubes de puntos que permita su uso práctico en videollamadas volumétricas o aplicaciones de uso en tiempo real, manteniendo a su vez un flujo de datos y un ancho de banda razonable.

1.5. Metodología general

Para alcanzar los objetivos anteriormente planteados, se desarrollará un sistema propio de reconstrucción 3D monocular que funcione en tiempo real, haciendo uso de las herramientas, frameworks y tecnologías que se detallarán posteriormente. La metodología a seguir en este proyecto se estructura en tres fases claramente diferenciadas, siendo la primera de estas, la adquisición de los datos con los que se van a trabajar, la segunda se correspondería con el procesamiento de esta información para alcanzar el resultado deseado, y finalmente la última fase sería la de la evaluación de los resultados y su comparación con otros sistemas existentes.

1.5.1 Herramientas y Tecnologías

Para el desarrollo de este proyecto se usará principalmente Python, acompañado por bibliotecas como OpenCV para el manejo y tratamiento de las imágenes, así como PyTorch para la ejecución de modelos de estimación de profundidad, como por ejemplo MiDaS. Para la visualización de los datos 3D resultantes se hará uso de Unity 3D, un motor de videojuegos y renderizado que cuenta con una gran versatilidad para incorporarle dispositivos, como por ejemplo visores de realidad virtual o aumentada y que cuenta, además, con sistemas de iluminación avanzados que podrían permitir cambiar el resultado en tiempo real, dotándolo de, por ejemplo, fuentes de iluminación 3D adicionales, o distintos efectos de postprocesado 3D. La interconexión entre ambas herramientas (procesamiento con Python y visualización con Unity) se llevará a cabo haciendo uso de un microservicio web basado en Flask por el lado de Python y las librerías UnityWebRequest por el lado de Unity.

1.5.2 Adquisición de datos

El sistema trabajará principalmente con una configuración de entrada de una única cámara a color (presumiblemente una cámara web) como entrada principal. Para tareas de entrenamiento, pruebas y evaluación, no se descarta el uso de sistemas adicionales, como cámaras de profundidad tipo Kinect o cámaras estereoscópicas cuando estén disponibles y así los requiera el sistema.

1.5.3 Procesamiento de Datos

Como ya se comentó en el apartado de tecnologías, esta fase estará principalmente implementada en Python y se corresponderá a su vez con varias etapas que son:

- Estimación de profundidad haciendo uso de modelos preentrenados como MiDaS u otras alternativas modernas
- Uso de técnicas de optimización de procesamiento antes o después de la estimación de profundidad como pueden ser reducciones de resolución de imágenes, identificación de zonas innecesarias de procesar, o usar un algoritmo de multirresolución que ayude a reducir la carga de trabajo y/o mejorar los resultados obtenidos
- Estabilización temporal de los datos, aplicando técnicas como por ejemplo medias móviles o predicción mediante flujo óptico para obtener una salida de vídeo 3D más uniforme entre fotogramas.
- Reducción del ruido, tanto el ocasionado por las propias cámaras, como los derivados de los propios algoritmos, en esta etapa se buscará aplicar algoritmos como el filtro bilateral o el desenfoque gaussiano con el fin de mejorar la consistencia visual del resultado final

1.5.4 Evaluación del sistema

El sistema será evaluado tanto desde un punto de vista cuantitativo como cualitativo y a su vez también será evaluado a través de métricas objetivas y subjetivas por los motivos que más adelante se detallarán.

Dentro de las métricas objetivas cabrán destacar las siguientes:

- Número de imágenes o Fotogramas por segundo que el sistema es capaz de generar (FPS)
- Latencia Media (tiempo que ocurre entre que el usuario realiza una acción y esta se ve reflejada en el sistema)
- Uso de CPU y GPU
- Estabilidad temporal (La varianza de la profundidad a lo largo del tiempo)

Dado que el sistema tiene como fin generar una imagen 3D que será visualizada por un usuario, además de las métricas objetivas que se puedan evaluar destacará también como métrica de evaluación subjetiva un test visual llevado a cabo por el usuario de percepción de calidad y estabilidad, así como su valor comparativo contra otros sistemas existentes.

1.6. Estructura del trabajo

Este trabajo se encuentra dividido en 5 capítulos, formando este texto parte del primero de ellos, cuyo nombre es “1. Introducción”, que, como su propio nombre indica, introduce el proyecto, comenzando con una contextualización de los temas a tratar en este proyecto (sección 1.1 del trabajo) así como una justificación de la motivación (sección 1.2) y un planteamiento del problema que se quiere resolver (sección 1.3), adicionalmente este capítulo establece los objetivos, tanto generales como específicos, que se quieren alcanzar (sección 1.4), así como la metodología general con la que será llevado a cabo el proyecto (sección 1.5). Antes de continuar, el lector podrá pararse en estas líneas (sección 1.6) donde podrá encontrar una descripción de estructura general del trabajo que le servirá como guía para poder navegar por este.

Una vez superado este apartado, el lector se encontrará frente al capítulo “2. Marco Teórico / Estado del arte”, un capítulo centrado en hacer comprender al lector, el estado (artículos, investigaciones, tecnología) desde la que parte este proyecto y para ello, este capítulo, al igual que el anterior, está dividido en varias secciones que son las siguientes: Fundamentos Teóricos (sección 2.1) en la que se explorarán los conceptos clave necesarios para entender el trabajo, así como definiciones técnicas o científicas relevantes. Siguiendo al anterior se encuentra la sección Revisión Bibliográfica (sección 2.2), esta sección se centrará en revisar los artículos académicos relacionados con el proyecto en cuestión, que, junto al resto de secciones de este capítulo ayudarán al lector a entender las decisiones tomadas en ciertos puntos clave del proyecto. Sumado a lo anterior y como cierre del capítulo en la sección 2.3 se hallará un análisis crítico de las soluciones y tecnologías existentes, revisando cuál es la tecnología actual que más se asemeja a lo que se quiere desarrollar, así como las carencias y limitaciones que se pueden encontrar en los enfoques actuales y qué oportunidades de mejora pueden existir respecto a los desarrollos actuales. Finalmente, la sección 2.4 le ofrecerá al lector una breve comparación de las tecnologías de la sección anterior, y servirá de marco de partida para algunas de las principales decisiones de diseño de la solución propuesta.

Finalizados los dos primeros capítulos de este documento, el lector ya contará con información suficiente para comprender la problemática a tratar, así como el estado del arte actual, el contexto y la fundamentación teórica que rodean a este proyecto, por lo que es el momento de pasar al capítulo “3. La solución aportada”, este capítulo, parte troncal y cénit de este proyecto, se enfoca en explicar la solución aportada y justificar las decisiones clave de su desarrollo. El capítulo comienza con una descripción de la solución (sección 3.1), argumentando la razón de esta y así como ciertos aspectos clave la envuelven. Posteriormente, se pasará

a analizar la arquitectura del sistema (sección 3.2), como esta se compone y se expondrán las razones de diseño de la misma. En este capítulo el lector también encontrará información acerca de las tecnologías y herramientas utilizadas (sección 3.3) así la justificación de uso. Siguiendo a lo anterior se encontrará la sección 3.4, “Diseño y configuración de los modelos Neuronales” que se centrará en explicar las principales redes neuronales utilizadas, así como su configuración, tras esta sección, la que sigue es “Desarrollo” (sección 3.5) una extensa sección en la que el lector podrá hacer un seguimiento del diseño de los componentes del sistema, las estructuras internas del proyecto y demás documentación técnica. Finalmente, este capítulo concluye con un repaso por la implementación del sistema (sección 3.6) desde el proceso de instalación y uso hasta su puesta en funcionamiento, haciendo hincapié en los problemas y retos a la hora de desplegar el proyecto, así como la solución de estos. Finalmente, como cierre de este capítulo y a modo de conclusión se presenta la sección 3.7, “Resumen del Capítulo”

Una vez superado el núcleo técnico del proyecto, y habiendo ya comenzado la implementación del sistema el lector podrá adentrarse en el capítulo “4. Medición y Evaluación”, este capítulo comprende desde un repaso más profundo a los objetivos de evaluación del proyecto (sección 4.1) hasta el análisis e interpretación de los resultados (sección 4.5), pasando como no puede ser de otro modo, por un repaso y defensa de las métricas usadas para la evaluación (sección 4.2), la metodología empleada para esta (sección 4.3), y un análisis pormenorizado de los resultados obtenidos (sección 4.4).

El presente documento culmina con el capítulo “5. Conclusiones y Trabajo Futuro”. Este apartado ofrece una visión completa del proyecto, comenzando con una revisión del trabajo realizado, así como la exposición de las conclusiones obtenidas (sección 5.1). A continuación, se analizan las aportaciones principales del proyecto (sección 5.2) y se examinan las limitaciones y restricciones surgidas durante el desarrollo del mismo (sección 5.3). Finalmente, se plantea el trabajo futuro (sección 5.4), donde se proponen mejoras funcionales, se sugieren nuevas líneas de trabajo derivadas y se ofrecen recomendaciones para quienes deseen profundizar en la materia.

Aunque el desarrollo del trabajo concluye en el Capítulo 5, el documento se complementa con un total de seis anexos, en los que el lector podrá encontrar desde la hoja de especificaciones de las pruebas (Anexo IV), hasta los cuestionarios realizados (Anexo II). Finalmente, destacar el último de los anexos, el Anexo VI, que, como ejercicio de transparencia, trata sobre la implicación y el uso de la IAG (Inteligencia Artificial Generativa), dentro del contexto de este trabajo.

2. Marco Teórico y Estado del Arte

2.1. Fundamentos teóricos

La reconstrucción 3D a partir de imágenes monoculares, así como su integración en entornos inmersivos, se sustenta en una serie de conceptos teóricos fundamentales provenientes de áreas como la visión por computador, la geometría proyectiva y el aprendizaje profundo. Comprender estos principios resulta esencial para el desarrollo e implementación de sistemas robustos y precisos.

En esta sección se presentan los fundamentos teóricos que permiten comprender el funcionamiento de los modelos de estimación de profundidad, las técnicas de reconstrucción tridimensional, y los mecanismos de codificación y transmisión necesarios para entornos interactivos en tiempo real.

2.1.1 Reconstrucción 3D a partir de imágenes

A lo largo de la historia reciente de la informática, y en particular en el campo de la visión por computador, ha existido un constante interés por comprender y representar el mundo en tres dimensiones a partir de imágenes. Este objetivo, en parte motivado por el deseo de imitar la percepción humana de la profundidad, ha dado lugar al desarrollo de numerosas técnicas y enfoques para la reconstrucción tridimensional. Desde métodos geométricos clásicos hasta sistemas basados en aprendizaje profundo, el problema de obtener información estructural del entorno a partir de imágenes bidimensionales continúa siendo un área de investigación activa y de gran relevancia en aplicaciones como la realidad aumentada, la robótica, la medicina o la telepresencia.

Desde un punto de vista académico, los métodos de reconstrucción 3D pueden agruparse en dos grandes categorías. Por un lado, se encuentran los enfoques clásicos, basados principalmente en técnicas geométricas que permiten reconstruir la estructura tridimensional de una escena a partir de múltiples vistas

de un objeto o entorno. Estas técnicas se fundamentan en la triangulación de puntos observados desde distintos ángulos y requieren, por tanto, información proveniente de más de una imagen.

Dentro de esta categoría destacan dos aproximaciones ampliamente utilizadas: la reconstrucción estereoscópica y la estructura desde el movimiento (Structure from Motion, SfM). La reconstrucción estereoscópica consiste en estimar la profundidad utilizando dos imágenes capturadas desde cámaras dispuestas de forma similar a los ojos humanos, lo que permite calcular disparidades entre las vistas. Por su parte, SfM, comúnmente conocida como fotogrametría 3D, consiste en capturar múltiples imágenes desde distintas posiciones mediante el desplazamiento de una única cámara alrededor del objeto o escena. Estas imágenes, combinadas con estimaciones de la posición y orientación de la cámara en cada toma, permiten recuperar la forma tridimensional de los elementos capturados. La figura 2.1 Muestra un ejemplo de reconstrucción 3D a partir de imágenes.

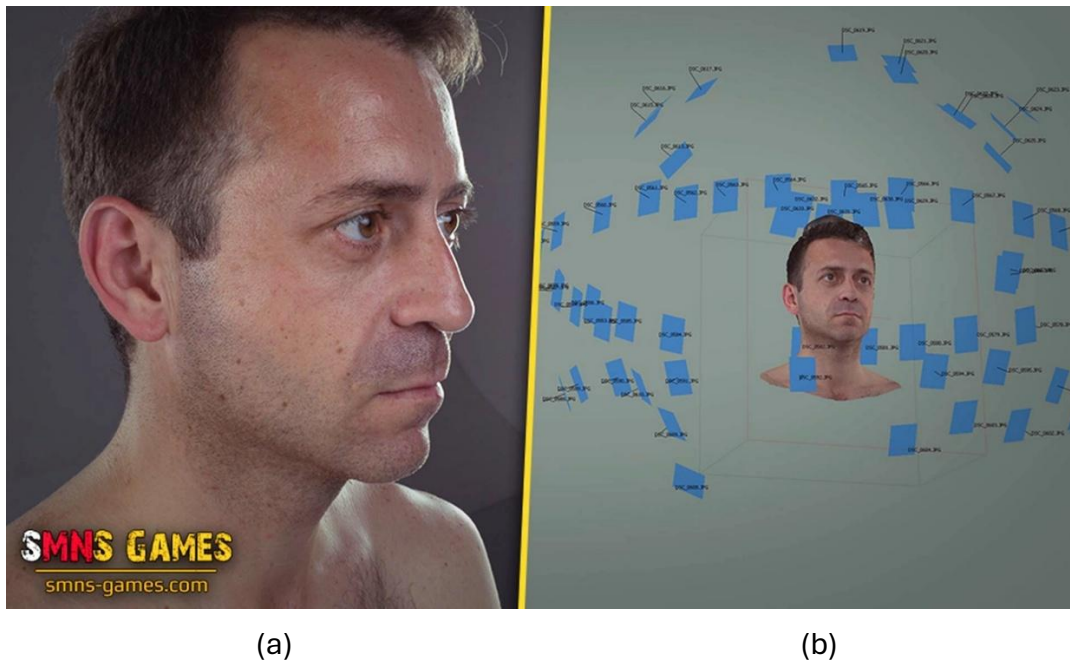


Figura 2.1 Ejemplo de modelo 3D obtenido mediante SfM, (a) el modelo junto con los recuadros azules que representan las posiciones de las cámaras desde donde se han capturado las imágenes, (b) imagen del modelo 3D resultante Fuente: smns-games.com

Junto a estos enfoques clásicos, en la actualidad han surgido nuevas técnicas basadas principalmente en el uso de redes neuronales profundas, que permiten abordar problemas tradicionalmente difíciles de resolver mediante métodos geométricos. Un ejemplo destacado es la estimación de profundidad a partir de una sola imagen, un problema inherentemente ambiguo que ha encontrado soluciones eficaces gracias al aprendizaje automático. Esta línea de trabajo será analizada en detalle a continuación en la sección 2.2.2. Estimación de profundidad monocular. Además de la estimación de profundidad, estos métodos permiten realizar tareas

como la segmentación semántica tridimensional, la reconstrucción de superficies continuas y la predicción de mapas de normales o de iluminación, todo ello a partir de entradas visuales limitadas. El uso de grandes conjuntos de datos anotados y arquitecturas avanzadas de redes, como U-Net o Transformers, ha sido clave en la mejora del rendimiento de estos sistemas, que actualmente alcanzan niveles de precisión y generalización notables en escenarios reales y complejos.

2.1.2 Enfoques Clásicos de reconstrucción

Los enfoques clásicos generalmente se basan en el uso de métodos geométricos y fotométricos (basados en las variaciones de intensidad lumínica) para determinar la tridimensionalidad de una escena a partir de imágenes bidimensionales. Estos métodos establecieron los fundamentos del campo y se caracterizan por un fuerte rigor matemático. Entre las visiones más extendidas, destacan la visión estereoscópica, la estructura desde el movimiento (SfM) y los métodos conocidos como métodos fotométricos

2.1.2.1 Visión Estereoscópica

La visión estereoscópica es el enfoque que más se asemeja a la visión humana. Hace uso de dos o más cámaras con posiciones y orientaciones fijas (para ello se hace un sistema previo de calibrado) para capturar una misma escena desde puntos de vista ligeramente diferentes. La predisposición más habitual de este sistema es la del uso de dos cámaras paralelas o cuasi paralelas en el eje horizontal, imitando la disposición del ojo humano.

El Principio fundamental de la visión estereoscópica se basa en la triangulación (Longuet-Higgins, 1981), esto es, al conocer la geometría del sistema (es decir, la posición y orientación de las cámaras), el objetivo radica en ubicar la posición de un mismo punto 3D en cada una de las cámaras obteniendo lo que se conoce como disparidad (Pajares & De la Cruz, 2007), es decir, la diferencia en la posición horizontal del punto en cada una de las cámaras, una vez que se ha obtenido esta disparidad, se puede calcular la profundidad, que es inversamente proporcional a la disparidad. A pesar de las muchas ventajas que presenta este método, no está exento de problemas, el más importante de ellos, radica en la necesidad de una calibración exacta de las cámaras, si se realiza una calibración poco precisa, dará como lugar un resultado a su vez poco preciso o erróneo, además, este método tiene problemas en regiones con poca textura (donde la correspondencia es ambigua), o en regiones con oclusiones (donde un punto es solo visible por una de las cámaras) o en superficies con altas reflexiones (como espejos) donde la luz no es absorbida por el material, lo que puede invalidar el resultado final.



Figura 2.2 Ejemplo de imagen estereoscópica, Como se puede observar ambas imágenes son similares, pero no idénticas, a partir de la diferencia de posición de cada elemento en ambas imágenes se puede determinar la distancia a la cámara de cada punto y por tanto la geometría 3D de la imagen. Fuente: artesvisualessemimara.blogspot.com

2.1.2.2 Estructura desde el movimiento

La estructura desde el movimiento o SfM generaliza el principio de la estereoscopia a un escenario más flexible. Tanto (Zisserman, 2004) como (Szeliski, 2010) son claves en esta área puesto que, en lugar de hacer uso de cámaras fijas calibradas, propusieron hacer uso de una secuencia de imágenes (o un video) tomadas por una única cámara que se desplaza a través de una escena, sin conocer a priori las posiciones de dicha cámara.

El objetivo de este método radica en estimar simultáneamente la estructura 3D de la escena, junto con el movimiento de la cámara para cada imagen. Esto dota a SfM de mayor potencia y flexibilidad que el uso de disposiciones estereoscópicas, pero, consecuentemente, a un coste computacional ampliamente mayor, además, uno de los principales problemas de este método radica en el “Drift”, esto es, que el algoritmo pueda acumular pequeños errores de pose en cada imagen, que se puedan ir acumulando, y causen disparidades importantes con la realidad en secuencias largas.

2.1.2.3 Métodos Fotométricos

Otra familia de enfoques clásicos son los métodos Fotométricos, que, no dependen, o al menos no enteramente de la correspondencia geométrica, sino de cómo la iluminación interactúa con las superficies. Dentro de esta familia de métodos destaca el estéreo Fotométrico de (Woodham, 1980) que a diferencia de la visión estereográfica requiere tan solo de una única cámara, desde la cual se toman varias imágenes bajo condiciones de iluminación diferentes pero conocidas,

(por ejemplo, con una luz proveniente de cada extremo de la pantalla en cada imagen).

Asumiendo un modelo de reflectancia concreto, la intensidad de un píxel dependerá de la orientación de la superficie y de la dirección de la luz, esto permite que, al someter al mismo píxel a distintas condiciones de luz conocidas, se pueda resolver algebraicamente su orientación exacta, que, al conjuntarlo con el resto de la imagen, permite obtener el mapa de profundidad de la imagen y por ende su forma 3D. Al igual que los casos anteriores, este método puede presentar desafíos fuera de entornos controlados, o con materiales de reflectancias muy altas.

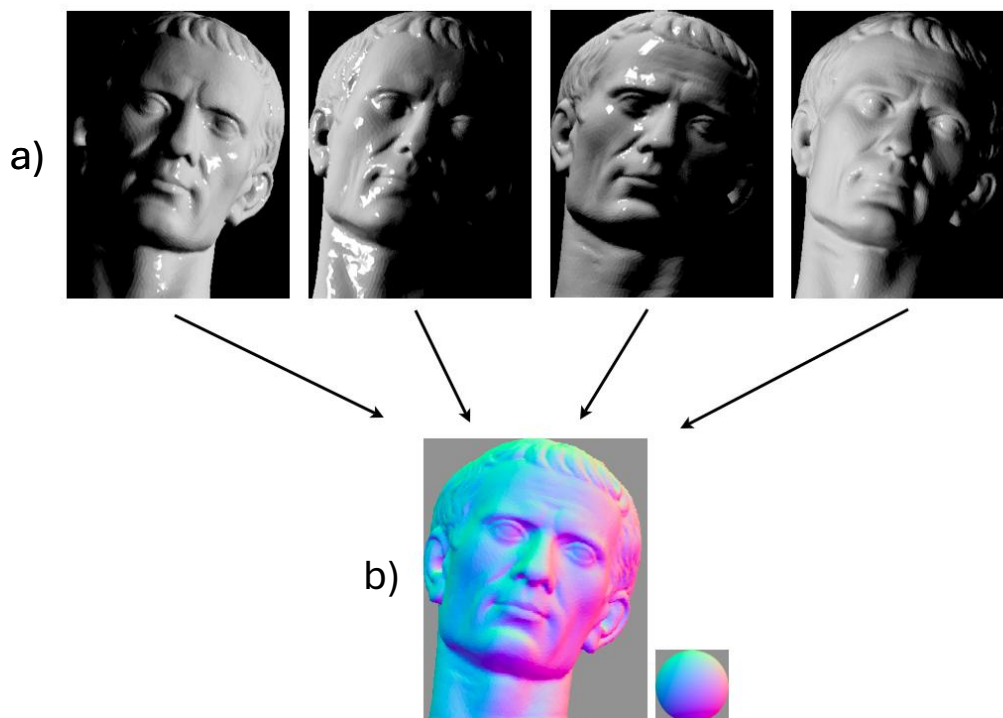


Figura 2.3 Ejemplo de uso del escaneo 3D mediante técnicas fotométricas. Se toman imágenes del objeto bajo distintas iluminaciones (a) y a partir de ellas, se reconstruye su mapa de normales (b). Fuente: dcmsistemas.com

2.1.2.4 Conclusión métodos clásicos

Si bien estos métodos son altamente precisos en entornos controlados, su dependencia de correspondencias fiables, calibración precisa o condiciones de iluminación controladas, los hace menos robustos para escenarios dinámicos o no controlados. Su aplicación en tiempo real en dispositivos portátiles presenta a su vez grandes desafíos computacionales. Estas limitaciones, han propiciado la transición hacia métodos más contemporáneos, principalmente hacia aquellos que hacen uso de aprendizaje profundo.

2.1.3 Estimación de profundidad monocular

Uno de los enfoques quizás más interesantes a la hora de trabajar con reconstrucciones 3D es la estimación de la profundidad a partir de sistemas

monoculares, es decir, a partir de la información de una sola cámara. Este enfoque contrasta con técnicas tradicionales como la visión estereoscópica, que emplean dos cámaras calibradas y dispuestas de forma similar a nuestros ojos para obtener una estimación directa de la profundidad mediante la triangulación de puntos correspondientes en ambas imágenes.

En los sistemas monoculares, en cambio, no disponemos de referencias adicionales que permitan calcular la distancia real a los objetos de una manera directa. Al final, una fotografía es una representación bidimensional de una escena tridimensional, lo que significa que parte de la información, en este caso la de profundidad se pierde en el proceso de captura.

Esta situación es comparable a intentar comprender la disposición espacial de un entorno 3D complejo con un ojo tapado. Al perder la visión estereoscópica, el cerebro ya no puede calcular la distancia a los objetos mediante la diferencia entre ambas imágenes oculares, por lo que debe recurrir a otras pistas visuales, como la iluminación, la perspectiva o las proporciones relativas entre objetos. Esta limitación puede dar lugar a ilusiones visuales como la famosa “Habitación de Ames” (Figura 2.4) un espacio tridimensional diseñado con referencias visuales distorsionadas que, al ser observado desde un punto específico o a través de una cámara, crea la impresión de que una persona puede cambiar de tamaño al desplazarse por la habitación.

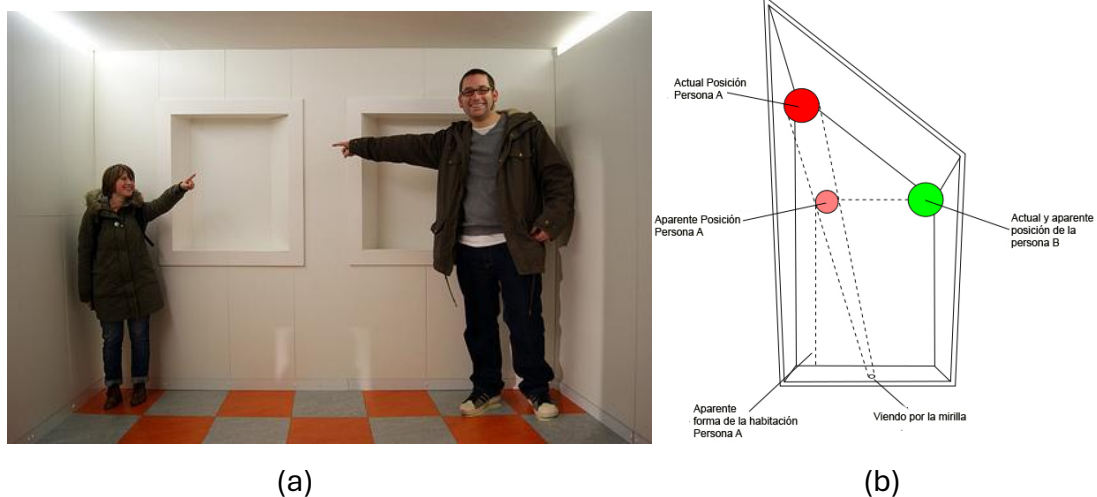


Figura 2.4 Un ejemplo de habitación de Ames. (a) fotografía en la que parece que ambas personas tienen tamaños completamente distintos. Fuente: Pinterest (b) esquema de cómo está diseñada esta ilusión. Fuente: elconfidencial.com

A pesar de este tipo de ilusiones, los métodos de estimaciones monoculares modernos tratan de superar sus limitaciones apoyándose en técnicas de inferencia basadas en el aprendizaje automático, especialmente en redes neuronales entrenadas con grandes cantidades de datos que le permiten al sistema informático extraer patrones visuales relacionados con la estructura tridimensional

de la escena. Estas pistas al igual que con el ojo humano pueden incluir elementos tales como la perspectiva, el tamaño relativo de los objetos, la oclusión entre ellos, los patrones de luces y sombras, las texturas de los elementos de la escena, o incluso al igual que los humanos el conocimiento previo de la forma y tamaño típico de ciertos objetos, tales como puertas, ventanas o edificios.

Respecto a su campo de aplicación, este tipo de estimación es especialmente útil en contextos en los que por restricciones ya sean físicas, económicas o de diseño no se pueda hacer uso de sistemas de múltiples cámaras, como son en, por ejemplo, dispositivos de consumo como ordenadores o algunos dispositivos móviles. A pesar de todas las limitaciones mencionadas, los avances recientes en redes neuronales convolucionales y transformadores (Varma y otros, 2022) han mejorado significativamente la precisión y robustez de los modelos de estimación de profundidad monocular, haciendo posible su aplicación en tiempo real y abriendo la puerta a sistemas de reconstrucción 3D cada vez más accesibles y versátiles.

2.1.4 Mapa de Profundidad

Los mapas de profundidad son una técnica que permite representar la distancia de los objetos en una imagen mediante una escala de colores, generalmente en escala de grises. A diferencia de otras técnicas de representaciones tridimensionales, estos mapas tienen la ventaja de poder almacenarse y procesarse en forma de imagen estándar, lo que facilita su uso junto a técnicas de optimización de imágenes como, por ejemplo, algoritmos de compresión de texturas. (Looking Glass, 2023)

Una de las principales ventajas de este sistema radica en que, al hacer uso de una escala de color para la profundidad, es incluso posible codificar los datos de profundidad en un solo canal de color, como por ejemplo el canal Alpha (de transparencia) que no es usado en imágenes generadas por cámaras convencionales. Esto permite combinar en una sola imagen tanto la información de color como la de profundidad, eliminando la necesidad de archivos adicionales y evitando problemas de sincronización entre ambos datos.

Sin embargo, los mapas de profundidad también cuentan limitaciones. La principal de ellas es que están condicionados por el punto de vista desde el que se generan. Por ejemplo, en la figura de esta sección se muestra el mapa de profundidad de una persona de frente, y como se puede apreciar, el mapa de profundidad solo es capaz de mostrar información de las áreas que son visibles, zonas como la espalda o la parte de atrás de la cabeza no son visibles por lo que el mapa de profundidad es incapaz de representarlo, esto significa que los datos de profundidad obtenidos son sesgados pues limitan la reconstrucción tridimensional a lo captable por el usuario a través de la cámara. En contraste, otras técnicas como las nubes de puntos

permiten capturar información de profundidad desde múltiples ángulos, proporcionando una representación más completa del entorno. (TOPS Marketing, 2025). La figura 2.5 Muestra un ejemplo de mapa de profundidad.

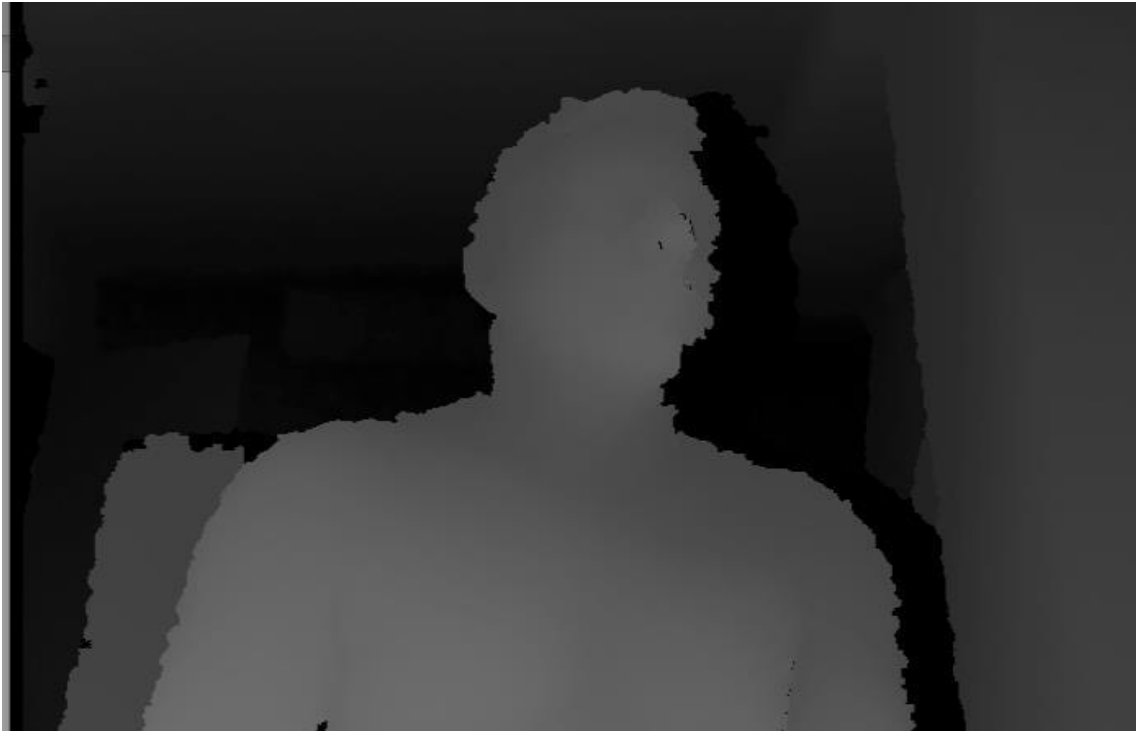


Figura 2.5 Ejemplo de mapa de profundidad. Fuente: researchgate.net

2.1.5 Redes neuronales: modelos convolucionales frente a modelos Transformers

Tal como se ha visto en las secciones anteriores, los mapas de profundidad constituyen una forma compacta y eficiente de representar la información tridimensional de una escena a partir de una imagen. Sin embargo, para generar dichos mapas a partir de una única imagen a color (una tarea intrínsecamente ambigua) es necesario recurrir a modelos capaces de inferir profundidad a partir de pistas visuales en muchas ocasiones sutiles.

En este contexto, el aprendizaje profundo ha demostrado ser especialmente eficaz, y dentro de este enfoque existen dos grandes familias de arquitecturas que han dominado este campo, las redes neuronales convolucionales y las arquitecturas basadas en Transformers. Esta sección se enfocará en definir ambas familias de arquitecturas, así como comparar las ventajas e inconvenientes de cada una.

2.1.5.1 Redes Neuronales

Una red neuronal es una solución de software que utiliza algoritmos de aprendizaje automático para «imitar» las operaciones del cerebro humano. Las redes

neuronales procesan los datos de forma más eficiente y ofrecen mejores capacidades de reconocimiento de patrones y resolución de problemas que los ordenadores tradicionales. Las redes neuronales también se conocen como redes neuronales artificiales (RNA) o redes neuronales simuladas (SNN). Estas redes son un subtipo de aprendizaje automático y un elemento esencial de los algoritmos de aprendizaje profundo. Al igual que su funcionalidad, la arquitectura de una red neuronal también se basa en el cerebro humano (Ortiz, 2025), ya que como su propio nombre indica, estas redes están compuestas por pequeñas neuronas artificiales (representadas en forma de función matemática). cada una de estas neuronas cuenta con distintas entradas y salidas, además de una función de activación que es la que define, que información se generara en las salidas a partir de los datos de entrada de la red. Estas neuronas no trabajan solas puesto que para generar una red es necesario agruparlas en diversas capas interconectadas que permitan el flujo de datos desde una capa inicial (entrada de datos) y una capa final (capa de salida/resultado). La figura 2.6 muestra de manera visual la estructura de una red neuronal. Las redes neuronales se clasifican en distintas familias en función de las capas y neuronas que las comprenden, existiendo en la actualidad muchas familias de redes neuronales, como las CNN, los Transformers, las redes recurrentes o las redes adversarias por nombrar algunas (Agdestein, 2025)

Deep neural network

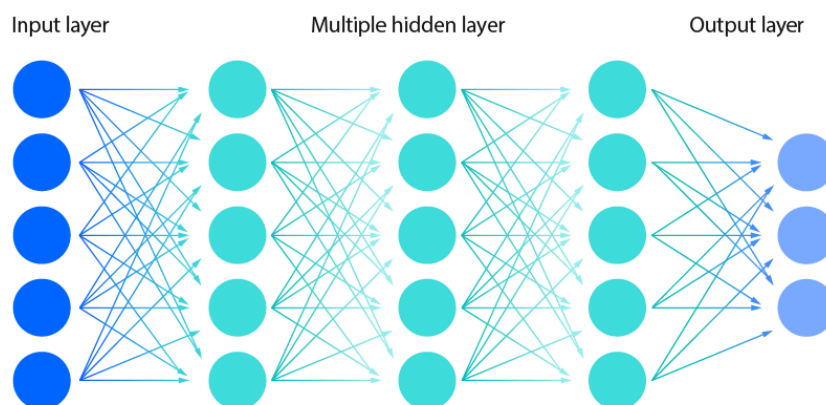


Figura 2.6 Esquema de una red neuronal, las bolas representan las neuronas y las flechas los flujos de datos que las conectan. Fuente: IBM

2.1.5.2 Redes Neuronales Convolucionales (CNNs)

Al igual que otras redes neuronales las CNNs están basadas en distintas capas, lo que la diferencia de otras arquitecturas radica en la forma en la que estas están ordenadas, que en este caso se dividen en tres tipos de capas:

- Capa convolucional

- Capa de agrupación
- Capa totalmente conectada

La capa convolucional es la primera capa de la red neuronal (Departamento de Matemática Aplicada, Universidad Politécnica De Madrid, 2021). Si bien las capas convolucionales pueden ir seguidas de otras capas convolucionales o de capas de agrupación, la capa final es la capa totalmente conectada. Con cada capa la CNN aumenta su complejidad, identificando mayores porciones de la imagen. Las primeras capas se centran en características simples, como colores y bordes. A medida que los datos de la imagen avanzan a través de las capas, la CNN comienza a reconocer elementos o formas más grandes hasta que finalmente identifica el objeto esperado. La Figura 2.7 permite ver visualmente el funcionamiento de este tipo de redes.

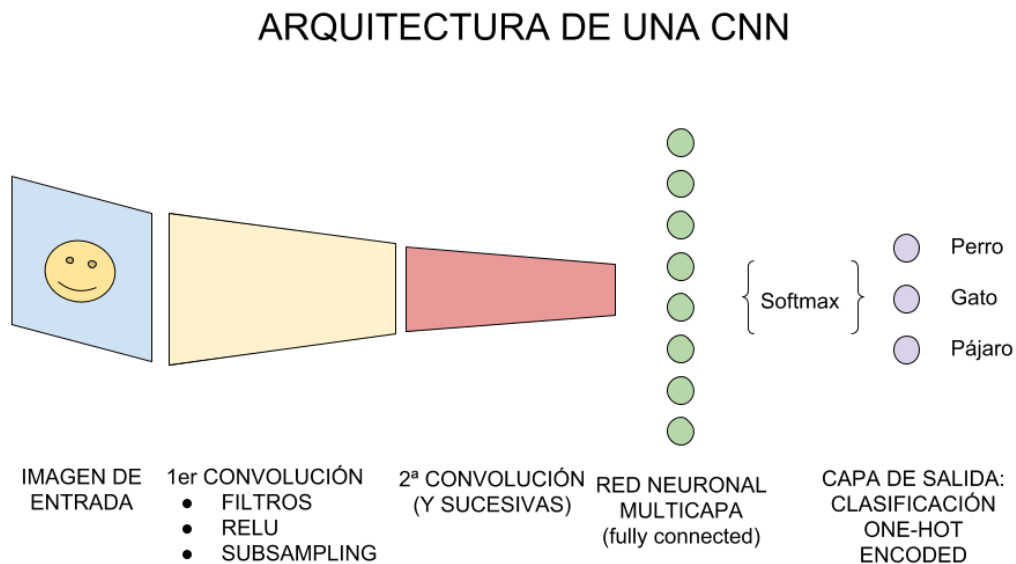


Figura 2.7 Arquitectura de una CNN. Fuente: aprendemachinellearning.com

Explicado de forma sencilla, imaginemos que queremos que una red neuronal sea capaz de reconocer una cara. La primera pregunta que podríamos hacernos es: ¿cómo identificamos una cara? ¿Qué características la definen? Podemos pensar en una cara como un conjunto de elementos más pequeños organizados de cierta manera: normalmente, dos ojos, una nariz y una boca dispuestos en una configuración reconocible.

Una vez identificamos esos elementos principales, nos hacemos las mismas preguntas para cada uno de ellos: ¿cómo distinguimos un ojo? ¿Qué lo caracteriza? Así, seguimos descomponiendo el problema en niveles cada vez más bajos, hasta llegar a identificar pequeños patrones presentes en grupos diminutos de píxeles.

El funcionamiento de una red neuronal convolucional (CNN) sigue esta misma lógica, pero en sentido inverso: comienza detectando patrones simples en regiones pequeñas de la imagen (bordes, texturas, contrastes locales), y a medida que se avanza por las distintas capas de la red, estos patrones se combinan para formar representaciones más complejas, como partes de objetos, y finalmente objetos completos, como una cara.

Esta manera de funcionar hace que este tipo de redes sean ideales para tareas como el reconocimiento de imágenes o, en el caso que nos ocupa, la estimación de profundidad de una forma ágil y eficiente. Sin embargo, como podrá imaginar el lector, también presentan ciertas limitaciones que deben tenerse en cuenta.

La principal de ellas, en el contexto de la estimación de profundidad, proviene del propio diseño jerárquico de las CNN, cuyo funcionamiento se basa en detectar primero patrones pequeños y simples (como bordes o esquinas) e ir combinándolos progresivamente para identificar estructuras más complejas y de mayor escala. Este enfoque ascendente permite abstraer información útil en muchos escenarios, pero también tiene un efecto secundario: la pérdida progresiva de detalle y de contexto global de la escena.

A medida que la información atraviesa las capas de la red neuronal, se va reduciendo la información que pasa de una capa a otra, lo que puede dificultar la reconstrucción precisa de la geometría tridimensional, especialmente en zonas con texturas uniformes, bordes difusos o relaciones espaciales de gran escala. Además, al no disponer de mecanismos explícitos de razonamiento estructurado o relaciones espaciales de largo alcance, las CNN pueden tener dificultades para comprender la disposición global de la escena o distinguir entre ambigüedades visuales.

Por ello, aunque las CNN han demostrado ser muy eficaces en tareas de estimación de profundidad monocular, especialmente en tiempo real, sus limitaciones arquitectónicas han motivado la aparición de nuevos enfoques basados en arquitecturas más expresivas, como los modelos tipo Transformer, que permiten capturar dependencias globales y conservar mejor la estructura espacial de las escenas.

2.1.5.3 Modelos Transformers

El modelo de transformador es un tipo de arquitectura de redes neuronales de reciente creación (su arquitectura fue descrita por primera vez en (Vaswani y otros, 2017)) que está revolucionando el panorama de la inteligencia artificial.

De hecho, el popular Chat-GPT que tantos cambios parece estar trayendo a nuestra vida, en su nombre hace referencia a GPT (Generative Pre-trained Transformer o

Transformer Generativo Preentrenado en español) lo cual ya le da una pista al lector de cuán relevante se está volviendo esta arquitectura en la actualidad.

Pasando a un enfoque más técnico, la arquitectura Transformer ha demostrado una notable eficacia en el procesamiento de datos secuenciales, y ha sido la base de los actuales modelos de lenguaje de gran tamaño (LLM, por sus siglas en inglés). No obstante, su aplicación no se limita al procesamiento del lenguaje natural. En los últimos años, los modelos basados en transformadores han alcanzado resultados de vanguardia en otras áreas clave de la inteligencia artificial (IA), como la visión por ordenador, el reconocimiento automático del habla y la predicción de series temporales. (IBM, 2025)

La característica distintiva de esta arquitectura es el mecanismo de autoatención (self-attention), que permite al modelo ponderar la relevancia de cada elemento de la entrada en relación con los demás. Esto le proporciona una capacidad superior para modelar dependencias a largo plazo y relaciones contextuales, incluso en secuencias muy extensas. A diferencia de las redes neuronales convolucionales (CNN) que utilizan filtros locales y tienen un campo receptivo limitado, los transformadores operan de forma paralela y global sobre la entrada. Además, su arquitectura se compone exclusivamente de bloques de atención y capas de alimentación directa (feedforward), lo que facilita el entrenamiento en paralelo y mejora la escalabilidad.

Esta flexibilidad estructural, combinada con su capacidad para captar relaciones complejas en los datos, ha propiciado su adopción en tareas de visión por ordenador como la segmentación semántica, la clasificación de imágenes y, más recientemente, en la estimación de profundidad monocular.

2.1.6 Detección de Objetos y Regresión de cajas

La detección de objetos es uno de los componentes fundamentales en visión por computador. A fin de cuentas, ¿De qué sirve ver si no podemos saber que estamos viendo? La detección de objetos no solo implica la clasificación de una imagen, sino también identificar la presencia, ubicación y extensión de múltiples objetos dentro de esta. A diferencia de la clasificación simple, este proceso requiere delimitar los objetos mediante cajas delimitadoras (o bounding boxes).

Desde un punto de vista técnico, la detección moderna se fundamenta en el uso de la regresión. Este concepto implica tratar la localización como un modelo matemático en el que una red neuronal procesa la imagen para predecir las coordenadas de las cajas y de las probabilidades de clase en un único paso.

Esta capacidad de “entendimiento” del contexto de una escena es un pilar esencial para tareas más complejas, como son por ejemplo el seguimiento de objetos, la

conducción autónoma o, en el caso de este trabajo, la interacción humano-máquina en entornos inmersivos.

2.1.6.1 El desafío del “Tiempo Real”

Si nos movemos a entornos en tiempo real, la detección de objetos puede volverse un aspecto clave (Zhong-Qiu y otros, 2019) puesto que nos permite saber qué clase de tratamiento debemos darle a cada parte de la escena. Por ejemplo, en el contexto actual, ¿Es relevante procesar la profundidad del fondo del escenario si solo estamos interesados en la persona de la pantalla? La respuesta depende del coste computacional, si el coste de detectar el fondo (o de aplicar una máscara substractiva a todo lo que quede fuera del usuario) es mayor que el coste de procesarlo, no nos merecerá la pena realizar esta tarea. Por el contrario, si procesar la escena detectando y anulando el fondo (usando una máscara de segmentación) tiene un coste computacional menor que si no lo hiciéramos, obtendríamos no solo una mejora en el rendimiento general del sistema, sino también en muchos casos una mejora visual de la imagen al cargar al modelo de procesamiento con menos datos (Yu y otros, 2018)

Una vez planteado el punto anterior, queda claro que el principal desafío aquí se halla en el equilibrio entre la precisión de la detección y la velocidad de inferencia. Históricamente, los modelos más precisos han solido ser también los más lentos, lo que hacía inviable su despliegue en dispositivos con recursos computacionales más limitados o en dispositivos cuyo uso estaba destinado al tiempo real.

Para abordar esa limitación, la investigación de los métodos basados en Deep learning evolucionó desde arquitecturas de “dos-etapas” (Girshick y otros, 2014), que primero proponen regiones de interés y luego ejecutan una clasificación de los elementos de cada una de esas regiones (como por ejemplo hace la familia R-CNN, Regions with CNN features), hacia arquitecturas de “una sola etapa” donde todo este proceso se realizaba de una sola vez, lo que reducía el coste computacional de estos modelos al prescindir de la necesidad de realizar múltiples pasos para el análisis y procesamiento de la imagen.

2.1.6.2 YOLO: Detección en un solo paso

En 2016, hubo una revolución en el campo con la presentación de YOLO (Redmon y otros, 2016), un detector de una única etapa. La filosofía de YOLO podría reducirse en tratar la detección de objetos como si fuera un problema de regresión, en lugar de un pipeline de varios pasos, YOLO aplica una única red neuronal convolucional (CNN) a la imagen completa, para ello, la red divide primero la imagen en una cuadrícula y predice simultáneamente las cajas delimitadoras (bounding boxes), además de las puntuaciones de confianza y las probabilidades de clase para cada celda de la cuadrícula en un solo paso.

Esta arquitectura dota a YOLO de una extraordinaria velocidad al eliminar la necesidad de generar propuestas de región, lo que permitía ya en sus primeras versiones que YOLO pudiera procesar imágenes en tiempo real con tiempos de respuestas inferiores a 25ms

2.1.6.3 Alternativas a YOLO

Aunque YOLO es una arquitectura dominante, no es la única en el ámbito de la detección de una etapa. SSD (Single Shot MultiBox Detector (Liu y otros, 2016) es otra alternativa relevante que compite dentro de este ámbito. La principal diferencia arquitectónica con YOLO radica en que SSD utiliza mapas de características de múltiples escalas para predecir objetos de diferentes tamaños, lo que a menudo le otorga cierta ventaja para detectar objetos pequeños

Sin embargo, a la hora de comparar velocidad y precisión, las versiones más modernas de YOLO (como YOLOv3 y posteriores) generalmente superan a SSD en velocidad pura (FPS), en ocasiones en hasta con el doble de rendimiento. (Redmon & Farhadi, 2018) Esta ventaja es clave en aplicaciones en tiempo real que necesitan apurar al máximo su coste computacional sin requerir de detecciones extremadamente precisas, como es el caso de este proyecto.

2.1.7 Superresolución mediante Aprendizaje Profundo

De forma análoga al desafío de la detección de objetos, muchas de las tareas más exigentes en visión por computador, como la estimación de profundidad monocular, requieren de la búsqueda de un equilibrio entre velocidad de inferencia y la resolución de salida, generalmente a más cantidad de datos de entrada, mayor será el tiempo de inferencia. Por ello para lograr el procesamiento en tiempo real, los modelos de redes neuronales requieren operar sobre imágenes de entrada de baja resolución como 320x240, lo que da como resultado un mapa de profundidad también de baja resolución, carente de detalles y con artefactos visuales. Es en este momento donde entran en juego los reescaladores, algoritmos o modelos de Aprendizaje Profundo que elevan la resolución de la imagen de entrada a una resolución superior, siendo capaces en algunos casos de dotar de incluso mayor detalle a la imagen de salida. Esto es una ventaja puesto que usar un modelo de estimación de baja resolución junto con un buen reescalador puede dar un resultado mejor y más detallado que simplemente usar el modelo de estimación con una imagen de alta resolución.

2.1.7.1 Métodos Tradicionales y sus limitaciones

Tradicionalmente, para aumentar la resolución de una imagen se recurre a algoritmos de interpolación, como la interpolación bilineal o bicúbica. (Gonzalez & Woods, 2018) Estos métodos, aunque computacionalmente muy rápidos, se basan en promedios matemáticos de los píxeles adyacentes a los que se van a añadir. Esto

presenta una limitación pues esta aproximación es incapaz de crear nueva información, simplemente “estiran” la información existente, lo que genera resultados borrosos y con falta de nitidez que no logran recuperar los llamados detalles finos de la escena.

2.1.7.2 Superresolución (SR) basada en Deep Learning

Para superar las limitaciones de las técnicas anteriores, aparecieron las técnicas de superresolución basados en redes neuronales, estas redes son entrenadas con imágenes en muy alta resolución junto con su equivalente en baja resolución. La red debe “aprender” a convertir la imagen de baja resolución en una versión aceptable de la de alta resolución.

Los primeros enfoques en esta área como SRCNN (Super-Resolution Convolutional Neural Network) (Dong & Loy, 2014), demostraron que era plausible el uso de CNNs para esta tarea. Sin embargo, el avance más significativo en esta área provino de la integración de las Redes Generativas Adversarias (GANs), que como su propio nombre indica se basa en una arquitectura en la que dos modelos se enfrentan, uno generando imágenes y otro detectando si la imagen es original o generada, ambos modelos “se enfrentan” mutuamente tratando de superar al otro, mejorando ambos en el proceso (Goodfellow y otros, 2014).

A pesar de esto, los métodos basados en redes neuronales convolucionales, lejos de caer en el ostracismo, evolucionaron para abordar otro gran desafío, el coste computacional (y la velocidad de inferencia). En este contexto, surge FSRCNN (Fast Super-Resolution Convolutional Neural Network) (Dong y otros, 2016), una arquitectura diseñada para acelerar el proceso de superresolución sin sacrificar la calidad de restauración.

A diferencia de su antecesor (SRCNN), que aumentaba la resolución de imagen antes de procesarla, FSRCNN hace uso de una estrategia de reescalado a posteriori, realizando las operaciones de convolución en baja resolución y aumentando la resolución solo en la capa final. Esto permite reducir drásticamente el tamaño de la red y, por ende, el coste computacional, permitiendo así velocidades de procesamiento aptas para el tiempo real, lo cual supone una ventaja sobre opciones más complejas y computacionalmente más costosas como las GANs.

2.1.8 Estabilización temporal en video

Hasta este punto se han revisado los fundamentos teóricos necesarios para convertir imágenes en representaciones tridimensionales. Sin embargo, aún queda por abordar un aspecto crucial cuando se trabaja con secuencias de imágenes: el video. Según la Real Academia Española, un vídeo es un <<sistema de grabación y reproducción de imágenes, acompañadas o no de sonidos, mediante cinta

magnética u otros medios electrónicos>>. En el contexto que nos ocupa, un vídeo puede entenderse como una sucesión de imágenes presentadas a una velocidad suficientemente alta para que el observador las perciba como una secuencia continua y fluida, en lugar de una simple sucesión de imágenes estáticas, como una presentación de diapositivas.

Para lograr esta fluidez visual, no solo es necesario mostrar las imágenes a una tasa adecuada de cuadros por segundo, sino también garantizar la coherencia espacial y temporal entre fotogramas consecutivos. Aquí es donde entra en juego la estabilización temporal de video: un conjunto de técnicas diseñadas para suavizar las transiciones entre fotogramas y reducir artefactos visuales o fluctuaciones no deseadas en la representación 3D a lo largo del tiempo.

Estas técnicas pueden ir desde enfoques simples como filtros de medias móviles, hasta métodos más sofisticados basados en el análisis del flujo óptico (donde se intenta predecir a partir de varias imágenes anteriores como se moverán los píxeles a continuación), que permiten detectar y corregir variaciones inconsistentes en la profundidad o la posición de los elementos reconstruidos. La estabilización temporal resulta especialmente relevante en el desarrollo de estimaciones de profundidad monoculares, donde estamos usando redes neuronales para inferir información que no se encuentra directamente en la imagen (información de profundidad), ya que incluso pequeñas inestabilidades o diferencias entre imágenes sucesivas pueden afectar negativamente al resultado.

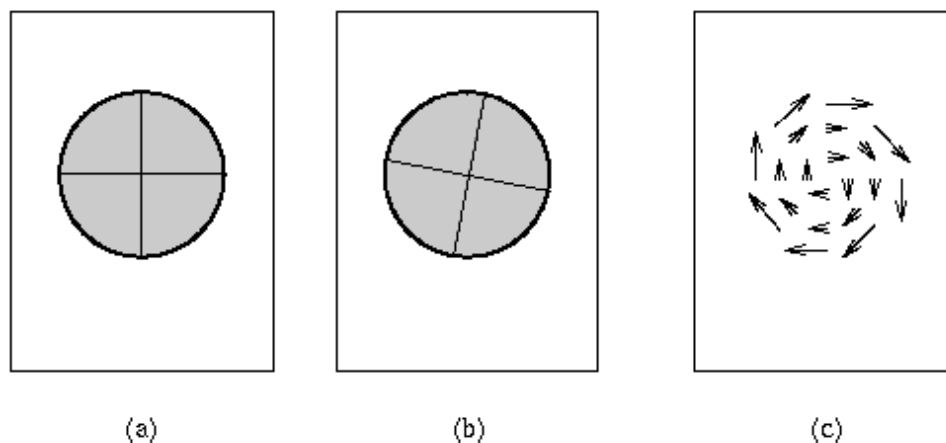


Figura 2.8 Ejemplo de análisis de flujo óptico, a: figura original, b: figura movida, c: flujo óptico representado por flechas del objeto de (a) a (b). Fuente: uiowa.edu

2.2. Revisión bibliográfica

La reconstrucción tridimensional a partir de imágenes ha sido objeto de intensa investigación dentro del campo de la visión por computador. Tradicionalmente, este

desafío ha sido abordado mediante técnicas multi-vista que emplean múltiples cámaras para recuperar la geometría de una escena, tales como la visión estereoscópica o la fotogrametría. Sin embargo, la creciente demanda de soluciones más accesibles, móviles y de bajo coste ha impulsado el desarrollo de métodos que operan con una única imagen como entrada: la reconstrucción 3D monocular.

Desde los trabajos pioneros en visión estereoscópica hasta los métodos basados en redes neuronales actuales, la reconstrucción 3D ha recorrido un camino que combina avances geométricos, computacionales y de hardware. Esta sección presenta una revisión de los principales trabajos académicos en torno a tres ejes fundamentales: los enfoques geométricos clásicos de reconstrucción, los avances en estimación de profundidad monocular mediante aprendizaje profundo y los desarrollos recientes orientados a la reconstrucción 3D en tiempo real para entornos inmersivos como la realidad virtual o la telepresencia.

2.2.1 Aprendizaje Profundo para estimar la profundidad

A diferencia del enfoque clásico basado en métodos geométricos para calcular la profundidad de una escena, los avances en aprendizaje profundo han permitido la llegada de nuevos métodos que han abierto la puerta a soluciones anteriormente inviables, como la estimación de profundidad a partir de una única imagen.

2.2.1.1 Fundamentos del aprendizaje supervisado

El trabajo fundacional de (Eigen y otros, 2014) fue el primero en demostrar la viabilidad del uso de Redes Neuronales Convolucionales (CNN) para la estimación monocular. Su propuesta introdujo una arquitectura multi-escala que predecía la profundidad a nivel grueso y la refinaba progresivamente. No obstante, este enfoque requería de datos etiquetados previamente, obtenidos normalmente a través de sensores LiDAR, lo que limitaba la capacidad de escalabilidad del sistema, así como su adaptación a nuevos entornos.

2.2.1.2 Transición al aprendizaje no supervisado

Para superar la dependencia del uso de sensores activos, (Godard y otros, 2017) plantearon un enfoque no supervisado que utilizaba pares de imágenes estéreo durante el entrenamiento. El objetivo de la red era reconstruir la imagen de un ojo a partir de la del otro, obteniendo por el camino los datos de profundidad del sistema. Este enfoque reducía los costes operativos, al mismo tiempo que mejoraba enormemente la escalabilidad del proyecto respecto al trabajo anterior.

2.2.1.3 Estado del arte y generalización (MiDaS)

Más recientemente, la arquitectura MiDaS (Ranftl y otros, 2020), ha redefinido el estado del arte en términos de robustez y generalización. Su éxito no solo se debe a una arquitectura de red más avanzada, sino fundamentalmente a dos aspectos

principales. En primer lugar, la integración de las redes neuronales de tipo Transformers (véase sección 2.1.5.3) a partir de MiDaS V3 que permite capturar el contexto global de la escena de manera más efectiva que las CNN convencionales. En segundo lugar, la estrategia de entrenamiento mixta, dado que MiDaS se entrenó en un conjunto de datos masivo y diverso que fusionaba varios datasets obtenidos a través de diversos sensores, lo que mejora su capacidad de generalizar escenas y adaptarse a situaciones nuevas.

A pesar de lo anterior, este tipo de sistemas todavía enfrentan desafíos importantes en términos de estabilidad temporal, precisión en texturas planas y eficiencia computacional, especialmente en contextos interactivos o en dispositivos de bajo consumo, esto motiva los objetivos de este proyecto.



Figura 2.9 Ejemplo de resultados de Intel MiDaS, (a) la imagen de entrada, (b) el mapa de profundidad obtenido por la red neuronal. Fuente: pyimagesearch.com

A pesar de estos avances en aprendizaje automático, la literatura señala que estos modelos aún deben enfrentar desafíos en términos de estabilidad temporal así, como de eficiencia computacional, sobre todo en sistemas de bajos recursos, aspectos clave que constituyen el núcleo de la investigación de este trabajo.

2.2.2 Reconstrucción y transmisión 3D en tiempo real

Más allá de la estimación de profundidad estática, el reto se traslada a la reconstrucción 3D en tiempo real, un requisito esencial para aplicaciones inmersivas como la realidad aumentada, la telepresencia o los sistemas de holografía interactiva, que demandan no solo la captura de geometrías sino

también su transmisión y renderizado con la mínima latencia y volumen de datos posibles.

2.2.2.1 Reconstrucción explícita y Captura Volumétrica

En esta línea destacan proyectos como Holoportation de Microsoft (Orts-Escolano y otros, 2016). Este sistema fue pionero en demostrar una telepresencia 3D de cuerpo completo y de alta calidad, ya que logra transmitir representaciones volumétricas de personas a través de redes de alta velocidad, usando cámaras RGB-D (Kinect) que rodean al sujeto. El resultado es un modelo 3D texturizado que se genera en tiempo real y puede ser visionado por otro usuario a través de un dispositivo de computación espacial como HoloLens. Sin embargo, como se verá más adelante (en la sección 2.3.1) esta tecnología no está exento de retos que se deben superar, como el coste del equipamiento, los artefactos de fusión o el ancho de banda requerido por el sistema.

Otro trabajo fundacional en la reconstrucción 3D en tiempo real (aunque enfocado en el escaneo y no en la telepresencia), es KinectFusion (Newcombe y otros, 2011). Este sistema demostró que con un solo sensor RGB-D (Kinect) en movimiento se podía escanear de manera incremental un entorno real denso en tiempo real haciendo uso de una representación volumétrica en la GPU.

2.2.2.2 Representaciones Neuronales (NeRF)

Para superar los problemas de almacenamiento y transmisión de los sistemas anteriores han aparecido nuevas líneas de investigación basadas en representaciones implícitas, es decir, que no almacenan la geometría directamente, sino que la codifican en pesos de una red neuronal.

El trabajo fundamental de esta línea NeRF (Neural Radiance Fields) (Mildenhall y otros, 2020) ha demostrado que es posible representar una escena 3D compleja haciendo uso de una función continua de bajo coste computacional. Esta red aprende a mapear una coordenada 3D y una dirección de visión al color y la densidad volumétrica en ese punto.

La ventaja de este sistema, radica en que la representación 3D de la escena se vuelve extremadamente compacta al limitarse a los pesos de la red neuronal.

2.2.3 Optimización de Pipelines mediante Regiones de Interés

Tal como se planteó en la sección 2.1.5 el uso de algoritmos y modelos de detección de objetos pueden ser de gran ayuda para reducir el área de trabajo y por tanto el tiempo de inferencia o procesamiento del algoritmo principal. Esta línea de trabajo no es original ya que en la literatura podemos encontrar amplios trabajos que hacen uso de este enfoque.

Desde un punto de vista histórico, uno de los ejemplos más influyentes y exitosos de este enfoque es el trabajo de (Viola & Jones, 2001) sobre detección de rostros en tiempo real. Su detector en cascada a pesar de que no utilizaba redes neuronales profundas, se basa en este mismo enfoque, hacer uso de una serie de clasificadores progresivamente más complejos aplicados a una región, si la región era rechazada por un clasificador simple y rápido, esta región ya no era analizada posteriormente por clasificadores más complejos y lentos, de esta manera el sistema ahorra recursos y optimizaba drásticamente la inferencia al poder descartar zonas de la imagen en etapas incipientes.

2.2.3.1 el enfoque de “dos etapas” en Deep learning

La idea de “filtrado” o “atención regional” ha sido adoptada extensamente en la era del Deep learning, consolidándose en la arquitectura de detectores de dos etapas.

La familia de trabajos iniciada por (Girshick y otros, 2014) con R-CNN (Regions with CNN features) es el exponente principal de esta línea.

- En R-CNN, la primera etapa consistía en un algoritmo externo (como Selective Search, (Uijlings y otros, 2013) que generaba un conjunto de “propuestas de región”. La segunda etapa era un clasificador CNN que se aplicaba solamente a estas regiones de interés en vez de sobre la imagen entera.
- Los trabajos subsecuentes como Fast R-CNN (Girshick, 2015) o más adelante Faster R-CNN (Ren y otros, 2016), integraron la primera etapa dentro de la propia red neuronal. Además, en Faster R-CNN, una subred es la que actúa como detector rápido de “atención”, proponiendo las regiones que luego son refinadas por el detector principal.

La literatura, evidencia que el uso de modelos de dos etapas (localizar-procesar) pueden dar resultados más rápidos, el procesamiento en una sola etapa.

2.2.3.2 Aplicaciones específicas para optimización del Pipeline

El enfoque de aplicar un prefiltro para descartar zonas sin interés, no es exclusivo a la detección genérica de objetos y ha sido la arquitectura de facto en dominios especializados que requieren alta precisión:

- **Análisis de imágenes médicas:** Numerosos estudios sobre detección de patologías emplean un pipeline de dos etapas. Primero un modelo de alta sensibilidad como U-Net (Ronneberger y otros, 2015) o similar, para localizar candidatos potenciales, para posteriormente, usando modelo clasificador más pesado sobre los candidatos, se pueda reducir el número de falsos positivos. (Litjens y otros, 2017)
- **Reconocimiento Óptico de Caracteres (OCR):** Los sistemas modernos de OCR, sobre todo aquellos que tienen lugar en entornos no controlados,

suelen hacer uso de este enfoque. Un primer modelo como por ejemplo CRAFT (Baek y otros, 2019) o EAST (Zhou y otros, 2017) detecta las regiones que podrían contener texto, y un segundo modelo procesa únicamente las regiones acotadas para reconocer los caracteres.

- **Estimación de Pose Humana:** En la estimación “Top-Down” esto es de reconocimiento más general a más específico en cada paso, un detector de personas localiza primero a cada individuo, y un modelo de estimación de pose, se ejecuta sobre el área de cada persona detectada. Esto resulta más preciso que los métodos “Bottom-Up”, esto es, de específico a general, que tratan de encontrar primero las articulaciones o “joints” e ir agrupándolas hasta tener a la persona completa.

Todos estos trabajos demuestran la utilidad de realizar una segmentación inicial en la imagen para captar elementos de interés de manera que pueda optimizarse el uso de recursos en el proceso o pipeline de un proyecto, lo cual es un aspecto crucial para proyectos que busquen la optimización de tiempo de ejecución o directamente la ejecución en tiempo real.

2.2.4 Optimización de Pipelines mediante Superresolución

Tal como se mencionó en la sección 2.1.6 y al igual que el punto anterior, el uso de escaladores puede suponer un más que interesante aporte en calidad y rendimiento a otros sistemas de visión por computador, ya que uno de los mayores desafíos que enfrenta esta área, sobre todo en aplicación en tiempo real como la telepresencia o la realidad aumentada, radica en la gestión de la carga computacional. La resolución de la imagen de entrada impacta de manera directa y crítica en el tiempo de inferencia de los modelos de Deep learning, por lo que trabajar con imágenes de baja resolución suele ser una manera eficiente computacionalmente de mejorar el rendimiento bruto de un sistema.

Es en este espacio donde entra en juego el concepto de pipeline desacoplado o por partes. Esta metodología desacopla la etapa de procesamiento de datos de la de presentación final, dividiéndola en dos fases diferenciadas.

- **Procesamiento a Baja Resolución:** El algoritmo principal, que es el que representa mayor coste computacional, se ejecuta sobre una imagen de resolución muy baja, esto permite que su procesamiento se haga de manera muy veloz.
- **Reescalado Inteligente (Superresolución):** La salida de baja resolución obtenida en el paso anterior se introduce en una segunda red neuronal, mucho más ligera que se encarga de generar un resultado en alta resolución.

Este modelo, basado en dos etapas (Procesamiento-Bajo + Reescalado) ha probado ser significativamente más eficiente que el modelo de una sola etapa

(Procesamiento alto) a través de estudios como (Shi y otros, 2016) que introducían un método para reescalados por CNN en tiempo real.

2.2.4.1 Reescaladores Clásicos versus Superresolución

Antes de dar por finalizada esta sección es crucial diferenciar entre los enfoques clásicos y la superresolución por IA, tal como se vio en la anterior sección del trabajo (véase 2.1.6.1). Los super reescaladores aportan mucho más que una mejora de resolución en la imagen, sino que son capaces de generar información o detalles nuevos si son entrenados para ello, esto no solo mejora la calidad aparente de la imagen sino que tiene utilidad práctica en escenarios en los que no es tan importante la exactitud a nivel de píxel del resultado final, sino la exactitud en características de la imagen. (Ledig y otros, 2017)

2.2.4.2 Validación de la Industria y la Academia

La validación más relevante de esta metodología, la podemos encontrar en la industria de los gráficos por computador. Tecnologías como NVIDIA DLSS (Deep Learning Super Sampling) o AMD FSR (FidelityFX Super Resolution) son la implementación directa de este concepto. El motor de videojuegos renderiza la escena a una resolución interna baja y una red neuronal escala esa imagen a una resolución mayor en tiempo real. Esto permite obtener tasas de fotogramas más altas que si la imagen se renderizara a resolución nativa y a veces, (gracias a la capacidad de “alucinar” de la red neuronal), obtener imágenes más detalladas que la que se obtendría por el método tradicional de un solo paso.

Más allá del sector de los gráficos por ordenador y entrando ya en un terreno más académico, podemos encontrar también implementaciones de esta metodología en el uso de algoritmos de reescalado por IA para mejorar la salida de sensores de bajo coste (como sensores de profundidad de baja resolución (Min y otros, 2017)) o incluso para optimizar el streaming de video, donde se envía un flujo en baja resolución al dispositivo cliente el cual se encarga de mejorar el resultado final antes de mostrárselo al usuario (Da Mata y otros, 2024).

Como conclusión, el uso de reescaladores de IA en el pipeline de este trabajo no se plantea como un mero postprocesado estético sino como una decisión metodológica clave para optimizar el núcleo algorítmico de manera eficiente y mejorar el resultado visual al mismo tiempo.

2.3. Análisis crítico de las soluciones y tecnologías existentes

Tras haber revisado las principales técnicas, herramientas y enfoques relacionados con la reconstrucción 3D, la estimación de profundidad monocular y los sistemas de visualización en tiempo real, en esta sección se realiza un análisis crítico de los

proyectos y productos existentes. El objetivo es identificar las fortalezas y limitaciones de las soluciones actuales, así como detectar vacíos o áreas de mejora que justifiquen la orientación del presente trabajo.

2.3.1 Microsoft Holoportation

Microsoft Holoportation es una tecnología desarrollada por Microsoft Research que permite la transmisión y reconstrucción 3D en tiempo real de personas a distancia, con el objetivo de generar experiencias inmersivas que simulan la presencia física remota. Se basa en capturar una representación tridimensional completa de una persona mediante un conjunto de cámaras RGB-D (color + profundidad), comprimir esa información, transmitirla a través de una red y renderizarla en tiempo real en un entorno 3D a través de dispositivos HoloLens (unos dispositivos de realidad mixta desarrollados por Microsoft)

Holoportation resuelve adecuadamente el problema de la compresión de datos y de la latencia, además de la generación de mallas 3D en tiempo real en el dispositivo resultante, pero cuenta con el hándicap de que requiere de varios sistemas de cámaras y profundidad (en este caso dispositivos Kinect) para llevar a cabo el escaneo. Esto es un factor limitante para la adopción de esta tecnología puesto que requiere que el usuario cuente con varios de estos dispositivos colocados y calibrados en la sala que quiere usar para hacer uso del sistema, además de un ordenador capaz de procesar la información de los sensores y otro para la reconstrucción temporal, además de hacer uso de una red interna de 10 gigabits, (Orts-Escolano y otros, 2016) esto lo aleja del objetivo de simplicidad y accesibilidad que trata de resolver el presente trabajo.



Figura 2.10 Microsoft Holoportation en funcionamiento, la persona de la izquierda es un modelo 3D retransmitido en tiempo real que la persona de la derecha puede ver a través de su dispositivo HoloLens.

Fuente: Microsoft

2.3.2 Meta Codec Avatars

La tecnología de Codec Avatar de Meta es una tecnología en proceso de desarrollo por la empresa Meta cuyo fin es el desarrollo de avatares realistas para la comunicación en tiempo real entre personas a través de dispositivos de realidad. Este proyecto tiene un enfoque totalmente distinto al aquí planteado, puesto que, a diferencia de este, Meta no hace uso de mapas de profundidad en tiempo real, ni de cámaras en tiempo real directamente. En su lugar Codec Avatar se divide en dos partes, una primera parte que hace uso de un dispositivo móvil, para escanear la cara del usuario mediante fotogrametría y una segunda fase en la que ya el usuario tiene el visor puesto y donde el sistema reconoce las expresiones faciales del usuario para transferirlas al modelo en cuestión (Ma y otros, 2021), este es un procedimiento inteligente de cara a reducir el uso de recursos en tiempo real pero más hermético puesto que requiere un escaneo personalizado para cada usuario en vez de una solución completamente en tiempo real.

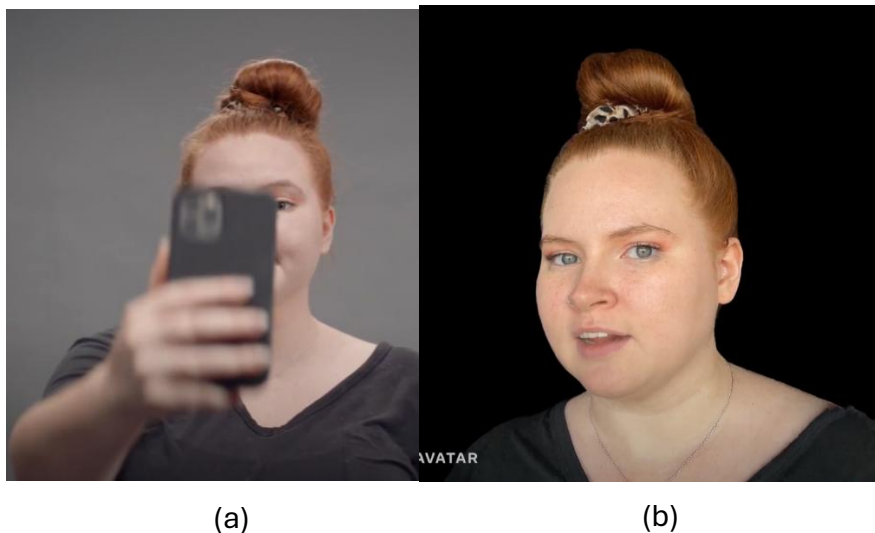


Figura 2.11 (a) Proceso de escaneo del usuario. (b) Codec Avatar Generado por Meta. Fuente Meta Platforms. Inc.

2.3.3 Google Beam

Google Beam, anteriormente conocido como Proyecto Starline durante sus fases iniciales de desarrollo, es una propuesta de comunicación en tiempo real que busca ofrecer una experiencia tridimensional realista sin necesidad de utilizar dispositivos de realidad virtual. Anunciado durante la conferencia Google I/O en 2021, este sistema permite a los usuarios mantener una conversación cara a cara con otra persona representada mediante un modelo 3D fotorrealista. La interacción se realiza de forma natural, incluyendo contacto visual, gestos y sincronización labial, gracias a una combinación de técnicas de visión por computador, aprendizaje automático, audio espacial y algoritmos de compresión en tiempo real. (Nartker, 2025)

En 2025, en colaboración con HP, se presentó el primer producto comercial basado en esta tecnología: HP Dimension, con un precio de lanzamiento de 25.000 dólares. Este sistema utiliza una pantalla holográfica de alta resolución junto con un conjunto de siete cámaras para capturar la imagen del usuario desde distintos ángulos, y está principalmente orientado a salas de reuniones empresariales o entornos corporativos de alto nivel.

Al igual que Holoportation de Microsoft, Google Beam representa un avance notable en comunicación inmersiva, pero su dependencia de un hardware específico, costoso y poco portátil limita su adopción generalizada. Además, su instalación requiere de un espacio fijo y controlado, lo cual entra en conflicto con los objetivos de simplicidad, accesibilidad y bajo coste que plantea el presente trabajo. No obstante, los algoritmos de compresión y transmisión desarrollados por Google en este contexto podrían representar una línea de investigación valiosa para futuras soluciones de comunicación volumétrica en tiempo real más asequibles.

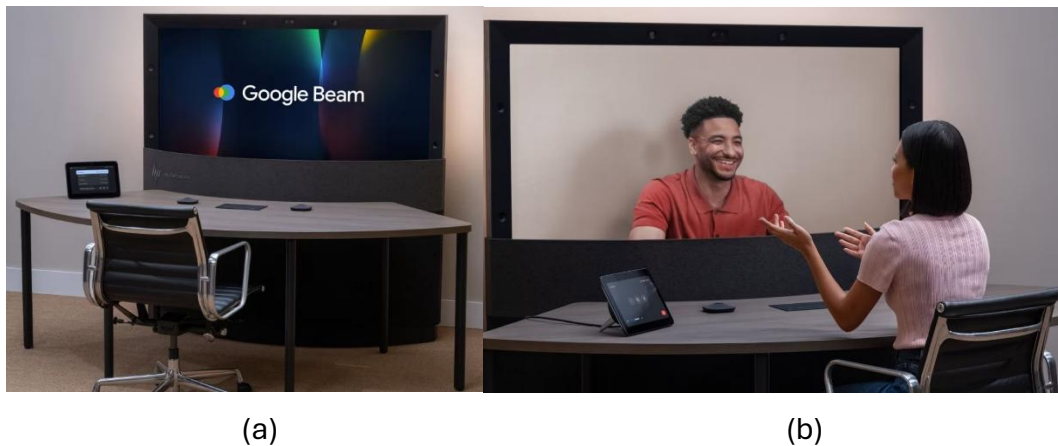


Figura 2.12 Google Beam. (a) sistema HP Dimension, (b) sistema usándose con Google Beam, donde la persona de rojo es un modelo 3D fotorrealista transmitido en tiempo real. Fuente: HP

2.3.4 Z-Buffer 3D (Z3D) de VorpX y similares

Uno de los sistemas cuyos resultados más se aproximan a lo que se pretende construir es Z3D de VorpX, una herramienta capaz de convertir en tiempo real una salida de imagen 2D de un videojuego en una representación 3D estereoscópica, permitiendo su uso con dispositivos como gafas de realidad virtual o monitores 3D. (VorpX, s.f.) Sin embargo, esta tecnología tiene una particularidad importante: aunque logra resultados con un alto grado de realismo visual en tiempo real, se basa en el uso del Z-Buffer, una estructura interna del motor gráfico que almacena información de profundidad por píxel y se emplea principalmente para calcular oclusiones entre objetos. (Microsoft Corporation, 2024) VorpX aprovecha estos datos, junto con la imagen de la pantalla, para generar la percepción tridimensional. Debido a esta dependencia del Z-Buffer (disponible únicamente dentro de motores de videojuegos) esta técnica no puede aplicarse de forma

general fuera de ese contexto. Por ello, en el marco de este proyecto, solo resultan útiles los algoritmos que generan profundidad a partir de una imagen ya renderizada, es decir, sin acceso al motor gráfico original.



(a)

(b)

Figura 2.13 Imagen 3D generada haciendo uso de un algoritmo de Z-Buffer. Pueden verse algunos defectos en la imagen generada (a) respecto a la original (b) entre los dos personajes debido a la falta de información de color a la hora de reconstruir la imagen. Fuente Paradise Decay

2.3.5 Soapbox

Soapbox es una plataforma de experiencias inmersivas en realidad aumentada y virtual (AR/VR), centrada principalmente en ofrecer actuaciones musicales en forma de hologramas 3D de artistas como T-Pain o Waka Flocka Flame, accesibles a través de dispositivos como Meta Quest. El sistema permite al usuario disfrutar de conciertos en formato volumétrico, a partir de modelos 3D previamente escaneados mediante un sistema de captura multicámara (hasta 48 cámaras), lo que garantiza un alto nivel de fidelidad geométrica y visual. (soapbox, s.f.)

Aunque el contenido no se transmite en tiempo real —sino que es grabado en estudio y descargado localmente en el visor del usuario—, Soapbox representa una de las experiencias más cercanas a una videollamada volumétrica actualmente disponibles para el gran público. Esto se debe a su capacidad de reproducir mallas 3D animadas en tiempo real sobre hardware limitado, como el Meta Quest 2, basado en la plataforma Qualcomm XR2 (Sag, 2019) (una versión adaptada del Snapdragon 865 utilizado en móviles como el Samsung Galaxy S20 o el OnePlus 8).

Este ejemplo demuestra que la representación de cuerpos humanos en 3D en movimiento es técnicamente viable incluso en dispositivos con capacidad de

cómputo moderada, lo que refuerza la propuesta de este trabajo de explorar soluciones ligeras de reconstrucción 3D para su uso hacia el gran público.



Figura 2.14 Soapbox. Un cantante virtual canta y se mueve dentro del espacio físico real del usuario, funcionando en unas Meta Quest 3/3S. Fuente: Horizon Store

2.4 Conclusiones

Tras el análisis comparativo de las soluciones existentes, queda patente que a pesar de que se han desarrollado avances significativos en la reconstrucción y transmisión 3D en tiempo real, la mayoría de estas propuestas dependen de sistemas e infraestructuras complejas acompañadas de hardware especializado, y, consecuentemente, elevados costes de implementación. Las siguientes tablas presentan una comparación resumida, así como una evaluación de la adecuación al presente proyecto de las tecnologías mostradas en la sección 2.3

Sistema	Tecnología	Hardware Requerido	Trasmisión en tiempo real	Realismo alcanzado
Microsoft Holoportation	Cámaras RGB-D + reconstrucción volumétrica multivista	Múltiples Kinects, PC de alto rendimiento, red 10 Gbps	Sí	Elevado
Meta Codec Avatar	Escaneo previo + redes neuronales para animación facial	Cámaras móviles + visor VR	Parcial (solo animación facial)	Elevado
Google Beam	Multicámara + compresión volumétrica + pantalla holográfica	Sistema cerrado con 7 cámaras y pantalla holográfica	Sí	Muy elevado

Z-Buffer 3D	Extracción del búfer de profundidad en motores gráficos	GPU compatible y software específico	Solo en entorno cerrado	Medio
Soapbox	Captura multicámara + renderizado volumétrico pregrabado	Múltiples cámaras (48) + visor VR	No (Pregrabado)	Elevado

Tabla 2.1 Comparación de las tecnologías analizadas

Sistema	Adecuación al presente trabajo	Justificación
Microsoft Holoportation	Poco viable	Coste muy alto, complejidad de instalación, no portátil
Meta Codec Avatar	Parcial	No genera profundidad en vivo, requiere calibración por usuario
Google Beam	Poco viable	Coste elevado, dependencia de hardware propietario
Z-Buffer 3D	Parcial	Solo aplicable a motores de videojuegos, no generalizable
Soapbox	No viable para comunicación	No en tiempo real, requiere estudio de captura

Tabla 2.2 Adecuación de las tecnologías al presente trabajo

A partir de la revisión anterior se pueden obtener las consecuentes conclusiones:

1. Dependencia de hardware especializado: Todas las soluciones analizadas requieren de sistemas especializados como múltiples cámaras o sensores de profundidad, esto restringe su escalabilidad y lo aleja de sistemas más limitados como son los contextos domésticos o móviles
2. Coste computacional y de infraestructura: Los sistemas más precisos como Holoportation o Beam requieren de redes de alta velocidad y de procesamiento distribuido, alejándolos de nuevo de su utilización en sistemas en tiempo real en equipos convencionales
3. Falta de equilibrio entre realismo y eficiencia: Existen soluciones extremadamente realistas, pero poco eficientes (como Codec Avatar) y otras muy ligeras pero limitadas en fidelidad y contexto (como Z-Buffer 3D). No se ha consolidado todavía ningún estándar capaz de combinar ambos aspectos
4. Oportunidad para el enfoque monocular: En este contexto, la estimación de profundidad monocular a través de redes neuronales emerge como una alternativa prometedora. Permite generar reconstrucciones tridimensionales aceptables usando hardware mínimo, alineándose con

los objetivos de accesibilidad, bajo coste y funcionamiento en tiempo real que persigue el actual proyecto.

Como conclusión, las soluciones revisadas muestran un alto grado de madurez tecnológica, pero presentan grandes carencias en el término de accesibilidad, eficiencia y flexibilidad. Estas limitaciones justifican la orientación del presente trabajo hacia el desarrollo de un sistema de reconstrucción 3D monocular optimizado para tiempo real, capaz de ofrecer una experiencia inmersiva sin depender de equipos especializados ni configuraciones de hardware/software complejas.

3. Solución Aportada

Tal como se ha visto en el segundo capítulo, aunque existen numerosas soluciones avanzadas para la reconstrucción 3D en tiempo real, muchas de ellas requieren hardware específico o infraestructuras complejas, lo que limita su adopción masiva. En este contexto, el presente trabajo se orienta hacia una solución más accesible basada en estimación de profundidad monocular, con especial atención a la eficiencia, portabilidad y coste. Este capítulo se centrará en definir la solución propuesta, cuál es su arquitectura y cuál ha sido su proceso de desarrollo e implementación.

3.1 Descripción de la Solución

La solución propuesta en este trabajo consiste en la evaluación de la viabilidad de un sistema dirigido a videollamadas en tiempo real basado en un pipeline desacoplado que permita capturar, procesar, transmitir entre dispositivos y renderizar en 3D imágenes a partir de una única fuente de vídeo haciendo uso solamente de técnicas de software, esto es, sin ningún tipo de hardware externo adicional (Kinect, LiDAR, etc.)

El núcleo de la solución lo compone un sistema desacoplado que divide la carga de trabajo en dos etapas críticas, captura e inferencia de la profundidad de la escena, por un lado, y reconstrucción inmersiva por el otro. Pudiendo consecuentemente, ambas etapas ejecutándose en equipos informáticos distintos (como en una videollamada tradicional)

A diferencia de las soluciones analizadas en el estado del arte como pueden ser Google Beam o Microsoft Holoportation, que dependen de hardware especializado y costoso junto con una gran demanda de recursos computacionales, esta propuesta busca democratizar el acceso a la comunicación volumétrica mediante el uso de modelos de aprendizaje profundo infiriendo la información tridimensional perdida durante la captura 2D.

La solución aborda este problema mediante tres procesos simultáneos, que son:

- Uso de algoritmos de detección para aislar al usuario del entorno, optimizando así el ancho de banda y centrando el cómputo en el sujeto de interés (el usuario)
- Generación de mapas de profundidad mediante modelos de aprendizaje profundo (MiDaS), mejorados posteriormente mediante técnicas de Superresolución personalizadas (FSRCNN + fine-tuning).
- Empaquetado y transmisión de la información codificada en forma de imagen RGBA con la profundidad en el canal Alpha que permita reconstruir posteriormente el modelo 3D en un cliente externo (Unity3D)

Esta arquitectura se ha seleccionado específicamente para evitar las barreras de entrada económicas detectadas en el estado del arte, al mismo tiempo que resolver el cuello de botella del procesamiento de imágenes en alta resolución haciendo uso de una estrategia de superresolución en dos etapas.

3.1.1 Objetivos del sistema

Para considerar la solución válida y alineada con los objetivos generales del proyecto, el sistema debe satisfacer los siguientes requisitos:

3.1.1.1 Requisitos Funcionales

- El sistema ha de ser capaz de generar una representación 3D coherente a partir de una única fuente de vídeo 2D
- Debe identificar y aislar a la persona del fondo sin intervención manual
- La información de color y profundidad debe transmitirse simultáneamente y de forma sincronizada para evitar artefactos visuales durante el movimiento.
- El sistema debe permitir la interconexión entre dos usuarios remotos, simulando una videollamada volumétrica.

3.1.1.2 Requisitos No Funcionales

- El sistema debe mantener una tasa de rendimiento estable y suficiente para la percepción de movimiento (con un objetivo de alcanzar al menos 20 FPS constantes en hardware de consumo actual o inmediatamente futuro)

- El tiempo transcurrido entre la captura y la visualización, esto es, la latencia del sistema, debe ser lo suficientemente bajo como para permitir una interacción comunicativa natural.
- Debe funcionar con cámaras web estándar y ordenadores orientados al mercado de consumo. Sin requerir sensores adicionales ni clústeres de procesamiento.
- Debe minimizar el parpadeo en la profundidad entre fotogramas consecutivos, un problema en la estimación de profundidad sin retroalimentación de fotogramas

3.2 Arquitectura del sistema y Flujo de datos del Backend

Esta sección se centrará en describir el funcionamiento a alto nivel del pipeline propuesto, empezando por la arquitectura general de la solución del sistema para posteriormente enfocarse en el flujo de información de los procesos en el Backend que es el que soporta la mayor carga de trabajo

3.2.1 Arquitectura del sistema

La arquitectura del sistema está basada una arquitectura de microservicios que une a un cliente desarrollado en Unity, con un Backend implementado en Python mediante Flask. El microservicio HTTP (expuesto desde el Backend) recibe solicitudes de reconstrucción de una imagen monocular. Cuando Unity realiza una petición, el Backend captura una imagen en directo desde una webcam y la procesa utilizando un modelo de estimación de profundidad (como MiDaS), codificando el mapa de profundidad en el canal Alpha de una imagen RGBA. Unity recibe esta imagen, interpreta el canal Alpha como información de profundidad y genera una nube de puntos en tiempo real para su visualización por parte del usuario, a continuación, la figura 3.1 muestra el diagrama C4 de nivel 2 (Contenedores) del proyecto, que ilustra de manera gráfica este funcionamiento.

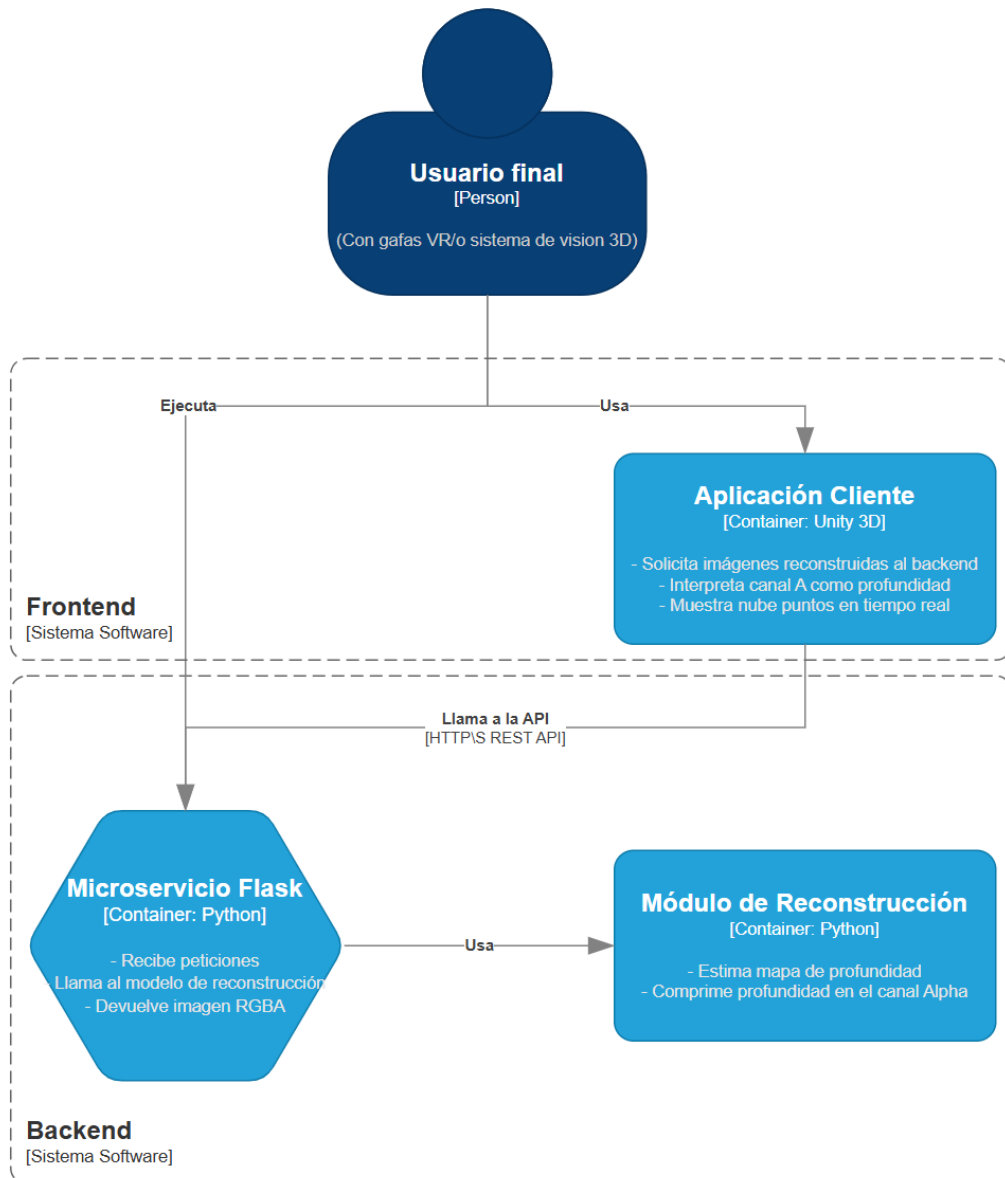


Figura 3.1 Diagrama C4 de nivel 2 (Contenedores) del funcionamiento del sistema con un solo usuario

Dividir el sistema en dos partes diferenciadas, a las que llamaremos Frontend y Backend, supone una serie de ventajas y desafíos que han influido en el diseño arquitectónico adoptado.

Por un lado, la ejecución del Backend como un conjunto de microservicios introduce cierto coste computacional y de latencia, ya que el cliente debe generar una solicitud HTTP, enviarla al servidor, esperar la respuesta y luego procesar el resultado, a pesar de esto el diseño del flujo de datos del servidor, que se verá más en detalle en la sección 3.2.2 saca casi todo el cómputo pesado fuera de Flask, lo que permite que la ejecución de los microservicios sea lo suficientemente ligera como para funcionar en tiempo real.

Además, dado que el sistema está pensado para entornos de comunicación inmersiva, el uso de microservicios ofrece una ventaja estructural clave: permite

interconectar múltiples instancias de Frontend y Backend entre distintos usuarios. De esta forma, basta con establecer conexiones cruzadas entre los microservicios para construir una aplicación de comunicación volumétrica punto a punto (sin audio), como se ilustra en la figura 3.2.

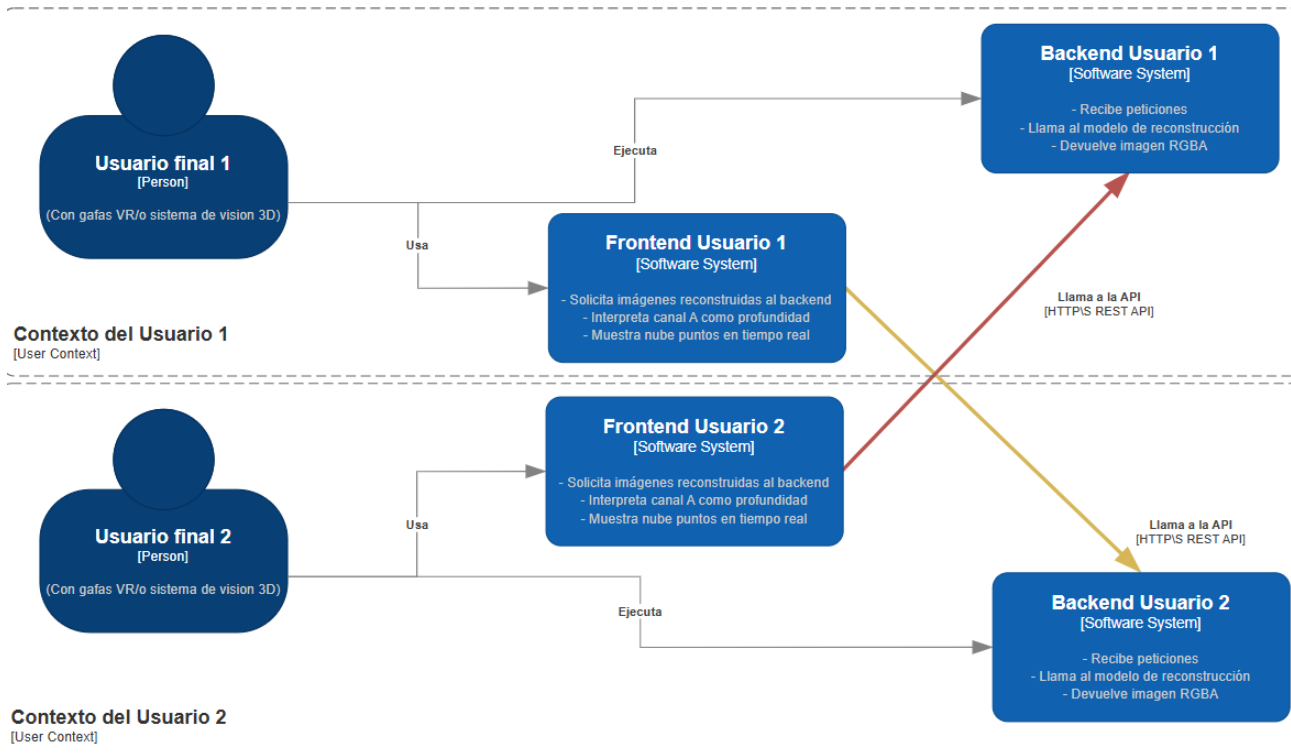


Figura 3.2 Diagrama C4 de nivel 1 (Contexto) del funcionamiento del sistema con dos usuarios interconectados

3.2.2 Flujo de datos de los procesos del Backend

Una vez revisada la arquitectura general del sistema, es relevante echar un vistazo más de cerca al Backend, puesto que es la parte con más complejidad algorítmica y por tanto donde más problemas potenciales (ya sean de funcionamiento o de rendimiento van a aparecer).

Dado que el Backend debe ejecutarse en tiempo real, se ha optado por una estructura de hilos paralelos intercomunicados que realicen las distintas tareas, de esta manera se evitan los “semáforos” y los tiempo de espera de CPU y GPU, además, se ha tenido en cuenta sobre qué recurso hardware (CPU o GPU) se ejecuta cada módulo para evitar cuellos de botella provenientes del uso del bus PCIe para el intercambio de datos entre la RAM (CPU) y la VRAM (GPU) dado que esta es una operación que dentro de los márgenes de trabajo (donde el procesamiento completo no debe de durar más de 66ms) es bastante costosa si se repite varias veces por ciclo de trabajo.

Teniendo en cuenta lo anterior el Backend se divide en cuatro procesos principales:

1. Módulo de captura. Este es el módulo que se encarga de conectarse al servidor RTMP (La entrada de vídeo desde OBS). Abre un flujo de datos, consume las imágenes y las elimina de la pila, este proceso debe ir independiente porque si el tiempo de procesamiento de este bloque supera a la velocidad de llenado del buffer de imágenes, puede producirse un buffer Overflow
2. Módulo de YOLO. Este módulo tiene como tarea identificar al usuario y generar la caja que lo contiene para reducir la carga al módulo principal (módulo de procesamiento)
3. Módulo de procesamiento. Este es el módulo principal, se encarga de generar y procesar la profundidad, recibe la imagen del buffer del módulo anterior y la carga, y la procesa directamente en GPU, ahorrando así el intercambio de datos en CPU. Como salida, guarda en segundo buffer la imagen original (recortada) del módulo de captura, junto con los datos enviados a CPU de la profundidad.
4. Módulo de compresión. Este es el último módulo, y a diferencia de los anteriores, no funciona en bucle sino bajo petición HTTP, obtiene los datos del módulo anterior, los comprime en una imagen PNG y los expone al Frontend.

La figura 3.3 muestra de manera visual la interrelación de estos 4 módulos

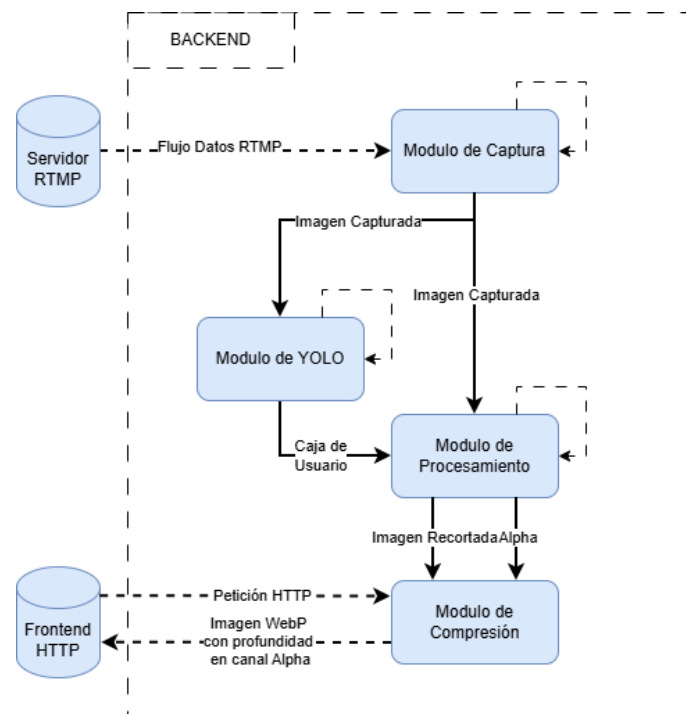


Figura 3.3 Diagrama de Flujo de Datos entre los módulos del Backend

3.3. Tecnologías, Metodologías y Herramientas aplicadas

En esta sección se revisarán las principales tecnologías, metodologías y herramientas aplicadas, justificando su uso y el por qué son claves para el proyecto. Finalmente, en la sección 3.3.7 se verán otras tecnologías que se valoraron para el proyecto pero que fueron descartadas en alguna de las etapas del desarrollo.

3.3.1 Python

La elección de Python como lenguaje del Backend responde a su hegemonía en el ecosistema del Deep Learning y de la visión por computador. Esto permite la integración nativa y cohesionada de las librerías del proyecto, especialmente PyTorch (Para MiDaS) y OpenCV (para la manipulación de imágenes), lo que evita la necesidad de desarrollar sistemas alternativos (y menos optimizados) para estas tareas.

Más allá de la inferencia, Python cuenta con una integración web sencilla a través de Flask. Esto le permite conectar la lógica de IA con una interfaz web ligera que facilita el desarrollo rápido y el mantenimiento de código. A pesar de ser un lenguaje interpretado (y no compilado como C#), su uso en el proyecto, no compromete el rendimiento puesto que delega las operaciones pesadas a las implementaciones de bajo nivel (C++/CUDA) de las librerías utilizadas.



Figura 3.4 Logo de Python. Fuente: www.python.org

3.3.2 YOLO

La selección de la arquitectura YOLO para este proyecto se fundamenta en su capacidad para realizar la detección de objetos en una única fase o etapa, esto le otorga una ventaja crítica en velocidad de ejecución respecto de otros modelos de dos etapas como Faster R-CNN. Esta característica resulta clave a la hora de cumplir los requisitos de latencia del sistema en tiempo real propuesto.

Además, en este trabajo la función de YOLO no se limita simplemente a la detección de elementos, sino que funciona como un filtro de atención selectiva que permite acotar la Región de interés (ROI) del usuario y descartar el fondo. Esto permite optimizar el ancho de banda y el cómputo del Backend. Además, aprovechando que en el entorno de las videollamadas no se suelen producir movimientos bruscos, se ha diseñado una estrategia de ejecución intermitente donde YOLO se ejecuta solo cada cierto intervalo de tiempo, esta estrategia, permite diluir en gran medida su coste computacional sin llegar a comprometer el seguimiento del sujeto.

3.3.3 Intel MiDaS

La adopción de MiDaS, desarrollado por Intel y el grupo Intel-ISL supone el núcleo del módulo de reconstrucción gracias a su capacidad de generalización a múltiples situaciones y entornos. Esto es crítico para la viabilidad del sistema, ya que permite inferir mapas de profundidad en entornos domésticos no controlados, donde la iluminación o la disposición de elementos son completamente impredecibles.

A pesar de que actualmente, el estado del arte este evolucionando de los modelos CNN a los modelos Transformers (Como MiDaS v3 o DPT) en este proyecto se ha priorizado el uso de la versión 2 de MiDaS basada en modelos convolucionales debido a su mayor rendimiento base (a costa de un peor resultado). Esta decisión se fundamenta en el compromiso técnico entre precisión y latencia, se ha priorizado la velocidad sobre la precisión de inferencia para poder alcanzar los objetivos establecidos de tiempo real en hardware de consumo. Aun así, se deja abierta la integración de modelos Transformers en futuras iteraciones del proyecto.

3.3.4 Flask

En el proyecto se ha optado por Flask como interfaz de comunicación entre el Frontend y el Backend debido a su mínima huella en memoria y su bajo consumo de CPU. Esta característica es vital en entornos de tiempo real donde el servidor debe convivir con otros procesos de alta demanda computacional (como YOLO o MiDaS) sin introducir latencia adicional a la hora de transmitir de manera asíncrona las texturas RGBA hacia el motor gráfico (Unity)



Figura 3.5 Logo de Flask. Fuente: flask.palletsproject.com

3.3.5 Unity

La elección de Unity como Frontend del sistema se basa en su capacidad para implementar un renderizado programable eficiente. A diferencia de un visor pasivo, en la solución, Unity actúa como un procesador visual de alto rendimiento capaz de gestionar la carga de la geometría dinámica en tiempo real, permitiendo ejecutar el Frontend del proyecto en distintas familias de dispositivos como son Ordenadores, teléfonos móviles, o dispositivos de VR. Además, su integración con las peticiones HTTP, permite el consumo de las texturas del Backend.

Sumado a lo anterior, Unity permite ejecutar lógica personalizada directamente en la tarjeta gráfica (GPU). A través del desarrollo de VFX Graphs personalizados, Unity interpreta el canal Alpha de la textura como información de profundidad y desplaza las partículas directamente desde la tarjeta gráfica. Este enfoque evita el cuello de botella de calcular las posiciones en la CPU para luego mandársela a la GPU, permitiendo un rendimiento de hasta diez veces superior garantizando la fluidez suficiente para una experiencia en tiempo real inmersiva.



Figura 3.6 Logo de Unity. Fuente: Unity

3.3.6 Docker

Finalmente, la inclusión de Docker responde a la necesidad de reproducibilidad del Backend en la fase de despliegue. Dado que el sistema depende de una combinación estricta ya no solo de librerías (PyTorch, OpenCV), sino de versiones específicas de estas, el uso de contenedores permite encapsular todas las dependencias en una imagen inmutable. Esto permite desplegar el Backend del equipo anfitrión de manera inmutable a cualquier otro dispositivo sin caer en problemas de dependencias.

A diferencia de las máquinas virtuales tradicionales, que requieren de emular un sistema operativo completo mediante un hipervisor, Docker comparte el kernel del sistema anfitrión. Esta arquitectura ligera minimiza la sobrecarga de la CPU y la Memoria, lo que es decisivo para las aplicaciones que funcionen en tiempo real. Además, esta tecnología facilita el acceso directo al hardware, permitiendo que el contenedor utilice aceleración de GPU nativa para la inferencia de redes neuronales sin las penalizaciones clásicas de la virtualización.

3.3.7 Evaluación tecnológica y “Tech Radar” del proyecto

Con el fin de respaldar la selección de las herramientas descritas en las secciones anteriores, se elaboró un “Tech Radar” (una técnica que sintetiza el grado de madurez y adecuación de las tecnologías existentes para un proyecto dado) en el ámbito de la reconstrucción 3D en tiempo real.

El concepto de Tech Radar se basa en el modelo propuesto por ThoughtWorks (Thoughtworks, Inc, 2024) ampliamente utilizado para la toma de decisiones tecnológicas estratégicas.

Este radar se estructura en cuatro cuadrantes correspondientes a las principales áreas implicadas en el desarrollo del proyecto: Visión por Computador,

Visualización 3D, Infraestructura de Comunicación y Optimización. Asimismo, se organiza en cuatro niveles de madurez y adopción tecnológica: Adoptar, Probar, Evaluar y Descartar.

Cada uno de estos niveles refleja el grado de análisis y experimentación alcanzado por las distintas tecnologías consideradas:

Adoptar: incluye aquellas tecnologías que han sido efectivamente integradas en el proyecto, demostrando un rendimiento adecuado, estabilidad y una elevada correspondencia con los objetivos planteados.

Probar: agrupa las tecnologías que, pese a resultar inicialmente prometedoras, fueron descartadas durante la fase de experimentación debido a limitaciones en aspectos como la escalabilidad, la adecuación funcional o el rendimiento.

Evaluar: comprende las tecnologías que fueron objeto de un análisis más exhaustivo, pero que finalmente se descartaron en favor de alternativas con mayor grado de madurez o mejor alineación con los requerimientos del sistema.

Descartar: recoge aquellas tecnologías que, tras una revisión preliminar, no se consideraron viables o pertinentes para los propósitos del proyecto.

La principal ventaja de este sistema es que permite visualizar de manera clara qué tecnologías se han utilizado, cuáles no, y en qué etapa fueron descartadas. La figura 3.7 muestra el Tech Radar aplicado en este proyecto.

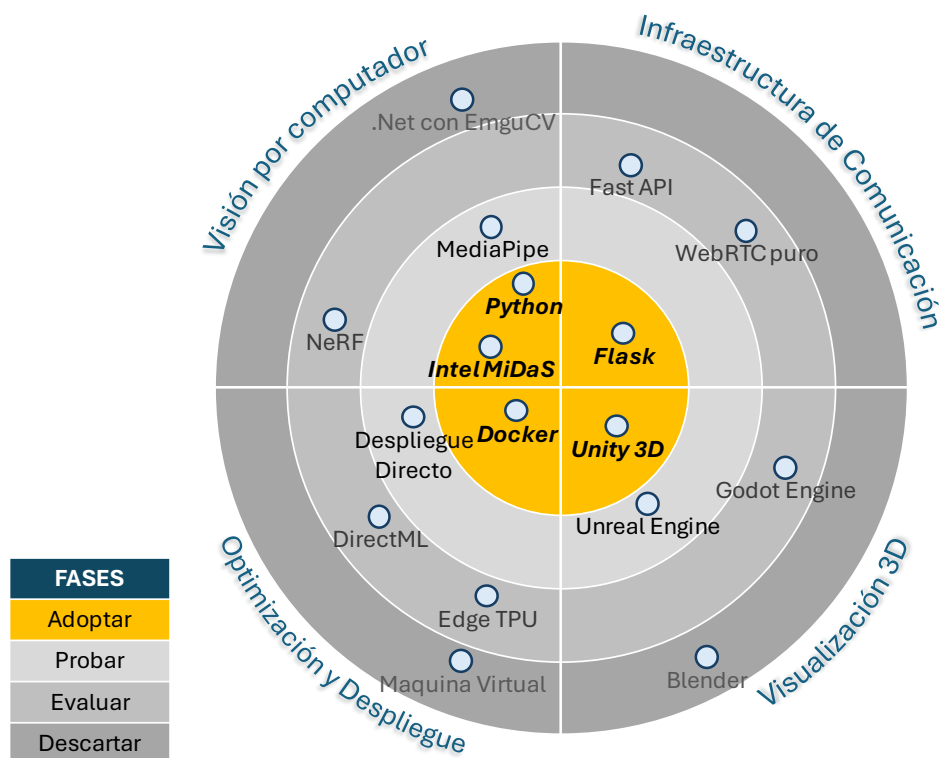


Figura 3.7 Diagrama "Tech Radar" del proyecto

En conjunto con lo anterior, las siguientes tablas muestran de manera resumida la razón de descarte de las tecnologías analizadas, pero no utilizadas.

Visión por computador	Fase	Motivo de Descarte
.Net con EmguCV	Descarte	EmguCV es una modificación sobre OpenCV para .Net, es un ecosistema más cerrado y menos flexible y adaptable que trabajar directamente en Python, además, la versión para Unity cuenta con importantes limitaciones
NeRF	Evaluación	Tiempo de inferencia y latencia demasiado altos para un sistema en tiempo real. Incluso las implementaciones optimizadas (Müller y otros, 2022) requieren tiempos de inferencia del orden de cientos de ms por vista.
MediaPipe	Prueba	Si bien MediaPipe ofrece alta eficiencia en detección de poses y segmentación, no dispone de componentes orientados a reconstrucción 3D densa, por lo que no resultó adecuado para este propósito.

Tabla 3.1 Tecnologías descartadas del cuadrante de visión por computador

Visualización 3D	Fase	Motivo de Descarte
------------------	------	--------------------

Unreal Engine	Prueba	Motor de gran potencia y calidad visual, pero con unos requisitos de hardware superiores a los de Unity, lo que lo hace menos adecuado para el alcance y las necesidades de este proyecto.
Godot Engine	Evaluación	A pesar de que su carácter de software libre constituye una ventaja significativa, en la fecha de desarrollo del proyecto Godot no dispone de un soporte multiplataforma plenamente maduro y presenta limitaciones en sus herramientas de visualización 3D avanzada.
Blender	Descarte	Permite la ampliación de funcionalidades mediante scripts en Python, pero no está concebido para ejecución en tiempo real ni para la compilación de aplicaciones ejecutables, por lo que resulta inadecuado para su distribución o uso generalizado.

Tabla 3.2 Tecnologías descartadas del cuadrante de visualización 3D

Infraestructura de Comunicación	Fase	Motivo de Descarte
FastAPI	Evaluación	Requiere de una mayor curva de aprendizaje que Flask, es más útil en proyectos que cuenten con una alta concurrencia.
WebRTC puro	Evaluación	Permite la transmisión de vídeo a través de un flujo en tiempo real, pero genera cierta complejidad a la hora de integrar la transmisión de vídeo con la sincronización de datos de reconstrucción 3D.

Tabla 3.3 Tecnologías descartadas del cuadrante de infraestructura de comunicación

Optimización y Despliegue	Fase	Motivo de Descarte
Edge TPU	Evaluación	Edge TPU permite ejecutar modelos de IA a través de accesorios que dotan a los sistemas de una TPU, este enfoque, a futuro, puede ser muy interesante, puesto que las TPU son más eficientes que las tarjetas gráficas para realizar tareas de IA, pero ahora mismo es aún una tecnología emergente y podría no ser prudente usarla en su estado actual, aunque no se descarta su uso en iteraciones posteriores
DirectML	Evaluación	Es una API de bajo nivel basada en DirectX 12 para el aprendizaje automático, la API parece prometedora pero aún está dando sus primeros pasos y su ecosistema es reducido en comparación con CUDA u otras alternativas, además, no cuenta con una comunidad tan amplia ni un sistema tan

		maduro como Python, aun así, podría tenerse en cuenta para trabajo futuro.
Máquina Virtual	Descarte	Permite agrupar aplicaciones junto con sus dependencias, encapsuladas dentro de un sistema operativo virtual, pero es menos eficiente que Docker.
Despliegue Directo	Prueba	Tiene algo más de rendimiento que depender de Docker, pero es dependiente del entorno y del sistema operativo del equipo anfitrión, lo que dificulta la portabilidad y replicabilidad.

Tabla 3.4 Tecnologías descartadas del cuadrante de Optimización

En su conjunto, el “Tech Radar” permitió estructurar la evaluación tecnológica y justificar las decisiones adoptadas, asegurando de tal manera un equilibrio entre madurez, rendimiento y compatibilidad con los objetivos del proyecto.

3.4 Diseño y configuración de los Modelos Neuronales

La arquitectura de inteligencia artificial del sistema se basa en el uso de modelos ligeros preentrenados optimizados para inferencia rápida, junto con arquitecturas personalizadas para las tareas de refinamiento. A continuación, se detalla la configuración técnica de los tres componentes principales: El detector de usuarios, el modelo de estimación de profundidad y la red de superresolución.

3.4.1 Configuración del Detector (YOLO)

Para la etapa de atención selectiva y segmentación del usuario se ha hecho uso del modelo YOLOv8 en su variante Nano (yolov8n.pt). Esta elección responde a la necesidad de reducir la huella de memoria y maximizar la velocidad de inferencia en paralelo al proceso principal de profundidad.

Esta versión (YOLOv8) representa una evolución respecto a sus predecesores al usar una arquitectura anchor-free (es decir, sin anclajes predefinidos), lo que reduce la cantidad de predicciones de cajas y acelera el proceso de inferencia.

Desde el punto de vista de la implementación del sistema, se ha hecho uso de la librería ultralytics bajo un esquema de ejecución asíncrona de hilos. Los parámetros de configuración que se han utilizado han sido los siguientes:

1. Precisión. Se ha usado FP16 media precisión para reducir el consumo de VRAM a la mitad y acelerar la inferencia en GPUs compatibles con núcleos Tensor, sin una pérdida significativa de precisión.
2. Filtro de Clases. Se ha configurado para filtrar exclusivamente la clase 0 (Personas) descartando todo lo demás para así evitar interferencias en el cálculo de la profundidad

3. Intervalo de Ejecución. A diferencia de la estimación de profundidad que ocurre fotograma a fotograma, la detección de YOLO se ejecuta solo cada medio segundo, esta estrategia asume que el movimiento del usuario, dado el contexto del sistema, será bajo. Esto permite liberar recursos de la GPU para el modelo de profundidad (MiDaS)

3.4.2 Modelo de Estimación de Profundidad (MiDaS)

MiDaS es sin duda el núcleo de la reconstrucción volumétrica del sistema, que en este caso se trata de su variante Small (v2.1). Esta versión ha sido seleccionada explícitamente por ofrecer el mejor equilibrio entre velocidad y precisión para aplicaciones en tiempo real, sacrificando detalles finos, que posteriormente serán recuperados en la etapa de superresolución.

A diferencia de una implementación estándar, el modelo ha sido sometido a un proceso de optimización para garantizar su ejecución dentro de los márgenes de tiempo real (<50ms por fotograma) que los requisitos del proyecto precisan, para ello se han hecho uso de tres estrategias diferenciadas:

En primer lugar se ha integrado la compilación del modelo mediante `torch.compile`, lo que permite una compilación Just-In-Time (JIT) que permite fusionar operaciones de capas consecutivas y optimizar la ejecución del modelo a la GPU objetivo, lo que reduce la sobrecarga del intérprete de Python, a costa de un periodo inicial de compilación (en las pruebas de entre 60-120 segundos) al arrancar la ejecución.

Por otro lado, se ha forzado al modelo a operar en precisión media (FP16), lo que además de reducir el consumo de memoria VRAM, permite aprovechar los Tensor Cores de las tarjetas gráficas modernas para acelerar las operaciones matriciales sin degradar la calidad percibida del mapa de profundidad.

Finalmente, uno de los cuellos de botella identificados durante el desarrollo fue la normalización de imágenes, por lo que en la implementación final se hizo uso de un pipeline híbrido de preprocesamiento, en el que por un lado se hizo uso de OpenCV para ajustar la imagen de entrada al tamaño nativo de la red (256x256) y por el otro, a diferencia de las implementaciones normales que normalizan la imagen en CPU, se optó por la carga completa de la imagen en GPU para posteriormente aplicar la normalización estándar de ImageNet directamente sobre los tensores en VRAM, eliminando así, en concordancia con la filosofía general del proyecto, ciclos innecesarios de transferencia de datos entre la RAM y la VRAM del sistema.

3.4.3 Arquitectura de Superresolución (FSRCNN)

Para abordar la limitación de los mapas de profundidad generados por MiDaS (limitados a 256x256), y tras descartar arquitecturas generativas complejas como

ESRGAN debido a su alta latencia ($>100\text{ms}$), se ha implementado una red FSRCNN (Fast Super-Resolution Convolutional Neural Network) personalizada.

Esta red ha sido diseñada específicamente para operar directamente sobre el tensor de profundidad normalizado de un solo canal, con un factor de escala de $\times 3$, elevando la resolución efectiva y restaurando los bordes geométricos de la imagen.

A diferencia de la implementación original de FSRCNN (Dong y otros, 2016), diseñada para imágenes RGB, para este proyecto se ha adaptado su uso para procesar tensores de un único canal, lo que reduce la carga computacional de la primera y la última capa (al tener que procesar solo un tercio de los datos).

Uno de los desafíos técnicos identificados durante el desarrollo fue la variabilidad en el rango de valores de profundidad generados por MiDaS. Esto generaba un problema con FSRCNN que requería una entrada normalizada para funcionar correctamente. Es por ello que se realizó un proceso de normalización de los datos de manera dinámica al rango $[0,1]$, acotando el resultado de manera relativa entre sus valores máximos y mínimos. Una vez obtenidos los resultados del reescalado estos datos se devolvían a su rango inicial realizando un proceso de desnormalización inverso al anteriormente mencionado.

Finalmente, para poder cumplir con los requisitos de procesamiento en tiempo real, al igual que con MiDaS, se ajustó el modelo para funcionar en precisión media (FP16), reduciendo el consumo de memoria VRAM y permitiendo el uso de Tensor Cores para una inferencia más rápida. Junto a lo anterior también se usó la misma estrategia de Compilación JIT que con MiDaS obteniendo las mismas ventajas que con este.

Esta configuración permitió que el módulo de superresolución tenga un tiempo de ejecución despreciable ($< 3\text{ms}$) respecto del resto del sistema, lo que valida su viabilidad para el pipeline propuesto.

3.4.3.1 Proceso de Entrenamiento y Fine-Tuning

A diferencia de los objetivos clásicos de la superresolución, donde el objetivo se limita a aumentar la densidad de puntos de una imagen, en este proyecto se ha buscado sacar provecho de la capacidad de alucinar de las redes neuronales para no solo obtener un resultado de profundidad de mayor resolución, sino de, además, dotar a la imagen resultante de mayor detalle fino, al mismo tiempo que se corrigen artefactos estructurales propios de la estimación monocular como las deformaciones geométricas o suavizados excesivos.

Es por ello, que el proceso de entrenamiento de FSRCNN no se ha limitado a un aprendizaje supervisado estándar, sino que se ha diseñado una metodología en tres fases para adaptar la red al dominio de trabajo, haciendo para ello uso, por un lado,

de un conjunto de datos híbridos (mapas de profundidades reales y sintéticos de personas) junto con una función de pérdida compuesta (Puede revisarse su código específico en el Anexo I), con distintos pesos para cada Fase.

3.4.3.1.1 Generación del Dataset

Empezando por la construcción del dataset, es relevante que la red aprenda la relación entre la estimación y la realidad física, es por ello que, en vez de entrenar la red simplemente reduciendo la resolución de imágenes de alta calidad, se hizo uso de un generador de datos personalizado que replica el pipeline real del sistema.

Para ello, el generador empieza trabajando con una imagen RGB real o sintética junto con su correspondiente mapa de profundidad, en el caso de las imágenes sensores, la imagen de profundidad se ha obtenido mediante un sensor Kinect V2 y en el caso de las sintéticas a partir del buffer de profundidad de Blender. Una vez que se tienen estos datos, el generador crea los datos a baja resolución usando MiDaS sobre la imagen de alta resolución RGB para dar como resultado una imagen 256x256 con imperfecciones. Una vez obtenidas todos los datos, dado que el dataset era bastante limitado (86 pares RGB-Depth Sintéticos y 196 pares reales, véase anexo IV para imágenes de referencia), se procedió a dividir los datos en parches de 32x32 píxeles (con un paso de 10 píxeles) para baja resolución y 96x96 para alta resolución. Para optimizar la cantidad de imágenes “útiles” que se usaban para entrenar la red, se aplicaron filtros de varianza (para descartar imágenes planas) o de valor 0 en las imágenes reales (para eliminar las zonas donde kinect diera error de profundidad).

3.4.3.1.2 Función de pérdida

Para la función de pérdida, se hizo uso de una función compuesta ya que las funciones estándar como el error cuadrático medio (MSE) tienden a generar resultados borrosos al promediar errores, lo que es contrario al objetivo de recuperar detalle fino. Es por ello que como función de pérdida se hizo uso de la siguiente configuración:

$$L_{total} = L_{pixel} + W_{bordes} \cdot L_{bordes} + W_{detalle} \cdot L_{detalle} \quad (1)$$

Donde:

- L_{pixel} es la función L1 estándar para garantizar la precisión global en la profundidad
- L_{bordes} aplica filtros de Sobel en los ejes X e Y para penalizar los errores en los contornos de la silueta

- $L_{detalle}$ aplica un operador Laplaciano que permite maximizar la detección de cambios de alta frecuencia, lo que es crucial para detalles internos del rostro y de la ropa
- W_{bordes} y $W_{detalle}$ son los pesos que se asignan en cada etapa a los filtros de bordes y detalle

3.4.3.1.3 Estrategia en tres fases

Para lograr una convergencia estable y evitar el sobreajuste a los datos sintéticos, el entrenamiento se estructuró en tres etapas secuenciales:

Etapas 1. Aprendizaje Geométrico

En esta etapa, la red se entrena haciendo uso exclusivo de los datos sintéticos de Blender, el objetivo de esta etapa radica en que la red aprenda la topología ideal del cuerpo humano sin tener en cuenta el ruido causado por el uso de sensores reales.

En esta etapa se hace uso de 60 épocas con una tasa de aprendizaje de 0.001. W_{bordes} y $W_{detalle}$ se ajustaron a 2.0 y 0.5 respectivamente, dando más prioridad a la generación de bordes definidos sobre la textura.

Etapas 2. Adaptación a un entorno Real

En esta segunda etapa, la red se entrena haciendo uso exclusivo de los datos reales capturados con Kinect V2. El objetivo de esta etapa radica en que la red aprenda a trabajar con las texturas orgánicas, pero sin olvidar lo aprendido en la fase anterior, es por ello, que, en esta etapa, se reduce la tasa de aprendizaje a 0.0001 al mismo tiempo que se reduce el número de épocas a 30. W_{bordes} y $W_{detalle}$ se ajustaron a 0.1 y 0.1 respectivamente, lo que reduce el peso de los bordes del paso anterior.

Etapas 3. Refinamiento

En esta última etapa se busca consolidar el conocimiento de las dos etapas anteriores, mitigando el olvido de la fase uno. Es por ello que en esta última fase se hace uso de la concatenación de ambos datasets, se mantienen las 30 épocas y se vuelve a reducir la tasa de aprendizaje a 0.00001. Finalmente, W_{bordes} y $W_{detalle}$ se ajustaron a 0.8 y 0.05 respectivamente.

3.5 Desarrollo

Esta sección cubrirá los aspectos más destacados del desarrollo del proyecto, como es la estructura del proyecto, la descripción y diseño detallado de los componentes claves que conforman el sistema, la justificación de las principales decisiones de diseño tomadas, así como las pruebas unitarias y la validación de componentes y sistemas durante el desarrollo del proyecto.

3.5.1 Diseño detallado de los componentes clave del sistema

- **Algoritmo Principal:** el Backend recibe la imagen RGB obtenida del servidor RTMP, la procesa mediante MiDaS, filtra el resultado, lo reescala y lo devuelve en forma de textura RGBA donde la profundidad se almacena en el canal Alpha.
- **Interfaz cliente-servidor:** el Frontend envía solicitudes HTTP al servidor Flask del Backend, Flask devuelve la imagen procesada por el algoritmo principal que posteriormente Unity convierte en una nube de puntos 3D.
- **Modelo de datos:** Se optó por usar el formato PNG con transparencia, ya que permite transmitir en una sola imagen toda la información requerida por el Frontend, en instancias iniciales, el rendimiento de compresión en PNG era un cuello de botella bastante importante puesto que tenía un tiempo de compresión de casi 100ms, por lo que se valoró el uso de otros formatos como WebP o BMP. Finalmente se hizo uso de PNG con la mínima compresión reduciendo el tiempo de procesamiento de 100ms a 15-30ms.

3.5.2 Estrategias de Implementación

- **Desacoplamiento de Inferencia y Renderizado:** La solución implementa una estricta separación de responsabilidad dividiendo la carga de trabajo en dos subsistemas independientes comunicados vía red, que, a su vez se dividen en pequeños hilos de trabajo independientes especializados cada uno en una tarea (captura de imágenes, reconstrucción, atención selectiva). El procesamiento de imágenes y la estimación de profundidad se ejecutan en un entorno de servidor con acceso a librerías de IA y aceleración por CUDA. Al mismo tiempo, la representación final se delega a un motor gráfico (Unity) capaz de compilar para diversos entornos de hardware, como PC, teléfonos móviles o dispositivos VR. Esta arquitectura permite que el cliente de visualización sea agnóstico a los requisitos de funcionamiento del servidor, lo que facilita su integración en dispositivos ligeros que solo consuman datos procesado remotamente.
- **Estimación Monocular en tiempo real:** El núcleo de la reconstrucción se implementa mediante la ejecución en tiempo real de la arquitectura neuronal Intel MiDaS, haciendo uso de su variante optimizada “Small” para garantizar tasa de inferencias altas (entre 40-60FPS) en equipos con GPUs de gama media. El sistema encapsula este modelo dentro de un pipeline de procesamiento estrictamente basado en software, eliminando la dependencia de sensores activos. Para mitigar los artefactos y la falta de detalles finos, la salida de la red no se procesa directamente, sino que se procesa posteriormente por varios filtros (reescalado, suavizado temporal, etc.) que mejoran los resultados obtenidos.

- **Atención Selectiva mediante Región de Interés Dinámica:** El módulo de detección opera de manera asíncrona, ejecutando YOLO cada medio segundo para actualizar las coordenadas de la caja delimitadora del usuario. Esta región de interés se extrae del buffer de la imagen original antes de ser enviada al modelo de profundidad, descartando automáticamente la información del entorno. Esto permite que la red neuronal dedique el total de la resolución de entrada (256x256) exclusivamente al usuario, maximizando así la cantidad de píxeles útiles que procesa y reduciendo el coste computacional de procesar zonas que igualmente se eliminarían posteriormente (fondo).
- **Reescalado Neuronal e Inferencia de detalle fino:** El mapa de profundidad base obtenido mediante MiDaS cuenta con una resolución de 256x256, bastante alejada de la resolución de la imagen original. Es por ello que antes de insertar esta imagen en el canal Alpha debe reescalarsse, para ello la imagen es procesada mediante un módulo de superresolución basado en FSRCNN, que, a diferencia de los algoritmos de interpolación clásicos, como el bicúbico o “el vecino más cercano”, puede generar información de alta frecuencia ausente en la fuente original. El modelo ejecuta una inferencia que reconstruye bordes difusos y detalle finos a partir de los patrones aprendidos durante el entrenamiento (fine-tuning), transformando el tensor de baja resolución en un mapa de profundidad de alta calidad y bordes más definidos antes de su transmisión.
- **Trasmisión de imágenes con profundidad en el canal Alpha:** Una vez obtenidos los 4 canales de información (Rojo, Verde, Azul y Profundidad) el sistema debe prepararlos para poder ser transmitidos al Frontend, para ello, se comprime la imagen en PNG, que permite obtener un resultado casi sin pérdidas, y con un tiempo de compresión aceptable al mismo tiempo que ocupa menos espacio que los datos en bruto. Esto permite exponer una imagen RGBA como único vector de salida de información por parte del servidor FLASK
- **Renderizado Volumétrico mediante Shaders en GPU:** La visualización de los datos en Unity se implementa a través de un sistema de partículas procedimental gestionado íntegramente mediante un “Visual Effect Graph”. En vez de generar una malla poligonal, que generalmente implica actualizar la topología en CPU y gestionar varias mallas simultáneamente, debido al límite de 65.535 vértices por malla de Unity, el sistema instancia una matriz de partículas estáticas. Estas partículas son modificadas en tiempo real directamente por la GPU según el sistema va recibiendo nuevas imágenes. Para evitar un enfoque de generación constante de partículas se optó por modificar el tamaño de las mismas, por lo que, si una partícula debiera desaparecer, esta simplemente sería de tamaño 0 y sería ignorada por la

GPU a la hora de dibujarla. Cuando la partícula vuelva a ser necesaria, se le vuelve a dotar de su tamaño original para que vuelva a aparecer. De esta manera la lógica del posicionamiento se paraleliza íntegramente en la GPU, eliminando el cuello de botella del traspaso de datos entre CPU y GPU permitiendo un renderizado más fluido de las nubes de puntos de alta densidad.

3.5.3 Desafíos Técnicos y Soluciones Aportadas

A lo largo de la fase de desarrollo del proyecto y, como es habitual en este tipo de proyectos, se encontraron una serie de contratiempos que estando o no contemplados en el diseño inicial fueron responsables de realizar cambios sustanciales en la solución aportada, estos contratiempos se pueden agrupar en tres categorías o ramas diferentes:

3.5.3.1 Consistencia Visual

Esta categoría abarca los desafíos surgidos de la discrepancia entre la geometría reconstruida (la profundidad estimada) y la estructura física real de la escena (la profundidad real). La causa última de este problema reside en la propia naturaleza de la estimación monocular. Esta técnica trabaja con inferencia probabilística (a través de redes neuronales) y no con mediciones directas (como con un LiDAR, esto causa dos principales problemas, por un lado la falta de referencias absolutas de distancias, esto es, es difícil saber la distancia exacta de un objeto a la cámara, esto a su vez nos lleva al segundo problema, MiDaS al igual que otros modelos similares analizan cada fotograma de manera independiente por lo que pequeños cambios en la iluminación o en las referencias de la escena pueden generar diferencias significativas con la profundidad real entre fotogramas. A este fenómeno se le conoce como flickering o parpadeo geométrico, y genera una rotura en la ilusión de la solidez de la escena, al mismo tiempo que puede generar una experiencia visual poco confortable.

Para mitigar estas inestabilidades y por ende tratar de garantizar una reconstrucción coherente, se implementaron las consiguientes correcciones:

En primer lugar, se aplicó un Suavizado Temporal, esto es, un sistema que “recuerda” las imágenes anteriores y las tiene en cuenta para suavizar la imagen actual. La fórmula usada para el suavizado temporal es la que sigue:

$$dst(x,y) = (1 - weight) * dst(x,y) + weight * src(x,y) \quad (2)$$

Donde:

- $dst(x,y)$: representa el historial o la media acumulado de ese píxel
- src : representa la imagen nueva (y por ende $src(x,y)$ representa el píxel en la posición (x,y) de src)

- *weight*: el peso que se le da a la nueva imagen respecto del histórico

A la hora de aplicar una media móvil ponderada, existen dos maneras principales, lineales, es decir, se almacenan los últimos N fotogramas y se calcula su valor medio, es problema de este método es su coste de almacenaje y procesamiento puesto que hay que guardar todos los N fotogramas en memoria y procesarlos cada vez que se ejecuta la función, es por ello que para este trabajo se ha optado por una media ponderada exponencial, que tal y como se puede ver en la fórmula anterior solo requiere de almacenar un único array bidimensional (habitualmente de mayor precisión que la imagen) como histórico, suponiendo así un ahorro de costes de almacenamiento en memoria y procesamiento. Además, este método posee otra ventaja adicional, que radica en que el peso de los fotogramas anteriores, se va “diluyendo” a lo largo de tiempo, lo que permite tener una mayor respuesta en movimientos rápidos que en una media ponderada lineal donde todos los fotogramas tienen el mismo peso.

Junto al suavizado temporal, se implementó un filtrado espacial (Un filtro Bilateral para ser concretos) para corregir el “ruido” en las superficies reconstruidas. Este filtro se aplica en la etapa de preprocesamiento, lo que asegura una mayor homogeneidad en los datos de entrada del modelo de estimación de profundidad, que a diferencia de un desenfoque convencional (como un desenfoque gaussiano) es capaz de preservar la nitidez de los bordes, lo que resulta primordial para mantener constante la figura del usuario, así como su separación con el fondo.

3.5.3.2 Nivel de Detalle

Si bien el modelo MiDaS (Monocular Depth Estimation) ofrece un desempeño notable en la reconstrucción de la profundidad en tiempo real, se identifica una limitación crucial en el nivel de detalle del mapa de profundidad resultante. La baja resolución y la inherente naturaleza de la inferencia monocular resultan en mapas que a menudo no capturan la geometría fina de los objetos, como se evidencia en

la dificultad para diferenciar la profundidad de estructuras faciales detalladas (nariz, ojos, etc.). La Figura 3.8 muestra esta disparidad de manera palpable.

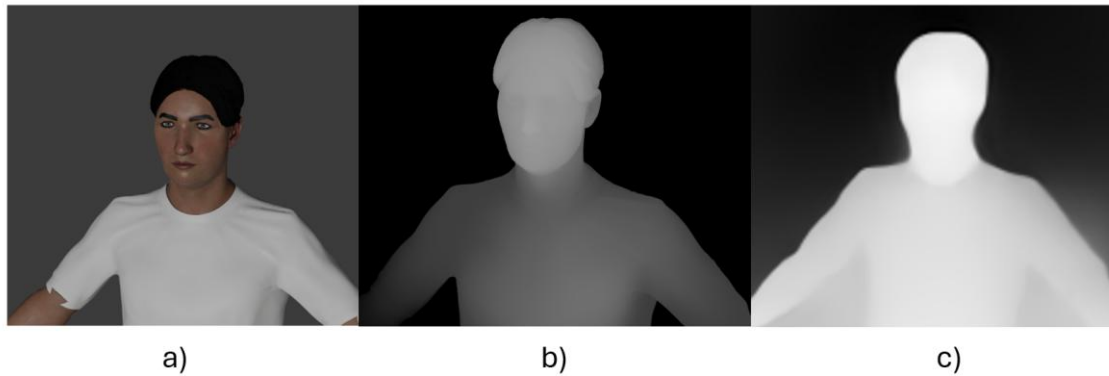


Figura 3.8 a) Imagen de referencia b) Mapa de profundidad real del modelo c) Mapa de profundidad generado por MiDaS

La notable diferencia entre (b) y (c) se atribuye a dos factores principales. En primer lugar, la limitación de resolución, ya que MiDaS estandariza la entrada a 256x256 píxeles, lo que implica una pérdida de información de alta frecuencia. En segundo lugar, el proceso de interpretación de la profundidad por la red neuronal a partir de una única imagen 2D.

Para solventar esta limitación se plantea un doble objetivo, en primer lugar, el aumento de resolución (superresolución), es decir, escalar el mapa de profundidad de baja resolución (c) a una resolución superior, y, en segundo lugar, el incremento del nivel de detalle (detalle fino), es decir, inyectar la geometría de alta frecuencia ausente para asemejar el resultado de (c) al mapa de profundidad real de (b).

Partiendo de la base de que se dispone del mapa de profundidad real y preciso (b) para cada imagen original, el problema se reformula como una tarea de Superresolución Condicionada a la Profundidad y Transferencia de detalle, esto es, emplear una red neuronal generativa de superresolución que no solo escale la imagen, sino que, además, tenga la capacidad de recuperar o de inferir el detalle fino de un mapa de profundidad real. Inicialmente se optó por el uso de ESRGAN modificado, pero resultando este proceso muy lento >100ms, se optó por utilizar una arquitectura basada en FSRCNN, mucho más veloz que ESRGAN (2-3ms)

Para optimizar esta transferencia de detalle, la red FSRCNN se ha sometido a un proceso de Fine-Tuning, este ajuste se realiza entrenando la arquitectura con los pares de datos específicos, el mapa de profundidad de baja resolución generado por MiDaS (como entrada) y el mapa de profundidad de alta resolución (como resultado a obtener). De esta manera, el modelo aprende a corregir los errores de estimación de MiDaS y a reconstruir activamente las características geométricas detalladas, actuando como un reescalador y refinador de profundidad personalizado.

3.5.3.3 Profundidad Relativa

Este es un desafío propio de la estimación monocular, dado que los mapas de profundidad generados por modelos como MiDaS no trabajan con unidades absolutas, sino con valores normalizados entre 0 y 1, que representan la profundidad mínima y máxima de la escena en cada fotograma.

Esto implica que la escala de profundidad es dinámica y dependiente del contenido: si un objeto, como la mano del usuario, se acerca a la cámara, se convierte en el nuevo punto “1”, y, como consecuencia, el rango dinámico de la escena se comprime y se produce un desplazamiento del resto de elementos, generando un efecto de movimiento hacia atrás y aplastamiento de por ejemplo el cuerpo.

Para evitar esto, se evaluaron dos estrategias de anclaje, en primer lugar, el anclaje físico (colocar un objeto en primer plano, que no debe ser traspasado por la persona, para que junto al fondo creen una escala de profundidad fija) y en segundo el anclaje virtual (identificar un elemento en la imagen y utilizarlo como pivote para calcular la profundidad relativa).

Dado que requerir la colocación de objetos de calibración afecta negativamente a la usabilidad y experiencia de usuario, para la solución se ha optado por el uso de la segunda opción, la normalización anclada al sujeto con un factor de escala constante.

Esta estrategia consiste, en primer lugar y aprovechando la integración de YOLO descrita en la sección 3.5.2, el sistema identifica la caja delimitadora (o bounding box) de la persona cada cierta cantidad de tiempo (0.5 segundos en las pruebas). A partir de esa información se elimina el fondo y se toma una pequeña región de por ejemplo 10x10 en el centro de la caja delimitadora (que presumiblemente coincidirá con parte del torso del usuario). De esta región se extraerá la mediana, cuyo valor de profundidad será usado como “Ancla”. Para asegurar una cierta estabilidad temporal, y que el ancla no sufra cambios bruscos en movimientos rápidos, esta mediana se pasará por un filtro de media móvil como el planteado en la sección 3.5.3.1. Una vez estabilizado el valor, se mapeará el valor de esta profundidad a un valor de 0.5 o 128 en el canal Alpha. El resto de valores se mapearán de acuerdo a un factor de escala constante, si los objetos se salen de la escala se ajustan a 0 o a 255 según corresponda. De esta manera la diferencia de profundidad entre el torso y la cabeza se mantienen estables, aunque se pase una mano del usuario por encima. Matemáticamente, esta operación se define como sigue:

$$P_{final} = (P_{pixel} - P_{torso}) \times Escala + Compensación \quad (3)$$

Donde:

- P_{final} es la profundidad final del píxel
- P_{pixel} es la profundidad del píxel extraído de la matriz de tensores de MiDaS
- P_{torso} es la profundidad del torso, el ancla definida anteriormente
- $Escala$ es un valor fijo que representa la escala de profundidad
- $Compesación$ es un vector de desplazamiento que permite mover todos los píxeles hacia adelante y hacia atrás según se necesite. Se establece mediante una constante fija

La principal fortaleza de esta estrategia es que permite sacar partido a parte del procesamiento ya realizado (YOLO) y es un método cuya complejidad computacional lineal $O(n)$ (donde n es el número de píxeles) por lo que es fácilmente escalable a grandes volúmenes de datos sin consumir muchos recursos.

3.5.3.4 Optimización de Rendimiento

Otro de los principales retos de este proyecto radica en el rendimiento (con vistas en el funcionamiento en tiempo real). Desde competencia entre algoritmos por la captación de recursos, como tiempos de procesamiento muy lentos pueden generar mermas en el rendimiento general del sistema, así como provocar un desaprovechamiento en sus recursos. Empezando por el Backend, las primeras versiones del sistema apenas podían alcanzar los uno o dos imágenes por segundo, esto era debido en primer lugar a la arquitectura inicial, de carácter responsivo, esto es, el sistema esperaba a que llegara una petición del Frontend para realizar todo el trabajo. Este era un proceso extremadamente lento y que contaba con un alto grado de latencia interna, es por esto que lo primero que se decidió fue separar el sistema en hilos independientes que funcionarían todo el tiempo, de manera que se pudiera adelantar la mayor cantidad posible de información. Esto ayudó a aumentar el rendimiento hasta cerca de los 3 a 4 frames por segundo. Posteriormente a esto, se observó que el módulo principal, que en ese momento estaba compuesto por YOLO y MiDaS competían por los recursos del sistema, por lo que se optó por separarlos, pero además, se observó un problema aún más crítico, el flujo constante de datos entre CPU y GPU debido a que una parte de la lógica del sistema se procesaba en el procesador y otra en el procesador gráfico (GPU), es por ello que se transfirió y aisló todo el proceso para funcionar exclusivamente en GPU, esto permitió aumentar el porcentaje de uso de GPU de un 20% inicial hasta cerca de un 80% de uso, al mismo tiempo que el módulo de procesamiento pasaba de unos 5-8 frames por segundo a cerca de 45. Finalmente, el último cuello de botella del sistema se radicaba en la compresión a PNG que, inicialmente, era un proceso lento en CPU que requería de entre 70 a 100ms lo que limitaba el rendimiento del sistema máximo por debajo de los 10 FPS, en primera instancia, se valoró el uso de alternativas como WEBP o BMP que reducían el tiempo de procesamiento.

Finalmente, también se hizo uso del modo de mínima compresión de PNG reduciendo también el tiempo de procesamiento.

Formato	Tiempo de CPU	Tamaño archivo	Ancho de Banda Requerido (30FPS)
WebP	~60 ms	40KB	~10 Mbps
BMP	~5 ms	1050KB	~250 Mbps
PNG (Min. Compresión)	~18 ms	350KB	~80 Mbps

Tabla 3.5 Comparación entre formatos de imagen valorados

Como se puede observar en la tabla 3.5 la elección del formato final estaba condicionada por dos aspectos, primero el tiempo de compresión (que no debía pasar de 50ms (20FPS)) y el tamaño del archivo resultante, que no debía saturar el ancho de banda de la red. Es por ello que finalmente se optó por usar PNG en modo de compresión mínima, puesto que tenía un tiempo de procesamiento mínimo por debajo del máximo establecido por los requisitos (<50ms) al mismo tiempo que requería de un ancho de banda sustancialmente menor a BMP.

Por el lado del Frontend también se encontraron problemas de rendimiento, sobre todo en equipos sin GPU dedicada. El sistema de partículas inicial, aunque más rápido que el uso de mallas, hacia uso de un script en C# (CPU) para posicionar las partículas que la GPU debía procesar. Este proceso, lento, al tener que intercambiar datos constantemente entre CPU y GPU, era un cuello de botella limitante en el ordenador de pruebas secundario (véase capítulo 4) a funcionar a unos 20 FPS con imágenes de 1024x1024, para solventar este problema, de nuevo se trasladó toda la lógica de la posición a través de un Shader personalizado (un script que se ejecuta en la GPU) a la tarjeta gráfica, de manera que la CPU solo tenía que encargarse de transferir la imagen recibida del Backend a la memoria de la GPU, encargándose esta del resto (El esquema de este Shader personalizado pueden encontrarse en el Anexo I). Este cambio supuso una gran mejora en el rendimiento del Frontend pudiendo ahora manejar imágenes 2048x2048 a cerca de 60 imágenes por segundo en el ordenador secundario y más de 300 FPS en el principal. La tabla 3.6 muestra las mejoras de rendimiento obtenidos en cada paso:

Backend	
Mejora	Rendimiento tras mejora (FPS)
Implementación Inicial	1-2 FPS Total
Programación Multihilo	3-4 FPS Total
Separación tareas en CPU y GPU para reducir pasos entre RAM y VRAM	45 FPS (Procesamiento) 8 FPS (Total)
Optimización del formato de compresión	40 FPS Total
Frontend	
Rendimiento tras mejora (FPS)	Rendimiento tras mejora (FPS)
Implementación Inicial	20 FPS
VFX Graph (Calculo de profundidad en GPU)	60+ FPS

Tabla 3.6 Mejoras de rendimiento del sistema

3.5.4 Pruebas y validación durante el desarrollo

Para validar el buen funcionamiento del sistema mientras se iban sucediendo los cambios, se procedió a diseñar scripts y sistemas de prueba que validaran las distintas partes del sistema, entre ellas destacan:

- **Pruebas de independencia de los microservicios:** estos scripts iban destinados a poder trabajar independientemente en cada parte del sistema (Unity/Python) sin necesidad de hacer funcional la otra parte, esto reduce el coste computacional durante las pruebas y permite probar el funcionamiento de cada parte por separado, en el caso de Python, se utilizó el plugin del navegador “Easy Auto Refresh” que permite refrescar una página de manera periódica (en este caso la URL de la imagen servida por Flask) junto con el intercambio de los canales azul y Alpha para facilitar el visionado de la estimación de profundidad. Por el lado de Unity se usó XAMPP para dar de alta un servidor apache con una imagen de prueba a la que se conectaba Unity para visualizar los datos
- **Pruebas de coherencia visual en Unity:** Se realizaron pruebas visuales entre el mapa de profundidad y el resultado generado, haciendo uso de un sistema de malla (una matriz rectangular de vértices 3D que contenían la textura de la imagen en el que la posición de los vértices en el eje de profundidad cambiaba según los datos del canal Alpha), donde se evalúa, que el mapa de profundidad se trasladara correctamente a la malla y posteriormente se verificaba que la nube de partículas generase un resultado coherente con el resultado de la malla.
- **Verificación de tiempo de respuestas en distintos escenarios:** se realizaron pruebas en distintos escenarios de red para evaluar la latencia y velocidad en conexiones de uso doméstico, tanto en redes wifi (2.4 y 5 Ghz) como en conexiones cableadas.

3.6 Implementación del sistema

Tras la fase de conceptualización, diseño y desarrollo, la implementación final del sistema se generó como un ecosistema que integra el procesamiento de IA en Python y la visualización en Unity. En esta sección se detallan los aspectos claves para su puesta en funcionamiento y el control de versiones aplicado.

3.6.1 Entorno de ejecución y despliegue

La implementación se basa en un sistema con una arquitectura de microservicios basados en Docker. El uso de contenedores ha permitido encapsular las dependencias críticas del Backend, principalmente aquellas referentes a Python (como PyTorch u OpenCV). La estructura y los requisitos finales del sistema son los que siguen:

3.6.3.1 Backend (Servidor)

El Backend requiere de dos recursos clave, en primer lugar, el contenedor de Docker junto a una instalación actualizada de este software, en segundo lugar, para garantizar la compatibilidad más allá del sistema operativo, Docker no tiene acceso directo a los recursos hardware del sistema, por lo que se ha optado por montar un servidor de retransmisión de vídeo dentro de un contenedor Docker (El fichero docker-compose.yml y dockerfile pueden revisarse en el Anexo I), al que el usuario deberá conectar el streaming de vídeo a través de un software de transmisión como por ejemplo OBS.

3.6.3.2 Frontend (Cliente)

El Frontend por otro lado es más sencillo de instalar puesto que requiere de una única aplicación para funcionar, siendo esta, el ejecutable compilado de Unity, que es capaz de funcionar en cualquier dispositivo. Este Frontend es capaz de conectarse a la dirección IP del Backend para funcionar, haciendo uso de la clase integrada UnityWebRequest, el Frontend puede solicitar de forma asíncrona la textura procesada en el Backend, posteriormente, usando un sistema de partículas personalizado basado en un shader que se ejecuta directamente en la GPU, interpreta el canal Alpha para desplazar las partículas sobre el eje Z, reconstruyendo así la figura del usuario en tiempo real. El sistema ha sido diseñado para que la mayor parte de la carga de procesamiento en el servidor lo que permite que la ejecución del cliente sea bastante liviana.

3.6.2 Interfaz y flujo de usuario

El sistema se ha diseñado con la intención de ser prácticamente “plug and play”. Al iniciar ambos módulos, el usuario recibe a través de la consola de Docker la información acerca del estado del sistema. Por otro lado, en el Frontend, Unity permite al usuario ajustar parámetros de visualización como la cantidad de

partículas a utilizar respecto al número total de píxeles, o aplicar “eliminar” las partículas cuya profundidad se encuentre por debajo de un umbral dado, las figuras 3.9, 3.10 y 3.11 muestran la interfaz de usuario del Frontend. Por otro lado, la figura 3.12 Muestra la interfaz gráfica del Backend.

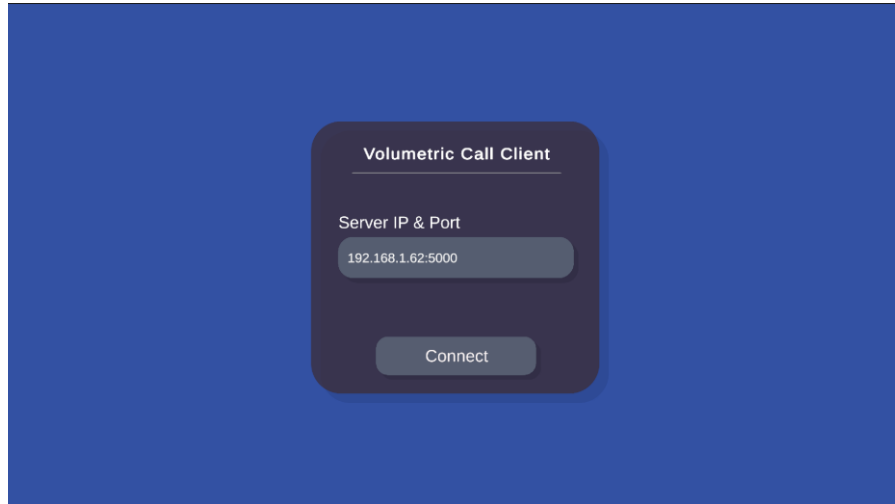


Figura 3.9 Interfaz del Frontend. Conexión del Backend

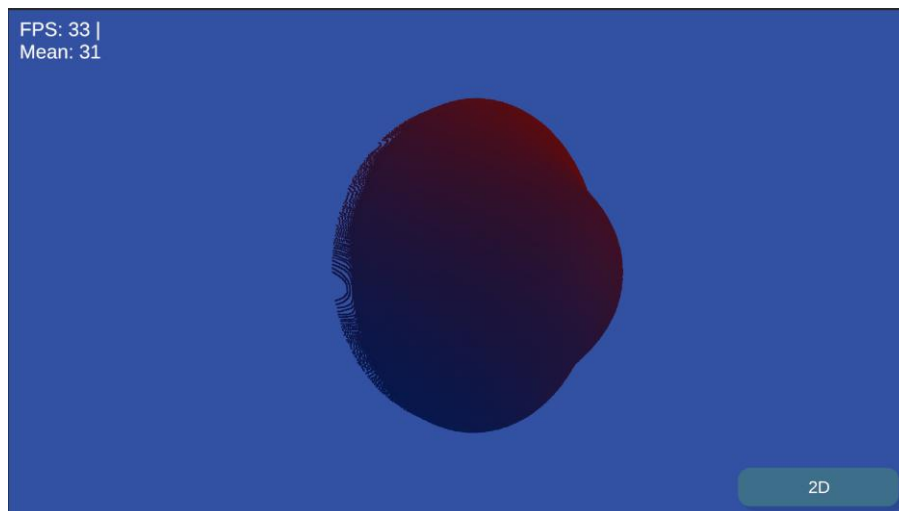


Figura 3.10 Interfaz del Frontend. Imagen de pruebas

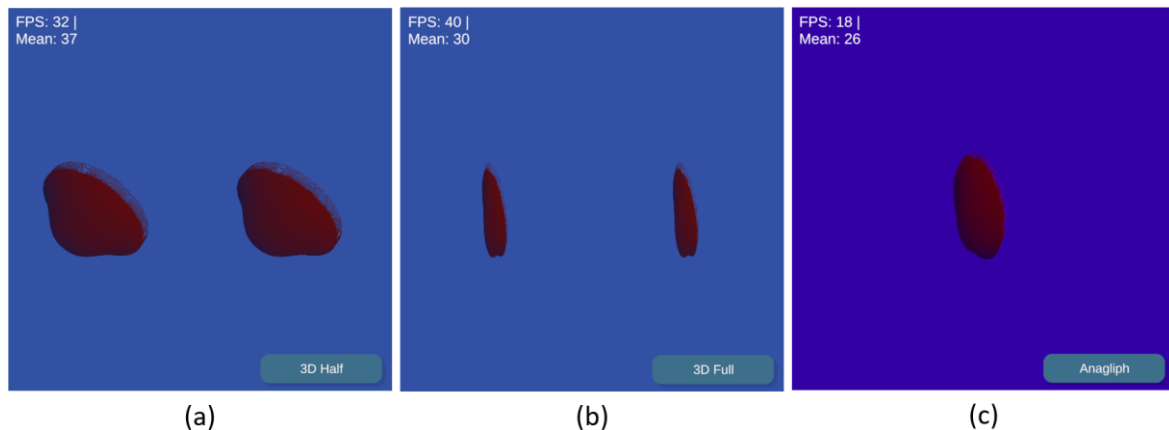


Figura 3.11 Frontend. Modos de visualización Alternativos. (a) Half SBS (para dispositivos VR). (b) Full SBS (para Proyector 3D). (c) Anaglifo

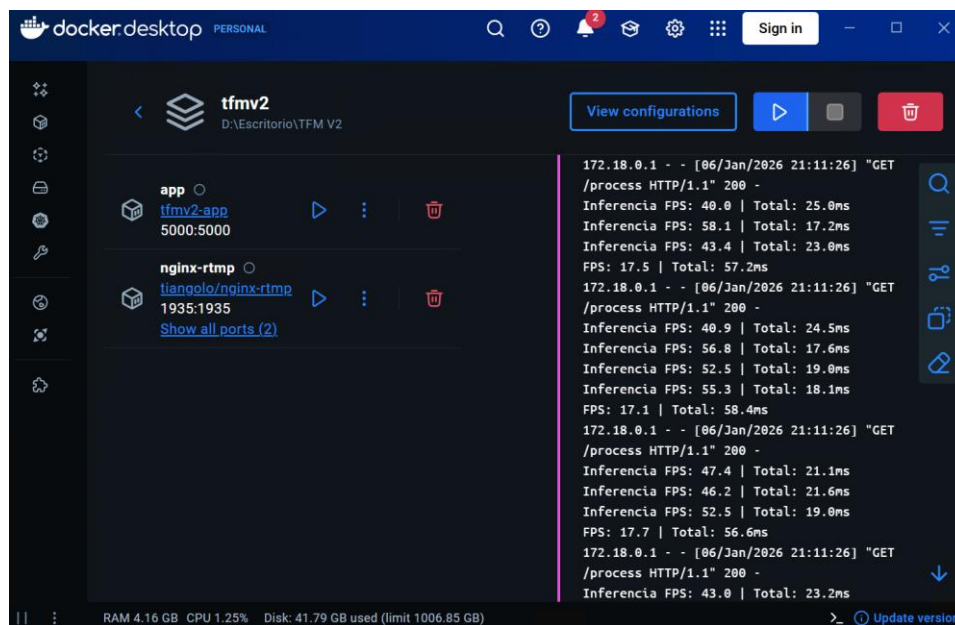


Figura 3.12 Interfaz del Backend. Para iniciar el Backend basta con iniciar el contenedor (botón de play) e iniciar la retransmisión desde la fuente de vídeo (OBS Studio)

Tal y como se presentó a inicios del capítulo, el flujo desde el punto de vista del usuario es el que sigue, una vez los dos usuarios han instalado sendos servidores y clientes, ambos deberán iniciar primero el servidor, una vez este haya arrancado, se deberá conectar el flujo de streaming de vídeo al Backend ya sea a través de un Streaming de OBS o de alguna otra herramienta. Una vez completado el proceso anterior, y con el Backend funcionando, ambos usuarios podrán ahora iniciar el Frontend, donde se les pedirán los datos de IP al que se quieren conectar, que se corresponderán con la IP mostrada en el Backend del otro usuario (y en este caso para las pruebas, la IP LAN/Privada del servidor al que se quiere conectar el cliente). Una vez introducida o seleccionada la IP, el cliente tratará de conectarse al servidor. En caso de que la conexión sea exitosa, comenzará el flujo de datos entre los servidores y los clientes de los respectivos usuarios. Sumado a lo anterior, el usuario podrá también ajustar mediante deslizadores, la calidad visual (esto es, la

cantidad de partículas utilizadas para la representación 3D) así como establecer un delimitador de profundidad, que haciendo uso de un proceso de umbralización, no renderiza las partículas cuya profundidad quede por debajo del portal. Adicionalmente, para facilitar el uso del sistema por parte de los usuarios, se diseñó un manual de usuario (Véase Anexo V) para facilitar el uso del prototipo desarrollado

3.6.3 Gestión de problemas en la implementación

Durante la fase de integración final y despliegue sucedieron una serie de retos específicos derivados de la interacción entre el entorno de contenedores, el hardware físico y la red local. A continuación, se detallan los principales obstáculos encontrados junto con las soluciones técnicas aplicadas.

Uno de los mayores desafíos encontrados al usar Docker en entornos de Windows (a través del subsistema de Windows para Linux WSL2) fue la imposibilidad de acceder de forma nativa a los dispositivos USB, concretamente a la webcam. Para solventar esto, se desarrolló una capa de streaming intermedio, se levantó un servidor RTMP que actuaba de intermediario con el programa. Este servidor recibía una fuente de vídeo mediante OBS Studio, para que posteriormente el servidor de Python pudiera consumir este recurso. Este procedimiento, añade una latencia adicional al sistema, cuyo impacto en la latencia será abordado en el capítulo cuarto, pero a su vez, también supone una ventaja, y es que al poder desvincular la cámara web de la entrada del sistema, ahora se pueden hacer uso de secuencias de vídeo pregrabado para revisar el funcionamiento del sistema, lo cual ayuda a la hora de evaluar la solución.

Otro de los problemas encontrados, radicaba en el alto consumo inicial de la VRAM y la gestión de los recursos de la GPU a causa de ejecutar simultáneamente varias redes neuronales en la tarjeta gráfica. Esto llevó en ocasiones a problemas de bloqueos del sistema. Para solucionar esto se llevaron a cabo varias medidas, en primer lugar, se redujo la precisión de cálculo de todos los sistemas a precisión media (FP16) para reducir el uso de memoria y los tiempos de carga. También se reestructuró el Backend por módulos para ser más eficiente y se creó una fase inicial de optimización de MiDaS al hardware de destino (esto genera un retraso de unos 90 segundos al arrancar el Backend, pero se compensa con un aumento de rendimiento una vez puesto en funcionamiento).

3.6.4 Control de versiones y mantenimiento

Para evitar la pérdida de datos y un sistema que permitiera volver a una versión anterior de software en caso de llegar a “un callejón sin salida” se hizo uso por un lado de Git, para el control de versiones, y del sistema de control de versiones de Unity (VCS) basado en Plastic SCM.

Por otro lado, la gestión de los pesos de las redes neuronales se llevó a cabo por separado debido principalmente al tamaño que ocupaban en disco, por lo que su gestión se llevó a cabo en local, almacenándose en una carpeta y cargándose dinámicamente según necesidad.

3.7 Resumen del Capítulo

Este capítulo sintetiza todo el proceso de desarrollo, desde el diseño inicial de la arquitectura hasta el proceso de despliegue del pipeline propuesto para la reconstrucción volumétrica y transmisión en tiempo real de datos de videollamada.

El pipeline propuesto se basa en un sistema desacoplado que separa la inferencia lógica (Backend en Python) de la representación visual (el Frontend en Unity). Esta estructura permite aprovechar la potencia de procesamiento de modelos de aprendizaje profundo y la versatilidad de los motores gráficos modernos de manera simultánea.

Se han descrito así mismo, las decisiones tecnológicas clave, destacando la adopción de Intel MiDaS para la estimación de profundidad por su capacidad de generalización y el uso de YOLO para optimizar el cómputo mediante la segmentación del sujeto de interés. Además, se ha justificado la implementación de una red neuronal personalizada basada en FSRCNN como una etapa clave para optimizar el rendimiento y refinar la calidad visual “recuperando” el detalle fino de la escena.

Finalmente, el capítulo ha abordado la resolución de desafíos técnicos fundamentales como el flickering o parpadeo de profundidad, mediante técnicas de suavizado temporal, y la gestión de la profundidad relativa a través de un sistema de anclaje por software y el afinamiento por redes neuronales. La implementación de este sistema basado en microservicios y contenedores de Docker, establece una base técnica sólida y portátil que permite la comunicación volumétrica sin depender de hardware especializado.

4. Medición y Evaluación

Una vez completada la fase de diseño e implementación del sistema de reconstrucción 3D monocular, es necesario someter la solución a un proceso de validación, tanto comparativo con otras alternativas, como una validación interna en términos técnicos y operativos de la solución. Este capítulo se centra en determinar si el sistema desarrollado cumple con los objetivos de rendimiento, estabilidad y calidad necesarias para una experiencia de comunicación volumétrica viable que funcione sobre hardware de consumo.

La evaluación busca no solo cuantificar la capacidad de respuesta del sistema sino de analizar, además, si las optimizaciones introducidas, como son, por ejemplo, el suavizado temporal o el reescalado mediante una red neuronal personalizada basada en FSRCNN suponen una mejora sobre lo ya existente, así como en el impacto que tienen sobre la experiencia final del usuario.

4.1 Objetivos de la evaluación

El proceso de evaluación presentado se ha diseñado para dar respuestas a las hipótesis que se han planteado al inicio de este trabajo, estas son, en primer lugar, validar el rendimiento en tiempo real de la solución, es decir, determinar si el sistema es capaz de mantener una tasa de fotogramas (FPS) aceptables en equipos de nivel de consumo, en segundo lugar, medir la latencia del sistema, esto es, el tiempo transcurrido desde la captura inicial de la imagen por parte del equipo servidor hasta su visualización por parte del equipo del cliente, lo que es clave para una interacción fluida. Sumando a lo anterior también es relevante medir y analizar la precisión y la consistencia visual final, comprobando la eficacia del algoritmo de estabilización para reducir el ruido en la imagen final, así como la coherencia de la nube de puntos generada. Así mismo, también cabe destacar la identificación de

cuellos de botella, a través de la monitorización del uso de recursos de CPU y GPU para detectar posibles limitaciones a la hora de escalar el sistema. Finalmente, también es de suma relevancia comparar la solución con otras soluciones analizadas en marco teórico de este trabajo, para ponderar como se posiciona la solución aportada respecto del resto de soluciones del mercado.

4.2 Métricas utilizadas

Para evaluar de manera objetiva el rendimiento y eficacia del sistema, se han seleccionado una serie de métricas cuantitativas y cualitativas. Estas métricas permiten analizar el comportamiento del sistema desde cuatro enfoques distintos: rendimiento computacional, fluidez de la comunicación, calidad estructural y calidad percibida por el usuario final.

4.2.1 Rendimiento y Fluidez

Para la evaluación y medición del rendimiento y la fluidez del sistema se ha hecho uso de dos métricas fundamentalmente:

- **Tasa de fotogramas por segundo (FPS)**, esto es, la cantidad de cuadros o imágenes de vídeo procesados, transmitidos y renderizados por el sistema en un segundo. Una tasa estable de alrededor de 20 FPS en hardware de consumo proporcionaría una experiencia aceptable de vídeo en tiempo real.
- **Latencia promedio (ms)**, que se define como el tiempo total transcurrido en milisegundos desde que la cámara captura el movimiento del usuario en el Backend hasta que se renderiza la nube de puntos correspondientes en el Frontend. Esta métrica se desglosa a su vez en:
 - Latencia media de Inferencia: Tiempo de procesamiento de MiDaS y FSRCNN.
 - Latencia media de Red: Tiempo de transferencia de la imagen RGBA vía HTTP.
 - Latencia de visualización: Tiempo de conversión de la imagen RGBA a nube de puntos y de los procesamientos propios del Frontend.

Ambas métricas, tanto la tasa de fotogramas por segundo como la latencia promedio, son esenciales para poder establecer una comunicación fluida y parcial o completamente en tiempo real.

4.2.2 Eficiencia de Recursos

Dado que la solución busca adaptarse a un amplio abanico de hardware de consumo, es importante monitorizar el uso medio de recursos del sistema por parte del sistema, principalmente en lo referente al Backend/servidor dado que es la parte que hace uso de una mayor cantidad de recursos. Es por ello que en este sentido la métrica a utilizar será:

- **Carga de CPU y GPU (%)**, esto es, el porcentaje de uso de los procesadores (tanto el principal como la GPU) durante la ejecución tanto por separado como conjuntamente del Backend como del Frontend. Es crítico evaluar como el desacoplamiento en microservicios optimiza o sobrecarga estos recursos.
- **Consumo de RAM y VRAM**, es decir, cuanta memoria del sistema y memoria de vídeo necesita el sistema para funcionar, esto es especialmente interesante en el caso del Backend, sobre todo la memoria de vídeo que hoy por hoy es la más escasa y no es ampliable sin cambiar la tarjeta gráfica entera.
- **Peso de las imágenes y ancho de banda consumido**, esta métrica es relevante puesto que muestra la capacidad del sistema a adaptarse a distintos entornos de conexión y de red.

4.2.3 Precisión de Detección y Segmentación

Estas métricas evalúan la precisión de los modelos de IA utilizados:

- **Precisión Media Promediada (mAP)**. Es la métrica estándar para evaluar la precisión en los modelos de detección. Para este proyecto se hará uso específico del mAP@0.5, que considera válida una predicción siempre y cuando la intersección entre la caja predicha y la caja real es superior al 50% de la unión de superficie de ambas cajas. Un mAP alto garantiza que el recorte del usuario es preciso y no incluye demasiado ruido de fondo ni recorta partes del usuario. Sumado a lo anterior, se medirá también el mAP@0.5:0.95, una métrica más moderna que mide además de lo anteriormente mencionado como de bien se adapta la caja predicha por el YOLO a la caja real de la imagen. Esta última métrica, permitirá comparar de manera más precisa en trabajos posteriores, la exactitud del clasificador utilizado en este proyecto.
- **Exhaustividad (Recall) y Precisión**. Recall mide la capacidad del modelo para encontrar todas las instancias positivas reales, lo que en este contexto sirve para cuantificar la habilidad de YOLO para detectar al usuario en cada llamada a su función. Esto implica también evaluar su capacidad de falsos negativos, es decir, que el sistema no encuentre y procese al usuario, a pesar de que este se halle en la escena. Este apartado es crítico para la continuidad de la presencia dentro de una videollamada. Por otro lado, y correlacionado a lo anterior, la precisión mide la calidad de las detecciones positivas; es decir, qué porcentaje de las instancias identificadas como “persona” son realmente el usuario. A pesar, de que el principal interés es medir la exhaustividad, ambas métricas están correlacionadas, puesto que a más flexibilidad se le dé al algoritmo para identificar a algo como persona, mayor será la tasa de positivos (ya sean falsos o verdaderos). Esta

correlación entre Exhaustividad y Precisión podrá apreciarse gráficamente más adelante en la figura 4.3.

4.2.4 Estabilidad y Calidad Visual

Estas métricas evalúan la eficacia de los algoritmos de suavizado y superresolución propuestos:

- **La Varianza de Profundidad (Estabilidad Temporal)**, que mide la fluctuación de los valores de profundidad en píxeles estáticos entre fotogramas consecutivos. Esta métrica permite cuantificar la reducción del parpadeo o flickering tras aplicar la media móvil ponderado exponencial. Para aplicar esta métrica se hará uso de la fórmula:

$$v^2_{temp} = \frac{1}{N} \sum (d_t - \bar{d})^2 \quad (4)$$

Donde d es el valor de profundidad normalizada.

- **El Pico de Relación Señal-Ruido (PSNR)**, que permite evaluar la fidelidad de profundidad reescalada por FSRCNN frente al mapa de profundidad Real utilizado durante el fine-tuning.
- **Índice de Similitud Estructural (SSIM)**, que cuantifica en porcentaje la degradación de la calidad de la imagen causada por el procesamiento de esta, es una métrica que requiere de dos imágenes, la de referencia y la que se va a evaluar. A diferencia de PSNR que se basa en el error cuadrático medio, SSIM evalúa la imagen basándose en cambios de luminancia, contraste y estructura. Esta métrica es de vital importancia a la hora de verificar si el modelo de superresolución mantiene la coherencia estructural de los rasgos faciales del usuario.
- **Similitud de parches de imágenes perceptuales aprendidas (LPIPS)**, mide la distancia entre la imagen reconstruida y la real utilizando una red neuronal profunda (VGG o AlexNet) para extraer las características. Esta métrica destaca sobre las anteriores en que penaliza la falta de nitidez y textura, siendo una pieza fundamental para validar la recuperación del detalle fino (como las arrugas o el cabello) a través del reescalador utilizado

4.2.5 Calidad Percibida por el usuario

Estas métricas a diferencia de las anteriores son de naturaleza subjetiva puesto que se basan en la calidad visual percibida por el usuario final. Por ello, se ha optado por integrar un cuestionario post uso basado en una escala de Likert de 5 puntos (donde 1 es “Muy en desacuerdo” y 5 es “Totalmente de acuerdo”)

Esto permite medir, más allá de una nota general, cada una de las dimensiones del sistema por separado, la tabla 4.1 recoge las preguntas que se le hicieron a los usuarios, junto con sus correspondientes dimensiones de evaluación:

Ítem	Dimensión Evaluada	Descripción
L1	Calidad de la Reconstrucción	El modelo 3D representa fielmente la figura y los rasgos faciales
L2	Estabilidad Visual	La imagen es sólida y no presenta parpadeos o saltos molestos en la profundidad
L3	Fluidez Percibida	El movimiento del avatar 3D se percibe suave y natural durante la interacción
L4	Latencia Percibida	El retraso entre el movimiento real y el visualizado es suficientemente bajo o aceptable para una videollamada
L5	IPQ-G1	En el entorno visualizado, tuve la sensación de que la otra persona estaba “allí” conmigo. (Adaptación de: <i>In the computer generated world I had a sense of "being there"</i>)
L6	IPQ-SP1	Sentí que la imagen reconstruida tenía volumen y no era solo una foto plana (Adaptación de: <i>Somehow I felt that the virtual world surrounded me</i>)
L7	IPQ-REAL1	La persona virtual me pareció más real en comparación con una videollamada tradicional. (Adaptación de: <i>How real did the virtual world seem to you?</i>)

Tabla 4.1 Dimensiones de evaluación de la escala de Likert. Las preguntas L5, L6 y L7 han sido extraídas del Igroup Presence Questionnaire

Para poder evaluar de manera estandarizada el sistema (y poder comparar los resultados subjetivos con otros proyectos existentes), las preguntas L5, L6 y L7 fueron extraídas y adaptadas del Igroup Presence Questionnaire (IPQ) (Shubert y otros, 2001) donde se adaptó su escala original de 7 puntos a una escala Likert de 5 puntos para mantener la coherencia con el resto del cuestionario. Esta adaptación de escala no es original, puesto que ya ha sido validado previamente en estudios de telepresencia (Vasconcelos-Raposo y otros, 2016) demostrando mantener la fiabilidad psicométrica de la evaluación.

Una vez contestadas las anteriores preguntas, al usuario se le hace una última pregunta, de carácter abierto (L8) “En su opinión, que es lo que más destacaría del sistema”. Esta pregunta, permite obtener un feedback por parte de los usuarios de sus preferencias y valoraciones acerca del uso del sistema. El cuestionario real pasado a los usuarios, puede verse en el Anexo II

4.3 Metodología de evaluación

La metodología de evaluación ha sido diseñada para someter al sistema a diversas condiciones similares a las que podríamos encontrar en un entorno de uso real. El proceso de evaluación se divide en tres partes fundamentales:

4.3.1 Escenarios de prueba y entorno técnico

Las mediciones se realizaron primero en un entorno compuesto por un equipo que actúa de servidor (Backend) mientras que otro actúa como cliente (Frontend). Para garantizar la adaptación de los datos a entornos reales se definieron los siguientes escenarios:

- Conexión por cable ethernet entre ambos dispositivos con un router como dispositivo intermediario (esta es la disposición de conexión ideal)
- Conexión inalámbrica variable a través de las bandas wifi de 2.4 y 5 GHz para evaluar el sistema en distintos escenarios de cobertura, ancho de banda y latencias de señal
- Entornos de carga variados, esto es, simular un uso “normal” del sistema junto con otras aplicaciones funcionando, como navegadores web y otras aplicaciones que compitan por los recursos del sistema

Estos escenarios permiten evaluar el sistema en distintos entornos, así como aislar el impacto de cada entorno en el resultado final de manera aislada e independiente.

4.3.2 Protocolo de recolección de los datos

Para la evaluación cuantitativa del sistema se siguió un protocolo que permitiese aislar y minimizar anomalías como la ejecución inesperada de tareas en segundo plano o la pérdida momentánea de paquetes es por ello que, en primer lugar, se realizaron sesiones de 10 minutos en cada escenario planteado, registrando los FPS y latencia en periodos estables de 5 segundos a través de logs automáticos generados por Unity y Flask, además, gracias al uso de OBS como fuente de vídeo, se pudo usar una secuencia de vídeo estática para poder medir la eficacia del sistema de reducción de parpadeo basado en suavizado temporal.

Finalmente, se usaron pares de imágenes (mapa de profundidad estimado vs real) para calcular el PSNR (Pico de relación señal-ruido) y validar la eficacia del modelo FSRCNN con fine-tuning. Para la obtención de los datos de profundidad real se hizo uso por un lado de un sensor Kinect V2 para las imágenes reales y la extracción de la profundidad del z-buffer en el caso de los avatares virtuales.

4.3.3 Ejecución del test subjetivo de usuario

Para aplicar la escala Likert definida anteriormente en este capítulo, se seleccionó un grupo de usuarios de prueba a quienes se les solicitó que realizaran una videollamada volumétrica de aproximadamente 5 minutos usando un entorno de Proyección 3D o de realidad virtual en función de los recursos disponibles, en esta prueba, el usuario podía ver por un lado, al interlocutor tanto en 2D (imagen extraída directamente del servidor RTMP) como en reconstrucción volumétrica (imagen procesada por el backend). La figura 4.1 ilustra lo que los usuarios con visor de realidad virtual podían ver durante las pruebas.



Figura 4.1 Ejemplo de lo que los usuarios veían durante las pruebas

Inmediatamente después de finalizar la experiencia, al usuario se le suministró el cuestionario definido en el apartado 4.2.5 a través de un formulario digital (Google Forms), garantizando así que las respuestas reflejaran su percepción inmediata sin sesgos de memoria a largo plazo.

Para la validación subjetiva del sistema se seleccionó una muestra de 7 participantes. Dado las características y limitaciones de este trabajo (se requiere un dispositivo VR o unas gafas anaglíficas para la visualización 3D) y siguiendo los principios de ingeniería de usabilidad de (Nielsen & Landauer, 1993), una muestra entre 5 y 10 usuarios es suficiente para identificar el 85% de los problemas de usabilidad y obtener una tendencia clara sobre la Calidad Percibida. El grupo estuvo compuesto por usuarios con edades entre 21 y 60 años, con diverso grado de familiaridad con las tecnologías utilizadas.

4.4 Resultados obtenidos

En esta sección se presentan los datos recopilados durante las pruebas, organizados según de las dimensiones de rendimiento técnico, estabilidad visual y la percepción del usuario. Para la evaluación del sistema se hizo uso de dos ordenadores, por un lado, un PC de sobremesa con una RTX 3060TI, Ryzen 9 5900X, 32Gb de RAM DDR4 3200Mhz CL16 (Equipo principal) y un portátil con un i5 11300H, gráfica integrada y 16 Gb de RAM LPDDR4X a 4300 MHz (Equipo secundario). También se llevaron a cabo algunas pruebas en una Tablet Lenovo P11 Pro 2º Gen (Mediatek Kompanio 1300T, 8GB de RAM) y en unas Meta Quest 3 para evaluar la adaptabilidad del Frontend a dispositivos móviles y dispositivos VR standalone. Información adicional sobre el entorno y los dispositivos de prueba pueden encontrarse en el Anexo IV.

4.4.1 Validación de los Modelos de IA

La validación de los modelos de IA constituye el núcleo de la evaluación técnica de este trabajo, ya que la fidelidad y el resultado de la reconstrucción volumétrica depende directamente de la precisión de la inferencia realizada en el Backend. Esta sección se centra en analizar el desempeño de dos de los pilares algorítmicos del sistema, en primer lugar, el detector de regiones de interés (YOLO) y en segundo lugar la red de superresolución (FSRCNN).

4.4.1.1 Subsistema de Detección (YOLO)

La validación del subsistema de detección es especialmente relevante, ya que este módulo actuar como la primera etapa del pipeline, encargándose de la segmentación espacial del usuario (Región de interés o ROI). Un error en esta etapa (falso negativo o caja delimitadora errónea) se puede propagar al estimador de profundidad produciendo artefactos visuales o cortes en la malla 3D final.

Para la evaluación se ha utilizado la arquitectura YOLOv8 Nano, seleccionada por su arquitectura anchor-free y su eficiencia en entornos de recursos limitados.

Para evaluar el modelo se ha hecho uso de un subconjunto de datos de validación mucho más diverso que los que YOLO encontrará en el sistema final (Un usuario frente a la cámara) como es el dataset COCO 2017 al cual se le ha aplicado un filtro estricto para la clase “Persona”. La evaluación se realizó usando precisión media (FP16). A continuación, se detallan los valores obtenidos de las métricas evaluadas.

Métrica	Valor Obtenido
mAP@0.5	0.745
mAP@0.5:0.95	0.514
Precision	0.787
Recall	0.653

Tabla 4.2 Métricas del Subsistema de Detección

El análisis detallado de estas métricas muestra un comportamiento altamente favorable para los objetivos del sistema. La Precisión (0.787) superior al Recall (0.653) indica que el modelo prioriza la exactitud, es decir, que rara vez confunde un objeto de fondo con una persona (falsos positivos bajos)

Esta característica es importante para la etapa de segmentación, ya que incluir por error zonas del fondo obligaría a MiDaS a procesar información irrelevante, lo que daría un peor resultado visual. Por otro lado, un Recall más moderado es aceptable en este contexto, ya que, en el entorno real del sistema, el usuario tenderá a ocupar un gran espacio en el centro de la imagen, lo que facilita su detección continua.

Para visualizar este comportamiento, se presentan a continuación tanto la matriz de confusión normalizada (Figura 4.2) como la curva de precisión-Recall (Figura 4.3) obtenidas durante la validación.

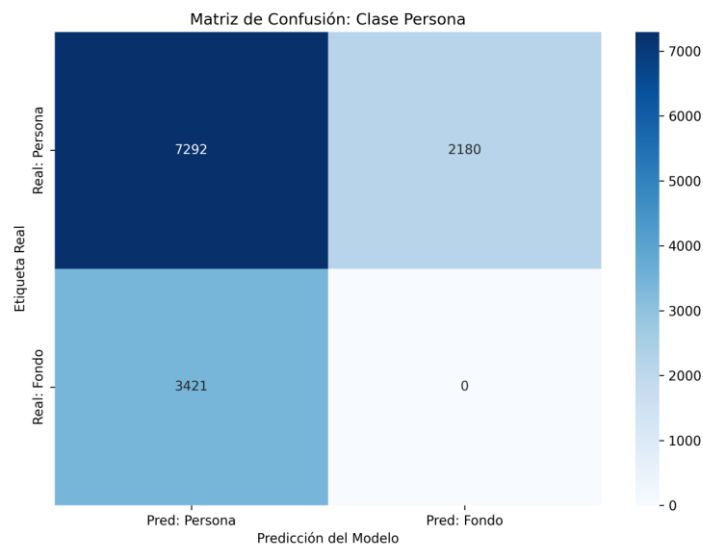


Figura 4.2 Matriz de confusión YOLOv8 Clase Persona

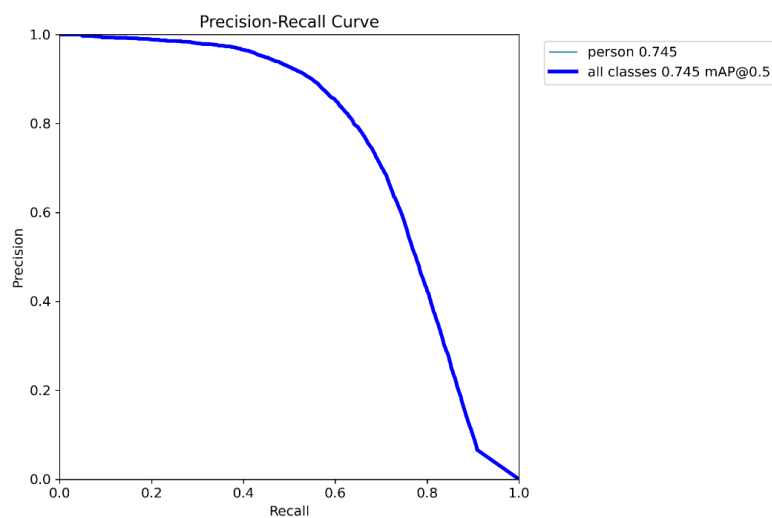


Figura 4.3 Gráfica de Precisión-Recall de YOLOv8 Nano

4.4.1.2 Subsistema de Superresolución (FSRCNN)

La validación del módulo de superresolución se ha llevado a cabo analizando tanto la convergencia del entrenamiento como el impacto cuantitativo en la reconstrucción de la profundidad. El objetivo de la etapa, más allá de aumentar la resolución, radicaba en recuperar parte de la información de alta frecuencia perdida durante la inferencia de MiDaS.

El proceso de fine-tuning de FSRCNN se monitorizó a través de la función de pérdida definida en la metodología. Tal y como puede observarse en la Figura 4.4, el modelo

muestra una adaptación clara en las tres fases de la estrategia de entrenamiento propuesta.

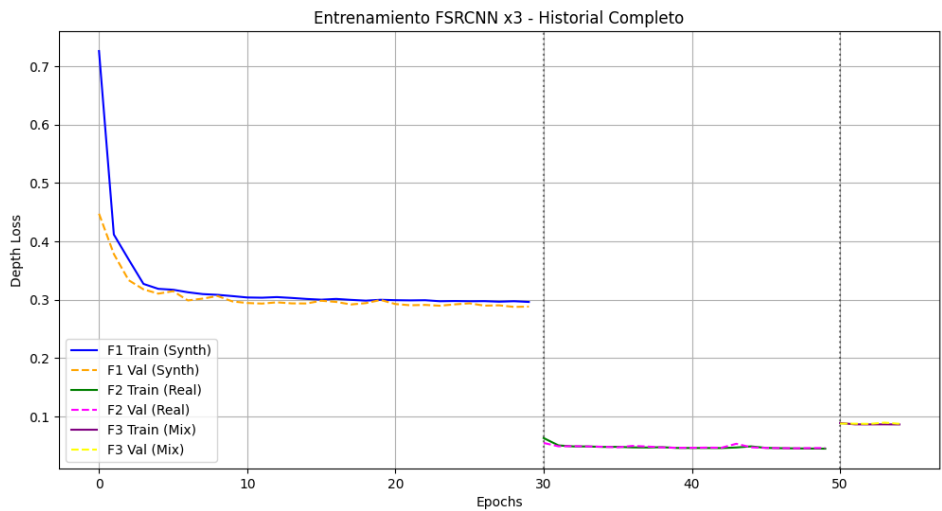


Figura 4.4 Gráfica de Depth Loss del fine-tuning de FSRCNN

En la fase sintética el sistema empieza a entender la topología humana ideal. En la fase real, se puede observar una caída progresiva desde 0.06 y una estabilización del Loss en torno a 0.04 lo que infiere que la red ha aprendido a gestionar el ruido de los sensores reales. Finalmente, en la tercera fase se puede observar una convergencia final del Loss a 0.09 que equilibra la geometría ideal con la textura orgánica, evitando así el olvido catastrófico.

Para medir la eficacia del modelo, se compararon tres escenarios usando el algoritmo de interpolación bicúbica a partir de los datos de MiDaS como línea base para la comparación. Los resultados, se compararon tanto con FSRCNN Genérico adaptado a una sola dimensión (promediando los tres canales de color RGB) como con el FSRCNN personalizado sometido al proceso de fine-tuning. Los resultados, presentados a continuación (tabla 4.3) evidencian el comportamiento generativo de la red:

Modelo FSRCNN Comparado	Referencia	PSNR (dB)	SSIM
Genérico	Bicúbico	50.10	0.9960
Fine-Tuned	Bicúbico	23.44	0.9683
Fine-Tuned	FSRCNN Genérico	23.35	0.9669

Tabla 4.3 Comparativa de métricas de similitud entre el modelo base y el modelo con fine-tuning

El análisis de estos datos revela un aspecto fundamental, por un lado, el modelo genérico (sin fine-tuning) presenta un PSNR extremadamente alto (50.10 dB) y un SSIM casi perfecto (0.9960) respecto al bicúbico. Esto indica que la red original, sin entrenar para profundidad, apenas altera la imagen de entrada, limitándose a

suavizar los píxeles sin aportar nueva información geométrica. Por otro lado, el modelo propuesto (con fine-tuning) presenta un descenso significativo en el PSNR (23.44 dB) y en el SSIM (0.9683). Al contrario de lo que pudiera parecer, este resultado no es negativo, sino que, valida el éxito del entrenamiento, al usar una función de pérdida que penaliza los bordes difusos, la red se ve obligada a “alucinar” y generar bordes y estructuras que difieren de la borrosidad propia de la interpolación bicúbica.

Esta discrepancia, cuantifica la ganancia de detalle, tal y como se puede observar en la figura 4.5, mientras que el modelo genérico preserva la borrosidad y la falta de detalle de la textura de profundidad de MiDaS (alto PSNR), el modelo propuesto reconstruye una nueva estructura de profundidad (bajo PSNR en relación al bicúbico), lo que se alinea con el diseño de recuperar los detalles geométricos perdidos durante la inferencia de profundidad.

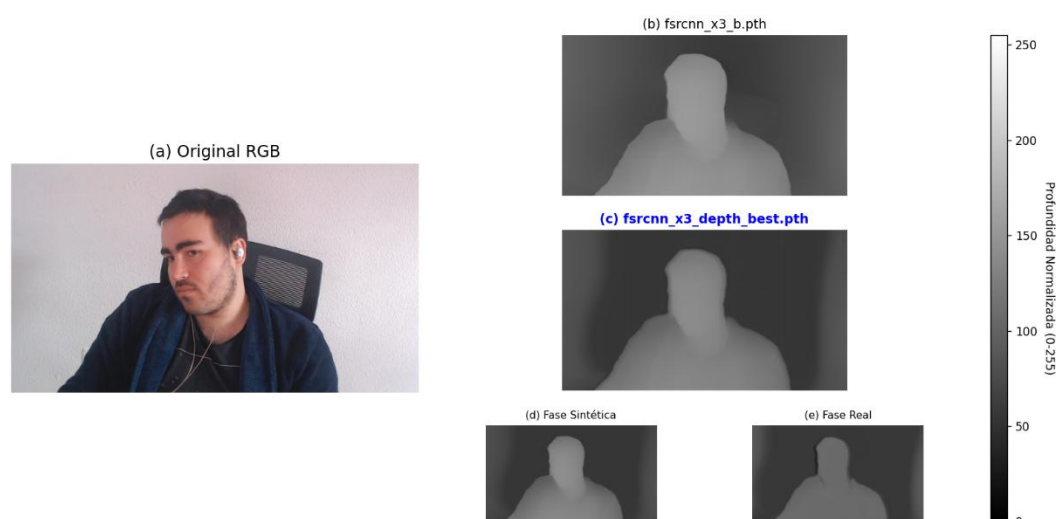


Figura 4.5 Comparativa visual. (a) Imagen RGB de partida. (b) MiDaS + FSRCNN Genérico. (c) MiDaS + FSRCNN Personalizado. (d) y (e) MiDaS + FSRCNN Personalizado hasta Primera y Segunda etapa respectivamente.

4.4.2 Rendimiento y Latencia del Sistema

En esta sección se analiza el comportamiento del pipeline desacoplado bajo diversas condiciones de carga y resolución. Los resultados aquí presentados se corresponden a los datos medios obtenidos a lo largo de sesiones de 10 minutos, utilizando el equipo principal para los procesos del Backend (Y los del Frontend para la tabla 4.6) y el equipo secundario para la visualización del Frontend.

4.4.2.1 Análisis del rendimiento del Backend

El Backend, tal y como se muestra en la tabla 4.4, ejecutado en el equipo principal, demostró una alta capacidad de procesamiento gracias a la ejecución integral en GPU y en el uso de precisión media (FP16)

	Promedio FPS	1% Más Alto	1% Más Bajo	Desviación Típica
Inferencia (MiDaS + FSRCNN)	65.8 FPS	89.1 FPS	47.6 FPS	± 10.8 FPS
Detección (YOLOv8)	27.0 FPS	33.7 FPS	24.5 FPS	± 2.4 FPS
Compresión PNG (Salida)	58.1 FPS	78.2 FPS	29.8 FPS	± 10.4 FPS

Tabla 4.4 Tabla de rendimiento (FPS) del Backend en el equipo de pruebas principal

Tal y como muestran los resultados, el subsistema de inferencia alcanza picos de hasta 89.1 FPS, lo que confirma que el modelo de IA no constituye un límite o un cuello de botella crítico del sistema. Sin embargo, el rendimiento real de salida, se ve condicionado por la etapa de compresión PNG que promedia los 58 FPS

4.4.2.2 Eficiencia en el Uso de Recursos

La monitorización de recursos durante una ejecución estable refleja la eficiencia de la arquitectura de microservicios y contenedores

Fase	CPU Promedio	GPU Promedio	RAM Promedio	VRAM (Reservada)
Arranque	0% - 50%	12-15%	685 MB	0.10 GB
Warmup / Compilación	85% - 112%	17-20%	1.2 GB	0.11 GB
Ejecución Estable	116%	90-96%	1.72 GB	0.41 GB

Tabla 4.5 Uso de recursos del Backend en equipo de pruebas principal

Tal y como se puede ver en la tabla 4.5, el sistema mantiene una huella de memoria muy reducida, consumiendo un promedio de 1.72 GB de RAM y reservando apenas 0.41GB de VRAM para el uso del sistema. Respecto al consumo de los recursos de procesamiento, en cuanto a consumo de CPU y GPU, el procesador registra picos de uso de un 116% (este valor no es un error de lectura, sino que al ser un programa multihilo, la carga de trabajo se divide entre varios núcleos, y al sumarse pueden dar valores superiores al 100% que representaría la carga de trabajo completa de un único núcleo. En este contexto, un 116% significa que el programa está usando tan solo el equivalente a 1.16 núcleos físicos), un uso bastante comedido teniendo en cuenta que los procesadores modernos de gama doméstica suelen contar con 4, 6, 8 o 12 núcleos. Por otro lado, la GPU opera en un rango del 90-96% de uso, lo que implica que el programa está aprovechando todos los recursos de la tarjeta gráfica para su funcionamiento.

4.4.2.3 Impacto de la Resolución y la Compresión en la Latencia

La latencia total del sistema se ha desglosado entre el tiempo de procesamiento mostrado por el Backend, y el mostrado por el Frontend (Real), la sobrecarga del

transporte hace referencia al rendimiento perdido en el transporte entre ambos sistemas.

Nivel Compresión PNG	Peso PNG (MB)	Ancho de Banda (Mbps)	Latencia del Backend (ms)	Sobrecarga de Transporte HTTP (ms)	Latencia Total (ms)	FPS Reales
720P Nivel 0	1.4	319.1	16.2	37.0	53.2	18.8
720P Nivel 1	0.55	62.1	52.3	30.0	82.3	12.1
480P Nivel 0	0.85	218.4	12.4	21.4	33.8	29.5
480P Nivel 1	0.30	51.2	24.9	49.4	74.3	13.5

Tabla 4.6 Comparación de latencias a distintas resoluciones y compresiones

Los datos de la Tabla 4.6 confirman que la configuración 480P sin compresión ofrece la experiencia más fluida, alcanzando casi los 30FPS reales con una latencia total de 33.8 ms. Por el contrario, usar niveles de compresión más altos aumenta drásticamente la latencia del Backend (hasta 52.3 ms en 720P) incluso haciendo uso de librerías optimizadas como pyvips, aunque, por otro lado, también permite reducir ampliamente el tamaño de las imágenes y el ancho de banda (pasando de 319.3Mbps a 62.1Mbps en 720P). También se observa un cuello de botella importante en la sobrecarga de la capa de transporte que puede llegar a consumir hasta el 66% de la latencia, en 480P con compresión de Nivel 1.

4.4.2.4 Impacto del medio de transporte en la Latencia Final

En esta sección se analiza la capacidad del medio físico (Wifi, Ethernet, etc) que conecta el Backend y el Frontend para afectar a la latencia de la transmisión y por ende a la fluidez del sistema. Para las pruebas se usó la configuración identificada como más eficiente en el apartado anterior (480p Nivel 0), además el Host, siempre estuvo conectado mediante ethernet durante las pruebas.

Escenario de Conexión	Ancho de Banda (Mbps)	Latencia Transmisión (ms)	Latencia Total (ms)	FPS Reales	Sobrecarga de Red (ms)
Ethernet (Cable)	186.4	31.3	43.7	22.9	+ 9.9 ms (+ 46.2%)
Wifi 5 GHz	137.6	46.2	58.6	17.1	+ 24.8 ms (+ 115.9%)
Wifi 2.4 GHz	91.5	76.8	89.2	11.2	+ 55.4 ms (+ 258.9%)

Tabla 4.7 Resultados obtenidos en los escenarios de conectividad y latencia a 480P Nivel 0

Los resultados obtenidos, muestran que el ancho de banda y la estabilidad de la red son factores críticos a la hora mantener la experiencia en tiempo real. Por un lado, la conexión por cable Ethernet se erige como la más estable, teniendo una latencia de 31.3 ms sumando 9.9 ms a la sobrecarga de Transporte de la Tabla 4.6. Esto da

unos 22.9 FPS reales, superando el requisito de fluidez mínima de 20 FPS establecido en los requisitos.

Por otro lado, la conexión Wifi muestra un peor desempeño. La conexión Wifi 5 GHz se posiciona como una alternativa inalámbrica viable para entornos domésticos. Aunque la latencia de transmisión aumenta hasta los 46.2 ms (14.9 ms más que mediante Ethernet), el sistema aún es capaz de mantener una tasa estable de 17.1 FPS, que, aunque más baja que la anterior sigue dentro del rango de aceptación establecido en los requisitos.

Finalmente, la conexión Wifi 2.4 GHz se posiciona como la peor de todas, llevando la latencia de transmisión hasta los 76.8 ms (45.5 ms peor que Ethernet) reduciendo el rendimiento hasta unos 11.2 FPS, lo cual queda por debajo de los requisitos de fluidez establecidos por el proyecto.

4.4.2.5 Rendimiento y Latencia del Frontend

Para el análisis del rendimiento del Frontend (Unity 3D) se ha incluido además de los Equipos Principal y Secundario, una Tablet Android (Cuyas especificaciones pueden encontrarse en la introducción de la sección 4.4) con el fin de analizar el funcionamiento del Frontend en dispositivos de distintas capacidades técnicas. Para normalizar todas las pruebas, todos los dispositivos hicieron uso de una resolución interna de renderizado de 1080P (1920x1080)

Dispositivo	Promedio FPS	1% Más Bajo	1% Más Alto	Desviación Típica
Equipo Principal	399.5	99.1	604.1	1.15 ms
Equipo Secundario	102.5	49.8	314.2	1.81 ms
Tablet	62.3	16.8	120.1	9.16 ms
Meta Quest 3	69.5	48.5	91.6	1.43 ms

Tabla 4.8 Rendimiento del Frontend

Los resultados obtenidos en los tres dispositivos de prueba, confirman la eficiencia del renderizado procedimental mediante VFX Graph y Shaders personalizados.

El Equipo Principal (RTX 3060Ti) alcanza un promedio de 400 FPS usándose dentro del editor de Unity (que acarrea algunas pérdidas respecto a la versión compilada). Esta cifra demuestra que, en hardware de alto rendimiento, la carga computacional del Frontend es sumamente despreciable.

Por otro lado, en el Equipo Secundario (I5 con Gráfica Integrada), el Frontend mantiene un promedio de 102.5 FPS. El hecho de superar los 100FPS en un procesador de gama media portátil sin GPU dedicada confirma que la estrategia de desplazar el cálculo de posiciones a los Shaders de la GPU elimina el cuello de botella de la CPU y valida el uso del Frontend en dispositivos de bajos recursos.

Yendo incluso más allá, en la Tablet (Lenovo P11 Pro 2º Gen) el sistema registró un promedio de 62.3 FPS. Esta evaluación es algo diferente a las anteriores, puesto que la Tablet por defecto funciona con un bloqueo de FPS que le obliga a trabajar, según el rendimiento en los modos de 30, 40, 60 y 120 FPS, saltando entre estos cuando el rendimiento lo permite, esto justifica el aumento de la desviación típica obtenido. Aun así, el rendimiento superior a 60 FPS en un procesador móvil, valida la viabilidad de la solución en dispositivos móviles y entornos de computación espacial ligera.

Finalmente, la solución también se probó de manera experimental en un dispositivo Meta Quest 3 a resolución nativa (4128x2208 entre ambos ojos) alcanzando unos 65 FPS de media, haciendo simultáneamente el streaming de la imagen 3D y de la imagen plana, y usando el modo realidad mixta, que tiene un mayor consumo de recursos. Lo que pone en valor esta solución en dispositivos de realidad virtual standalone.

4.4.3 Estabilidad y Calidad de Reconstrucción

Para validar la eficacia de las optimizaciones de suavizado temporal y superresolución se compararon los resultados del modelo MiDaS base, frente al sistema optimizado (MiDaS + FSRCNN + Suavizado). En el caso de la prueba de fidelidad visual también se evaluaron ambas opciones frente a el Ground Truth (Kinect)

4.4.3.1 Varianza de Profundidad y LPIPS

Para validar la eficiencia de las optimizaciones implementadas en el proyecto, concretamente aquellas relacionadas a la dimensión temporal y a la calidad visual (esto es, el suavizado temporal y la etapa de superresolución) se llevó a cabo una comparativa directa entre tres escenarios, MiDaS Base (sin post-procesado), el sistema propuesto (MiDaS + FSRCNN con Fine-Tuning + Suavizado Temporal) y el Ground Truth (Datos reales obtenidos mediante Kinect)

Uno de los problemas propios de la estimación monocular, reside en la falta de retroalimentación temporal, es decir, el sistema estima la profundidad a partir de un único frame cada vez, sin tener en cuenta las imágenes anteriores, esto puede dar lugar a que pequeñas variaciones en las imágenes acaben provocando grandes cambios en la profundidad de un frame a otro.

Para mitigar este fenómeno y evaluar la eficacia del algoritmo de suavizado temporal basado en medias móviles ponderadas exponenciales, se llevó a cabo una prueba de estabilidad sobre una secuencia de vídeo controlada (pares RGB y Depth extraídos con Kinect) con un sujeto estático. Se utilizó la métrica de Varianza de Profundidad definida en la sección 4.2.4 comparando la señal de profundidad extraída directamente de MiDaS frente a la estabilizada por el sistema.

Modelo	Métrica	Valor Obtenido
Ground Truth (Kinect)	Varianza de Profundidad	1594.80
MiDaS	LPIPS	0.3945
	Varianza de Profundidad	7625.97
MiDaS + FSRCNN + Suavizado Temporal	LPIPS	0.1822
	Varianza de Profundidad	1203.33

Tabla 4.9 Métricas temporales. Resultados

Como se puede observar en los datos, el filtro de suavizado temporal consigue reducir la varianza temporal. Esta reducción confirma la idea de que la coherencia temporal puede ser parcialmente recuperada mediante software.

Por otro lado, también se puede observar una mejora crítica en la métrica de LPIPS (en la que se usa una segunda red neuronal, para evaluar la similitud porcentual, penalizando al mismo tiempo a las imágenes borrosas o carentes de texturas). Los resultados muestran una reducción del error de 0.39 en el modelo base hasta 0.1822 lo que nos permite inferir una mejora en la nitidez de la imagen, así como en el detalle fino de sus texturas. La figura 4.6 muestra de manera visual la diferencia entre los distintos modelos extrayendo la imagen de profundidad resultante del primer frame, además, la imagen ha sido codificada como imagen de temperatura (donde los negros y morados representan mayor profundidad y los rojos y amarillos representan cercanía) para facilitar la identificación visual de patrones en las imágenes de profundidad.



Figura 4.6 Comparación entre Ground Truth contra MiDaS y contra el modelo propuesto

4.4.3.2 Fidelidad Visual

Más allá de las métricas numéricas, al ser este un sistema orientado hacia la interacción humana, es también importante hacer hincapié en la calidad visual percibida. Para ello, se ha comparado la reconstrucción de la salida entre MiDaS base y el algoritmo propuesto desde más cerca, para poder contemplar texturas y bordes que no pueden observarse en la figura 4.6

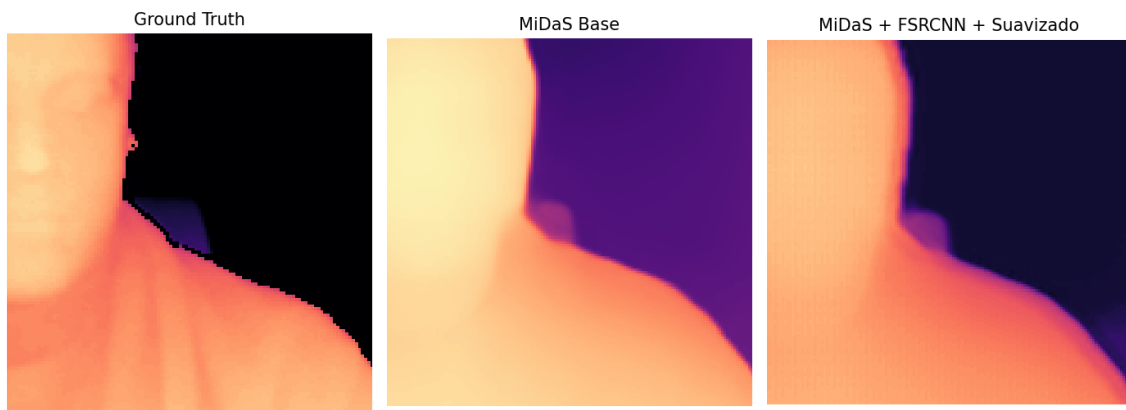


Figura 4.7 Comparación más detallada de MiDaS Base (MiDaS + Bicúbico) con el modelo propuesto

Empezando por el lado negativo, como puede verse en la figura 4.7 el modelo entrenado ha aprendido a inferir el ruido de la imagen, produciendo un efecto rejilla producto de las alucinaciones, pero, por otro lado, también ha aprendido a interpretar de mejor manera la profundidad, generando transiciones más suaves en la cabeza y en el hombro. Además, ha aprendido a detectar mejor algunos contornos como el de la mejilla, que en MiDaS Base parece “meterse” hacia el interior de la cara, esto prueba la capacidad del sistema para corregir defectos de MiDaS, también por otro lado, el modelo ha aprendido a suavizar las líneas del borde del hombro, pero aplicar ruido en la zona del pelo, esto probablemente debido a la mayor exposición en el entrenamiento a modelos sintéticos (con cuerpos más estilizados) que a modelos reales. La figura 4.8 Muestra estos resultados aplicados directamente sobre el Frontend.

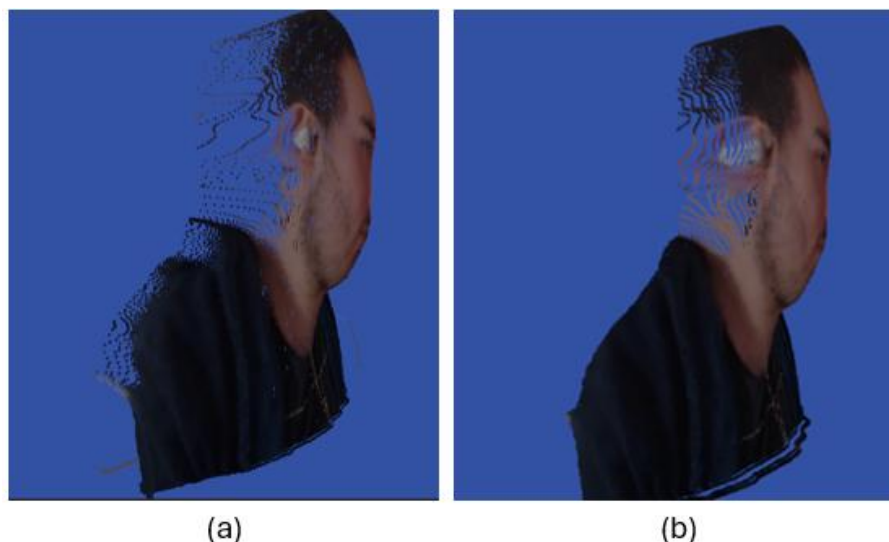


Figura 4.8 Comparación Fine-Tuning. (a) FSRCNN genérico. (b) FSRCNN con Fine-Tuning

4.4.4 Evaluación Subjetiva (Escala Likert)

Los resultados del test subjetivo realizado a los usuarios se muestran promediados en la tabla 4.10 haciendo uso de una escala entre 1 (Muy en desacuerdo) a 5

(Totalmente de acuerdo). Los datos brutos del cuestionario pueden consultarse en el Anexo III.

Ítem	Dimensión Evaluada	Media	Desviación Estándar
L1	Calidad de la Reconstrucción	4.71	0.49
L2	Estabilidad Visual	4.14	0.69
L3	Fluidez Percibida	4.57	0.53
L4	Latencia Percibida	3.86	0.69
L5	IPQ-G1	4.57	0.53
L6	IPQ-SP1	5	0
L7	IPQ-REAL1	4.71	0.49

Tabla 4.10 Resultados obtenidos de la escala Likert

Además de lo anterior, el análisis de las respuestas abiertas revela una acogida positiva hacia la tecnología propuesta. El total de la muestra (N=7) destacó la sensación de presencia social como el factor diferencial del sistema. Los usuarios describieron la experiencia como “natural” y con una “sensación de cercanía” mayor que en las videollamadas tradicionales, destacando la percepción de volumen y la visibilidad de los gestos no verbales.

Desde un punto de vista cultural, la experiencia recordó a referentes cinematográficos de la ciencia ficción (como las comunicaciones de la saga Star Wars), describiendo el sistema como una materialización de “comunicaciones futuristas en la vida real”. Finalmente, en términos de usabilidad, el sistema fue calificado como “cómodo y agradable”

4.5 Análisis e interpretación de resultados

Una vez expuestos los resultados técnicos y experimentales en los apartados anteriores, la presente sección se dedica al análisis e interpretación de dichos hallazgos. El objetivo de este bloque radica en ir más allá de la presentación de cifras para comprender las implicaciones prácticas del sistema en un entorno real.

Este análisis permitirá confirmar si la arquitectura propuesta cumple con la hipótesis original de democratizar el acceso a las videollamadas y el “streaming” volumétrico sin requerir de hardware específico.

4.5.1 Equilibrio entre latencia y Calidad (El compromiso para conseguir el Tiempo Real)

El mayor desafío técnico del proyecto residía en alcanzar una experiencia de comunicación volumétrica en tiempo real en hardware de consumo. Para ello se estableció un umbral de fluidez de 15 a 20 FPS con la mínima latencia posible. El análisis de los resultados obtenidos en este capítulo confirma que el sistema no solo cumple con este requisito, sino que es capaz de superarlo gracias a un diseño

centrado en alcanzar un compromiso entre resolución de procesamiento y refinamiento algorítmico.

Los datos de rendimiento del Backend demuestran que usar la versión small de MiDaS (256x256) es la decisión que permite desbloquear el tiempo real. Si se hubiera optado por procesar la imagen a una resolución superior como la original, los tiempos de inferencia habrían escalado de manera exponencial, lo que alejaría al sistema de su objetivo primordial. Al delegar la recuperación del detalle fino a una segunda etapa con FSRCNN, cuyo coste computacional es despreciable (<3 ms), se logra escapar de la baja calidad obtenida al procesar imágenes a baja resolución. Las métricas de PSNR y SSIM validan este enfoque, aunque la similitud matemática disminuye respecto al uso de reescaladores geométricos, la “alucinación” de detalles geométricos por parte de la red personalizada aporta una riqueza visual que los métodos geométricos no pueden ofrecer.

En definitiva, estas métricas confirman que el modelo FSRCNN con fine-tuning no solo escala la imagen, sino que reconstruye con éxito detalles geométricos que la interpolación bicúbica ignora. Además, la implementación de medias móviles ponderadas exponenciales ha logrado mitigar el fenómeno del parpadeo o flickering, consolidando así, una base visual más estable para la experiencia de usuario

Un hallazgo de suma importancia derivado de la evaluación radica en que el cuello de botella principal no se encuentra en los modelos de IA sino en la etapa de transporte de datos. Tal y como se observa en la Tabla 4.6, existe una relación inversamente proporcional entre el nivel de compresión y la latencia. Además, la sobrecarga del transporte supone el foco principal de generación de latencia pudiendo ser el responsable de hasta el 66% de la latencia total.

4.5.2 Independencia del hardware

El análisis del rendimiento del Backend en el equipo principal (Tabla 4.3) confirma la escalabilidad de la arquitectura propuesta. Las pruebas realizadas sobre la configuración de referencia (Ryzen 9 5900X + RTX 3060 Ti) arrojan una tasa de inferencia promedio combinada (MiDaS + FSRCNN) de 65.8 FPS, con picos superiores a los 89 FPS.

Sin embargo, el rendimiento efectivo de salida del sistema se sitúa en 44.1 FPS promedio, limitado por la etapa de compresión a PNG previo a la transmisión, esta decisión, justificada en la sección 3.5.3.4, si se compara con los objetivos de fluidez establecidos para el proyecto (al menos 20 FPS), se observa que el sistema opera con un margen de seguridad superior al 200%

Este amplio margen de rendimiento permite inferir la viabilidad del sistema en una gama extensa de hardware. Incluso en escenarios teóricos donde se utilice una

GPU de gama de entrada con aceleración CUDA y Tensor Cores (como una RTX 3050 o RTX 2060), cuya potencia de cómputo es aproximadamente un 40-50% inferior a la del equipo de pruebas, la tasa de inferencia proyectada se mantendría en el rango de los 30-35FPS, lo cual sigue superando tanto el umbral mínimo de fluidez como, en muchos casos, la propia velocidad de captura de las cámaras web estándar (30 FPS), se concluye que la solución es plenamente funcional en toda la familia de tarjetas gráficas RTX, eliminando la necesidad de hardware de gama alta para su ejecución.

Adicionalmente, el análisis de consumo de la memoria (Tabla 4.5) respalda la eficiencia del diseño del software. Durante la fase de ejecución estable, el Backend ha registrado un consumo de memoria RAM de apenas 1.72GB, y, lo que es aún más relevante, una ocupación de VRAM (memoria de vídeo) de apenas 0.41GB

Este dato es especialmente relevante para la democratización del sistema, ya que el bajo consumo de VRAM garantiza que el modelo de Inteligencia Artificial no saturará la memoria gráfica, permitiendo su ejecución incluso en tarjetas gráficas con capacidad de memoria limitada como 2GB o 4GB. Asimismo, la baja huella en RAM asegura que el sistema pueda coexistir con otras aplicaciones necesarias para la videollamada (como OBS, navegadores o clientes de chat) sin provocar cuellos de botella por falta de memoria en equipos con 8GB o 16GB de RAM.

Respecto al aprovechamiento de los recursos de procesamiento, el CPU registra picos de uso del 116%. Este valor no es un error de lectura, sino que, al tratarse de un software multihilo, la carga se distribuye entre varios núcleos, representando el uso equivalente a solo 1.16 núcleos físicos. Este consumo resulta notablemente eficiente, teniendo en cuenta que los procesadores domésticos modernos cuentan con configuraciones de 4, 6 u 8 núcleos físicos habitualmente. En paralelo, la GPU opera en un rango del 90-96%, lo que confirma que el sistema desplaza la carga pesada a los núcleos CUDA y Tensor, optimizando el hardware especializado sin saturar el sistema principal

Por otro lado, al delegar la inferencia pesada (YOLO + MiDaS + FSRCNN) al Backend, el cliente (Frontend en Unity) ha podido ejecutarse en el equipo secundario (que no contaba con gráfica dedicada) manteniendo tasas estables por encima de los 100 FPS y en la Tablet con tasas estables de 60 FPS sostenidos.

Esto valida la capacidad del sistema para desplegar la visualización volumétrica en una gama muy amplia de dispositivos de consumo, sin obligar al usuario final a disponer de hardware de alto rendimiento.

4.5.3 Comparación con el estado del arte

Tras analizar los resultados obtenidos, es posible situar la solución propuesta dentro del ecosistema actual, comparándola con las soluciones tecnológicas

analizadas en el marco actual de este trabajo. El principal factor diferenciador radica en la democratización tecnológica, lograda al sustituir las infraestructuras de alto coste por algoritmos de inferencia monocular.

A diferencia de sistemas como Microsoft Holoportation o Google Beam que requieren de múltiples sensores RGB-D, cámaras de alta velocidad o pantallas holográficas, el sistema desarrollado funciona íntegramente con una cámara web estándar y hardware de consumo. Mientras que Google Beam tiene un coste de despliegue cercano a los 25.000 dólares, esta solución permite una experiencia volumétrica con una inversión de hardware prácticamente nula para el usuario que ya disponga de una GPU de gama media. El sistema ha demostrado una tasa de salida de 44.1 FPS, lo que supera significativamente la fluidez de muchas soluciones académicas que, aunque precisas, no logran alcanzar el tiempo real en dispositivos domésticos. Frente a Holoportation, el sistema planeado no requiere de una red interna de 10 Gbps, dado que es capaz, en su estado actual, de dar tasas de imágenes por segundo decentes (17,1 FPS) incluso a través de una conexión Wifi 5GHz. Frente a técnicas como NeRF que ofrecen un mayor grado de realismo a costa de un mayor tiempo de inferencia, la combinación de MiDaS y FSRCNN es capaz de generar una respuesta inmediata (< 15 ms en 720p), esencial para la interacción humana natural.

Otras alternativas como Codec Avatar de Meta requieren de un proceso previo de escaneo por parte del usuario, requerimiento que no es necesario en la arquitectura de carácter “Plug and Play”.

A pesar de lo anterior, la solución planteada tiene retos que resolver, soluciones como Soapbox o Holoportation ofrecen una mayor fidelidad al capturar el volumen del usuario desde 360 grados. Sin embargo, mientras que soluciones como Soapbox o Holoportation priorizan la fidelidad geométrica total, este trabajo demuestra la viabilidad de un modelo centrado en la optimización de recursos. Los resultados confirman que es posible obtener una reconstrucción frontal coherente y en tiempo real, priorizando la accesibilidad del sistema sobre la complejidad de la infraestructura física.

4.5.4 Interpretación de la Percepción del Usuario

Los resultados obtenidos durante la evaluación subjetiva (Tabla 4.10) arrojan una valoración global altamente positiva del sistema, cabe destacar el Ítem L6 (Sensación de presencia espacial), que tuvo un consenso completo en 5 (totalmente de acuerdo) indicando un consenso unánime en la capacidad inmersiva del prototipo.

Asimismo, la calidad de reconstrucción (L1) con una valoración de 4.71 y la Fluidez Percibida (L3) con 4.57, validan la viabilidad de la arquitectura propuesta,

demostrando que las optimizaciones de software ayudaron a tener una experiencia visual estable.

Por otro lado, la valoración más moderada recayó en la latencia Percibida (L4) que obtuvo una puntuación más moderada (3.86) y la mayor dispersión (0.69). Este resultado es consistente con las métricas objetivas y refleja el compromiso técnico de priorizar la calidad visual y la profundidad sobre la latencia ultra baja, siendo este uno de los principales vectores de mejora para el trabajo futuro.

4.5.5 Viabilidad y Cumplimiento de Hipótesis

El análisis conjunto de las métricas técnicas y de la percepción subjetiva de los usuarios, permiten validar de manera concluyente la hipótesis inicial de este trabajo, esto es, la viabilidad de desarrollar un sistema de reconstrucción volumétrica funcional en tiempo real, que no requiera de hardware especializado. Desde un punto de vista técnico, el sistema no solo supera en un 200% el umbral mínimo de fluidez, alcanzando 44.1 FPS de medias, sino que lo hace manteniendo un uso bajo de los recursos hardware (a excepción de la GPU). La arquitectura modular que deriva el consumo de recursos hacia el Backend, ha sido la clave que permite que dispositivos sin gráfica dedicada, como tablets o portátiles de oficina puedan participar en el visionado de contenido volumétrico en tiempo real. Se confirma, por tanto, que la solución es funcional en hardware de diverso formato y capacidad técnica, y por ende, permite democratizar una tecnología que hasta ahora estaba limitada a infraestructuras de alto coste.

5. Conclusiones y Trabajo Futuro

Tras completar la experimentación y haber analizado los resultados técnicos en el capítulo anterior, el presente trabajo llega a su etapa final de síntesis y reflexión crítica acerca de todo el trabajo realizado. Este capítulo tiene como propósito consolidar los hallazgos obtenidos durante el proceso de análisis, desarrollo y validación del sistema de reconstrucción 3D monocular propuesto.

El desarrollo de este proyecto nació con la necesidad de democratizar el acceso a la comunicación volumétrica, un sistema que permite interacciones más cercanas y reales que las videollamadas 2D habituales, para ello, era necesario encontrar una alternativa que permitiera sustituir la dependencia de hardware especializado y costoso, por algoritmos avanzados de aprendizaje profundo que fueran capaces de operar en tiempo real sobre dispositivos de consumo. A través de las siguientes secciones se realizará un balance sobre la adecuación final del proyecto a los objetivos iniciales, evaluando si la integración de tecnologías como MiDaS, YOLO o el refinamiento mediante FSRCNN han logrado alcanzar el equilibrio deseado entre eficiencia computacional y fidelidad visual.

Asimismo, se expondrán las limitaciones encontradas durante el desarrollo, lo que servirá como base para proponer nuevas líneas de investigación que permitan profundizar más allá de este trabajo hacia una solución de videollamada inmersiva plenamente escalable y comercial.

5.1 Conclusiones y Resumen del Trabajo

Este trabajo final de máster ha cumplido con el diseño, implementación y evaluación de un sistema de reconstrucción 3D monocular optimizado para su uso en entornos de videollamadas volumétricas en tiempo real. El proyecto inició con el objetivo de democratizar el acceso a las comunicaciones inmersivas, eliminando

la necesidad de hardware especializado de profundidad, como sensores LiDAR o cámaras estereoscópicas, sustituyéndolos por algoritmos de visión por computador y aprendizaje profundo.

Tras el análisis de los resultados expuestos en el capítulo anterior, se han llegado a las siguientes conclusiones:

En primer lugar, se ha validado la hipótesis inicial, esto es, que la reconstrucción volumétrica en tiempo real es viable para su uso en hardware de consumo a través de una arquitectura de software optimizada. El sistema no solo alcanzó la tasa de fluidez objetiva, sino que la superó llegando a alcanzar un rendimiento promedio de 44.1 FPS en resolución de 720P. Un hallazgo clave en este sentido, radica en que los modelos de IA, optimizados en precisión media (FP16), operaron por encima de los 65 FPS, demostrando así, que el aprendizaje profundo ya no es el cuello de botella principal para la interactividad, recayendo dicha limitación en otros factores como los procesos de compresión y transporte de los datos.

En segundo lugar, se ha demostrado que el sistema en dos bloques (Backend en Python y Frontend en Unity) ha demostrado ser una estrategia altamente escalable. Combinar YOLO y MiDaS junto con algoritmos de suavizado temporal, así como con un reescalador inteligente basado en FSRCNN con fine-tuning, ha permitido mitigar o reducir eficazmente defectos característicos de la estimación monocular, como son la pérdida de detalle fino, o el parpadeo frame a frame (flickering). Por otro lado, delegar la visualización a Unity mediante un sistema de partículas, permitió alcanzar unas cotas de fluidez de más de 100 FPS en portátiles sin gráfica dedicada, y de 60 FPS en tablets Android, abriendo así la puerta al streaming volumétrico en tiempo real para dispositivos de bajos recursos.

Más allá de esta conclusión general, a partir del trabajo llevado a cabo en los capítulos tercero y cuarto, se pueden extraer conclusiones adicionales. La primera de ellas radica en que se ha podido alcanzar una tasa de fluidez aceptable (15-20 FPS) de manera holgada. El sistema ha demostrado una capacidad de procesamiento, incluso con una salida de imagen en alta resolución a 720P una capacidad de procesamiento de 44.1 FPS en el equipo de pruebas principal. Estando esta tasa limitada por la compresión de imagen y no por los modelos de IA (que se esperaba inicialmente que fueran el principal cuello de botella) que fueron capaces de operar con un rendimiento medio por encima de los 65 FPS, lo que confirma que los algoritmos de aprendizaje profundo cuando se ejecutan en media precisión (FP16) y con las optimizaciones adecuadas, ya no representan un impedimento para la interactividad inmediata.

Otra de las conclusiones técnicas alcanzadas radica en que la comunicación volumétrica ya no requiere de equipos de alta gama. El consumo de recursos registrados, con apenas 0.41 GB de memoria VRAM reservada y 1.72 GB constantes

de memoria RAM usada en el servidor, valida la capacidad del sistema para ejecutarse en tarjetas gráficas con aceleración de inferencia por hardware de gama de entrada (Toda la gama RTX de Nvidia) sin saturar la memoria. Sumado a lo anterior, la arquitectura desacoplada, ha permitido que el cliente de visualización (Frontend) funcione con una fluidez extrema, alcanzando más de 100 FPS en portátiles sin gráfica dedicada e incluso 60 FPS estables en Tablets Android de gama media. Esto demuestra la alta escalabilidad del sistema y abre la puerta del Streaming Volumétrico unidireccional a un gran abanico de hardware de bajos recursos computacionales.

En tercer lugar, un aporte técnico relativo de este trabajo ha sido la integración de una red FSRCNN con fine-tuning. Aunque las métricas de similitud de píxeles (PSNR) peores que con el reescalado bicúbico, se logró una mejora sustancial en la calidad perceptual, LPIPS (0.18 frente a 0.39). Esto permite concluir que el modelo es capaz de “imaginar” y recuperar detalles finos perdidos durante la estimación monocular, ofreciendo así, una reconstrucción visualmente más rica que los métodos de reescalado tradicionales.

En cuarto lugar, el sistema ha destacado por su ligereza en el uso de recursos hardware, con un consumo de VRAM de apenas 0.41 GB en plena carga, lo que garantiza su funcionamiento en tarjetas gráficas de diversas gamas (dentro de la familia Nvidia RTX) incluyendo así a GPUs más modestas como las RTX 2060 o RTX 3050. No obstante, el estudio de la latencia reveló que la mayor parte del retraso total se halla en la infraestructura de transporte. Este hecho, posiciona a la migración hacia protocolos de flujo continuo, como WebRTC como el siguiente paso lógico para alcanzar un mayor grado de excelencia en la comunicación volumétrica inmersiva.

Finalmente, más allá de los datos técnicos, la validación con los usuarios confirma el impacto del proyecto. Con una puntuación media de 4.71 sobre 5 en la calidad de la reconstrucción y un consentimiento unánime sobre la sensación de presencia espacial, se concluye que el sistema logra su propósito fundamental: mejorar la sensación de cercanía y conexión humana en la comunicación digital, superando con creces las limitaciones de la videollamada tradicional.

5.2 Aportaciones

El presente trabajo realiza diversas aportaciones tanto al campo de la visión por computador como al de las comunicaciones inmersivas, enfocándose principalmente en los aspectos de eficiencia y accesibilidad.

5.2.1 Contribuciones Técnicas y Metodológicas

- Se ha diseñado e implementado un Pipeline de Reconstrucción Desacoplado, basado en una arquitectura que separa el proceso de inferencia (en Python) de la representación visual (Unity3D). Esta estructura permite aprovechar librerías de IA de última generación junto con motores gráficos modernos simultáneamente, facilitando la escalabilidad del sistema.
- Se hace uso de una red neuronal FSRCNN modificada mediante fine-tuning específico para el ámbito de aplicación (mapas de profundidad) obteniendo así un refinamiento de profundidad mediante el uso de Superresolución. Este proceso permite “imaginar” y recuperar detalles finos de la geometría, como lo son por ejemplo los rasgos faciales, que se suelen perder en los modelos de estimación monocular debido a su baja resolución de entrada.
- Se ha propuesto el uso de un algoritmo de estabilización temporal basado en medias móviles ponderadas exponenciales, lo que permite una mayor respuesta ante movimientos rápidos mientras que mantiene los costes de memoria muy por debajo de un algoritmo basado en medias lineales tradicionales.
- Se ha propuesto el uso de canal Alpha de una imagen PNG para encapsular la profundidad junto al color de la imagen de manera que pueda hacer una transmisión de datos 3D de manera ligera y robusta. Este método elimina los problemas de desincronización y la necesidad de protocolos adicionales de datos como JSON o XML que requerirían de un mayor coste computacional

5.2.2 Impacto y Valor Añadido

- Este trabajo demuestra que es posible generar experiencias inmersivas usando solo una cámara web estándar y hardware de consumo. Eliminando la necesidad de hardware especializado como el que requieren algunas de las soluciones comerciales referidas en el estado del arte (como Microsoft Holoportation o Google Beam), fomentando así la democratización de la comunicación volumétrica.
- Se ha integrado YOLO para aislar al sujeto del entorno, lo que permite centrar el cómputo de profundidad exclusivamente en la región de interés. Esto permite obtener un mayor nivel de detalle y reduce el ruido de escala que produce el entorno del sujeto.
- La metodología de evaluación y análisis crítico de tecnologías proporcionan una hoja de ruta útil para otros investigadores que busquen implementar sistemas de reconstrucción monocular en tiempo real.
- La relevancia de los datos obtenidos ha impulsado la difusión del trabajo a la comunidad científica. Se ha enviado una versión sintetizada al Simposio

CEA de Robótica, Bioingeniería, Visión por computador y Automática Marina en Bilbao 2026, y dada la solidez de la propuesta, se contempla así mismo la elaboración de un artículo científico extendido para su posterior publicación en una revista de impacto, contribuyendo así a la investigación en telepresencia y visión por computador.

5.3 Limitaciones del trabajo

A pesar de haber alcanzado los objetivos generales propuesto del sistema, el sistema planteado presenta ciertas limitaciones inherentes tanto a la tecnología actual, como al enfoque planteado. Estas restricciones definen el marco de operación del sistema y deben ser tenidas en cuenta para futuras iteraciones.

5.3.1 Falta de información en las zonas ocluidas

Al ser un sistema basado en una única cámara, el mapa de profundidad solo puede representar las áreas visibles por esta misma. Objetos ocultos o zonas como la espalda del propio usuario no son visibles por la cámara, por lo que no pueden ser a priori procesadas (aunque sí inferidas). Lo que limita la reconstrucción a una “carcasa” o Malla frontal en lugar de a un volumen completo 3D.

5.3.2 Dependencia del hardware en el Backend

El rendimiento general del Backend es sumamente dependiente a los recursos de hardware del sistema del servidor, específicamente requiere una GPU dedicada con soporte CUDA/Tensor Cores para poder funcionar óptimamente.

5.3.3 Estimación de la Profundidad

Tal y como se ha apuntado a lo largo del trabajo, el sistema estima o “predice” la profundidad, no la mide o calcula directamente como haría un sensor LiDAR. Esto conlleva a errores de escala absoluta y, aunque se hayan implementado correcciones por software, la profundidad estimada puede variar en cierta medida de la profundidad real.

5.3.4 Latencia Acumulada

El sistema es actualmente muy sensible al ancho de banda y la estabilidad de red, debido a que hasta un 66% de la latencia total se pierde en la transferencia de datos entre Backend y Frontend debido al uso de imágenes PNG completas por HTTP, lo cual es actualmente el mayor cuello de botella de rendimiento del sistema. Aunque el sistema es viable en redes domésticas, pudiendo funcionar adecuadamente en redes de 5Ghz, el uso de conexiones inestables (como Wifi 2.4G) o alejadas del router, pueden ser, actualmente un punto crítico de fallo.

5.3.5 Limitaciones de Iluminación

Aunque MiDaS ofrece una buena generalización, las condiciones de iluminación extremas como la falta de luz o los contraluces intensos, pueden afectar a la segmentación de YOLO y a la consistencia en los mapas de profundidad, generando diversos artefactos visuales en la nube de puntos resultante

5.3.6 Limitaciones del proceso de fine-tuning

El proceso de fine-tuning realizado a FSRCNN para mejorar el detalle fino se ha realizado a partir de un conjunto limitados (menos de 300 pares de imágenes entre reales y sintéticos), que, además, en el caso de los datos reales, han sido obtenidos por un sensor antiguo (Kinect V2) y con un mapa de profundidad de baja resolución (512x424). Esto provoca que el sistema presente una menor ganancia de detalle en ropas o rasgos faciales de los que podría llegar a tener con un dataset mayor o con sensores de profundidad más modernos o de mayor resolución (como sensores LiDAR)

5.4 Trabajo Futuro

El desarrollo de este sistema ha sentado las bases para una nueva línea de investigación en comunicaciones volumétricas de bajo coste. Para avanzar una solución completamente escalable y fotorrealista, se proponen las siguientes líneas de trabajo.

5.4.1 Uso de modelos generativos para la reconstrucción 360°

Para superar la limitación de visión de la cámara propuesta en el apartado anterior, se plantea el uso de redes neuronales generativas, para no solo reescalar la imagen y darle detalle fino, sino también, para poder “rellenar los huecos” dejados por el sistema actual, es decir, implementar una red neuronal que permita, por ejemplo renderizar la espalda del usuario, o la parte trasera de la cabeza a partir de ya sea, datos reales del usuario (por ejemplo cuando gira o se levanta) o bien datos inferidos a partir de la información presente. Obteniendo, de esta manera una vista completa 3D en 360 grados del usuario a partir de la información de una única cámara.

5.4.2 Migración a protocolos de baja latencia como WebRTC

La arquitectura actual, basada en el consumo de vídeo mediante RTMP y la transmisión de resultados mediante peticiones HTTP, fue seleccionada para desacoplar los microservicios y simplificar la fase de desarrollo y depuración del algoritmo de reconstrucción. Si bien este enfoque ha permitido validar con éxito la viabilidad técnica del proyecto, se ha identificado que el protocolo de transporte constituye actualmente el principal cuello de botella del sistema, pudiendo

representar hasta el 66% (o más en caso de conexiones WIFI) de la latencia total del sistema, introduciendo una latencia acumulada estructural (debido al buffering de RTMP) que aleja la experiencia de una interacción natural.

Es por ello, que la línea de trabajo futuro más prioritaria radicaría en la sustitución de la capa de transporte actual por WebRTC (Web Real-Time Communication). La adopción de este estándar permitiría transitar de un modelo de petición-respuesta a uno de flujo de datos continuo sobre UDP. Esta migración eliminaría la necesidad de servidores intermedios de retransmisión y evitaría la acumulación de buffers, permitiendo reducir la latencia de los actuales 3-4 segundos a rangos por debajo de un segundo, un requisito indispensable para alcanzar una comunicación volumétrica fluida y comercialmente viable.

5.4.3 Adopción de técnicas de representación visual más eficientes como 3D Gaussian Splatting

Otra mejora destacada radica en el uso de sistemas de representación 3D. A pesar de que la nube de puntos actual es bastante eficiente, dando buenos resultados, (por encima de 60 FPS) en equipos sin gráfica integrada, el uso de técnicas como Gaussian Splatting permitiría alcanzar un nivel de fotorrealismo superior, manteniendo la capacidad de renderizado en tiempo real de la GPU.

5.4.4 Optimización para dispositivos “Standalone”

Otra línea prioritaria radica en el despliegue del sistema completo dentro de visores de realidad virtual autónomos, como Meta Quest 3, dados los resultados actuales en la Tablet, parece bastante viable portar el Frontend a este tipo de dispositivos, estando, por tanto, el reto, en portar el Backend (inferencia en Python) a la arquitectura y capacidades técnicas del visor. Esto requeriría optimizar los modelos de MiDaS y YOLO mediante técnicas de cuantización para su ejecución en procesadores móviles o NPUs específicas.

5.4.5 Aceleración mediante el uso de Hardware Emergente

Sería interesante evaluar el impacto de nuevas tecnologías como las Edge TPU o la API DirectML, que permiten desplazar carga computacional de la GPU hacia núcleos especializados en inteligencia artificial pueden afectar a los resultados del proyecto.

5.4.6 Retroalimentación de superresolución a partir de información RGB

Otra línea de investigación que podría llevar a una mejora en la consistencia temporal del sistema vendría de poder retroalimentar el modelo de superresolución con información de frames pasados. Actualmente el algoritmo trabaja a nivel de imágenes independientes, y al ser un modelo de naturaleza probabilística,

pequeños cambios en las imágenes de entrada pueden dar lugar a variaciones en los datos de alta frecuencia de la imagen final, generando así un efecto de ruido o de inconsistencia del vídeo. Facilitarle información de las imágenes anteriores al programa podría ayudar a que el detalle fino de un frame a otro se mantenga con cierta consistencia, aumentando así la inmersividad del todo el sistema.

Bibliografía

- Agdestein, I. (27 de 02 de 2025). *Arquitecturas de Modelos de IA: CNNs, RNNs y Transformadores*. <https://focalx.ai/>. <https://focalx.ai/es/inteligencia-artificial-es/arquitecturas-de-modelos-de-ia-cnns-rnns-y-transformadores/>
- Baek, Y., Lee, B., Han, D., Yun, S., & Lee, H. (2019). Character Region Awareness for Text Detection. <https://arxiv.org/abs/1904.01941>
- Da Mata, J., Oliveira, J., & Melo, C. (2024). Super-resolution with perceptual quality for improved live streaming delivery on edge computing. *Computer Networks*, 248, pág. 110463. <https://doi.org/10.1016/j.comnet.2024.110463>
- Deng, H., Zhang, Q., Jin, H., & Chang-Hun, K. (2023). Real-Time Interaction for 3D Pixel Human in Virtual Environment. *Applied Sciences*, 13(2), 966. <https://doi.org/10.3390/app13020966>
- Departamento de Matemática Aplicada, Universidad Politecnica De Madrid. (2021). *Redes Neuronales Convoluciones*. Introducción al Aprendizaje Automático. https://dcain.etsin.upm.es/~carlos/bookAA/05.7_RRNN_Convoluciones_CIFAR_10_INFORMATIVO.html
- Dong, C., & Loy, C. H. (2014). Learning a Deep Convolutional Network for Image Super-Resolution. En *Computer Vision – ECCV 2014*. https://doi.org/10.1007/978-3-319-10593-2_13
- Dong, C., Change, C., & Tang, X. (08 de 2016). Accelerating the Super-Resolution Convolutional Neural Network. <https://doi.org/10.48550/arXiv.1608.00367>
- Eigen, D., Puhrsch, C., & Fergus, R. (2014). *Depth Map Prediction from a Single Image using a Multi-Scale Deep Network*. <https://arxiv.org/abs/1406.2283>

- Fernández, R. (25 de 09 de 2024). *Statista*. Statista. <https://es.statista.com/estadisticas/562838/porcentaje-de-personas-que-hizo-llamadas-o-videollamadas-online-por-pais-en-la-union-europea/>
- Girshick, R. (2015). Fast R-CNN. <https://arxiv.org/abs/1504.08083>
- Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. <https://arxiv.org/abs/1311.2524>
- Godard, C., Aodha, M., & Brostow, G. (2017). *Unsupervised Monocular Depth Estimation with Left-Right Consistency*. <https://arxiv.org/abs/1609.03677>
- Gonzalez, R., & Woods, R. (2018). *Digital Image Processing* (4 ed.). New York: Pearson.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (06 de 2014). Generative Adversarial Networks. *Advances in Neural Information Processing Systems*, 3. <https://doi.org/10.1145/3422622>
- Gutierrez, L., & Alarcón, C. (2020). Extensión de la arquitectura Docker para el despliegue automático de contenedores. *Ingeniare*, 11-26. <https://doi.org/10.18041/1909-2458/ingeniare.29.7432>
- IBM. (25 de 03 de 2025). ¿Qué es un modelo de transformador?. <https://www.ibm.com/es-es/think/topics/transformer-model>
- Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., & Shi, W. (2017). Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. <https://arxiv.org/abs/1609.04802>
- Litjens, G., Kooi, T., Bejnordi, B., Setio, A., Ciompi, F., Ghafoorian, M., J. v. d., van Ginneken, B., & Sánchez, C. (2017). A survey on deep learning in medical image analysis. *Med Image Anal.*, 42, 60-88. <https://doi.org/10.1016/j.media.2017.07.005>
- Liu, W. A., Erhan, D., Szegedy, C., Reed, S., Fu, C., & Derg, A. (2016). SSD: Single Shot MultiBox Detector. En *Computer Vision – ECCV 2016*. https://doi.org/10.1007/978-3-319-46448-0_2
- Longuet-Higgins, H. (1981). A computer algorithm for reconstructing a scene from two projections. *Nature*(293), 133–135. <https://doi.org/10.1038/293133a0>

- Looking Glass. (19 de 01 de 2023). *Depth Maps: How Software Encodes 3D Space*. <https://blog.lookingglassfactory.com/depth-maps-how-software-encodes-3d-space/>
- Ma, H., Simon, T., Saragih, J., Wang, D., Li, Y., De La Torre, F., & Sheikh, Y. (2021). *Pixel Codec Avatars*. <https://arxiv.org/abs/2104.04638>
- Microsoft Corporation. (05 de 07 de 2024). *Búferes de profundidad*. <https://learn.microsoft.com/es-es/windows/uwp/graphics-concepts/depth-buffers>
- Mildenhall, B., Srinivasan, P., Tancik, Y., Barron, J., & Ramamoorthi, R. N. (2020). *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. <https://arxiv.org/abs/2003.08934>
- Min, N., Jianjun, L., Runmin, C., Zheng, K., Peng, B., & Xiaoting, F. (11 de 2017). Color-Guided Depth Map Super Resolution Using Convolutional Neural Network. *IEEE Access, PP*. <https://doi.org/10.1109/ACCESS.2017.2773141>
- Müller, T., Evans, A., Schied, C., & Keller, A. (07 de 2022). Instant neural graphics primitives with a multiresolution hash encoding. *Association for Computing Machinery, 41*. <https://doi.org/10.1145/3528223.3530127>
- Nagy, A., Spyridis, Y., Mills, G., & Argyriou, V. (10 de 2024). User Experience Evaluation of AR Assisted Industrial Maintenance and Support Applications. <https://doi.org/10.48550/arXiv.2410.17348>
- Nartker, A. (20 de 05 de 2025). *Google Beam: Our AI-first 3D video communication platform*. <https://blog.google/technology/research/project-starline-google-beam-update/>
- Newcombe, R., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A., Kohi, P., & Shotton, J. (2011). KinectFusion: Real-time dense surface mapping and tracking. *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 127-136. <https://doi.org/10.1109/ISMAR.2011.6092378>
- Nielsen, J., & Landauer, T. (1993). A mathematical model of the finding of usability problems. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems* (págs. 206–213). New York, NY, USA: Association for Computing Machinery. <https://doi.org/10.1145/169059.169166>
- Ortiz, M. (2025). Redes neuronales artificiales. *Ingenio y Conciencia Boletín Científico de la Escuela Superior Ciudad Sahagún*, 12(23), 38-44.
- Orts-Escalano, S. a., Fanello, S., Chang, W., Kowdle, A., & Degtyarev, Y. (2016). Holoportation: Virtual 3D Teleportation in Real-time. *Proceedings of the 29th*

- Annual Symposium on User Interface Software and Technology*, 741–754.
<https://doi.org/10.1145/2984511.2984517>
- Pajares, G., & De la Cruz, J. (2007). *Visión por computador, Imágenes digitales y aplicaciones* (2 ed.). RA-MA.
- Pallets. (s.f.). *Flask Documentation*. <https://flask.palletsprojects.com/en/stable/>
- Quantum News. (21 de 09 de 2024). Quantum Zeitgeist.
<https://quantumzeitgeist.com/python-and-artificial-intelligence/>
- Ranftl, R., Lasinger, K., Hafner, D., Schindler, K., & Koltun, V. (2020). *Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer*. arXiv (preprint). <https://doi.org/10.48550/arXiv.1907.01341>
- Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement.
<https://arxiv.org/abs/1804.02767>
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (08 de 06 de 2016). You Only Look Once: Unified, Real-Time Object Detection.
<https://doi.org/10.48550/arXiv.1506.02640>
- Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.
<https://arxiv.org/abs/1506.01497>
- Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentationv. <https://arxiv.org/abs/1505.04597>
- Rovinelli, M. (2021). *Realtime monocular depth estimation on mobile phones*. Trabajo Fin De Master, Università di Bologna, Bologna.
https://amslaurea.unibo.it/id/eprint/24159/1/thesis_marco%20%288%29.pdf
- Sag, A. (05 de 12 de 2019). *Qualcomm Announces XR2 5G Aimed At Broader VR/AR Features*. <https://www.uploadvr.com/xr2-qualcomm/>
- Shi, W., Caballero, J., Huszar, F., Totz, J., Aitken, A., Bishop, R., Rueckert, D., & Wang, Z. (2016). Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network.
<https://arxiv.org/abs/1609.05158>
- Shubert, T., Friedmann, F., & Regenbrech, H. (2001). The experience of presence: Factor analytic insights. *Presence: Teleoperators and Virtual Environments*, 10(3), 266-281.
- soapbox. (s.f.). soapbox. <https://www.soapbox.us/>

- Szeliski, R. (2010). *Computer Vision: Algorithms and Applications* (1 ed.). Springer London. <https://doi.org/10.1007/978-1-84882-935-0>
- Thoughtworks, Inc. (2024). *Technology Radar, Volume 31: An opinionated guide to today's technology landscape*.
- TOPS Marketing. (13 de 06 de 2025). *Depth Maps and Point Clouds: A Comparative Analysis*. <https://www.takeoffpros.com/blog/depth-map-sequences-vs-point-clouds/>
- Uijlings, J., Van de Sande, K., Gevers, T., & Smeulders, A. (2013). Selective Search for Object Recognition. *International Journal of Computer Vision*, 104, 154-171. <https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013>
- Unity Technologies. (s.f.). *Unity Documentation*. <https://docs.unity3d.com/Manual/index.html>
- Van Rossum, G. (1995). *Python tutorial*. Department of Computer Science [CS]. CWI.
- Varma, A., Chawla, H., B, Z., & Elahe, A. (7 de 2 de 2022). *Transformers in Self-Supervised Monocular Depth Estimation with Unknown Camera Intrinsics*. <https://doi.org/10.5220/0010884000003124>
- Vasconcelos-Raposo, J., Bessa, M., Melo, M., Barbosa, L., Rodrigues, R., Teixeira, C., Cabral, L., & Sousa, A. (2016). Adaptation and Validation of the Igroup Presence Questionnaire (IPQ) in a Portuguese Sample. *Presence: Teleoperators and Virtual Environments*, 25(3), 191-203. https://doi.org/10.1162/PRES_a_00261
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., & Gomez, A. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 5998-6008. <https://doi.org/10.48550/arXiv.1706.03762>
- Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. CVPR 2001, 1.
- VorpX. (s.f.). *VorpX Features*. <https://www.vorpx.com/features/>
- Webb, M. (12 de 2021). Zoom Fatigue and How to Prevent It. *J Registry Manag.*, 48(4), 181-182. <https://pmc.ncbi.nlm.nih.gov/articles/PMC10198405/>
- Woodham, R. (1980). Photometric method for determining surface orientation from multiple images. *Optical engineering*, 139-144. <https://www.cs.ubc.ca/~woodham/papers/Woodham80c.pdf>

- Yu, C., Wang, J., Peng, C., Gao, C., Yu, G., & Sang, N. (2018). BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation. <https://arxiv.org/abs/1808.00897>
- Zhang, Y., & Yin, X. (2021). Design of remote control system for smart home based on Unity and the Internet of Thing. *Journal of Physics: Conference Series*, 1744(2), Articulo 022099. <https://doi.org/10.1088/1742-6596/1744/2/022099>
- Zhong-Qiu, Z., Peng, Z. S.-t., & W, X. (2019). Object Detection with Deep Learning: A Reviewv. <https://arxiv.org/abs/1807.05511>
- Zhou, X., Yao, C., Wen, H., Y, W., Zhou, S., He, W., & Liang, J. (2017). EAST: An Efficient and Accurate Scene Text Detector. <https://arxiv.org/abs/1704.03155>
- Zisserman, H. (2004). *Multiple View Geometry in Computer Vision* (2 ed.). Cambridge University Press.
- Zollhöfer, M., Thies, J., Garrido, P., Bradley, D., Beeler, T., Perez, P., & Stamminger, M. (2018). State of the Art on Monocular 3D Face Reconstruction, Tracking, and Applications. *Computer Graphics Forum*, 37(2), 523-550. <https://doi.org/10.1111/cgf.13382>

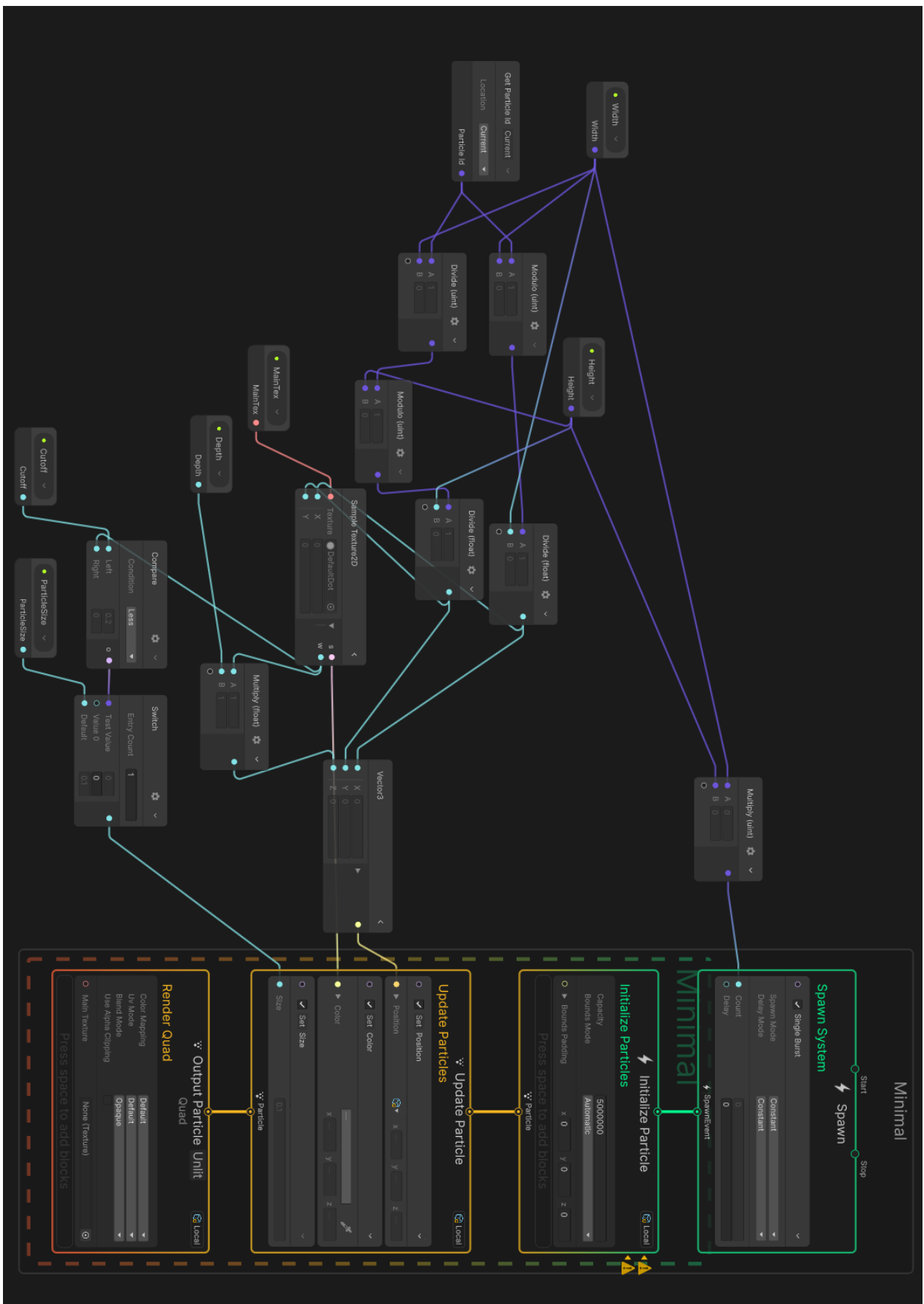
Siglas y Acrónimos

VR	Realidad Virtual
IA	Inteligencia Artificial
ML	Machine Learning
3D	En Tres Dimensiones / Tridimensional
CNN	Red Neuronal Convolucional (Convolutional Neuronal Network)
SFM	Estructura desde el movimiento (Structure From Motion)
API	Interfaz de Programación de Aplicaciones (Application Programming Interface)
DLSS	Supermuestreo por Aprendizaje Profundo (Deep Learning Super Sampling)
FSRCNN	Red neuronal convolucional de súper resolución rápida (Fast Super-Resolution Convolutional Neural Network)
MIDAS	Estimación de profundidad Monocular mediante Redes Multiescala
NERF	Campos de Radiancia Neuronal (Neural Radiance Fields)
YOLO	Solo Miras Una Vez (You Only Look Once)
NPU	Unidad de procesamiento Neuronal (Neural Processing Unit)
TPU	Unidad de procesamiento de Tensores (Tensor Processing Unit)
FPS	Frames Per Second / Fotogramas por Segundo
PNG	Gráficos de Red Portátiles (Portable Network Graphics)
RGBA	Modelo de color Rojo, Verde, Azul y Alfa (transparencia)

Notas Adicionales

- I. **Canal Alpha:** Se utiliza en este proyecto no para transparencia, sino para codificar y transmitir la información de profundidad sincronizada con el color
- II. **Pipeline Desacoplado:** Se refiere a la separación lógica de procesamiento (Backend en Python) de la representación visual (Frontend en Unity)

Anexo I. Código Fuente y diagramas de componentes críticos



VFX Graph personalizado del Frontend (Unity)

```
def forward(self, pred, target, sobel_weight=2, laplacian_weight=0.2):
    loss_pixel = self.l1(pred, target)

    # Bordes Sobel
    pred_sx = torch.nn.functional.conv2d(pred, self.k_sobel_x, padding=1)
    pred_sy = torch.nn.functional.conv2d(pred, self.k_sobel_y, padding=1)
    target_sx = torch.nn.functional.conv2d(target, self.k_sobel_x, padding=1)
    target_sy = torch.nn.functional.conv2d(target, self.k_sobel_y, padding=1)
    loss_sobel = self.l1(pred_sx, target_sx) + self.l1(pred_sy, target_sy)

    # Bordes Laplaciano
    pred_lap = torch.nn.functional.conv2d(pred, self.k_laplacian, padding=1)
    target_lap = torch.nn.functional.conv2d(target, self.k_laplacian, padding=1)
    loss_laplacian = self.l1(pred_lap, target_lap)

    return loss_pixel + sobel_weight * loss_sobel + laplacian_weight * loss_laplacian
```

Entrenamiento de FSRCNN. Función de Pérdida.

```
FROM pytorch/pytorch:2.1.0-cuda11.8-cudnn8-runtime

ENV DEBIAN_FRONTEND=noninteractive
ENV TZ=Etc/UTC

# Dependencias del sistema
RUN apt-get update && apt-get install -y \
    build-essential \
    libgl1-mesa-glx \
    libgl1-mesa-dev \
    libsm6 \
    libxext6 \
    libxrender1 \
    libgl1-mesa-dev \
    v4l-utils \
    wget \
    unzip \
    libffi-dev \
    libvips-dev \
    && rm -rf /var/lib/apt/lists/*

# Resolución de conflictos
ENV LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libffi.so.7:\
/usr/lib/x86_64-linux-gnu/libobject-2.0.so.0

# Variables de entorno
ENV YOLO_CONFIG_DIR=/tmp/.config/Ultralytics
RUN mkdir -p /tmp/.config/Ultralytics && \
    chmod -R 777 /tmp/.config/Ultralytics

WORKDIR /app

# Instalación de librerías
RUN pip install --no-cache-dir pyvips mkl-service

# Requisitos
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Limpieza y más arreglos de compatibilidad
RUN pip uninstall -y opencv-python opencv-python-headless \
    opencv-contrib-python mediapipe && \
    pip install --no-cache-dir \
    opencv-python-headless==4.8.0.74 \
    mediapipe

EXPOSE 5000

# Comando por defecto
CMD ["python", "main.py"]
```

```
Run All Services
services:
  Run Service
  app:
    build: .
    working_dir: /app
    volumes:
      - ./app
    ports:
      - "5000:5000"
    environment:
      PYTHONUNBUFFERED: "1"
      DISPLAY: "host.docker.internal:0.0"
      OPENCV_VIDEOIO_PRIORITY_MSMF: "1"
    extra_hosts:
      - "host.docker.internal:host-gateway"
    runtime: nvidia
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]
      stdin_open: true
      tty: true

  Run Service
  nginx-rtmp:
    image: tiangolo/nginx-rtmp
    container_name: nginx-rtmp
    ports:
      - "1935:1935"
      - "8080:80"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
    restart: unless-stopped
```

Configuración de Despliegue. Dockerfile y docker-compose.yml

Cuestionario de Calidad Percibida por el Usuario

Evaluación subjetiva de la calidad visual y la experiencia tras el uso del sistema de telepresencia 3D, utilizando una escala Likert de 5 puntos.

1. Evalúe las siguientes dimensiones de la experiencia. (1 = Muy en desacuerdo, 5 = Totalmente de acuerdo)

Marca solo un óvalo por fila.

	1 - Muy en desacuerdo	2	3	4	5 - Totalmente de acuerdo
El modelo 3D representa fielmente la figura y los rasgos faciales	☐	☐	☐	☐	☐
La imagen es sólida y no presenta parpadeos o saltos molestos en la profundidad	☐	☐	☐	☐	☐
El movimiento del avatar 3D se percibe suave y natural durante la interacción	☐	☐	☐	☐	☐
El retraso entre el movimiento real y el visualizado es suficientemente bajo o aceptable para una videollamada	☐	☐	☐	☐	☐

2. Evalúe su sensación de presencia. (1 = Muy en desacuerdo, 5 = Totalmente de acuerdo)

Marca solo un óvalo por fila.

	1 - Muy en desacuerdo	2	3	4	5 - Totalmente de acuerdo
En el entorno visualizado, tuve la sensación de que la otra persona estaba 'allí' conmigo.	☐	☐	☐	☐	☐
Sentí que la imagen reconstruida tenía volumen y no era solo una foto plana	☐	☐	☐	☐	☐
La persona virtual me pareció más real en comparación con una videollamada tradicional.	☐	☐	☐	☐	☐

3. En su opinión, ¿qué es lo que más destacaría del sistema?

Anexo III. Tabla de Respuestas de Usuarios

L1	L2	L3	L4	L5	L6	L7	L8
5	3	4	3	5	5	5	hay algo de retraso entre el movimiento real y el que se ve en el programa, pero a parte de eso, es bastante impactante poder ver a alguien en 3D como si fuera un holograma sacado de Star Wars
5	4	4	4	4	5	5	El sistema trae las comunicaciones futuristas del cine a la vida real en VR
4	5	5	4	5	5	5	
5	4	5	4	5	5	5	Hay mayor sensación de cercanía de la persona. Llama la atención los gestos, se percibe más próxima la persona
5	5	5	5	4	5	4	El uso de este sistema me resultó cómodo y agradable. La calidad visual fue buena y el efecto tridimensional me hizo sentir la experiencia más realista y cercana.
5	4	4	3	5	5	5	Increíble la sensación de volumen, aunque la resolución no es muy alta y tiene algo de retraso, se compensa con la sensación de profundidad
4	4	5	4	4	5	4	Se pierden algunos detalles respecto a ver a la persona real, pero el efecto 3D está muy bien conseguido. Se siente muy natural interactuar así.

Anexo IV. Hoja de especificaciones de Prueba y Entorno experimental

1. Infraestructura de Hardware

Nombre Dispositivo	CPU / SOC	GPU	RAM	Rol
Equipo Principal (PC Clónico)	AMD Ryzen 9 5900X	NVIDIA GeForce RTX 3060 Ti	32 GB DDR4 a 3200 MHz (CL16)	Backend y Validación High-End
Equipo Secundario (Lenovo yoga Slim 7i)	Intel Core i5-11300H	Intel Iris Xe Graphics 80 EUs (Integrada)	16 GB LPDDR4X a 4300 MHz	Frontend Portátil
Tablet (Lenovo Tab P11 Pro (2ª Gen))	MediaTek Kompanio 1300T	---	8 GB	Frontend Android
Meta Quest 3	Snapdragon XR 2 Gen 2	Adreno 740	8 GB	Frontend XR

2. Entorno de Software y Dependencias

Sistema	Tipo	Entorno y Dependencias
Backend	Contenedores	Docker Desktop (WSL2 en Windows) - Imagen Base: Python 3.9 (app) y nginx-rtmp (gestión vídeo)
	Lenguaje y librerías	Python - Flask - PyTorch 2.1.0 (+cuda 11.8) - Ultralytics YOLO (yolov8n.pt) - OpenCV-Python - Intel MiDaS (MiDaS_small) - Timm - Numpy - Request
Frontend	Motor Gráfico	Unity 3D 6000.3.0f1
	Renderizado	VFX Graph (Visual Effect Graph) con Shader personalizado en GPU
	API de Red	UnityWebRequest (para consumo HTTP de texturas)

3. Escenarios de red

Conexión Backend	Conexión Frontend	Observaciones	Latencia Promedio (480p)
Ethernet 1Gbps	Ethernet 1Gbps	Escenario Ideal	43.7 ms
Ethernet 1Gbps	WIFI 5 GHz	Cliente WIFI a 2 metros, visión directa al rúter	58.6 ms
Ethernet 1Gbps	WIFI 2.4 GHz	Cliente WIFI a 2 metros, visión directa al rúter	89.2 ms

4. Especificaciones del Dataset (Fine-Tuning)

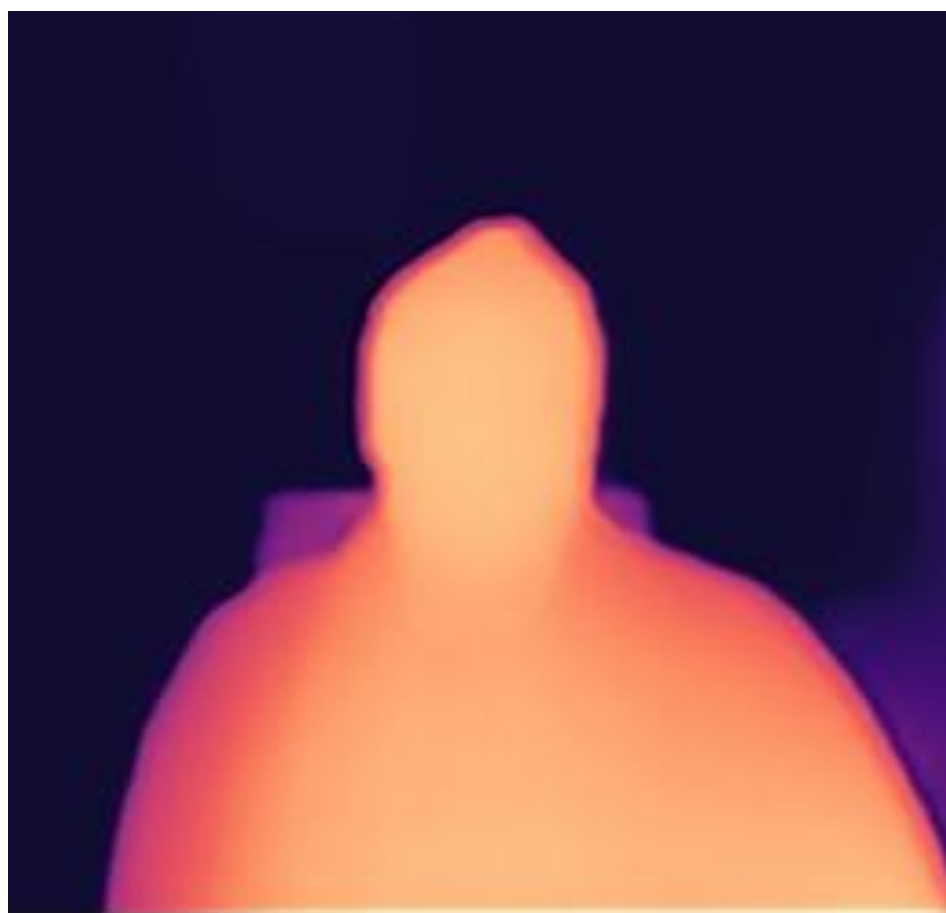
- **Total de pares de imágenes:** 248 pares (RGB + Mapa profundidad)
- **Pares Sintéticos:** 66 pares generados mediante Blender (z-buffer + $0.001 \cdot \text{normal}$)
- **Pares Reales:** 182 pares capturados con Kinect V2
- **Estrategia de entrenamiento:** 3 Fases (Sintética $\rightarrow$ Real $\rightarrow$ Mixta) con data augmentation



Par de imágenes sintéticas de referencia



Par de imágenes reales de referencia



**TELEPRESENCIA MONOCULAR:
DEL 2D AL 3D EN TIEMPO
REAL MEDIANTE MODELOS DE
IA SIN SENSORES DEDICADOS**

Guía de
usuario

PRERREQUISITOS



Backend (Servidor de Procesamiento)

Hardware: PC con GPU NVIDIA RTX

Software: Docker Desktop (WSL2)

Requisitos Adicionales: OBS Studio (o cualquier software que permita inyectar video a un servidor RTMP)



Frontend (Cliente Visualización)

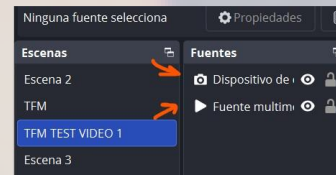
Hardware: PC / Android

Red: Ethernet (Servidor) + Ethernet / Wifi 5GHz (Cliente). Ambos en la misma red Local

PASO 1. PREPARAR LA TRANSMISIÓN RTMP

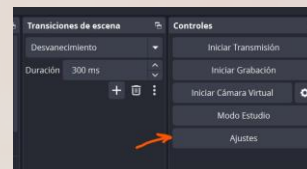
El uso de OBS es opcional para este paso, cualquier sistema capaz de retransmitir video al servidor RTMP debería funcionar. A continuación se explica como hacerlo con OBS.

Una vez instalado OBS Studio, el primer paso es configurar la escena, (lo que se va a transmitir al servidor RTMP)



En el apartado "Fuentes" se debe pulsar sobre el botón de "+" para añadir una nueva fuente, hay varias opciones pero para este caso interesan dos, "Fuente Multimedia" que permite poner fichero de video (en la carpeta del Backend hay un video de ejemplo llamado TestVideo.mp4), o Dispositivo de captura de video que permite capturar la imagen desde una Webcam o de una captura de video.

Una vez elegida la fuente de captura y ajustada al recuadro de pantalla el siguiente paso es configurar los parámetros que permitan transmitir al servidor RTMP. Para ello hay que hacer 'click' en el botón de ajustes del apartado "Controles"

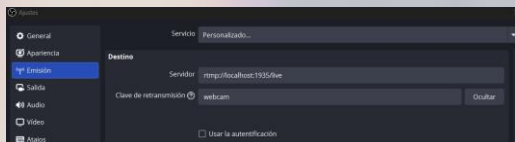


PASO 1. PREPARAR LA TRANSMISIÓN RTMP

Una vez en Ajustes hay que tocar dos apartados, en primer lugar "Video"



El campo Resolución de salida permite ajustar la resolución a la que se enviarán los datos al Backend, lo recomendable aquí es ajustarlo a 480P o 720P (o superior) en función de la capacidad de procesamiento del ordenador del Backend.



La pestaña Emisión deberá configurarse tal y como se ve en la imagen.

NOTA: Si el ordenador en el que se ejecuta OBS fuera distinto del de Docker, deberá cambiarse la IP de "localhost" a la IP del servidor con Docker

PASO 2: DESPLIEGUE DE DOCKER

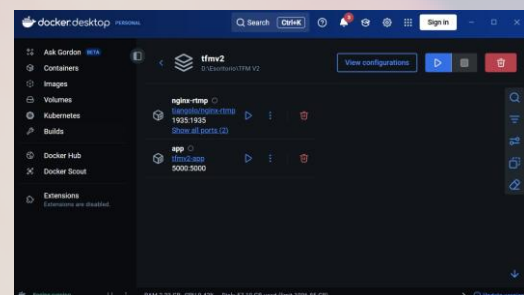
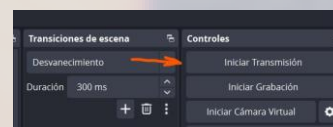
Si es la primera vez

1. Abrir la consola de Windows en la misma carpeta que el fichero Dockerfile
2. Ejecutar el comando "docker compose up" para levantar el contenedor
3. El sistema levantará y ejecutará el contenedor, que podrá verse desde la interfaz de Docker desktop

Si ya se ha levantado el contenedor anteriormente

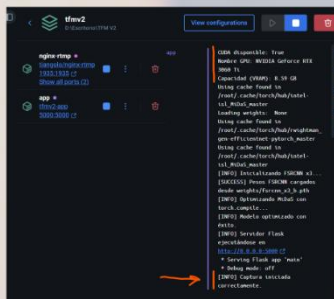
1. En Docker Desktop pulsar el botón de "Play" sobre el contenedor (Ver imagen)

Una vez levantado el contenedor, ya se puede transmitir la imagen al servidor RTMP desde OBS (si el sistema no puede abrir el flujo de datos mostrará un error, por lo que habrá que reiniciar el contenedor) para ello simplemente habrá que pulsar sobre el botón "Iniciar Transmisión" de OBS



PASO 3: EJECUCION DEL BACKEND

Una vez realizados los pasos anteriores, el Backend empezará a funcionar. El sistema mostrará al principio un mensaje con el soporte de CUDA y el dispositivo en el que se ejecutará el sistema (CPU o GPU). Si después del mensaje de "Servidor Flask ejecutándose ..." aparece un mensaje de "Captura Iniciada correctamente" (ver imagen), significa que el software está recibiendo correctamente la secuencia de imágenes de OBS. Si saliese un error, se deberá reiniciar el contenedor y e "iniciar transmisión" de nuevo desde OBS

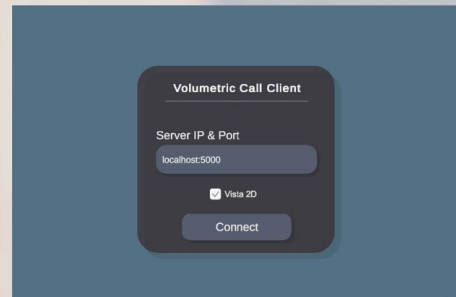


Una vez pasado este punto, se deberá esperar 1 o 2 minutos mientras el sistema compila los modelos (torch.compile) para adaptarlos al hardware de su equipo. Sabrá que este proceso ha terminado cuando vea que empiezan a aparecer mensajes de "Inferencia FPS...". A partir de ese momento ya podrá establecer la conexión desde el Frontend.

```
Inferencia FPS: 0.2 | Total: 6293.8ms
Inferencia FPS: 75.2 | Total: 13.3ms
Inferencia FPS: 74.6 | Total: 13.4ms
```

PASO 4: EJECUCION DEL FRONTEND

En la carpeta del Frontend encontrará dos carpetas, Windows (Que contiene el ejecutable para PC) y Android (Que contiene el .apk para Android). Si desea ejecutar el Frontend en Android deberá copiar el .apk a su dispositivo móvil y abrirlo para que se instale la aplicación. Si desea ejecutar el Frontend en Windows deberá copiar el contenido de la carpeta Windows a su ordenador y abrir el fichero .EXE de su interior



Una vez abierto el Frontend verá esta interfaz, en server IP & Port deberá colocar la dirección IP (si no sabe cual es, revise la ultima diapositiva del manual) del Ordenador del Backend (o dejarlo como esta si es el mismo ordenador) y colocar el puerto 5000.

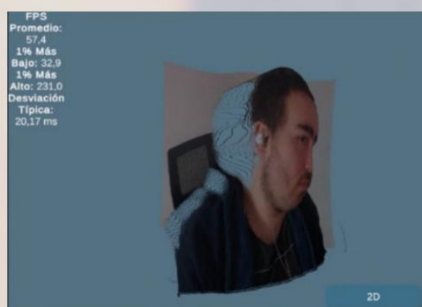
La opción Vista 2D habilita ver en la esquina superior derecha la imagen recibida del servidor RTMP sin procesar para poder usarla como referencia. Una vez configurados los parámetros deberá pulsar sobre el botón "Connect".

Si una vez pulsado no se ve nada en la pantalla, la dirección IP o el puerto están mal escritos o el servidor no es accesible, en ese caso deberá cerrar la aplicación e intentarlo de nuevo.

PASO 4: EJECUCION DEL FRONTEND

Si todo ha ido bien, debería poder ver la imagen 3D en movimiento, en la esquina superior izquierda puede ver las métricas de rendimiento del programa (no las de la transmisión) y en la esquina superior derecha verá un botón que dice "2D" pulsando este botón podrá cambiar entre 4 modos:

- **2D:** Modo estándar, muestra la imagen en la pantalla
- **3D Half SBS:** Muestra dos imágenes, derecha e izquierda, es el modo para usar con gafas VR tipo Google Cardboard
- **3D Full SBS:** Igual que el anterior pero con las imágenes mas "achatas", ideal para usarse con proyectores 3D
- **Anaglyph:** Modo anaglifo (Rojo-Azul), ideal para su uso con gafas 3D que usen esta tecnología



Para ver mejor la representación de la profundidad, puede **pulsar y arrastrar el ratón** por la pantalla para mover la cámara alrededor del sujeto.

Finalmente para cerrar la ejecución del programa, cierre la aplicación del Frontend, y detenga el contenedor de Docker del Backend (Tras esto puede cerrar OBS también)

EXTRA: GESTIÓN DE ERRORES

BACKEND

Obtener dirección IP. Para obtener la dirección IP en Windows basta con abrir una terminal y escribir "ipconfig" y buscar el campo "Dirección IPv4"

El puerto 5000 ya está en uso. Si Docker no puede iniciar el contenedor por que alguno el puerto 5000 está siendo usado por otros servicios. Se pueden cambiar los puertos editando el fichero docker-compose.yml cambiando "5000:5000" por "5000:X" donde X es el nuevo puerto, que deberá especificarse también en el frontend

No aparece el mensaje de "Captura Iniciada" en Docker. OBS no está transmitiendo al servidor RTMP, revise el paso 1

FRONTEND

Al pulsar en "Connect" no aparece nada. El servidor aún no ha arrancado (debe esperar a que muestre los mensajes de Inferencia FPS) o no se ha podido establecer la conexión con el (Revise la IP y el Puerto que está introduciendo en el frontend)

Anexo VI. Uso ético de modelos de IAG durante el desarrollo del trabajo

Para el desarrollo del presente trabajo se ha hecho uso de la herramienta Gemini (modelo 2.5 flash primero y 3 flash, y 3 pro después) para tareas de apoyo técnico y auxiliar, asegurando siempre que la autoría intelectual y el redactado final pertenezcan al estudiante.

A este respecto la IAG se ha utilizado, exclusivamente en su modalidad de texto, esto es, sin generación de imágenes ni videos, para los siguientes fines:

1. En desarrollo software

- 1.1 En gestión de errores, como alternativa a foros especializados como StackOverflow para la gestión de errores. Cuando el software ha arrojado errores de consola, se ha consultado a Gemini en primera instancia a la hora de identificar y entender el error, así como buscar posibles soluciones.
- 1.2 Para la revisión de código, a través de la técnica de “programación por pares” usando a Gemini como compañero virtual. Se le han suministrado fragmentos de código, con el fin de revisar problemas de seguridad o para la detección de oportunidades de mejora de optimización y limpieza de código, haciendo uso de prompts como: “¿Habría alguna manera de que esto se ejecute más rápido?” o “Este código está quedando muy denso, ¿podría simplificarse?”.
- 1.3 Como apoyo en la implementación de funciones específicas, como la captura de imágenes con Kinect para el dataset o la creación de promedios de FPS en Unity3D. Dado que Kinect es un dispositivo de 2014 sin soporte oficial desde 2017, y las librerías asociadas como pykinect2 presentan errores y problemas con el ecosistema de software actual, la IA fue clave a la hora de poder subsanar errores de compatibilidad internos de la propia librería de software para poder utilizarla e integrarla dentro la lógica del proyecto.

2. En la memoria

- 2.1 Para las primeras etapas de la Estructura y para brainstorming, se ha usado prompts como: “este es el capítulo 4.7 que trata sobre... cómo crees que debería estructurarlo?” o “sería buena idea añadir un punto 3.3 que hable de...” o “si voy a incluir un punto que trate sobre sobre debería ponerlo en el 4.4.7, en el 4.2.5 o crees debería crear un apartado nuevo para eso”. El objetivo ha sido el de aclarar ideas y mejorar la comunicación.

Un ejemplo de esto, puede verse en el capítulo “3.3 Tecnologías, Metodologías y Herramientas aplicadas”. La primera versión de esa sección, se centraba en definir las herramientas utilizadas (Python es... Se creo en... con el fin de...). El estudiante tenía dudas respecto a la adecuación de esta sección al resto del trabajo por el enfoque que esta

tenía. Tras consultarlo con la IAG, se sugirió, que efectivamente, dado el perfil técnico del tribunal, esta sección no debería centrarse en describir las herramientas, sino en justificar su adecuación y relevancia para el proyecto. Esta sugerencia, validada y aplicada por el estudiante, le ayudó a replantearse que si en esa sección se justificaba la adecuación de las herramientas, sería interesante también hablar sobre las herramientas descartadas, esta inspiración, junto a investigaciones posteriores en internet, llevaron a la creación e integración del “Tech Radar” que hoy puede ubicarse en la sección 3.3.7 de este trabajo, que no habría existido de mantenerse el enfoque inicial.

2.2 Para revisiones ortográficas, principalmente en las últimas etapas del trabajo, se le ha facilitado a la IA el pdf del trabajo con el siguiente prompt “Revisa el documento y busca todas las faltas de ortografía, dame además el total de faltas, y la ratio de faltas por página”. Este proceso requirió de supervisión humana y de relanzarse múltiples veces, ya que la herramienta a veces alucinaba errores inexistentes o pasaba por alto otra buena parte de ellos. Aun así, esta técnica permitió corregir manualmente cerca de 150 errores ortográficos (del tipo “vídeo” o “Máster” sin tilde, o palabras mal escritas como ngx en vez de ngxn o convlucionales en vez de convolucionales). Esta metodología permitió llevar una corrección más exhaustiva y enfocada del trabajo, ahorrando horas de corrección, mitigando problemas y asegurando en general una mayor calidad gramatical que si no se hubiera utilizado.

2.3 Finalmente, también se ha hecho uso de Gemini para solventar bloqueos de redacción, para, por ejemplo, cuando se quieren transmitir dos ideas, pero resulta difícil encontrar la manera de conectarlas. En estos casos, en ocasiones, se le facilitaba el texto ya redactado a Gemini, junto con lo que se quería decir, con el fin de que la herramienta generare un borrador preliminar uniendo estos conceptos, que, posteriormente, sirviera de base inspiradora con el objetivo, de ser reescrito y adaptado por el estudiante.

Cierre y conclusión personal

Para este trabajo el estudiante considera que se ha hecho un uso ético de la IAG. En ningún caso se ha copiado y pegado el texto generado por la IA para hacerlo pasar como propio. El 100% del texto final ha sido redactado, revisado y validado por el estudiante.

En definitiva, la IAG ha funcionado como un vehículo para impulsar la creatividad (brainstorming), evitar bloqueos creativos y ayudar a reenfocar el trabajo del estudiante en su búsqueda de la excelencia, en ningún momento, la IAG ha sustituido al estudiante, sino que ha servido como soporte (al igual que otras muchas técnicas y tecnologías) para permitirle alcanzar nuevos niveles de calidad y excelencia académica.