UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Implementación de reconocimiento de pose a partir de video RGB plano en aplicación 3D

Trabajo Fin de Grado

Grado en Tecnología Digital y Multimedia

AUTOR/A: Baldoví Murcia, Aitana

Tutor/a: Rey Solaz, Beatriz

CURSO ACADÉMICO: 2022/2023

Resumen

En el campo de las aplicaciones multimedia y la producción audiovisual, es innegable hoy en día la influencia de los sistemas de captura de movimiento y reconocimiento de pose. Aplicable a sectores tan variados como el cine o la medicina, esta tecnología ofrece un amplio espacio para la innovación y el desarrollo. Sin embargo, el alto coste de los sistemas Motion Capture (MoCap) utilizados en la industria, limita el acceso a ella significativamente.

En este proyecto, se analizarán las distintas alternativas disponibles al público general para el reconocimiento de pose en imagen plana, es decir, sin información de profundidad. Tras realizar una comparativa basada en su accesibilidad y capacidades se escogerá la opción que mejor se adapte a la siguiente etapa.

La parte práctica de este trabajo, tendrá como objetivo hacer uso de la librería escogida anteriormente. Se implementará como parte del desarrollo de una aplicación, que reflejará movimiento humano, capturado en vivo con una webcam, en una escena 3D.

El objetivo de este Trabajo de Final de Grado es estudiar la actualidad del reconocimiento de pose y poner en práctica los conocimientos adquiridos, acercando así sus múltiples aplicaciones a un sector demográfico más amplio. Además de la implementación de una de estas librerías, sentando las bases para su aplicación en el entrenamiento de habilidades motoras de forma remota.

Palabras clave: Captura de Movimiento; Reconocimiento de Pose; Inteligencia Artificial; 3D; Python; Unity;

Resum

Al camp de les aplicacions multimèdia i la producció audiovisual, és innegable avui dia la influència dels sistemes de captura de moviment i reconeixement de posi. Aplicable a sectors tan variats com el cinema o la medicina, aquesta tecnologia ofereix un ampli espai per a la innovació i el desenvolupament. No obstant això, l'alt cost dels sistemes MoCap utilitzats a la indústria, limita l'accés a ella significativament.

En aquest projecte, s'analitzaran les diferents alternatives disponibles al públic general per al reconeixement de posició en imatge plana, és a dir, sense informació de profunditat. Després de fer una comparativa basada en la seva accessibilitat i capacitats s'escollirà l'opció que millor s'adapti a la següent etapa.

La part pràctica d'aquest treball tindrà com a objectiu fer ús de la llibreria triada anteriorment. S'implementarà com a part del desenvolupament d'una aplicació, que reflectirà moviment humà, capturat en viu amb una càmera web, en una escena 3D.

L'objectiu d'aquest Treball de Final de Grau és estudiar l'actualitat del reconeixement de posi i posar en pràctica els coneixements adquirits, apropant així les seues múltiples aplicacions a un sector demogràfic més ampli. A més de la implementació d'una d'aquestes llibreries, sentant les bases per la seua aplicació a l'entrenament d'habilitats motores de forma remota.

Paraules clau: Captura de Moviment; Reconeixement de Pose; Intel · ligència Artificial; 3D; Python; Unity;

Abstract

In the field of multimedia applications and audiovisual production, the influence of motion capture and pose recognition systems is undeniable today. Applicable to sectors as varied as cinema or medicine, this technology offers ample space for innovation and development. However, the high cost of the MoCap systems used in the industry limits access to it significantly.

In this project, the different alternatives available to the general public for pose recognition in a flat image, that is, without depth information, will be analyzed. After making a comparison based on its accessibility and capabilities, the option that best suits the next stage will be chosen.

The practical part of this work will aim to make use of the library chosen previously. It will be implemented as part of the development of an application, which will reflect human movement, captured live with a webcam, in a 3D scene.

The objective of this Final Degree Project is to study the current state of pose recognition and put the knowledge acquired into practice, thus bringing its multiple applications closer to a broader demographic sector. In addition to the implementation of one of these libraries, laying the foundations for its application in training motor skills remotely.

Keywords: Motion Capture; Pose Recognition; Artificial intelligence; 3D; Python; Unity;

Agradecimientos

A mi tutora, Bea. Gracias por acompañarme y guiarme durante todo el curso.

A mi familia y mis amigos. Gracias por acompañarme y guiarme durante toda la vida.

Índice general

Índice de figuras

Listado de siglas empleadas

AR Realidad Aumentada.

CNN Convolutional Neural Network.

EMG Electromiografía.

GAN Generative Adversarial Networks.

GUI Graphic User Interface.

IA Inteligencia Artificial.

IDE Entorno de Desarrollo Integrado.

ML Machine Learning.

MoCap Motion Capture.

SCV Sistema de Control de Versiones.

TCP Transmision Control Protocol.

UDP User Datagram Protocol.

UI Interfaz de Usuario.

VR Realidad Virtual.

Introducción

1.1 Contexto y motivación

En el siempre cambiante mundo del desarrollo tecnológico, el papel de los sistemas de captura de movimiento y reconocimiento de pose ha emergido como una herramienta de vanguardia con un impacto innegable. Estos sistemas, conocidos comúnmente como MoCap (por sus siglas en inglés: Motion Capture), han trascendido su origen en la industria del cine y se han convertido en una tecnología omnipresente en campos tan diversos como la medicina, los videojuegos, la animación, la educación y el arte digital.

La capacidad de registrar y analizar los movimientos del cuerpo humano con precisión milimétrica ha revolucionado la forma en que interactuamos con el mundo digital y físico que nos rodea. Desde la creación de personajes animados en películas de Hollywood hasta la rehabilitación de pacientes con discapacidades motoras, pasando por el diseño de videojuegos que responden a los gestos del jugador, los sistemas de MoCap se han extendido a prácticamente todos los aspectos de nuestra vida cotidiana.

No obstante, a pesar de sus vastas aplicaciones y su potencial, la adopción de esta tecnología ha estado históricamente limitada por su alto coste. Los sistemas de MoCap profesionales pueden llegar a ser inaccesibles para la mayoría de las personas y organizaciones, restringiendo así su impacto a una demográfica mucho más reducida.

Este proyecto se enmarca en este contexto, donde la demanda de soluciones más asequibles y accesibles para el reconocimiento de pose está en constante crecimiento. Busca dar un pequeño paso más en la democratización de esta tecnología, brindando una oportunidad para que un público más amplio se beneficie de sus capacidades y aplicaciones.

Al analizar y comparar las alternativas disponibles para el reconocimiento de pose en imágenes planas, es decir, sin información de profundidad, este proyecto aspira a estudiar y aplicar una solución viable que permita a cualquiera con interés acceder a esta tecnología de manera más económica y sencilla.

La motivación es clara: hacer accesible el reconocimiento de pose y el MoCap a un público más amplio no solo democratizará el acceso a esta tecnología, sino que también abrirá nuevas posibilidades de aplicación en áreas que van más allá de la producción cinematográfica y el entretenimiento. Desde la telemedicina hasta la capacitación deportiva remota, la capacidad de capturar y analizar el movimiento humano de manera precisa y económica tiene

el potencial de mejorar numerosos aspectos de nuestras vidas y trabajos diarios.

Se podría decir que estas motivaciones se alinean con el cuarto Objetivo de Desarrollo Sostenible: Educación de Calidad, puesto que busca acercar los nuevos desarrollos tecnológicos a un entorno más asequible. De este modo hace más accesible el aprendizaje al respecto a todos aquellos interesados, sin las limitaciones de costes que implican otros tipos de sistemas.

En última instancia, este Trabajo de Final de Grado busca no solo estudiar la vanguardia tecnológica en el sector, sino también contribuir a ampliar su alcance y aplicaciones, allanando el camino para una revolución en la forma en que interactuamos con el mundo digital y físico que nos rodea.

1.2 Objetivos

Este proyecto tiene como objetivo principal explorar, comprender y aplicar las tecnologías relacionadas con el reconocimiento de pose y la captura de movimiento, fomentando el aprendizaje y la adquisición de habilidades en este campo. Los objetivos específicos son los siguientes:

1. **Investigación y evaluación:** Realizar una investigación exhaustiva de las tecnologías de reconocimiento de pose disponibles actualmente, así como de la historia que las precede, enfocándose en las soluciones basadas en imágenes planas y sin necesidad de información de profundidad. Todo ello para, más tarde, evaluar críticamente estas tecnologías en función de su accesibilidad, precisión y capacidad para adaptarse a aplicaciones prácticas.
2. **Selección de una solución:** Escoger la tecnología de reconocimiento de pose que mejor se adapte a los requisitos del proyecto, considerando tanto sus capacidades técnicas como su disponibilidad para el público general.
3. **Desarrollo práctico:** Implementar la tecnología de reconocimiento de pose seleccionada en una aplicación práctica. Diseñando una interfaz simple, haciendo uso de los conocimientos adquiridos a lo largo del grado y utilizando la solución previamente estudiada de manera efectiva.
4. **Optimización:** Optimizar el rendimiento de la aplicación para garantizar una experiencia fluida y en tiempo real. Siempre llevando a cabo pruebas exhaustivas para identificar y corregir posibles errores o carencias.

5. **Documentación y transferencia de conocimientos:** Elaborar una memoria detallada que describa el proceso de desarrollo, las tecnologías empleadas y las decisiones técnicas tomadas. Proporcionando así recursos y guías para que otros puedan comprender y expandir sobre la aplicación desarrollada.

Marco teórico

Previo al desarrollo de este proyecto, es necesario comprender los orígenes de la captura de movimiento y el reconocimiento de pose. Además de conocer el estado actual de estas tecnologías y sus aplicaciones más recientes.

Esto nos permitirá realizar un estudio informado de las soluciones discutidas más adelante y delimitar unos parámetros adecuados para su comparativa.

2.1 Historia

2.1.1 Rotoscopia

En 1915, durante la Primera Guerra Mundial, el animador Max Fleischer desarrolló una técnica conocida como "rotoscopia", la cual sentó las bases para la sofisticada tecnología MoCap que hoy en día disfrutamos.

La rotoscopia [Figura 2.1] implicaba el meticuloso proceso de trazar cada *frame* de una animación sobre una superficie de vidrio, a través de la cual se proyectaban secuencias de acción en vivo, con el fin de replicar de manera naturalista los movimientos humanos y de animales. Esta técnica permitía alcanzar niveles de realismo y fluidez considerablemente superiores a las animaciones de la época.[1]

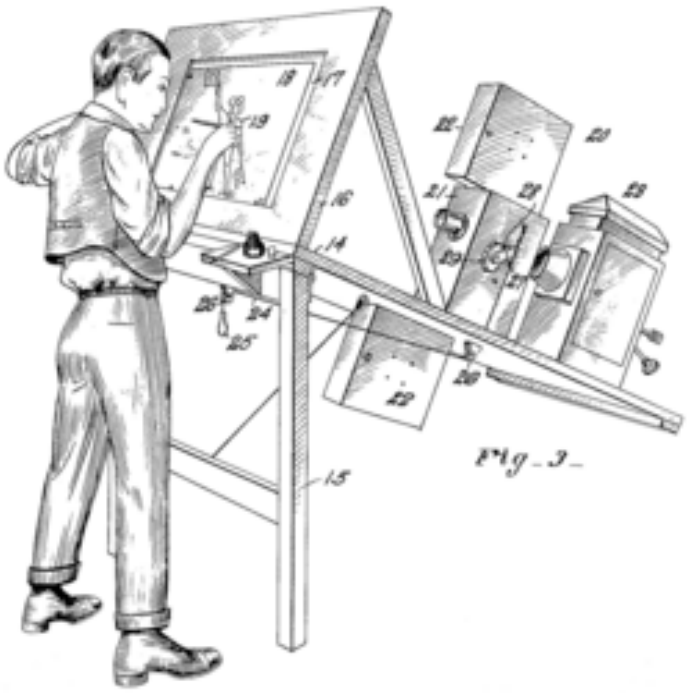

Figura 2.1. Rotoscopio original de Fleischer (1914). [1]

Desde entonces, los estudios de animación, en particular Fleischer Studios, utilizaron la rotoscopia de manera prominente hasta principios de la década de 1940. Durante este período, esta técnica desempeñó un papel central en obras influyentes como “Popeye”, “Betty Boopÿ “Los Viajes de Gulliver”.[2]

La captura de movimiento, tal como la conocemos hoy, estaba aún lejos de su desarrollo, pero la rotoscopia demostró que valía la pena copiar acciones de actores reales para incorporarlas en la animación. Aunque esta ha sido revisada y actualizada a lo largo del tiempo, su importancia radica en haber desencadenado la curiosidad sobre los métodos más efectivos para registrar y capturar el movimiento humano.

2.1.2 Primeras capturas

En 1959, Lee Harrison III desarrolló un traje equipado con potenciómetros (resistencias ajustables) [Figura 2.2] y logró registrar y animar los movimientos de un actor en tiempo real en una pantalla de tubo de rayos catódicos (CRT). Aunque rudimentario, este dispositivo bautizado como Scanimate representó la primera instancia de captura de movimiento en tiempo real.[2]

Figura 2.2. Bailarina vistiendo el traje de Scanimate en la producción de "Turn-On"[3]

Veinte años después, en la década de 1980, se popularizó el uso de trajes con marcadores activos y un conjunto de cámaras para rastrear los movimientos de los actores. Motivado por un crecimiento en el estudio del movimiento humano por parte de los laboratorios de biomecánica. Esta solución ofrecía imágenes digitales con un nivel de detalle y precisión muy elevado respecto al primer traje de Harrison.[4]

Sin embargo, incluso durante los noventa, todos los sistemas de captura de movimiento requerían cámaras del tamaño de una cabina telefónica, y los animadores asignaban manualmente cada marcador en cada *frame* para cada escena. Es decir, su precisión era mucho mayor, pero se trataba de una técnica tan costosa en tiempo y esfuerzo como la rotoscopia.

2.2 Estado del arte

En los últimos años, se han producido una serie de tendencias en la captura de movimiento, entre las que destacan el aumento de la precisión y la resolución, la reducción de los costes y el incremento de la versatilidad. Con estas mejoras, en la actualidad, el reconocimiento de pose se ha integrado en gran variedad de campos.

2.2.1 Sistemas actuales

A continuación se detallan las prestaciones ofrecidas por distintas empresas punteras y/o con especial significancia en el sector. La selección pretende ofrecer una vista general del panorama actual, sin embargo, es necesario remarcar que el mercado para estos sistemas se ha expandido exponencialmente en los últimos años y existe una variedad muy extensa de productos.

Vicon [5]

Vicon es una empresa líder en el mercado de la captura de movimiento. Sus productos se utilizan en una amplia gama de aplicaciones, desde la investigación científica hasta la producción cinematográfica.

Los productos de Vicon [Figura 2.3] para la captura de movimiento se basan en la tecnología óptica. Un conjunto de cámaras infrarrojas captan la luz de unos marcadores reflectantes que se colocan en el cuerpo del sujeto. El software de Vicon interpreta esta información y genera un modelo 3D del movimiento del sujeto.

Vicon ofrece una amplia gama de productos para satisfacer las necesidades de cada aplicación. Sus sistemas más avanzados pueden capturar el movimiento con una precisión de hasta 0,1 milímetros.

Figura 2.3. Producción con productos Vicon, hardware y software. [6]

Para establecer un marco de referencia, a continuación se listan algunos de sus productos, junto con su coste:

- **Cámaras:** Vicon ofrece una amplia gama de cámaras infrarrojas para la captura de movimiento. Las cámaras se pueden montar en techos, paredes o trípodes.
    1. Vicon MX-X1: Cámara infrarroja de alta resolución con un campo de visión de 120 grados. Precio: 12.000 USD.
    2. Vicon MX-X2: Cámara infrarroja de ultra alta resolución con un campo de visión de 120 grados. Precio: 24.000 USD.
    3. Vicon MX-X3: Cámara infrarroja de ultra alta resolución con un campo de visión de 60 grados. Precio: 28.000 USD.
- **Marcadores:** Los marcadores reflectantes se utilizan para rastrear el movimiento del sujeto. Vicon ofrece una amplia gama de marcadores de diferentes tamaños y colores.
    1. icon MX-Marker Mini: Marcador reflectante de 5 mm de diámetro. Precio: 75 USD.

2. Vicon MX-Marker: Marcador reflectante de 10 mm de diámetro. Precio: 100 USD.
3. Vicon MX-Marker Large: Marcador reflectante de 20 mm de diámetro. Precio: 150 USD.

- **Software:** El software de Vicon interpreta la información de las cámaras y genera un modelo 3D del movimiento del sujeto. El software también permite editar y analizar los datos de captura de movimiento.

1. Vicon Vero: Software de captura de movimiento de nivel básico. Precio: 25.000 USD.
2. Vicon Vantage: Software de captura de movimiento de nivel medio. Precio: 50.000 USD.
3. Vicon Nexus 2: Software de captura de movimiento de última generación. Precio: 100.000 USD.

Estos precios son solo orientativos y pueden variar según la configuración específica del sistema de captura de movimiento.

Myo[7]

Myo es una empresa que fabrica dispositivos de captura de movimiento basados en la tecnología Electromiografía (EMG). Los dispositivos Myo utilizan sensores para medir la actividad eléctrica de los músculos, lo que permite capturar el movimiento del cuerpo humano con precisión.

Myo ofrece dos productos principales para la captura de movimiento:

- **Myo Armband [Figura 2.4]:** Una pulsera que se coloca en el antebrazo. El brazalete Myo tiene ocho sensores que miden la actividad eléctrica de los músculos del antebrazo, la muñeca y la mano. Precio: 250 USD
- **Myo Grip:** Un dispositivo que se coloca en la mano. El Myo Grip tiene cuatro sensores que miden la actividad eléctrica de los músculos de la mano y la muñeca. Precio: 150 USD

Figura 2.4. Myo Armband de Thalmic Labs y los gestos que reconoce. [8]

Es importante tener en cuenta que los dispositivos Myo no son tan precisos como los sistemas de captura de movimiento tradicionales basados en la tecnología óptica. Sin embargo, ofrecen una precisión suficiente para la mayoría de las aplicaciones y se pueden percibir como respuesta a la necesidad de sistemas más accesibles. Su producción se descontinuo en 2019; sin embargo, siguen usándose en diversas entidades médicas.

Intuitive Surgical [9]

Intuitive Surgical es una empresa que fabrica sistemas quirúrgicos robóticos. Sus sistemas se utilizan en una amplia gama de procedimientos quirúrgicos, desde cirugías abdominales hasta cirugías torácicas.

Intuitive Surgical utiliza la captura de movimiento en sus sistemas quirúrgicos robóticos para mejorar la precisión y la seguridad de los procedimientos quirúrgicos. Estos sistemas utilizan una combinación de tecnología óptica y EMG para rastrear el movimiento del cirujano y de los instrumentos quirúrgicos.

Estas son las ventajas que ofrecen:

- **Mayor precisión:** Los sistemas de captura de movimiento ayudan a los cirujanos a realizar movimientos más precisos.
- **Mayor seguridad:** Los sistemas de captura de movimiento ayudan a los cirujanos a

evitar lesiones a los pacientes.

- **Mayor eficiencia:** Los sistemas de captura de movimiento pueden ayudar a los cirujanos a realizar procedimientos quirúrgicos de forma más eficiente.

A continuación, se describen los principales productos de Intuitive Surgical para la captura de movimiento:

- **Intuitive Surgical StealthStation:** El StealthStation es un sistema de captura de movimiento de última generación que utiliza una combinación de tecnología óptica y EMG para rastrear el movimiento del cirujano y de los instrumentos quirúrgicos. Precio: a partir de 1,5 millones de USD.
- **Intuitive Surgical da Vinci Xi [Figura 2.5]:** El da Vinci Xi es un sistema quirúrgico robótico que utiliza el StealthStation para capturar el movimiento del cirujano. Precio: a partir de 2,5 millones de USD.
- **Intuitive Surgical da Vinci X™:** El da Vinci X™ es un sistema quirúrgico robótico portátil que utiliza el StealthStation para capturar el movimiento del cirujano. Precio: a partir de 1,5 millones de USD.

Figura 2.5. Imagen promocional Intuitive Surgical daVinci Xi [9]

Incluyen los siguientes componentes:

- **Cámaras infrarrojas:** Las cámaras infrarrojas captan la luz de unos marcadores reflectantes que se colocan en el cirujano y en los instrumentos quirúrgicos.
- **Sensores EMG:** Los sensores EMG miden la actividad eléctrica de los músculos del cirujano.
- **Software:** El software interpreta la información de las cámaras y los sensores para generar un modelo 3D del movimiento del cirujano y de los instrumentos quirúrgicos.

Faceware [10]

Faceware Technologies es una empresa que fabrica software y hardware para la captura de movimiento facial. Sus productos se utilizan en una amplia gama de aplicaciones. Entre ellas,

la creación de efectos especiales fotorealistas, las animaciones de personajes en videojuegos o la creación de gráficos en vivo en programas de televisión. Además, Faceware se utiliza para rehabilitar lesiones faciales.

Estos productos se basan en la tecnología de seguimiento facial. El software de Faceware utiliza una combinación de cámaras infrarrojas y cámaras de alta resolución para rastrear los movimientos de la cara. También utiliza algoritmos de inteligencia artificial para identificar y procesar los rasgos faciales.

Los productos de Faceware incluyen los siguientes componentes:

- **Cámaras infrarrojas:** Las cámaras infrarrojas captan la luz de unos marcadores reflectantes que se colocan en la cara del sujeto.
- **Cámaras de alta resolución:** Las cámaras de alta resolución captan imágenes de la cara del sujeto.
- **Software:** El software interpreta la información de las cámaras para generar un modelo 3D del movimiento de la cara.

Y ofrecen las siguientes ventajas:

- **Alta precisión:** Capturan los movimientos faciales con una precisión de hasta 0,01 milímetros.
- **Rango dinámico amplio:** Pueden capturar los movimientos faciales en una amplia gama de condiciones de iluminación.
- **Rendimiento en tiempo real:** Los productos de Faceware pueden capturar los movimientos faciales en tiempo real.

A continuación, se describen los principales productos de Faceware [Figura 2.6] para la captura de movimiento facial:

- **Faceware Studio:** Software de captura de movimiento facial de última generación. Precio: a partir de 10.000 USD.
- **Faceware Headcam:** Cámara infrarroja portátil que se utiliza para capturar el movimiento facial. Precio: a partir de 25.000 USD.

Figura 2.6. Vista de escritorio del software de Faceware [10]

2.2.2 Aplicaciones

Tanto los sistemas detallados anteriormente como otros que aprovechan mecanismos de MoCap similares se emplean activamente en distintas áreas. En esta sección se detallan algunos ejemplos de como esta nueva tecnología se ha implantado en cada sector y los avances que ha promovido.

Medicina

En el Centro de Rehabilitación y Medicina Física de la Universidad de California, Davis, emplean el sistema de captura de movimiento Vicon para evaluar la función motora en pacientes con lesiones de la médula espinal. Utilizando un conjunto de cámaras infrarrojas que rastrean la posición y movimiento de marcadores en el cuerpo de los pacientes, se generan modelos tridimensionales de sus movimientos. Esto permite a los fisioterapeutas identificar patrones de movimiento inusuales y diseñar tratamientos personalizados.[11]

En el Hospital Universitario de Lovaina, Bélgica, se utiliza el sistema de reconocimiento de pose Myo para brindar retroalimentación a pacientes en rehabilitación de la mano. Este

sistema aprovecha sensores electromiográficos para capturar la actividad muscular, siguiendo el movimiento de las manos de los pacientes. Proporciona retroalimentación visual y auditiva sobre su técnica, lo que mejora la precisión y coordinación. [12]

En el Hospital Brigham and Women's de Boston, Massachusetts, el reconocimiento de pose controla un robot quirúrgico en cirugías de cataratas. Utilizan el sistema de reconocimiento de pose Intuitive Surgical da Vinci con una cámara para seguir la posición de las manos del cirujano. [13]

En el Instituto de Investigación de Parkinson de la Universidad de California, Berkeley, emplean la captura de movimiento para diagnosticar la enfermedad de Parkinson. Los pacientes realizan una prueba de marcha rastreada por un sistema de captura de movimiento, que detecta signos como temblores, rigidez o lentitud. Utilizan el sistema de captura de movimiento Qualisys con cámaras infrarrojas y marcadores en el cuerpo del paciente. [14]

Deporte

El equipo de baloncesto de los Golden State Warriors utiliza el sistema de reconocimiento de pose Qualisys 3D con 12 cámaras para brindar retroalimentación a sus jugadores en cuanto a su técnica de tiro. Los sistemas identifican errores en la técnica y ofrecen recomendaciones para mejorar la precisión. [15]

El equipo de fútbol de la selección alemana hace uso del Vicon Nexus con 60 cámaras infrarrojas para crear simuladores de entrenamiento. Estos simuladores proporcionan a los jugadores una experiencia realista de jugar en estadios llenos, lo que les ayuda a prepararse para los partidos de competición. [16]

Los científicos deportivos de la Universidad de Stanford también emplean Vicon Nexus para investigar la biomecánica del salto de longitud. Utilizan los datos obtenidos para identificar patrones de movimiento que contribuyen a un salto más largo. [17]

Cine y Videojuegos [18]

Para la película "Avatar", los animadores emplearon el Motion Capture Lab de Vicon para modelar a los personajes Na'vi y los animales de Pandora. Actores llevaron trajes con sensores que registraron sus movimientos, los cuales sirvieron como base para su animación.

Mientras, en la película "Puss in Boots", los animadores utilizaron Faceware para crear las expresiones faciales del protagonista, detectando los músculos faciales involucrados en las

expresiones del Gato con Botas y luego utilizándolos para animar al personaje.

Los desarrolladores de la aplicación de realidad virtual “HoloLens” emplean el sistema de captura de movimiento Kinect para proporcionar experiencias más inmersivas. Los datos capturados de movimiento registran los movimientos del usuario, creando así una experiencia de realidad virtual más auténtica.

En “Pokémon Go”, los desarrolladores utilizan el reconocimiento de pose ARKit de Apple para diseñar interfaces de usuario más naturales. Los sistemas de reconocimiento de pose identifican las partes del cuerpo involucradas en las acciones, lo que da lugar a interfaces de usuario más intuitivas.

Soluciones con Machine Learning e Imagen Plana

En los últimos años, la inteligencia artificial ha experimentado avances asombrosos que han revolucionado prácticamente todos los aspectos de nuestra vida cotidiana. Desde la atención médica hasta la industria automotriz, pasando por el comercio electrónico y la educación, la Inteligencia Artificial (IA) ha demostrado su capacidad para transformar radicalmente la forma en que hacemos las cosas.

Los algoritmos de aprendizaje profundo, el procesamiento del lenguaje natural y la visión por computadora han permitido que las máquinas comprendan y respondan a datos y comandos de una manera que antes parecía imposible. La IA también ha impulsado avances en la robótica y la automatización, mejorando la eficiencia y la precisión en una amplia gama de industrias. A medida que la inteligencia artificial continúa avanzando, estamos viendo cómo redefine nuestro mundo, desde la toma de decisiones empresariales hasta la medicina personalizada, y promete un futuro emocionante y lleno de posibilidades.

Las tecnologías de MoCap y reconocimiento de pose no han sido la excepción. En esta área, no solo ha presentado una amplia gama de posibilidades para mejorar y optimizar los sistemas existentes hasta algo cercano a la perfección, sino que ha abierto las compuertas al desarrollo de soluciones mucho más accesibles.

Estos nuevos avances no son únicamente menos costosos económicamente, sino que han reducido también la carga computacional que infieren en los sistemas que los ejecutan. Poniéndolos así al alcance de todo aquel interesado en aprender al respecto.

3.1 Reconocimiento de pose sobre imagen plana

Hoy en día, las técnicas basadas en Machine Learning (ML) son capaces de registrar el movimiento y la pose de un sujeto u objeto sin la necesidad de marcadores. Esto las hace más asequibles, fáciles de usar y adaptables a una gran variedad de situaciones.

Existen muchos enfoques diferentes para utilizar el aprendizaje automático para la captura de movimiento y la estimación de pose en 3D. Una de las técnicas más comunes es el diseño de redes neuronales convolucionales Convolutional Neural Network (CNN) para aprender a identificar características en una imagen 2D que se correlacionan con la posición

y orientación de los puntos de interés en 3D.

Otra técnica popular es implementar redes neuronales generativas, como Generative Adversarial Networks Generative Adversarial Networks (GAN), para crear imágenes 3D a partir de imágenes 2D. Con estas técnicas se pueden generar modelos tridimensionales de objetos y personas que son indistinguibles de los modelos producidos con técnicas tradicionales de captura de movimiento. [19]

El uso de aprendizaje automático para la captura de movimiento y la estimación de pose en 3D todavía está en sus primeras etapas, pero tiene el potencial de transformar el campo. Las técnicas basadas en aprendizaje automático han empezado a democratizar el acceso a esta tecnología que está presente en cada vez más sectores.

3.2 Alternativas

En la búsqueda de soluciones que encajaran dentro de las necesidades y aspiraciones de este proyecto, destacan dos candidaturas. En la siguiente sección se estudian las bases del funcionamiento de cada una y sus prestaciones, con el objetivo de realizar una comparativa informada y tomar una decisión coherente más adelante.

Vale la pena incidir en que se han estudiado también otras dos alternativas, siendo estas DeepCut [20] y PoseNet [21]. Sin embargo, por su menor volumen de documentación clara, comunidad y recursos disponibles, se descartaron como soluciones viables frente a OpenPose y MediaPipe.

No obstante se incluyen referencias a articulos explicativos de cada algoritmo. Haber revisado estas alternativas ha sido igualmente enrriquecedor en esta etapa de investigación, puesto que ha sacado al la luz importantes métricas en la evaluación de las mismas, como el soporte actural de cada plataforma y si facilidad de implementación.

3.2.1 OpenPose [22]

OpenPose es una biblioteca de detección de poses humanas en tiempo real para múltiples personas [Figura 3.1] que, por primera vez, muestra la capacidad de detectar conjuntamente el cuerpo humano, los puntos clave de los pies, las manos y la cara en imágenes individuales. OpenPose es capaz de detectar un total de 135 puntos clave. Este método ganó el desafío de puntos clave COCO 2016 y es conocido por su calidad decente y robustez en entornos con múltiples personas.

La técnica de OpenPose fue creada por Ginés Hidalgo, Yaser Sheikh, Zhe Cao, Yaadhav Raaj, Tomas Simon, Hanbyul Joo y Shih-En Wei. Sin embargo, actualmente es mantenida por Yaadhav Raaj y Ginés Hidalgo.

Algunas características destacadas de OpenPose incluyen:

- Detección de keypoints en 3D en tiempo real para una sola persona:
  - Triangulación en 3D con múltiples vistas de cámara.
  - Compatibilidad con cámaras Flir.
- Detección de keypoints en 2D en tiempo real para múltiples personas:
  - Estimación de keypoints del cuerpo/pies con 15, 18 o 27 keypoints.
  - Estimación de 21 keypoints para las manos.
  - Estimación de 70 keypoints para la cara.
- Seguimiento de una sola persona para acelerar la detección y suavizar la visualización.
- Toolbox de calibración para estimar parámetros de cámara extrínsecos, intrínsecos y de distorsión.

OpenPose tiene licencia para uso no comercial gratuito y redistribución bajo ciertas condiciones. Sin embargo, para aplicaciones comerciales no exclusivas, se requiere una tarifa anual no reembolsable de 25,000 USD.

Para utilizar OpenPose de manera más eficiente, existe una versión optimizada llamada Lightweight OpenPose que realiza inferencia en tiempo real en CPU con una pérdida de precisión mínima.

OpenPose admite varias fuentes de entrada de video, incluyendo imágenes, videos, transmisiones de cámaras web, cámaras Flir/Point Grey, cámaras IP (CCTV) y fuentes de entrada personalizadas. Puede ejecutarse en sistemas Ubuntu, Windows, Mac y Nvidia Jetson TX2.

La forma más rápida y sencilla de usar OpenPose es a través de plataformas como Viso Suite, que proporciona una solución integral para construir, implementar y escalar aplicaciones de OpenPose.

Figura 3.1. Ejemplo de visualización de Openpose[22]

Funcionamiento [23]

La biblioteca OpenPose inicialmente extrae características de una imagen utilizando las primeras capas. Las características extraídas se insertan luego en dos divisiones paralelas de capas de CNN. La primera división predice un conjunto de 18 mapas de confianza, cada uno de los cuales denota una parte específica del esqueleto de la pose humana. La siguiente rama predice otro conjunto de 38 "Part Affinity Fields"(PAFs) que indican el nivel de asociación entre las partes.

Las etapas posteriores se utilizan para limpiar las predicciones realizadas por las ramas. Con la ayuda de los mapas de confianza, se crean gráficos bipartitos entre pares de partes. A través de los valores de PAF, se eliminan los enlaces más débiles en los gráficos bipartitos. Ahora, aplicando todos los pasos mencionados, se pueden estimar y asignar esqueletos de pose humana a cada persona en la imagen. [Figura 3.2]

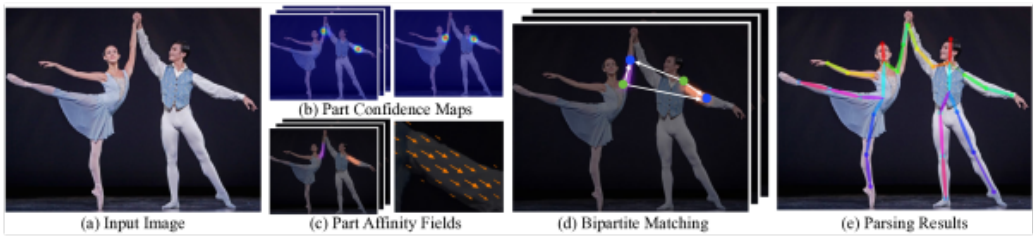

Figura 3.2. Pipeline de Openpose[23]

El proceso de OpenPose consta de múltiples tareas secuenciales:

1. Adquisición de la imagen completa como entrada (imagen o fotograma de video)
2. Dos redes neuronales convolucionales de dos ramas predicen conjuntamente mapas de confianza para la detección de partes del cuerpo
3. Estimación de los "Part Affinity Fields"(PAF) para la asociación de partes
4. Conjunto de emparejamientos bipartitos para asociar candidatos de partes del cuerpo
5. Ensamblar poses de cuerpo completo para todas las personas en la imagen.

La siguiente referencia ofrece una revisión mucho más profunda del funcionamiento de esta solución, profundizando en la arquitectura de capas de la IA implementada con referencias concretas en el código fuente [24].

En resumen, OpenPose es una importante biblioteca de detección de poses humanas en tiempo real que permite a las máquinas comprender y analizar visualmente a las personas y sus interacciones. Su versión ligera lo hace adecuado para aplicaciones de IA en el borde y para su implementación en hardware con recursos limitados.

3.2.2 MediaPipe [25]

MediaPipe representa un conjunto de herramientas de código abierto destinadas al desarrollo de aplicaciones de visión artificial, diseñadas tanto para dispositivos móviles como para plataformas de escritorio. Este conjunto de recursos fue presentado por Google en 2019 y se basa en tecnologías asociadas a Google Cloud.

Dentro del amplio abanico de funciones que MediaPipe ofrece para el procesamiento de imágenes y vídeos, destacan las siguientes: reconocimiento facial, reconocimiento de gestos, a análisis de movimiento, segmentación de imágenes y reconocimiento de objetos. [Figura 3.3]

Adicionalmente, MediaPipe proporciona una API que simplifica la incorporación de estas herramientas en aplicaciones de terceros. Entre las aplicaciones que aprovechan las capacidades de MediaPipe, encontramos ejemplos tales como: reconocimiento facial para desbloquear dispositivos móviles, control de dispositivos inteligentes a través de gestos y segmentación de imágenes y reconocimiento de objetos para aplicaciones de realidad aumentada o virtual.

MediaPipe emerge como una herramienta de gran potencia, adecuada para una extensa variedad de aplicaciones de visión artificial. Su naturaleza de código abierto y gratuidad la convierten en una herramienta accesible para desarrolladores de todos los niveles de experiencia.

Entre los beneficios notables de MediaPipe, destacan:

- **Facilidad de Uso:** MediaPipe pone a disposición una API intuitiva que facilita la integración de estas herramientas en aplicaciones desarrolladas por terceros.
- **Eficiencia:** Está especialmente diseñada para garantizar un rendimiento eficiente en dispositivos móviles y de escritorio.
- **Precisión:** MediaPipe se respalda en algoritmos avanzados que brindan resultados precisos.
- **Flexibilidad:** Su versatilidad permite su aplicación en una amplia variedad de casos relacionados con la visión artificial.

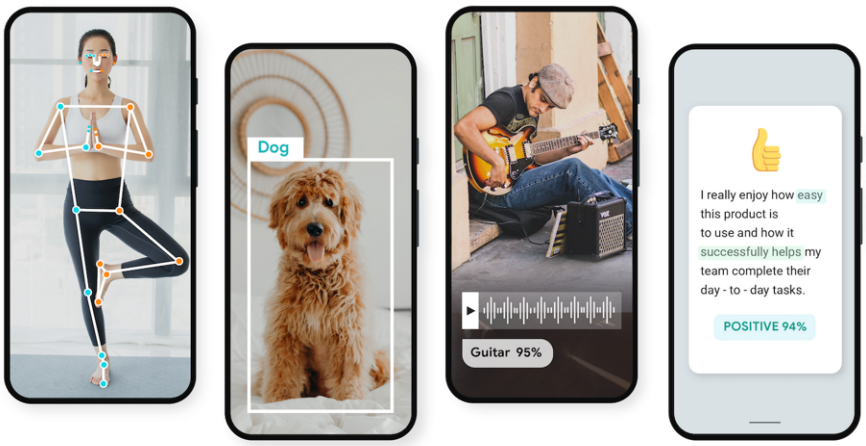

Figura 3.3. Muestra orientativa de algunas soluciones de Mediapipe[25]

3.2.3 Comparativa

En esta sección se proporciona una tabla comparativa que analiza como compiten OpenPose y MediaPipe en diferentes métricas. Estas han sido planteadas a raíz de los conocimientos obtenidos durante la investigación, además de incluir aquellas recurrentes en otros estudios similares.

Cuadro 3.1. Comparación entre MediaPipe y OpenPose

	MediaPipe	**OpenPose**
Desarrollador	Google	CMU-Perceptual-Computing-Lab
Licencia	Apache License 2.0	BSD 3-Clause License
Lenguajes	C++, Python	C++, Python
Precisión	Buen seguimiento facial y de mano	Preciso en detección de cuerpo
Velocidad	Rápido	Variable dependiendo del hardware
Dificultad de uso	Fácil de usar, muchas aplicaciones	Más complejo y especializado
Peso de procesado	Moderado	Alto
Soporte de OS	Windows, Linux, macOS	Windows, Linux, macOS, iOS
API y Bibliotecas	MediaPipe API, TensorFlow Lite	Caffe, OpenCV, PyTorch
Aplicaciones	Detección de cuerpo, seguimiento facial	Análisis de postura, seguimiento de mano

3.2.4 Conclusiones

La elección entre MediaPipe y OpenPose dependerá de varios factores, y la tabla realizada nos puede ayudar a tomar una decisión informada. A continuación, se explica por qué MediaPipe podría ser una elección adecuada en este contexto:

- **Velocidad:** MediaPipe tiene una buena capacidad de procesamiento en tiempo real, lo que es esencial para capturar movimiento en vivo y enviarlo a Unity sin una latencia significativa. Una velocidad más rápida garantiza una experiencia de usuario más fluida.
- **Dificultad de uso:** MediaPipe es considerado por su comunidad fácil de usar. Esto es beneficioso teniendo en cuenta la experiencia limitada en el área de trabajo y las restricciones de tiempo para el desarrollo del proyecto.
- **Peso de procesado:** MediaPipe tiene un peso de procesado moderado. Esto significa

que es eficiente en términos de recursos, lo que es importante si estás ejecutando todo en un PC personal. Un software con un peso de procesamiento moderado es menos exigente en términos de potencia de cómputo y memoria, lo que es beneficioso para mantener un rendimiento óptimo en esta situación.

- **API y Bibliotecas:** MediaPipe proporciona una API que es fácil de integrar con otros sistemas, como Unity. Esto simplifica la tarea de enviar datos de movimiento capturados en tiempo real.

En resumen, MediaPipe parece ser una elección adecuada para el proyecto debido a su velocidad, facilidad de uso, eficiencia en el procesamiento, soporte de sistemas operativos y capacidad de integración con Unity.

Mediapipe

En 2017, Google AI inició el proyecto interno “Project Marble” con el objetivo de desarrollar un marco para aplicaciones de visión por computadora de alta calidad. Este proyecto contó con un equipo compuesto por expertos en visión por computadora, aprendizaje automático y procesamiento de señales, que se basaron en su experiencia previa en proyectos como TensorFlow y OpenCV.

El año siguiente, en 2018, Project Marble evolucionó y se convirtió en MediaPipe, un proyecto de código abierto. Este paso marcó un hito importante en el mundo de la visión por computadora, al ofrecer una herramienta completa y fácil de usar para los desarrolladores.

MediaPipe se destacó por su arquitectura modular, que permitía a los desarrolladores ensamblar tuberías de inferencia de visión por computadora personalizadas. Además, incluía una amplia variedad de módulos prefabricados para tareas comunes de visión por computadora, como la detección de caras, el seguimiento de manos y el reconocimiento de objetos.

Desde su lanzamiento, MediaPipe experimentó un crecimiento constante en popularidad y se convirtió en una herramienta ampliamente adoptada por empresas como Google, Facebook, Microsoft y Amazon, así como por desarrolladores independientes.

En los años siguientes, MediaPipe continuó mejorando y adaptándose a las necesidades cambiantes del campo de la visión por computadora. En 2020, se lanzó una nueva versión con soporte para modelos de aprendizaje profundo, lo que permitió realizar tareas más complejas y precisas. En 2021, MediaPipe agregó soporte para dispositivos de hardware con recursos limitados, lo que lo hizo más accesible para aplicaciones móviles y de IoT. Y en 2022, se introdujeron nuevas características y mejoras, como modelos de aprendizaje profundo más potentes y un rendimiento optimizado.

A medida que avanzamos en 2023, MediaPipe continúa siendo un líder en el campo de la visión por computadora y se espera que siga creciendo en popularidad y adopción en los próximos años. Su arquitectura modular, rendimiento eficiente y flexibilidad lo convierten en una herramienta poderosa para una amplia variedad de aplicaciones. Además, su disponibilidad como código abierto fomenta la colaboración y el desarrollo comunitario, lo que contribuye aún más a su éxito continuo.

4.1 MediaPipe Hands[26]

MediaPipe Hands utiliza un conjunto de modelos de aprendizaje automático que trabajan juntos: un modelo de detección de palma que opera en la imagen completa y devuelve un cuadro delimitador orientado de la mano. Un modelo de landmarks [Figura 4.1] de mano que opera en la región de la imagen recortada definida por el detector de palma y devuelve landmarks de mano en 3D de alta fidelidad. Esta estrategia es similar a la utilizada en la solución MediaPipe Face Mesh, que utiliza un detector de rostros junto con un modelo de landmarks faciales.

Proporcionar la imagen de la mano recortada con precisión al modelo de landmarks de mano reduce drásticamente la necesidad de aumentación de datos (por ejemplo, rotaciones, traslaciones y escalas) y permite que la red dedique la mayor parte de su capacidad a la precisión en la predicción de coordenadas. Además, en nuestro flujo de trabajo, los recortes también pueden generarse en función de los landmarks de mano identificados en el fotograma anterior, y solo cuando el modelo de landmarks ya no puede identificar la presencia de la mano se invoca la detección de palma para volver a localizar la mano.

El flujo de trabajo se implementa como un gráfico de MediaPipe que utiliza un subgráfico de seguimiento de landmarks de mano del módulo de landmarks de mano y se renderiza utilizando un subgráfico de renderizado de mano dedicado. El subgráfico de seguimiento de landmarks de mano utiliza internamente un subgráfico de landmarks de mano del mismo módulo y un subgráfico de detección de palma del módulo de detección de palma.

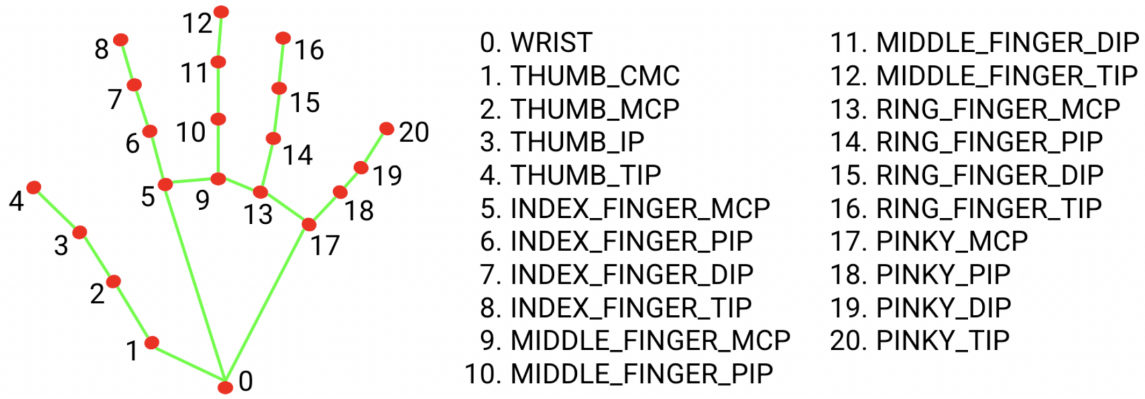

Figura 4.1. Landmarks obtenidas con Mediapipe Hands[25]

4.1.1 Modelos

Palm Detection Model

Para detectar las ubicaciones iniciales de las manos, se usa un modelo de detector *one-shot* optimizado para usos en tiempo real. Detectar manos es una tarea decididamente compleja: tanto el modelo ligero como el modelo completo deben funcionar en una variedad de tamaños de mano con un amplio rango de escala en relación con el marco de la imagen y ser capaces de detectar manos ocultas y autoocultas. Mientras que los rostros tienen patrones de alto contraste, por ejemplo, en la región de los ojos y la boca, la falta de tales características en las manos hace que sea comparativamente difícil detectarlas de manera confiable solo a partir de sus características visuales. En cambio, proporcionar contexto adicional, como características del brazo, cuerpo o persona, ayuda a localizar las manos con precisión.

Este método aborda los desafíos anteriores mediante diferentes estrategias. En primer lugar, se ha entrenado un detector de palmas en lugar de un detector de manos, ya que estimar cuadros delimitadores de objetos rígidos como palmas y puños es significativamente más simple que detectar manos con dedos articulados. Además, las palmas pueden ser modeladas usando cuadros delimitadores (*anchors* en la terminología de ML). En segundo lugar, se utiliza un extractor de características codificador-decodificador para tener en cuenta el contexto de la escena más grande, incluso para objetos pequeños.

Con las técnicas anteriores, se logra una precisión promedio del 95.7 % en la detección de palmas.

Hand Landmark Model

Después de la detección de la palma en toda la imagen, el modelo de landmarks subsecuente realiza una precisa localización de keypoints, es decir, coordenadas de las articulaciones de la mano en 3D, dentro de las regiones de la mano detectadas mediante regresión, es decir, predicción directa de coordenadas. El modelo aprende una representación interna de la postura de la mano consistente y es resistente incluso a manos parcialmente visibles y autoocultaciones.

El modelo está entrenado con alrededor de 30,000 imágenes etiquetadas a mano [Figura 4.2] del mundo real con 21 coordenadas en 3D, como se muestra a continuación (tomando el valor Z de un mapa de profundidad de imagen, si existe, para la coordenada correspondiente). Para abarcar mejor las posibles posturas de la mano y proporcionar supervisión adicional sobre la naturaleza de la geometría de la mano, también se renderizó un modelo de mano sintético de alta calidad sobre diversos fondos y se asignó a las correspondientes coordenadas en 3D.

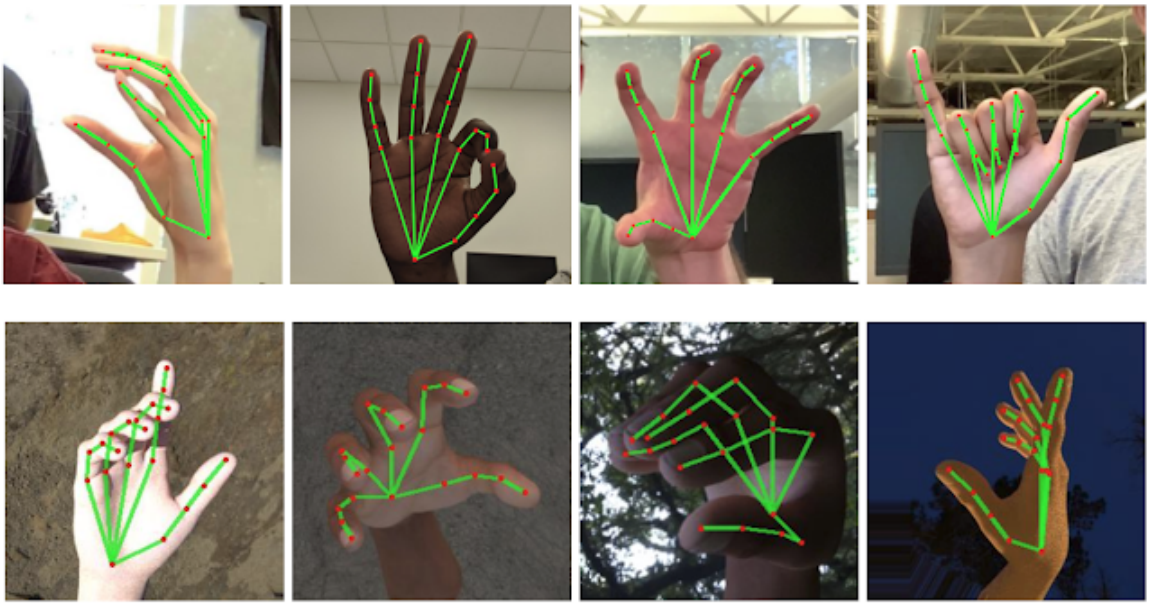

Figura 4.2. Imágenes reales y *renders* pasados al modelo para su entrenamiento. [27]

4.1.2 Opciones de configuración

- **`running_mode`**: Establece el modo en el que se ejecuta la solución.

  IMAGE: Detección de landmarks en una sola imagen.

  VIDEO: Detección de landmarks en un input de video.

  LIVE_STREAM: Detección de landmarks sobre un *stream* de video en vivo, por ejemplo obtenido de la webcam. En este modo, se debe llamar a `result_callback` para recibir los resultados de forma asíncrona.

  - Rango de valores: IMAGE, VIDEO, LIVE_STREAM
  - Valor por defecto: IMAGE

- **`num_hands`**: Máximo número de manos a detectar.

  - Rango de valores: int > 0
  - Valor por defecto: 1

- **`min_hand_detection_confidence`**: Mínimo *confidence score* para considerar la detección de una mano exitosa.

  - Rango de valores: 0.0 - 1.0

    - Valor por defecto: 0.5

- `min_hand_presence_confidence`: Mínimo *confidence score* para la presencia de una mano. En los modos Video y Live stream, si el *confidence score* es inferior, el `Hand Landmarker` llama a la detección de palma. Si no, un algoritmo ligero de `hand tracking` encuentra las manos para la subsecuente detección de landmarks.

    - Rango de valores: 0.0 - 1.0
    - Valor por defecto: 0.5

- `result_callback`:Establece el `result_listener` para recibir los resultados de forma asíncrona. Solo aplicable en el modo Live Stream.

Desarrollo práctico

5.1 Tecnologías empleadas

La elección de las tecnologías en un proyecto de desarrollo es un proceso crítico que puede influir significativamente en su éxito y viabilidad. Esto se debe a que las tecnologías seleccionadas deben estar alineadas con los objetivos del proyecto, ser eficientes y productivas, y considerar los costos totales de propiedad.

Además, la compatibilidad, escalabilidad, mantenibilidad y seguridad son factores clave a tener en cuenta. La experiencia del equipo de desarrollo y la comunidad en torno a las tecnologías también son determinantes, así como los requisitos legales y regulatorios específicos del proyecto. En resumen, tomar decisiones informadas sobre las tecnologías antes de iniciar un proyecto de desarrollo es esencial para garantizar su éxito y su capacidad para adaptarse a los desafíos a lo largo del tiempo.

En las siguientes secciones se detallan los motivos principales de la elección de cada una de las tecnologías utilizadas. Sin embargo, en la mayoría de los casos, se han escogido aquellas estudiadas a lo largo del grado, puesto que parte del objetivo de este proyecto es asentar los conocimientos preexistentes al respecto. Además de poder focalizar así la atención en los retos del desarrollo, en lugar de en el esfuerzo de usar tecnologías desconocidas.

5.1.1 Git

El uso de un sistema de control de versiones, como Git en este caso, proporciona una serie de beneficios sustanciales para cualquier proyecto de desarrollo de software:

- **Historial y seguimiento de cambios:** Git registra cada modificación realizada en el código fuente, lo que permite mantener un historial completo de cambios a lo largo del tiempo. Esto facilita la identificación de cuándo se realizó y por qué se hizo, lo que es fundamental para un trabajo extenso y la resolución de problemas.
- **Seguridad de datos:** Los repositorios Git almacenan datos de manera distribuida, lo que significa que si se produce una pérdida de datos en un servidor central, se pueden recuperar fácilmente desde las copias locales de los desarrolladores.
- **Gestión de versiones estables:** Git permite etiquetar versiones estables y realizar

ramas (branches) para el desarrollo de nuevas características o corrección de errores sin afectar la versión principal. Esto facilita la entrega de versiones funcionales y estables del software. En este caso, su versatilidad ha permitido el desarrollo de distintas pruebas paralelamente a un proyecto siempre funcional.

- **Retroceso a versiones anteriores:** Si se descubre un error en una versión más reciente del software, Git permite retroceder a versiones anteriores de manera sencilla. Esto es útil para solucionar problemas inesperados o mantener versiones anteriores en funcionamiento.

En este proyecto [Figura 5.1] su uso ha sido especialmente beneficioso a la hora de explorar nuevas *features*. Puesto que el código es extenso, el Sistema de Control de Versiones (SCV) nos permite desarrollar una nueva funcionalidad con la seguridad de no perturbar la última versión funcional. Y, una vez hecho el *merge* a la rama principal, identificar con facilidad donde se han realizado todos los cambios.

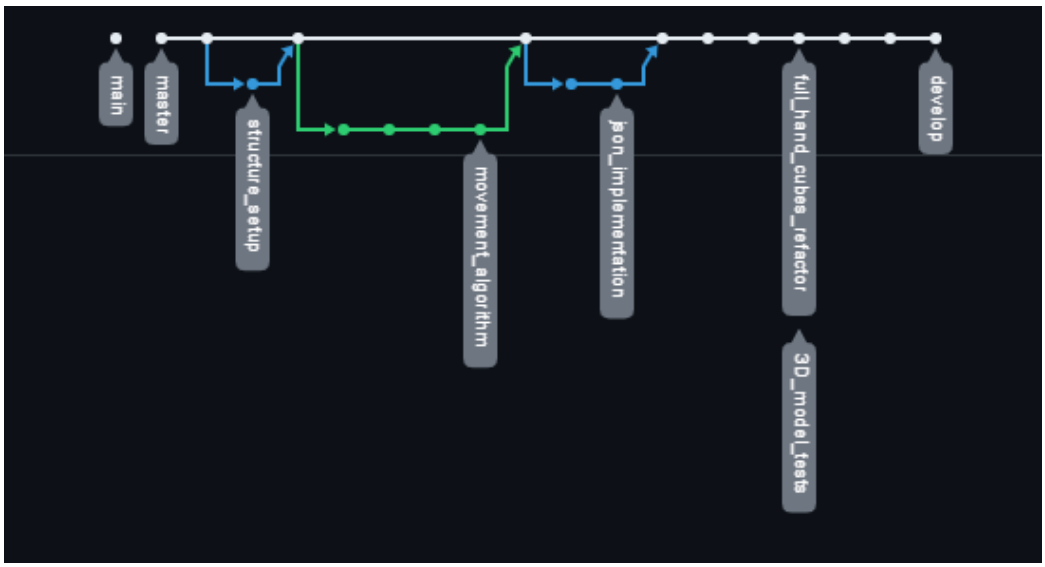

Figura 5.1. Network del repositorio empleado en GitHub

5.1.2 Python

Python es un lenguaje de programación extremadamente versátil y ampliamente utilizado que ofrece las siguientes ventajas:

- **Sintaxis legible y clara:** Python se destaca por su sintaxis limpia y legible, lo

que facilita la escritura y comprensión del código. La estructura clara de Python, que utiliza la indentación en lugar de llaves o corchetes, reduce la probabilidad de errores de sintaxis y hace que el código sea más fácil de mantener. Esto permite que los desarrolladores se centren en la lógica del programa en lugar de preocuparse por detalles de formato complicados.

- **Amplia biblioteca estándar:** Python incluye una amplia biblioteca estándar que abarca una variedad de tareas comunes de programación. Esto significa que los desarrolladores tienen acceso a una gran cantidad de funciones y módulos pre construidos que simplifican el desarrollo de aplicaciones. Ya sea para trabajar con archivos, gestionar redes, realizar operaciones matemáticas o interactuar con bases de datos, Python ofrece soluciones listas para usar, lo que ahorra tiempo y esfuerzo.
- **Gran comunidad y soporte:** Python tiene una comunidad de desarrolladores activa y comprometida en todo el mundo. Esto significa que hay una abundancia de recursos en línea, foros de ayuda y tutoriales disponibles para los desarrolladores que buscan orientación o solución de problemas. La comunidad también contribuye al continuo desarrollo y mejora del lenguaje, lo que asegura su relevancia a largo plazo.
- **Aprendizaje sencillo:** Python es conocido por ser un lenguaje de programación accesible para principiantes. Su sintaxis clara y legible, junto con una comunidad amigable, lo convierten en una excelente opción para aquellos que están comenzando a programar. La curva de aprendizaje suave de Python permite a los nuevos desarrolladores familiarizarse rápidamente con los conceptos de programación y comenzar a escribir código funcional en poco tiempo.
- **Código abierto y gratuito:** Python es un lenguaje de programación de código abierto y completamente gratuito. Esto significa que cualquiera puede descargar, usar y distribuir Python sin costos de licencia. La naturaleza de código abierto de Python fomenta la colaboración y la innovación, y garantiza que esté disponible para una amplia audiencia de desarrolladores y organizaciones sin restricciones financieras.

En este caso, además de los conocimientos preexistentes, se ha dado peso a la accesibilidad que otorgan su sintaxis clara y su aprendizaje sencillo, manteniendo así el enfoque al futuro uso de este proyecto.

OpenCV

OpenCV (cv2) en Python es una biblioteca esencial para el procesamiento de imágenes y videos. Ofrece una amplia variedad de funciones y herramientas, lo que la convierte en una elección poderosa para proyectos de visión por computadora. Además, cuenta con una comunidad activa que proporciona un valioso soporte y recursos, lo que facilita la implementación

exitosa de aplicaciones. Además, OpenCV se distribuye bajo una licencia de código abierto, lo que la hace accesible y versátil para desarrolladores y organizaciones sin restricciones financieras.

5.1.3 Sockets UDP

La transferencia de datos por *sockets* User Datagram Protocol (UDP), se trata de un protocolo de comunicación de red sin conexión. Cuando un programa desea enviar datos a través de UDP, crea un datagrama UDP que contiene los datos que se van a enviar. Este datagrama se empaqueta y se envía a la dirección IP y al puerto de destino especificados. A diferencia de Transmision Control Protocol (TCP), UDP no establece una conexión de tres vías antes de la transmisión de datos y no garantiza la entrega de paquetes ni el orden en que se entregan.

En el lado del receptor, se escucha en un puerto específico para recibir datagramas UDP entrantes. Cuando llega un datagrama, se procesa y se toma la acción correspondiente según la aplicación. Debido a que UDP no tiene mecanismos de control de flujo ni garantía de entrega, es responsabilidad de la aplicación gestionar la integridad y la secuencia de los datos si es necesario.

En resumen, los sockets UDP son una elección adecuada para aplicaciones donde la velocidad y la simplicidad son prioritarias, y donde la pérdida ocasional de datos no es crítica. Funcionan mediante el envío de datagramas sin conexión entre un emisor y uno o varios receptores, lo que los hace aptos para una variedad de situaciones, incluidas las comunicaciones entre Python y Unity.

Estas son las ventajas que se han tenido en cuenta al escoger entre el uso de UDP y TCP:

- **Velocidad y eficiencia:** UDP es un protocolo orientado a la velocidad y la eficiencia. No tiene la sobrecarga de establecer y mantener una conexión como TCP. En cambio, los datagramas UDP se envían sin confirmación de recepción, lo que permite una comunicación más rápida y menos latencia. Esto es particularmente útil en aplicaciones en tiempo real, donde la velocidad de transmisión es crucial.
- **Simplicidad:** UDP es más simple que TCP en términos de configuración y manejo. No se requiere un proceso de establecimiento de conexión y, por lo tanto, es más fácil de implementar y depurar, lo que lo hace adecuado para aplicaciones donde la simplicidad es una prioridad.
- **Tolerancia a la pérdida de paquetes:** Aunque UDP no garantiza la entrega de datos ni el orden en que se entregan, esto puede ser una ventaja en algunas situaciones.

En juegos y aplicaciones en tiempo real, a veces es preferible perder algunos datos obsoletos en lugar de retrasar la entrega de datos nuevos.

- **Transmisión de datos pequeños y ráfagas:** UDP es eficiente para la transmisión de pequeñas cantidades de datos o ráfagas de datos, lo que lo hace adecuado para enviar actualizaciones de posición, por ejemplo.

5.1.4 Unity

Unity es uno de los motores de desarrollo más populares y ampliamente utilizados en la industria de los videojuegos y la realidad virtual, pero también se utiliza en una variedad de aplicaciones más allá del entretenimiento. Destaca por su interfaz de usuario intuitiva y su facilidad de uso en comparación con otros motores de desarrollo. Esto lo convierte en una excelente opción tanto para desarrolladores principiantes como para aquellos con experiencia previa en desarrollo de juegos y aplicaciones 3D.

El entorno tiene una comunidad de desarrollo activa y una gran cantidad de recursos disponibles en línea, incluyendo tutoriales, documentación y foros de ayuda. Esto facilita el aprendizaje y la resolución de problemas durante el desarrollo del proyecto. Además, es compatible con una amplia gama de plataformas, incluyendo PC, consolas, dispositivos móviles, Realidad Virtual (VR) y Realidad Aumentada (AR). Esto significa que puedes desarrollar tu proyecto una vez y publicarlo en múltiples plataformas, lo que ahorra tiempo y recursos.

Unity Labs trabaja constantemente en mejoras y actualizaciones para el motor, lo que garantiza que esté al día con las últimas tendencias y tecnologías en desarrollo 3D. Del mismo modo, es ampliamente adoptado en la industria del desarrollo de videojuegos y aplicaciones 3D. Por lo tanto, aprender y trabajar con Unity puede aumentar tus oportunidades profesionales y colaborativas. Por último, Unity utiliza C# como lenguaje de programación principal, que es ampliamente conocido y utilizado en la industria.

Para la realización de este proyecto se ha utilizado Unity 2022.3.8f1. Dado que no se requerian librerías descontinuadas y se trata de un trabajo muy enfocado a su futuro desarrollo y expansión, se ha optado por el uso de la última versión con Long Term Supprt (LTS) en el momento de la creación del proyecto.

5.1.5 Hardware

En consecuencia con los objetivos de este proyecto, para su desarrollo y uso se han empleado únicamente un PC de gama media y su Webcam predeterminada. Del mismo modo, estos

son los únicos requerimientos de *hardware* para su ejecución. En este caso se trata de un LG Gram 15Z990-V con las siguientes especificaciones:

Cuadro 5.1. Especificaciones del LG Gram 15Z990-V

Procesador	Intel Core i7
Memoria RAM	16 GB
Almacenamiento	256 GB SSD
Pantalla	15.6 pulgadas Full HD
Peso	1.09 kg
Duración de la batería	Hasta 18.5 horas
Sistema operativo	Windows 11
Año	2019

Cuadro 5.2. Especificaciones de la Webcam del LG Gram 15Z990-V

Resolución de la Webcam	720p
Tipo de Webcam	Integrada
Funciones Adicionales	Reducción de ruido

5.2 Metodología

Utilizar una metodología iterativa en un proyecto es una estrategia efectiva que se centra en el desarrollo progresivo y la mejora continua. En este contexto, significa un compromiso a abordar el proyecto en pequeñas etapas o iteraciones, en lugar de intentar resolver todo el problema de una sola vez.

Este enfoque tiene varias ventajas notables. En primer lugar, reduce los riesgos al dividir el proyecto en iteraciones más pequeñas. Esto permite abordar y resolver los riesgos potenciales en etapas tempranas, lo que reduce la posibilidad de que surjan problemas graves en etapas posteriores del proyecto. Además, cada iteración resulta en una entrega funcional y utilizable del producto, lo que proporciona avances medibles y tangibles en cada etapa.

La metodología iterativa también facilita la obtención de retroalimentación continua de los usuarios o partes interesadas. Esta retroalimentación temprana es esencial para adaptar el proyecto según las necesidades reales, evitando que se desvíe de su objetivo principal. Además, si surgen cambios en los requisitos o en la visión del proyecto, una metodología iterativa permite realizar ajustes más fácilmente en comparación con un enfoque de desarrollo monolítico.

Otra ventaja clave radica en la mayor eficiencia y calidad. Al enfocarse en tareas más

pequeñas y manejables en cada iteración, se puede prestar una mayor atención a la calidad y a la optimización del código y la funcionalidad. La retroalimentación y la experiencia acumulada en cada iteración se utilizan para mejorar y refinar el proyecto a medida que avanza, lo que contribuye a una solución final más robusta y efectiva.

En resumen, la metodología iterativa implica abordar el problema en pequeñas etapas secuenciales, asegurándose de tener soluciones funcionales antes de avanzar. Esto proporciona ventajas como la reducción de riesgos, retroalimentación continua, entregas incrementales y mayor adaptabilidad, lo que contribuye a la eficiencia y la calidad del proyecto. Además, permite que el desarrollo se ajuste a medida que avanza y mejore continuamente la solución.

Como ejemplo, en la siguiente figura [Figura 5.2] se muestra uno de los primeros estados del proyecto de Unity, dónde se realizaron pruebas de movimiento con tres cubos que representaban tres *Landmarks* del dedo índice.

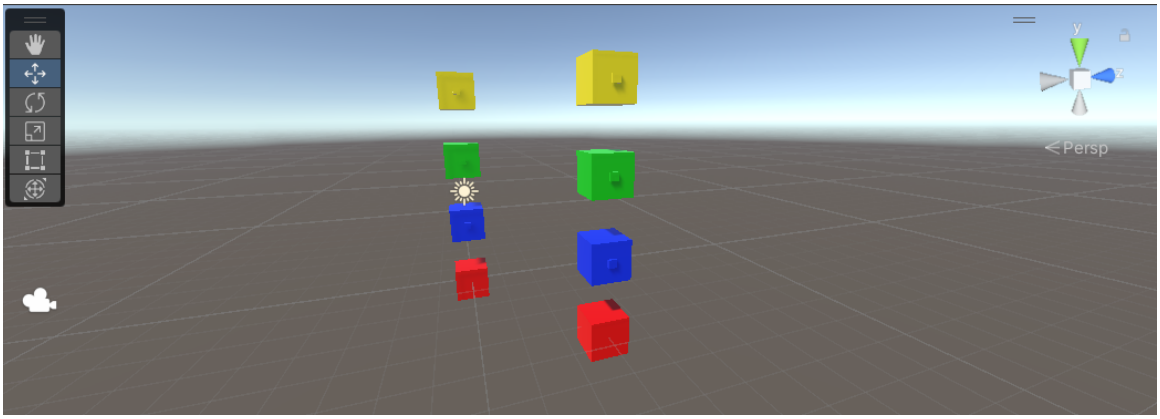

Figura 5.2. Una de las primeras versiones del proyecto.

5.3 Obtención de datos - Python

5.3.1 `mediapipe_capture.py`

El desarrollo de la primera parte de la aplicación, la captura de los datos y su evio, se ha realizado en Python, en el Entorno de Desarrollo Integrado (IDE) Visual Studio Code.

Este código se encarga de capturar video desde la cámara web, detectar las manos en el flujo de video utilizando la biblioteca MediaPipe, extraer información de las manos detectadas, y luego enviar esta información en formato JSON a través de un socket UDP a una dirección

IP y puerto especificados.

El código sigue la siguiete estructura:

1. Se importan las bibliotecas necesarias, incluyendo OpenCV (cv2) para el manejo de video y gráficos, MediaPipe (mp) para el procesamiento de manos y mediapipe.solutions para los componentes de MediaPipe.
2. Se definen las constantes `UDP_IP` y `UDP_PORT` que especifican la dirección IP y el puerto UDP al que se enviarán los datos JSON.
3. La función `process_and_send_data` se define para procesar la imagen de video, detectar las manos en ella, extraer información relevante y enviarla a través del socket UDP.
4. Se configura la cámara web utilizando `OpenCV` con `cv2.VideoCapture(0)` para capturar el video desde la cámara principal (0).
5. Se inicia el bucle principal con `while cap.isOpened()`. En cada iteración del bucle, se captura un fotograma del video.
6. Se llama a la función `process_and_send_data` para procesar la imagen capturada y enviar la información de las manos detectadas a través del socket UDP.
7. Si se presiona la tecla Esc (código ASCII 27), el bucle se rompe y se liberan los recursos de la cámara y el socket.
8. Finalmente, se cierra la cámara (`cap.release()`), se cierra el socket (`sock.close()]`) y se destruyen todas las ventanas abiertas por OpenCV (`cv2.destroyAllWindows()`).

La función **`process_and_send_data`** realiza las siguientes acciones:

1. Convierte la imagen de BGR a RGB para que sea compatible con MediaPipe.
2. Procesa la imagen con MediaPipe para detectar las manos.
3. Recopila datos sobre las manos detectadas, incluyendo información sobre la mano (izquierda o derecha), coordenadas 2D de los landmarks (puntos clave de la mano) y coordenadas 3D en el espacio.
4. Organiza estos datos en una estructura JSON.
5. Imprime el mensaje JSON en la consola para visualización. Este paso se incluye a modo de comprobación del correcto funcionamiento del algoritmo.

6. Envía el mensaje JSON codificado a través del socket UDP a la dirección IP y puerto especificados.
7. Muestra la imagen con las manos detectadas utilizando OpenCV.

En resumen, este código permite la detección de manos en tiempo real desde la cámara web, la extracción de datos relevantes de las manos y el envío de estos datos a través de un socket UDP para su posterior procesamiento y visualización.

5.4 Procesado de datos - Unity

Una vez preparada la captura y retransmision de datos, se ha creado un proyecto de Unity con el objetivo de procesar y representar los mismos. En esta parte del desarrollo se incluye en la escena una representación simplificada de cada mano, con 21 esferas para representar las articulaciones correspondientes a los landmarks capturados.

Cada una de las esferas tiene como componente el script `MyJoint.cs`, del mismo modo que las manos contienen `MyHand.cs`. Con esto, se reciben los datos y se procesan, para luego asignar a cada objeto sus rotaciones y posiciones pertinentes.

Los scripts que no se corresponden con ningún objeto se incluyen en la escena en un `EmptyObject`, como hijos de otro llamado Scripts. [5.3]

5.4.1 `MpDataReceiver.cs`

Este script configura una comunicación UDP para recibir datos y muestra los datos recibidos en la consola de Unity, a modo de comprobación. Los datos recibidos se almacenan en la variable `text` y se pueden utilizar en otras partes de tu proyecto de Unity.

1. Se importan las bibliotecas necesarias, incluyendo `System`, `System.Net`, `System.Net.Sockets`, `System.Text`, `System.Threading`, y `UnityEngine`.
2. Se crea una clase `MpDataReceiver` que hereda de `MonoBehaviour`, lo que significa que este script se adjuntará a un objeto en Unity.
3. Se definen variables y objetos para manejar la comunicación UDP:
    a) `Thread receiveThread`: Un hilo que se utilizará para recibir datos UDP en segundo plano.

   b) `UdpClient client`: El cliente UDP que se utilizará para recibir datos.
   c) `readonly int port = 5060`: El número de puerto UDP al que se conectará el cliente UDP.
   d) `readonly int port = 5060`: El número de puerto UDP al que se conectará el cliente UDP.
   e) `public string text`: Una variable pública que almacenará los datos recibidos en formato JSON. Se inicializa con un valor de ejemplo.

4. La función **`InitUDP()`** se encarga de inicializar la comunicación UDP. Crea un nuevo hilo (`receiveThread`) que ejecutará la función `ReceiveData()` para recibir datos UDP.

5. La función **`ReceiveData()`** se ejecuta en un bucle infinito y se encarga de recibir datos UDP. Utiliza el objeto `UdpClient` para recibir datos en el puerto especificado. Los datos recibidos se convierten de bytes a una cadena de texto utilizando UTF-8 y se almacenan en la variable `text`. También se registran mensajes de depuración en la consola de Unity para mostrar los datos recibidos.

6. En la función **`Start()`**, se llama a InitUDP() para iniciar la comunicación UDP cuando el script se inicia.

5.4.2 `MpData.cs`

Este código define varias clases que se utilizan para almacenar los datos recibidos en un formato estructurado.

1. **`class MpData`**: Esta clase se utiliza para representar el conjunto de datos que contiene información sobre las manos. Contiene un campo público llamado `data` que es un *array* de objetos `HandData`. Esto significa que puede almacenar datos de múltiples manos en un solo objeto MpData.

2. **`class HandData`**: Esta clase se utiliza para representar información específica sobre una mano. Contiene dos campos públicos:

   a) `handedness`: Un objeto de la clase `Handedness` que almacena información sobre la lateralidad (izquierda o derecha) y su puntuación de clasificación.
   b) `landmarks`: Una matriz de *floats* (`float[][]`) que almacena las coordenadas de los landmarks (puntos clave) de la mano. Cada landmark tiene tres coordenadas (x, y, z), y estas coordenadas se almacenan en un arreglo de *floats*.

3. **`class Handedness`**: Esta clase se utiliza para representar información sobre la lateralidad. Contiene dos campos públicos:

a) `classification`: Una cadena de texto que indica la clasificación de la mano ("Left"para izquierda o "Right"para derecha).

b) `score`: Un valor *float* (entre 0.0 y 1.0) que representa la puntuación de la confianza en la clasificación.

5.4.3 `MyJoint.cs`

Esta clase se ha diseñado para ajustar la posición y rotación del `GameObject` (representado por `Target`) para que coincida con la posición y rotación del objeto al que está adjunto (`MyJoint`). De este modo podemos definir todas las posiciones y rotaciones necesarias y después asignarlas el conjunto.

Esta es una explicación en más detalle:

1. `public Transform Target`: Esta es una variable pública que representa el objeto (`Transform`) al que se ajustará la posición y rotación. Se asigna más tarde desde el Inspector de Unity.
2. `public void UpdateTransform()`: Esta es la función que se llama para actualizar la posición y rotación del objeto `Target` para que coincida con la posición y rotación del objeto al que está adjunto (`MyJoint`).
3. `if (Target == null) return`: Esta línea de código verifica si la variable Target está configurada como nula (no se ha asignado un objeto de destino). Si es nula, la función sale temprano y no realiza ningún ajuste.
4. `Target.SetPositionAndRotation(this.transform.position, this.transform.rotation)`: Esta línea de código utiliza la función `SetPositionAndRotation` del componente `Transform` del objeto de destino (`Target`) para establecer su posición y rotación. La posición y rotación se establecen en base a la posición y rotación actual del objeto al que está adjunto (`MyJoint`). En otras palabras, esta línea de código copia la posición y rotación de `MyJoint` en el objeto de destino.

5.4.4 `DepthCalibrator.cs`

Este script representa una clase que se utiliza para calibrar la profundidad de la mano en función de la longitud del pulgar utilizando una simple fórmula lineal.

1. `private float m` y `private float c`: Variables privadas que almacenan los coeficientes de la ecuación lineal. m representa la pendiente de la línea y c representa la ordenada al origen.
2. `public DepthCalibrator(float m, float c)`: Este es el constructor de la clase DepthCalibrator. Toma dos argumentos: m y c, que son los coeficientes de la ecuación lineal. Cuando creas una instancia de DepthCalibrator, debes proporcionar estos valores para calibrar el objeto.
3. `public float GetDepthFromThumbLength(float length)`: Esta es una función pública que toma la longitud del pulgar como argumento y devuelve la profundidad calculada en función de la longitud.
4. `if (length == 0) return 0`: Esta línea de código verifica si la longitud proporcionada es igual a cero. Si es así, devuelve cero como profundidad. Esto evita divisiones por cero en el siguiente cálculo.
5. `return m / length + c`: Esta es la parte principal de la función. Calcula la profundidad utilizando la ecuación lineal *m / length + c*. Divide la pendiente `m` por la longitud `length` y luego suma la ordenada al origen `c`. Esto da como resultado la profundidad estimada en función de la longitud proporcionada.

5.4.5 MyHand.cs

Este script contiene toda la lógica asociada a una mano, así como todas sus propiedades.

1. **Variables:**
   - `public HandData handData`: Esta variable pública almacena datos de la mano.
   - `public string handedness`: Almacena la clasificación de la lateralidad (izquierda o derecha).
   - `public bool FlipXY = false`, `public bool NegateX = false` y `public bool NegateY = false`: Ajustes para la traducción de los ejes de imagen de Mediapipe a los ejes de la escena en Unity.
   - `private float thumbModelLength = 0.05f`: Almacena la longitud del pulgar en la escena.
   - `private float scale`: Se utiliza para almacenar la escala aplicada al modelo de mano.
   - `private DepthCalibrator depthCalibrator = new DepthCalibrator(-0.0719f, 0.439f)`: Crea una instancia de la clase DepthCalibrator con coeficientes específicos para calibrar la profundidad en función de la longitud del pulgar.

- `private MyJoint[] myJoints`: Un *array* de objetos `MyJoint` utilizados para actualizar las transformaciones de las articulaciones de la mano.
- `private float smoothingFactor = 0.1f`: Factor de suavizado utilizado en el proceso de actualización de posiciones.
- `private List<Vector3>smoothedLandmarks` y `private List<Vector3>scaledLandmarks`: Listas utilizadas para almacenar landmarks suavizados y escalados.

2. **`Start()`**: En este método, se obtienen todas las referencias a los objetos `MyJoint` que están bajo el objeto `MyHand`. Esto se hace para poder actualizar sus transformaciones más adelante.

3. **`SetInPlace()`**: Esta función se llama para actualizar y ajustar la posición y rotación del modelo de mano en función de los landmarks proporcionados en `handData`. Aquí se realizan varias acciones:

   - Los datos de landmarks se convierten en una lista de vectores `Vector3` con `LandmarksToVector3List(handData.landmarks)`.
   - Se actualizan las posiciones de las articulaciones de la mano (`UpdatePositions(landmarks)`).
   - Se calcula la escala del modelo de mano(`UpdateScale(landmarks)`).
   - Se actualiza la rotación de la muñeca(`UpdateWristRotation()`).
   - Se actualizan las transformaciones de los objetos `MyJoint` iterando sobre `MyJoints` (`j.UpdateTransform()`).

4. **`UpdatePositions(List<Vector3>landmarks)`** realiza una serie de cálculos y transformaciones para actualizar las posiciones de las articulaciones de la mano y ajustar el modelo de mano en Unity:

   a) `if (smoothedLandmarks == null) { smoothedLandmarks = new List<Vector3>(landmarks)}`: Se verifica si la lista `smoothedLandmarks` es nula. Si es así, se inicializa como una nueva lista copiando los valores de `landmarks`. Esta lista se utiliza para almacenar landmarks suavizados.

   b) `if (scaledLandmarks == null) { scaledLandmarks = new List<Vector3>(smoothedLandmarks)}`: Se verifica si la lista `scaledLandmarks` es nula. Si es así, se inicializa como una nueva lista copiando los valores de `smoothedLandmarks`. Esta lista se utilizará para almacenar landmarks escalados.

   c) `var offset = landmarks[0]`: Se toma el primer landmark (`landmarks[0]`) como punto de referencia para calcular las diferencias entre las posiciones de los landmarks y aplicarlas posteriormente.

   d) El bucle `for (int i = 1; i <landmarks.Count; i++)` itera a través de todos los landmarks (a partir del segundo landmark, ya que el primero se usó como punto de referencia).

e) `var x = landmarks[i].x - offset.x, var y = landmarks[i].y - offset.y, var z = landmarks[i].z - offset.z`: Se calcula la diferencia entre las coordenadas de `landmarks[i]` y las coordenadas de referencia `offset` en cada dimensión (X, Y y Z).

f) `if (x == 0 y == 0 z == 0) return`: Se verifica si las diferencias son todas cero, lo que podría indicar una posición incorrecta de landmarks. Si es así, la función sale temprano sin realizar más cálculos.

g) `if (NegateX) x *= -1;, if (NegateY) y *= -1, if (FlipXY) (x, y) = (y, x)`: Se aplican transformaciones opcionales a las coordenadas `x` e `y` en función de las variables `NegateX`, `NegateY` y `FlipXY`. Estas transformaciones permiten ajustar la orientación del modelo de mano según sea necesario.

h) `smoothedLandmarks[i] = Vector3.Lerp(smoothedLandmarks[i], new Vector3(x, y, z), 1 - smoothingFactor)`: Se aplica un suavizado a las coordenadas de `landmarks[i]` utilizando la función `Lerp` de Unity. El suavizado se controla mediante el parámetro `smoothingFactor`.

i) `scaledLandmarks[i] = smoothedLandmarks[i] * scale`: Las coordenadas suavizadas se escalan multiplicándolas por el valor de `scale`. Esto ajusta el tamaño del modelo de mano en función de la longitud del pulgar detectada.

j) `myJoints[i].transform.localPosition = scaledLandmarks[i]`: Se actualizan las transformaciones locales de los objetos `MyJoint` correspondientes a las articulaciones de la mano, estableciendo sus posiciones locales según las coordenadas escaladas.

k) `float depth = depthCalibrator.GetDepthFromThumbLength(scale)`: Se calcula la profundidad en función de la longitud del pulgar detectada utilizando la instancia de `depthCalibrator`.

l) `Vector3 handPosition = new(offset.x * scale, (-offset.y * scale) + 1, depth * scale)`: Se calcula la posición de la mano ajustada en función de la escala y la profundidad calculada. Esto coloca la mano en la posición correcta en el espacio 3D.

m) `this.transform.localPosition = Vector3.Lerp(this.transform.localPosition, handPosition, 1 - smoothingFactor)`: Se aplica un suavizado final a la posición de la mano en función de la posición calculada previamente. Esto asegura una transición suave entre las posiciones de la mano.

5. `UpdateScale(List<Vector3>landmarks)` calcula la escala del modelo de mano en función de la longitud detectada del pulgar. A continuación, se explica en detalle el código dentro de esta función:

   a) `var pointA = landmarks[0], var pointB = landmarks[1];`: Se toman los dos primeros landmarks de la lista `landmarks` como puntos de referencia para medir la longitud del pulgar.

b) `var thumbDetectedLength = Vector3.Distance(pointA, pointB)`: Se calcula la distancia entre los dos puntos de referencia utilizando la función `Vector3.Distance`. Esta distancia representa la longitud detectada del pulgar.

c) `if (thumbDetectedLength == 0) return`: Se verifica si la longitud del pulgar detectada es cero. Si es así, la función sale temprano sin calcular la escala. Esto evita divisiones por cero y problemas de escalado inapropiado.

d) `scale = thumbModelLength / thumbDetectedLength`: Se calcula la escala dividiendo la longitud del modelo del pulgar (`thumbModelLength`) por la longitud detectada del pulgar (`thumbDetectedLength`). Esto ajusta el tamaño del modelo de mano en función de la longitud del pulgar detectada.

6. **`UpdateWristRotation()`** se encarga de calcular la rotación de la muñeca del modelo de mano en función de las posiciones de los dedos de la siguiente forma:

   a) `var wristTransform = myJoints[0].transform`, `var indexFinger = myJoints[5].transform.position`, `var middleFinger = myJoints[9].transform.position`: Se obtienen las referencias a transformaciones y posiciones relevantes para el cálculo de la rotación de la muñeca. `wristTransform` representa la transformación de la muñeca, `indexFinger` y `middleFinger` son las posiciones de los dedos índice y corazón, respectivamente.

   b) `var vectorToMiddle = middleFinger - wristTransform.position`, `var vectorToIndex = indexFinger - wristTransform.position`: Se calculan vectores que apuntan desde la posición de la muñeca hacia las posiciones de los dedos corazón e índice.

   c) `Vector3.OrthoNormalize(ref vectorToMiddle, ref vectorToIndex)`: Se normalizan los vectores `vectorToMiddle` y `vectorToIndex`, asegurando que sean ortogonales entre sí. Esto es importante para calcular una rotación adecuada.

   d) `Vector3 normalVector = Vector3.Cross(vectorToIndex, vectorToMiddle)`: Se calcula un vector normal utilizando el producto cruz (`Cross`) de los vectores `vectorToIndex` y `vectorToMiddle`. Este vector normal define la orientación de la muñeca.

   e) `wristTransform.rotation = Quaternion.LookRotation(normalVector, vectorToIndex)`: Se asigna la rotación de la muñeca (`wristTransform.rotation`) utilizando la función `Quaternion.LookRotation`. Esta función utiliza el vector normal y el vector que apunta al dedo índice para determinar la orientación de la muñeca.

5.4.6 SceneController.cs

El `SceneController` es el script encargado de gestionar la información recibida y aplicarla a las manos izquierda y derecha en la escena. A continuación, se detalla el funcionamiento del código:

1. `public MyHand leftHand`, `public MyHand rightHand`: Estas variables públicas representan las manos izquierda y derecha en la escena. Están conectadas a objetos de tipo `MyHand` que controlan la representación de las manos en Unity.
2. `public MpDataReceiver mpDataReceiver`, `public MpData mpData`: `mpDataReceiver` es un objeto que recibe datos de movimiento de las manos desde un servidor UDP. `mpData` almacena los datos de las manos en un formato adecuado para su uso en Unity.
3. `public bool invertedCamera`: Esta variable booleana determina si la cámara está invertida o no. Dependiendo de esta configuración, se ajusta la orientación de las manos.
4. `private void Update()`: En el método `Update()`, se ejecutan las siguientes acciones:
   a) `DeserializeAndAssignData()`: Se deserializan los datos recibidos desde el servidor UDP y se asignan a `mpData`.
   b) `leftHand.SetInPlace()`, `rightHand.SetInPlace()`: Se actualiza la posición y la orientación de las manos izquierda y derecha en la escena, respectivamente.
5. `private void DeserializeAndAssignData()`: En esta función se realiza la deserialización de los datos JSON recibidos desde el servidor UDP. El proceso es el siguiente:
   a) Se obtiene el JSON de `mpDataReceiver.text`.
   b) Se intenta deserializar el JSON en un objeto de tipo `MpData`, que contiene información sobre las manos detectadas.
   c) Se manejan excepciones en caso de errores en la deserialización y se muestra un mensaje de error en la consola.
   d) Se itera a través de los datos de cada mano en `mpData.data`, ajustando la clasificación (`handedness`) si la cámara está invertida y asignando los datos de la mano a la mano izquierda o derecha según corresponda.
6. `private string InvertHandedness(string handedness)`: Esta función invierte la clasificación de la mano (izquierda por derecha o viceversa) si la cámara está invertida. Retorna la clasificación ajustada.
7. `private void AssignHandData(HandData handData, string handedness)`: Dependiendo de la clasificación de la mano (`handedness`), esta función asigna los datos de la mano a la mano izquierda o derecha (`leftHand.handData` o `rightHand.handData`).

5.5 Escena - Unity

En este apartado se expondrán los scripts relacionados con la Interfaz de Usuario (UI), así como la configuración de la escena en GUI de Unity.

5.5.1 `UIController.cs`

El `UIController` se encarga de gestionar la interfaz de usuario (UI) en la escena. Este es el funcionamiento del código:

1. `public MyHand leftHand`, `public MyHand rightHand`: Estas variables públicas representan las manos izquierda y derecha en la escena. Están conectadas a objetos de tipo `MyHand` que controlan la representación de las manos en Unity.
2. `public TextMeshProUGUI leftHandText`, `public TextMeshProUGUI rightHandText`: Estos son objetos de texto (TextMeshProUGUI) utilizados para mostrar la información de lateralodad sobre las manos izquierda y derecha en la UI.
3. `public Slider smoothnessSlider`: Este es un objeto deslizante (Slider) que permite al usuario ajustar la suavidad (*smoothness*) de los movimientos de las manos.
4. `void Start()`:
   a) Se verifica si los objetos de texto `leftHandText` y `rightHandText` no son nulos (`!= null`). Si no son nulos, se establecen textos iniciales para mostrar la clasificación y puntuación de las manos.
5. `void Update()`:
   a) Se verifica si los objetos de texto `leftHandText` y `rightHandText` no son nulos (`!= null`). Si no son nulos, se actualizan los textos para mostrar la clasificación y puntuación actualizadas de las manos izquierda y derecha.
   b) Se ajusta el valor de `smoothingFactor` de las manos izquierda y derecha al valor actual del `smoothnessSlider`, lo que permite al usuario controlar el factor de suavizado de los movimientos de las manos.

En resumen, este controlador de UI se encarga de actualizar la información mostrada sobre las manos izquierda y derecha en la interfaz de usuario y permite al usuario ajustar la suavidad de los movimientos de las manos mediante un *slider*.

5.5.2 `CameraController.cs`

El `CameraController` es un componente de Unity encargado de gestionar la posición y el comportamiento de la cámara en la escena. De esta forma se garantiza que las manos, que se pueden desplazar a lo largo de la escena, estén siempre centradas en el *display* y se aprecie su movimiento de forma óptima.

1. `public MyHand leftHand`, `public MyHand rightHand`: Estas variables públicas representan las manos izquierda y derecha en la escena. Están conectadas a objetos de tipo `MyHand` que controlan la representación de las manos en Unity.
2. `public Camera mainCamera`: Este es el objeto de la cámara principal que se utilizará en la escena.
3. `public float zoomSpeed = 2.0f`, `public float minFOV = 5.0f`, `public float maxFOV = 60.0f`: Estas variables públicas permiten ajustar la velocidad de zoom de la cámara, así como los valores mínimos y máximos del campo de visión (FOV) de la cámara.
4. `private Vector3 cameraTargetPosition`, `private Vector3 cameraInitialPosition`: Estas variables privadas almacenan la posición objetivo de la cámara y la posición inicial de la cámara en la escena.
5. `void Start()`: En el método `Start()`, se asigna la posición inicial de la cámara a la variable `cameraInitialPosition` para recordar la posición original de la cámara.
6. `void Update()`:
   a) Se calcula el punto medio (`midpoint`) entre las posiciones de las manos izquierda y derecha en la escena.
   b) Se calcula la distancia (`distance`) entre las manos izquierda y derecha.
   c) Se establece la posición objetivo de la cámara (`cameraTargetPosition`) en función del punto medio y la distancia entre las manos. La cámara se posiciona de manera que mire hacia el punto medio, con un ligero desplazamiento hacia arriba y hacia adelante.
   d) Se utiliza la función `Vector3.Lerp()` para suavizar el movimiento de la cámara hacia la posición objetivo. El valor `Time.deltaTime` se utiliza para controlar la velocidad de la interpolación.

En resumen, este controlador de cámara ajusta la posición de la cámara en función de la posición de las manos izquierda y derecha en la escena, lo que permite a la cámara seguir el movimiento de las manos y mantener el punto medio entre ellas en el centro de la vista de la cámara.

5.5.3 Escena

Dado que el objetivo de este proyecto se centraba en la investigación e implementación de la captura de movimiento, se ha decidido mantener una UI mínima. En esta se pueden ver los dos *clusters* de esferas que representan las articulaciones (*joints*) de las manos y dedos.

Cada `GameObject` que representa tiene una gama de colores asignada, para lo que se han creado dos sets de materiales monocromáticos. De este modo se pueden reconocer fácilmente tanto el movimiento de cada uno de los dedos como la mano a la que corresponden los datos de lateralidad mostrados en la parte superior izquierda.

Además, se han incluido en la escena dos planos y dos cubos que, manteniendo el aspecto minimalista, permiten al usuario apreciar con facilidad el movimiento de las manos en el espacio y los cambios de profundidad.

En las figuras siguientes podemos observar tanto la configuración de la escena en la GUI de Unity [Figura 5.3] como el aspecto de la escena final con el proyecto arrancado [Figura 5.4].

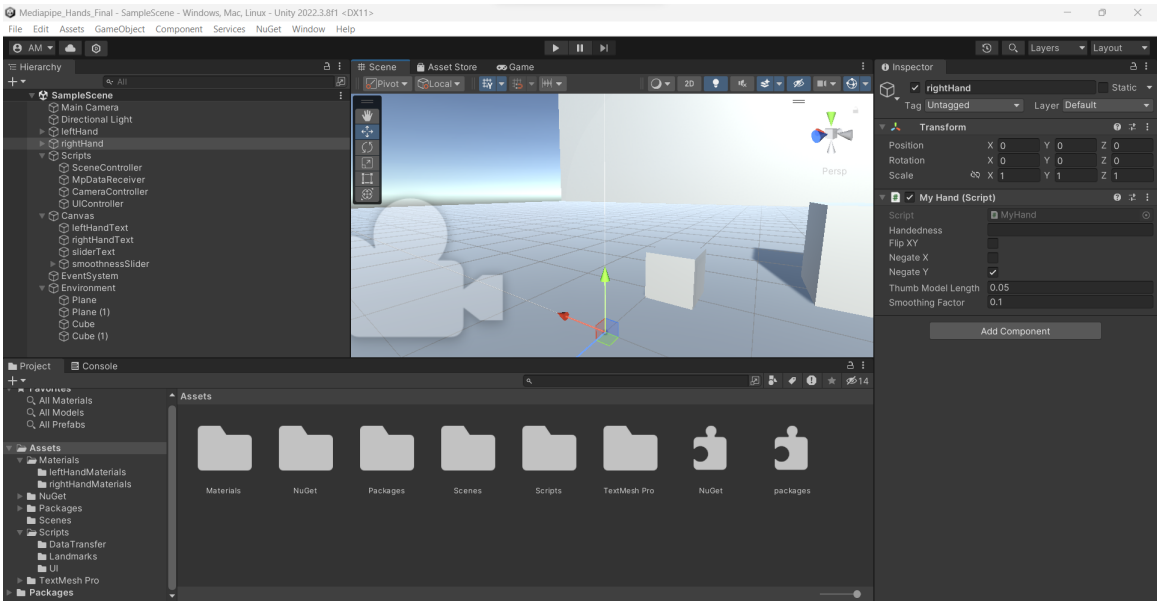

Figura 5.3. Proyecto en el GUI de Unity.

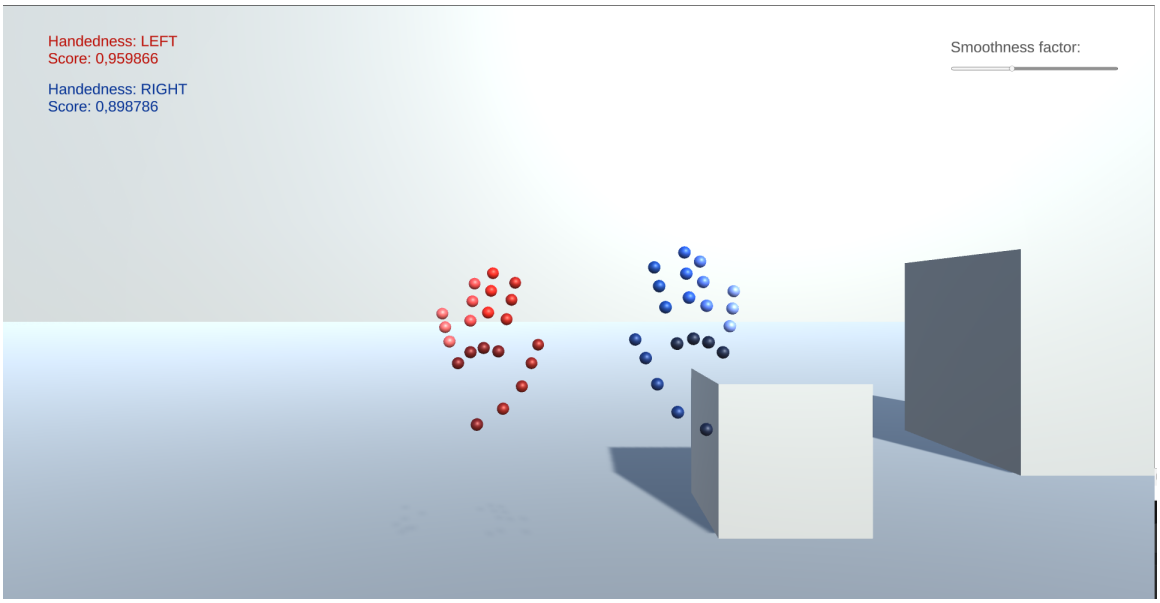

Figura 5.4. Escena arrancada en Unity.

5.6 Consideraciones

A lo largo del desarrollo se han ido revelando algunos puntos de inflexión, en esta sección se hablan sobre algunos de los más relevantes.

5.6.1 Testing

Inicialmente, durante la revisión de las buenas prácticas de programación en el proyecto, se consideró la posibilidad de implementar pruebas unitarias para garantizar la calidad del código y facilitar futuras actualizaciones y mantenimiento. Sin embargo, después de un análisis más profundo, se llegó a la conclusión de que las pruebas unitarias podrían resultar ineficaces dadas las características específicas de la aplicación.

En su lugar, se han optado por pruebas de integración debido a la naturaleza en tiempo real de la aplicación y la complejidad de las tecnologías involucradas. La aplicación se basa en múltiples componentes interconectados que trabajan en conjunto para lograr un rendimiento en tiempo real y una comunicación fluida entre ellas. Esto hace que sea más efectivo evaluar cómo interactúan estos componentes en su conjunto mediante pruebas de integración, en lugar de intentar aislar unidades individuales de código para pruebas unitarias.

Además, la complejidad del código y las numerosas dependencias entre los componentes harían que las pruebas unitarias fueran complicadas de implementar y mantener. Las pruebas de integración proporcionan una forma más eficiente de garantizar que todos los componentes funcionen coherentemente en tiempo real, lo que es esencial para el éxito de la aplicación.

5.6.2 Retos y dificultades

A lo largo del proyecto, se han enfrentado varios desafíos significativos. Inicialmente, la falta de experiencia en la captura de movimiento y el uso de la inteligencia artificial para este fin planteó un obstáculo considerable. Las etapas iniciales de investigación y aprendizaje fueron un tanto lentas, ya que era necesario adquirir un profundo entendimiento de los conceptos subyacentes y las tecnologías implicadas. Sin embargo, este período de aprendizaje también fue fundamental para construir una base sólida y asegurarse de que las decisiones posteriores estuvieran fundamentadas en un conocimiento sólido.

Además, debido a la naturaleza en constante evolución del campo de la inteligencia artificial y las tecnologías de captura de movimiento, se ha enfrentado el reto de mantenerse al día con las últimas actualizaciones y avances. Las bibliotecas y soluciones utilizadas en el proyecto pueden haber experimentado cambios significativos a lo largo del tiempo, lo que podría haber resultado en información desactualizada en el proyecto. Para abordar este problema, se ha realizado un esfuerzo constante por mantenerse al tanto de las novedades y ajustar el proyecto según sea necesario para aprovechar las últimas capacidades y mejoras.

En particular, el *framework* de Mediapipe ha pasado por una gran refactorización de su código fuente y, en particular, de su página de documentación. Sin embargo, aunque inconveniente, esto solo se puede valorar como un avance positivo en la evolución de este tipo de tecnologías y su popularización.

Resultados

Una vez desglosado en detalle todo el proyecto, esta sección muestra las vistas del mismo [Figura 6.1][Figura 6.2], para solidificar los conceptos desarrollados a lo largo de la memoria. Hecho esto, se discutirán los resultados y se realizará una evaluación personal del trabajo realizado.

6.1 Overview del proyecto

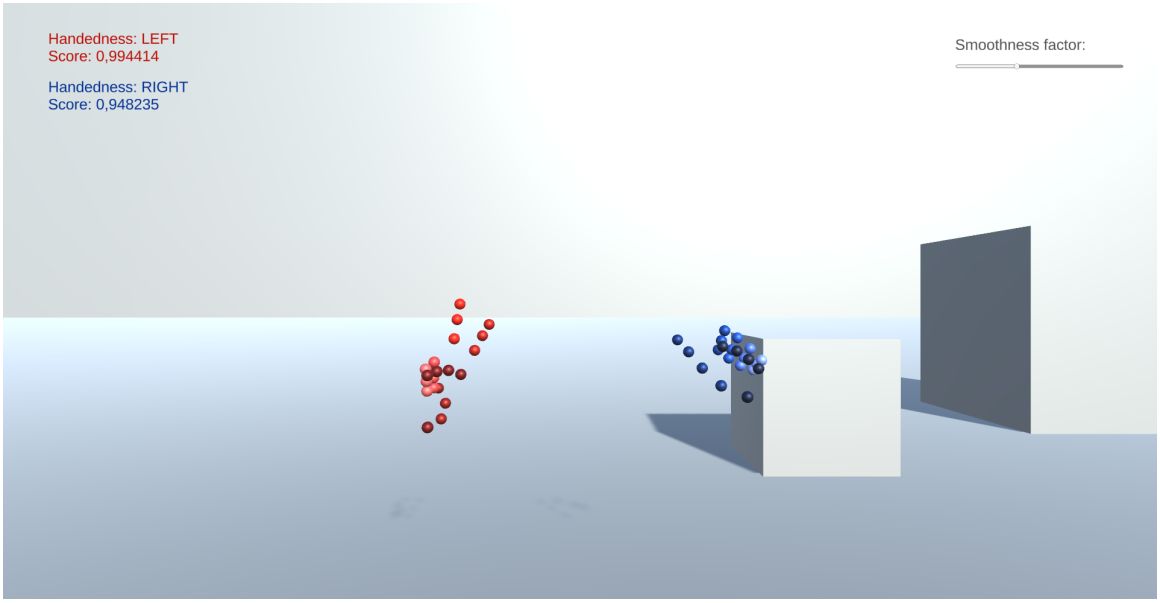

Figura 6.1. Escena replicando el movimiento capturado.

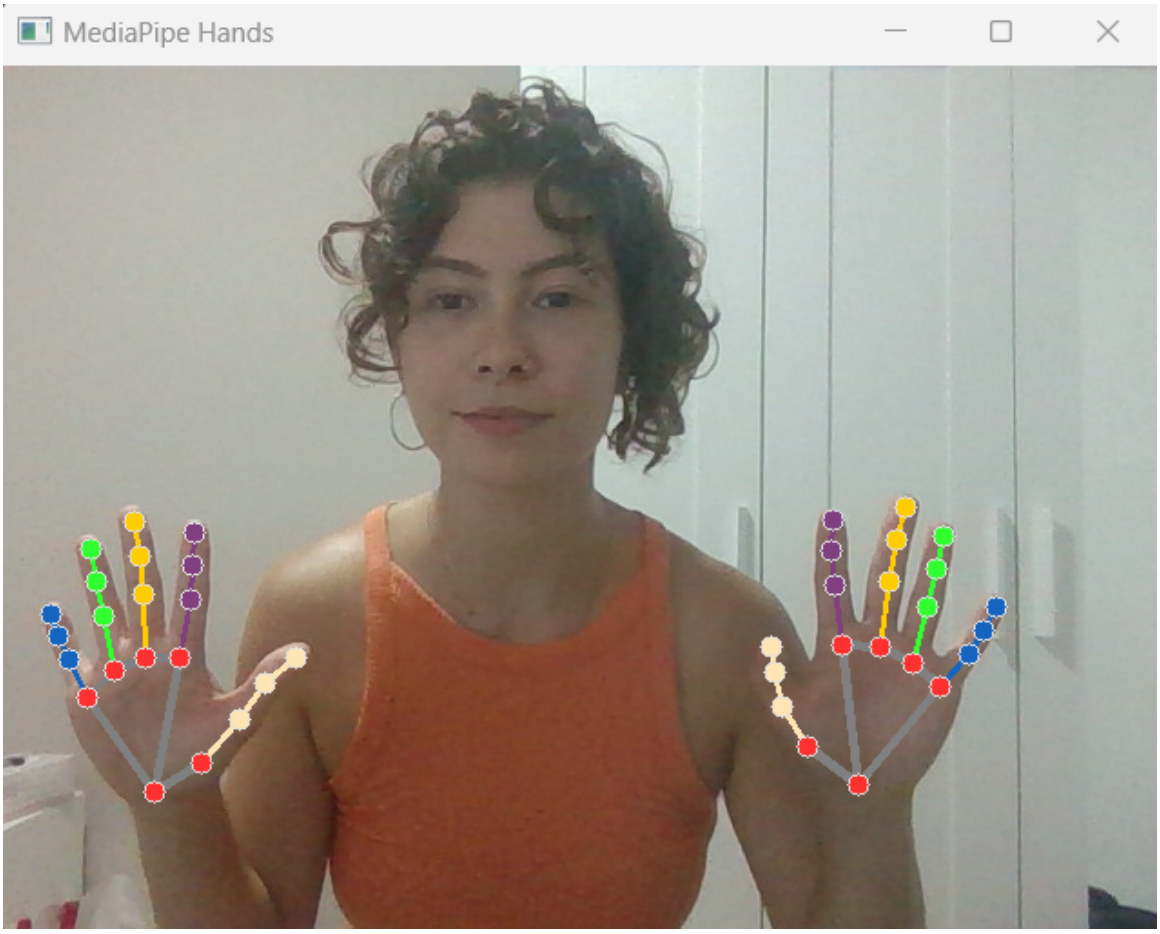

Figura 6.2. *Prompt* del código en Python con los *Landmarks* detectados.

6.2 Pruebas y análisis

Con la aplicación terminada de forma satisfactoria, se ha realizado una pequeña fase de pruebas. Estas se han enfocado principalmente a comprobar que el proyecto no estaba sesgado por el uso del mismo equipo (un PC personal) durante toda la fase de desarrollo, ni por la comprobación de funcionamiento, realizada siempre con las mismas manos reales.

Por estos motivos, se ha clonado el proyecto y ejecutado en dos sistemas distintos al propio, que cumplían los prerrequisitos de instalaciones pertinentes, y se ha comprobado que se ejecutaba de forma satisfactoria. Además de comprobar que la obtención de dados y los cálculos de escalado funcionaban de forma efectiva con las manos de una variedad de usuarios ejemplo.

6.3 Evaluación

Estoy muy satisfecha con los resultados de este proyecto. A pesar de los desafíos y retos en el desarrollo, el resultado final es un sistema que funciona de manera muy satisfactoria. Si bien tuve que comprometer algunos de mis grandes planes iniciales debido a la complejidad de las tecnologías involucradas y la naturaleza en tiempo real del proyecto, la experiencia fue increíblemente enriquecedora.

Uno de los aspectos más gratificantes de este proyecto fue el aprendizaje. Inicialmente, no tenía un conocimiento profundo de la captura de movimiento ni de cómo se aplicaba la inteligencia artificial en este contexto. Sin embargo, a lo largo del proyecto, tuve la oportunidad de adentrarme en estos campos y aprender mucho más de lo que esperaba. Desde comprender los detalles de la biblioteca Mediapipe hasta la integración de Unity y Python mediante sockets UDP, cada paso fue una oportunidad para adquirir nuevos conocimientos y habilidades.

Esta experiencia me ha brindado una comprensión más profunda de la intersección entre la tecnología y la creatividad. Ver cómo los movimientos humanos se traducen en un entorno 3D en tiempo real es verdaderamente fascinante. Este proyecto ha sido un recordatorio de que el aprendizaje continuo y la adaptación son esenciales en el mundo en constante evolución de la tecnología.

Aplicaciones y futuros trabajos

Este proyecto sienta las bases para el reconocimiento de pose de manos en aplicaciones de Unity utilizando Python y MediaPipe, añadiendo un granito de arena a la extensa investigación de este tipo de tecnologías y demostrando una implementación sencilla que acerca este tipo de soluciones a estudiantes como yo. Sin embargo, aun habiendo conseguido los objetivos del proyecto, hay oportunidades para futuros trabajos que pueden mejorar y expandir esta aplicación:

Se puede investigar y aplicar técnicas adicionales para mejorar la precisión del reconocimiento de pose de manos. Esto puede incluir el uso de modelos de aprendizaje profundo personalizados o una fase de calibración previa personalizada para cada usuario individual.

También se podría integrar el reconocimiento de gestos. Actualmente, la aplicación se centra en el seguimiento básico de las manos, pero se pueden explorar gestos más complejos y para permitir una interacción más rica en la escena de Unity. El mismo paquete de mediapipe puede proporcionar las herramientas necesarias para una tarea como esta.

Tal vez el camino de progreso más evidente es la expansión del alcance de la captura de movimiento a modelos de todo el cuerpo. Combinando distintas soluciones, se podría escalar este proyecto para reflejar en una escena 3D las posiciones en tiempo real de un humano entero.

Por último, se podría añadir más portabilidad. Actualmente, la aplicación se desarrolla principalmente para sistemas de escritorio. Se puede trabajar en la adaptación y portabilidad de la aplicación a dispositivos móviles y de realidad virtual, lo que ampliaría su alcance y aplicaciones.

En resumen, este proyecto es un primer paso en la dirección de hacer que el reconocimiento de pose de manos sea más accesible y versátil. Futuros trabajos pueden explorar y expandir aún más estas posibilidades, creando aplicaciones más avanzadas y diversas en el campo del MoCap.

Conclusiones

En conclusión, este proyecto ha logrado desarrollar una aplicación que demuestra el potencial del reconocimiento de pose de manos en tiempo real utilizando Python y MediaPipe, junto con la representación tridimensional en Unity. Se han alcanzado los objetivos de investigación, selección de tecnología, desarrollo práctico y optimización, proporcionando una base sólida para futuros desarrollos en este campo.

La aplicación no solo ha demostrado la viabilidad técnica de esta tecnología, sino que también ha subrayado su importancia en la democratización del reconocimiento de pose y la captura de movimiento, abriendo nuevas posibilidades en campos que van más allá del entretenimiento, como la telemedicina, la educación y el arte digital.

A medida que continuamos explorando y mejorando esta tecnología, podemos esperar un futuro lleno de aplicaciones innovadoras y versátiles que permitirán una interacción más intuitiva y precisa entre los usuarios y el mundo digital que los rodea. Este proyecto, por lo tanto, quiere motivar a seguir evolucionando en la forma en que interactuamos con la tecnología y el entorno que nos rodea.

Anexos

9.1 mediapipe_capture.py

C: > Users > aitan > Escritorio > TFG-Aitana_Baldovi_Murcia > mediapipe_capture.py > ...

```python
import cv2
import mediapipe as mp
import numpy as np
import socket
import json

UDP_IP = "127.0.0.1"
UDP_PORT = 5060

def process_and_send_data(image, landmarks_data, sock):
    image.flags.writeable = False
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = hands.process(image)

    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    data = []

    if results.multi_hand_landmarks and results.multi_hand_world_landmarks:
        for hand_landmarks, world_landmarks, handedness in zip(
            results.multi_hand_landmarks, results.multi_hand_world_landmarks, results.multi_handedness):

            handedness_data = {
                "classification": handedness.classification[0].label,
                "score": handedness.classification[0].score
            }

            landmarks_data = [
                [
                    round(landmark.x, 6),
                    round(landmark.y, 6),
                    round(landmark.z, 6)
                ] for landmark in hand_landmarks.landmark
            ]

            hand_data = {
                "handedness": handedness_data,
                "landmarks": landmarks_data
            }

            data.append(hand_data)

            mp_drawing.draw_landmarks(
                image,
                hand_landmarks,
                mp_hands.HAND_CONNECTIONS,
```

Figura 9.1. Código de Python en Visual Studio Code. Parte 1.

```
48                  mp_drawing_styles.get_default_hand_landmarks_style(),
49                  mp_drawing_styles.get_default_hand_connections_style())
50
51      json_message = json.dumps(data)
52
53      print("JSON message:", json_message)
54
55      sock.sendto(json_message.encode(), (UDP_IP, UDP_PORT))
56
57      cv2.imshow('MediaPipe Hands', cv2.flip(image, 1))
58
59
60  if __name__ == '__main__':
61      #MediaPipe
62      mp_drawing = mp.solutions.drawing_utils
63      mp_drawing_styles = mp.solutions.drawing_styles
64      mp_hands = mp.solutions.hands
65
66      #socket
67      sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
68
69      #webcam
70      cap = cv2.VideoCapture(0)
71
72      with mp_hands.Hands(
73          model_complexity=0,
74          min_detection_confidence=0.5,
75          min_tracking_confidence=0.5,
76          max_num_hands=2) as hands:
77
78          while cap.isOpened():
79              success, image = cap.read()
80
81              if not success:
82                  print("Ignoring empty camera frame.")
83                  continue
84
85              process_and_send_data(image, [], sock)
86
87              if cv2.waitKey(5) & 0xFF == 27:
88                  break
89
90      cap.release()
91      sock.close()
92      cv2.destroyAllWindows()
93
```

Figura 9.2. Código de Python en Visual Studio Code. Parte 2.

9.2 MpDataReceiver.cs

```
Unity Script (1 asset reference) | 1 reference
public class MpDataReceiver : MonoBehaviour
{
    // VARIABLES UDP
    Thread receiveThread;
    UdpClient client;
    readonly int port = 5060;
    public string text = "[{\"handedness\": {\"classification\": \"Left\", \"score\": 0.9339931011199951}, \"landmarks

//FUNCIONES UDP
1 reference
private void InitUDP()
    {
        print("UDP Initialized");

        receiveThread = new Thread(new ThreadStart(ReceiveData))
        {
            IsBackground = true
        };
        receiveThread.Start();
    }

    1 reference
    public void ReceiveData()
    {
        client = new UdpClient(port);
        while (true)
        {
            try
            {
                IPEndPoint anyIP = new IPEndPoint(IPAddress.Parse("0.0.0.0"), port);
                byte[] data = client.Receive(ref anyIP);

                text = Encoding.UTF8.GetString(data);
                Debug.Log("Received Data: " + text);
            }
            catch (SocketException socketException)
            {
                Debug.LogError("SocketException: " + socketException.Message);
            }
            catch (Exception e)
            {
                Debug.LogError("Exception: " + e.ToString());
            }
        }
    }

    Unity Message | 0 references
    void Start()
    {
        InitUDP();
    }
}
```

Figura 9.3. El script que se encarga de recibir los datos por UDP.

9.3 MpData.cs

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

2 references
public class MpData
{
    public HandData[] data;
}
4 references
public class HandData
{
    public Handedness handedness;
    public float[][] landmarks;
}

[System.Serializable]
1 reference
public class Handedness
{
    public string classification;
    public float score;
}

```

Figura 9.4. Estructura para almacenar los datos recividos.

9.4 MyJoint.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (42 asset references) | 2 references
public class MyJoint : MonoBehaviour
{
    public Transform Target;

    1 reference
    public void UpdateTransform()
    {
        if (Target == null)
            return;
        Target.SetPositionAndRotation(this.transform.position, this.transform.rotation);
    }
}

```

Figura 9.5. Clase para cada articulación de la mano.

9.5 DepthCallibrator.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
   3 references
5  public class DepthCalibrator
6  {
7      private float m;
8      private float c;
9
       1 reference
10     public DepthCalibrator(float m, float c)
11     {
12         this.m = m;
13         this.c = c;
14     }
15
       1 reference
16     public float GetDepthFromThumbLength(float length)
17     {
18         if (length == 0)
19             return 0;
20         return m / length + c;
21     }
22 }
23
```

Figura 9.6. Calculadora de profundidad de la mano.

9.6 MyHand.cs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    Unity Script (2 asset references) | 6 references
5   public class MyHand : MonoBehaviour
6   {
7       public HandData handData;
8
9       public string handedness;
10      public bool FlipXY = false;
11      public bool NegateX = false;
12      public bool NegateY = false;
13
14      [SerializeField]
15      private float thumbModelLength = 0.05f;
16      private float scale;
17
18      private DepthCalibrator depthCalibrator = new DepthCalibrator(-0.0719f, 0.439f);
19
20      private MyJoint[] myJoints;
21
22      [SerializeField]
23      public float smoothingFactor = 0.1f;
24      private List<Vector3> smoothedLandmarks;
25      private List<Vector3> scaledLandmarks;
26
27
        Unity Message | 0 references
28      private void Start()
29      {
30          myJoints = this.transform.GetComponentsInChildren<MyJoint>();
31      }
32
        2 references
33      public void SetInPlace()
34      {
35          var landmarks = LandmarksToVector3List(handData.landmarks);
36
37          if (landmarks == null || landmarks.Count == 0)
38          {
39              return;
40          }
41
42          UpdatePositions(landmarks);
43          UpdateScale(landmarks);
44          UpdateWristRotation();
45
46          foreach (var j in myJoints)
47          {
48              j.UpdateTransform();
49          }
50      }
51
```

Figura 9.7. Lógica de las manos. Parte 1.

```
1 reference
public List<Vector3> LandmarksToVector3List(float[][] landmarks)
{
    if (landmarks == null)
    {
        return null;
    }

    List<Vector3> vector3List = new List<Vector3>();

    for (int i = 0; i < landmarks.Length; i++)
    {
        Vector3 landmarkVector = new Vector3(
            landmarks[i][0],
            landmarks[i][1],
            landmarks[i][2]
        );

        vector3List.Add(landmarkVector);
    }

    return vector3List;
}

1 reference
private void UpdatePositions(List<Vector3> landmarks)
{
    if (smoothedLandmarks == null)
    {
        smoothedLandmarks = new List<Vector3>(landmarks);
    }

    if (scaledLandmarks == null)
    {
        scaledLandmarks = new List<Vector3>(smoothedLandmarks);
    }

    var offset = landmarks[0];

    for (int i = 1; i < landmarks.Count; i++)
    {
        var x = landmarks[i].x - offset.x;
        var y = landmarks[i].y - offset.y;
        var z = landmarks[i].z - offset.z;

        if (x == 0 && y == 0 && z == 0)
            return;

        if (NegateX) x *= -1;
        if (NegateY) y *= -1;
        if (FlipXY) (x, y) = (y, x);

        smoothedLandmarks[i] = Vector3.Lerp(smoothedLandmarks[i], new Vector3(x, y, z), 1 - smoothingFactor);
```

Figura 9.8. Lógica de las manos. Parte 2.

```
103
104                 scaledLandmarks[i] = smoothedLandmarks[i] * scale;
105
106                 myJoints[i].transform.localPosition = scaledLandmarks[i];
107             }
108
109             float depth = depthCalibrator.GetDepthFromThumbLength(scale);
110
111             Vector3 handPosition = new(offset.x * scale, (-offset.y * scale) + 1, depth * scale);
112
113             this.transform.localPosition = Vector3.Lerp(this.transform.localPosition, handPosition, 1 - smoothingFactor);
114         }
115
                1 reference
116         private void UpdateScale(List<Vector3> landmarks)
117         {
118             var pointA = landmarks[0];
119             var pointB = landmarks[1];
120             var thumbDetectedLength = Vector3.Distance(pointA, pointB);
121
122             if (thumbDetectedLength == 0)
123             {
124                 return;
125             }
126
127             scale = thumbModelLength / thumbDetectedLength;
128         }
129
                1 reference
130         private void UpdateWristRotation()
131         {
132             var wristTransform = myJoints[0].transform;
133             var indexFinger = myJoints[5].transform.position;
134             var middleFinger = myJoints[9].transform.position;
135
136             var vectorToMiddle = middleFinger - wristTransform.position;
137             var vectorToIndex = indexFinger - wristTransform.position;
138
139             Vector3.OrthoNormalize(ref vectorToMiddle, ref vectorToIndex);
140
141             Vector3 normalVector = Vector3.Cross(vectorToIndex, vectorToMiddle);
142
143             wristTransform.rotation = Quaternion.LookRotation(normalVector, vectorToIndex);
144         }
145     }
146
```

Figura 9.9. Lógica de las manos. Parte 3.

9.6.1 SceneController.cs

```
using UnityEngine;
using Newtonsoft.Json;

Unity Script (1 asset reference) | 0 references
public class SceneController : MonoBehaviour
{
    public MyHand leftHand;
    public MyHand rightHand;

    public MpDataReceiver mpDataReceiver;
    public MpData mpData;

    public bool invertedCamera;

    Unity Message | 0 references
    private void Update()
    {
        DeserializeAndAssignData();

        leftHand.SetInPlace();
        rightHand.SetInPlace();
    }

    1 reference
    private void DeserializeAndAssignData()
    {
        string jsonData = mpDataReceiver.text;

        try
        {
            mpData = JsonConvert.DeserializeObject<MpData>("{\"data\":" + jsonData + "}");
        }
        catch (JsonException jsonException)
        {
            Debug.LogError("Error deserializing JSON: " + jsonException.Message);
            return;
        }

        foreach (HandData handData in mpData.data)
        {
            string handedness = handData.handedness.classification;

            if (invertedCamera)
            {
                handData.handedness.classification = InvertHandedness(handedness);
            }

            handedness = handData.handedness.classification;

            AssignHandData(handData, handedness);
        }
    }

```

Figura 9.10. Script gestor de la escena en Unity. Parte 1.

```
1 reference
private string InvertHandedness(string handedness)
{
    if (handedness == "Right")
    {
        return "Left";
    }
    else if (handedness == "Left")
    {
        return "Right";
    }
    return handedness;
}

1 reference
private void AssignHandData(HandData handData, string handedness)
{
    if (handedness == "Right")
    {
        rightHand.handData = handData;
    }
    else if (handedness == "Left")
    {
        leftHand.handData = handData;
    }
}
}
```

Figura 9.11. Script gestor de la escena en Unity. Parte 2.

9.7 UIController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

Unity Script (1 asset reference) | 0 references
public class UIController : MonoBehaviour
{
    public MyHand leftHand;
    public MyHand rightHand;

    public TextMeshProUGUI leftHandText;
    public TextMeshProUGUI rightHandText;

    public Slider smoothnessSlider;

    Unity Message | 0 references
    void Start()
    {
        if (leftHandText != null && rightHandText != null)
        {
            leftHandText.text = "Handedness: - \nScore: -";
            rightHandText.text = "Handedness: - \nScore: -";
        }
    }

    Unity Message | 0 references
    void Update()
    {
        if (leftHandText != null && rightHandText != null)
        {
            leftHandText.text = "Handedness: " + leftHand.handData.handedness.classification.ToUpper() + "\nScore: " + l
            rightHandText.text = "Handedness: " + rightHand.handData.handedness.classification.ToUpper() + "\nScore: " +
        }

        leftHand.smoothingFactor = smoothnessSlider.value;
        rightHand.smoothingFactor = smoothnessSlider.value;
    }


}
```

Figura 9.12. Código para los elementos de la UI.

9.8 CameraController.cs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.XR;
5
    Unity Script (1 asset reference) | 0 references
6   public class CameraController : MonoBehaviour
7   {
8       public MyHand leftHand;
9       public MyHand rightHand;
10
11      public Camera mainCamera;
12
13      public float zoomSpeed = 2.0f;
14
15      public float minFOV = 5.0f;
16      public float maxFOV = 60.0f;
17
18      private Vector3 cameraTargetPosition;
19      private Vector3 cameraInitialPosition;
20
        Unity Message | 0 references
21      void Start()
22      {
23          cameraInitialPosition = mainCamera.transform.position;
24      }
25
        Unity Message | 0 references
26      void Update()
27      {
28          Vector3 midpoint = (leftHand.transform.position + rightHand.transform.position) / 2f;
29
30          float distance = Vector3.Distance(leftHand.transform.position, rightHand.transform.position);
31
32          cameraTargetPosition = midpoint - mainCamera.transform.forward * (distance / 2f);
33          cameraTargetPosition.y += 0.2f;
34          cameraTargetPosition.z += 0.8f;
35
36          mainCamera.transform.position = Vector3.Lerp(mainCamera.transform.position, cameraTargetPosition, Time.deltaTime);
37      }
38  }
39
```

Figura 9.13. Control del movimiento de cámara.

Bibliografía

[1] Wikipedia. *Rotoscopio — Wikipedia, La enciclopedia libre*. [Internet; descargado 29-junio-2023]. 2023. URL: https://es.wikipedia.org/w/index.php?title=Rotoscopio&oldid=152165257.

[2] Jessica Conditt. *100 years of motion-capture technology*. URL: https://www.engadget.com/2018-05-25-motion-capture-history-video-vicon-siren.html.

[3] Ernie Smith. *This is what 1970s motion capture tech looked like*. Mar. de 2017. URL: https://www.vice.com/en/article/wnkbzz/this-is-what-1970s-motion-capture-tech-looked-like.

[4] Wikipedia contributors. *Motion capture suit — Wikipedia, The Free Encyclopedia*. [Online; accessed 4-September-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Motion_capture_suit&oldid=1170936448.

[5] Sep. de 2023. URL: https://www.vicon.com/.

[6] URL: https://cgicoffee.com/blog/tag/ue4.

[7] Dorin Sabin Copaci et al. *Evaluación y desempeño de los sensores myo armband Y Mindrove*. Ene. de 1970. URL: http://hdl.handle.net/2183/31353.

[8] Diego R. Faria. URL: https://www.researchgate.net/figure/Myo-armband-sensor-by-Thalmic-labs-The-bottom-images-present-the-possibilities-for_fig4_328676359.

[9] URL: https://www.intuitive.com/en-us.

[10] URL: https://facewaretech.com/.

[11] Department of Physical Medicine UC Davis Health y Rehabilitation. *Congratulations to department chair dr. Craig McDonald*. URL: https://health.ucdavis.edu/pmr/.

[12] URL: https://www.uzleuven.be/en.

[13] URL: https://www.brighamandwomens.org/.

[14] Ago. de 2023. URL: https://vcresearch.berkeley.edu/.

[15] Gswbwitt. *Dubs deep dives*. Abr. de 2017. URL: https://www.nba.com/warriors/deepdive/mocap/20170415.

[16] Mar. de 2023. URL: https://www.vicon.com/resources/case-studies/elite-sports-motion-analysis/.

[17] Abr. de 2023. URL: https://humanperformance.stanford.edu/programs/.

[18] Mar. de 2023. URL: https://woz-u.com/blog/motion-capture-ultimate-guide/.

[19] Mar. de 2020. URL: https://www.foundry.com/insights/machine-learning/motion-capture.

[20] Sik-Ho Tsang. *Review: Deepcut amp; Deepercut - multi person pose estimation (human pose estimation)*. Mar. de 2020. URL: https://sh-tsang.medium.com/review-deepcut-deepercut-multi-person-pose-estimation-human-pose-estimation-da5b469cbbc3.

[21] Gaurav Patil. *PoseNet: Your gateway to gesture detection*. Nov. de 2020. URL: https://medium.com/globant/posenet-your-gateway-to-gesture-detection-a15d0ed0ae40.

[22] Gaudenz Boesch. *The Complete Guide to openpose in 2023*. Ene. de 2023. URL: https://viso.ai/deep-learning/openpose/.

[23] Zhe Cao et al. *Realtime multi-person 2D pose estimation using part affinity fields*. URL: https://www.arxiv-vanity.com/papers/1611.08050/.

[24] Peter Tanugraha. *Understanding openpose (with code reference)- part 1*. Sep. de 2019. URL: https://medium.com/analytics-vidhya/understanding-openpose-with-code-reference-part-1-b515ba0bbc73.

[25] URL: https://developers.google.com/mediapipe.

[26] URL: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker.

[27] URL: https://github.com/google/mediapipe/blob/master/docs/solutions/solutions.md.