

# Uso de VR y AR para el entrenamiento de actividades físicas



Trabajo de Fin de Grado

Raúl Fernández Pérez

Departamento de Ingeniería del Software e Inteligencia Artificial

Facultad de Informática

Universidad Complutense de Madrid

Septiembre 2020



# Uso de VR y AR para el entrenamiento de actividades físicas

*Trabajo de Fin de Grado*  
**Ingeniería del Software e Inteligencia Artificial**

*Dirigido por los Doctores*  
**Gonzalo Méndez Pozo, Pablo Gervás Gómez-Navarro**

**Departamento de Ingeniería del Software e Inteligencia Artificial**  
**Facultad de Informática**  
**Universidad Complutense de Madrid**

**Septiembre 2020**



# Dedicatoria

*A mis familiares por darme todo y apoyarme para conseguir siempre lo que me proponga*

*A mi novia Ana, por su apoyo y siempre alentarme a ser mejor*

*A todos mis amigos que me han acompañado y ayudado todo este tiempo*

*A todos ellos le dedico este trabajo porque sin ellos no habría sido posible*



# Agradecimientos

*Quisiera agradecer a Gonzalo Méndez Pozo y a Pablo Gervás Gómez-Navarro, los directores de este trabajo, por todo el apoyo que me han proporcionado para realizarlo.*

*También me gustaría agradecer a los miembros de mi familia, mis padres y amigos que me han mostrado su apoyo y ánimos durante la creación de este trabajo.*

*A mi amiga de la infancia Ana Galindo Lobato por su dedicación, gracias por ayudarme a llegar tan lejos, sin ti no lo hubiera conseguido.*

*Esto va también por vosotros frescos(SOF), mis grandes hermanos.*

*También quería agradecérselo a esa persona que me hace la vida más fácil cada día, a mi novia Ana.*



# Resumen

Este proyecto tiene como meta el diseño

Palabras clave:Kinect, capoeira, captura movimiento, Mocap, Reactive-Virtual Trainer, RVT, Unity 3D, entrenador personal, comparación y corrección de movimientos.ix



# **Abstract**

The target of this Project is developing



# Índice general

<b>Dedicatoria</b>	<b>v</b>
<b>Agradecimientos</b>	<b>vii</b>
<b>Resumen</b>	<b>ix</b>
<b>Abstract</b>	<b>xI</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Objetivos del trabajo . . . . .	1
1.2. Plan de Trabajo . . . . .	1
1.3. Estructura de la memoria . . . . .	1
<b>2. Introduction</b>	<b>3</b>
2.1. Objetivos del trabajo . . . . .	3
2.2. Plan de Trabajo . . . . .	3
2.3. Estructura de la memoria . . . . .	3
<b>3. Estado del arte</b>	<b>5</b>
3.1. Captura de movimiento . . . . .	5
3.2. Historia de la captura de movimiento . . . . .	5
3.2.1. Precursores . . . . .	5
3.2.2. Nacimiento de la captura de movimiento . . . . .	6
3.3. Métodos de captura de movimiento . . . . .	9
3.3.1. Captura de movimiento no óptico . . . . .	10
3.3.2. Captura de movimiento óptico . . . . .	11
3.4. Métodos de captura de movimiento emergentes . . . . .	13
3.4.1. Visión artificial . . . . .	13
3.4.2. Realidad virtual . . . . .	16
3.5. Trabajo relacionado . . . . .	18
<b>4. Tecnologías empleadas</b>	<b>19</b>
4.1. Software empleado . . . . .	19
4.1.1. Unity . . . . .	20

4.1.2. Adobe Fuse Character Creator . . . . .	23
4.1.3. Mixamo . . . . .	24
4.2. Hardware empleado . . . . .	25
<b>5. Desarrollo del Proyecto</b>	<b>27</b>
5.1. Investigación para el desarrollo de <i>Mocap</i> en Realidad Virtual y Realidad Aumentada . . . . .	28
5.1.1. SteamVR . . . . .	28
5.1.2. Final IK . . . . .	30
5.1.3. ARCore . . . . .	32
5.2. Grabación y desarrollo de <i>Mocap</i> en VR . . . . .	34
5.2.1. Investigación base . . . . .	34
5.2.2. Desarrollo de la grabación de movimientos . . . . .	36
5.3. Análisis de <i>Mocap</i> en AR . . . . .	39
5.3.1. Investigación base . . . . .	39
5.3.2. Desarrollo del análisis de movimientos . . . . .	41
5.4. Diseño y creación de avatares . . . . .	44
5.4.1. Investigación base . . . . .	45
5.4.2. Diseño de los avatares . . . . .	46

# Índice de figuras

3.1.	El caballo en movimiento de Eadweard Muybridge . . . . .	6
3.2.	Rotoscopia en la película Blancanieves y los siete enanitos .	7
3.3.	Mocap en la película Los vengadores . . . . .	8
3.4.	Mocap en el videojuego The Last Of Us . . . . .	9
3.5.	Azure Kinect DK . . . . .	14
3.6.	Detección Facial con OpenCV . . . . .	15
3.7.	<i>Deepfake</i> de Barack Obama . . . . .	16
3.8.	<i>Deepfake</i> de la Princesa Leia en Star Wars . . . . .	16
3.9.	Oculus Quest, modelo <i>all in one</i> de VR . . . . .	17
3.10.	Ejemplo de <i>Mocap</i> con <i>trackers</i> Vive . . . . .	18
4.1.	Orden de ejecución para los eventos en Unity . . . . .	22
4.2.	Configuración de un avatar con Adobe Fuse Character Creator	24
4.3.	Servicio de animaciones de Mixamo . . . . .	25
5.1.	Diferencia entre <i>Forward Kinematics</i> e <i>Inverse Kinematics</i> .	31
5.2.	Patrón <i>Singleton</i> en Unity . . . . .	38
5.3.	Proyección en el plano utilizada en <i>Depth API</i> . . . . .	40
5.4.	Asignación automatizada de los movimientos grabados en el <i>Mocap</i> . . . . .	43
5.5.	Auto-Rig con 65 huesos de Mixamo . . . . .	46
5.6.	<i>Hand tracking</i> en Oculus Quest . . . . .	47



# **Capítulo 1**

## **Introducción**

En la actualidad los sistemas de captura de movimiento

### **1.1. Objetivos del trabajo**

### **1.2. Plan de Trabajo**

### **1.3. Estructura de la memoria**



## **Capítulo 2**

# **Introduction**

En la actualidad los sistemas de captura de movimiento

**2.1. Objetivos del trabajo**

**2.2. Plan de Trabajo**

**2.3. Estructura de la memoria**



# Capítulo 3

## Estado del arte

### 3.1. Captura de movimiento

Los sistemas de captura de movimiento o **Mocap**, son un conjunto de técnicas que se utilizan para registrar los movimientos, ya sea de objetos, animales o mayormente personas, trasladándolos a un modelo 3D. En la actualidad, esta técnica se utiliza en la industria del cine y de los videojuegos ya que facilita mucho la labor de los animadores al realizar un modelado más realista. En el cine se utiliza como mecanismo para almacenar los movimientos realizados por los actores, y así poder animar los modelos 3D de los diferentes personajes que tenga el film. En cambio, en el sector de los videojuegos se utiliza para naturalizar los movimientos de los personajes. De ese modo se obtiene una mayor sensación de realismo.

### 3.2. Historia de la captura de movimiento

#### 3.2.1. Precursores

Ya en la antigua Grecia, Aristóteles (384-322 AC) escribió el libro “*De Motu Animalium*” (Movimiento de los animales). El no solo veía los cuerpos de los animales como sistemas mecánicos, sino que perseguía la idea de cómo diferenciar la realización de un movimiento y cómo poderlo hacer realmente, por lo que podrá ser considerado el primer biomecánico de la historia.

Aproximadamente dos mil años después, Leonardo da Vinci (1452-1519) trató de describir algunos mecanismos que utiliza el cuerpo humano para poder desplazarse, como un humano puede saltar, caminar, mantenerse de

pie, etc.

Como pionero en la edad moderna, Eadweard Muybridge (1830-1904) fue el primer fotógrafo capaz de diseccionar el movimiento humano y animal, a través de múltiples cámaras tomando varias fotografías para captar instantes seguidos en el tiempo. Este experimento llamado “el caballo en movimiento”, (véase Figura 3.1), utiliza esta técnica de fotografía.[1] [2]

Figura 3.1: El caballo en movimiento de Eadweard Muybridge

### 3.2.2. Nacimiento de la captura de movimiento

Al principio de la industria cinematográfica, en la década de 1970, cuando empezaba a surgir la posibilidad de realizar animaciones de personajes por ordenador, se conseguía naturalizar los movimientos mediante técnicas clásicas de diseño, como la técnica de rotoscopia. Esta técnica consiste en reemplazar los frames de una grabación real por dibujos calcados en cada frame. Los estudios *Walt Disney Pictures* utilizaron esta técnica en la película de 1937 “Blancanieves y los siete enanitos” (véase Figura 3.2), para animar a los personajes del príncipe y Blancanieves.

En la década de 1980, en los laboratorios de biomecánica hubo un crecimiento en el análisis del movimiento humano mediante el uso de ordenadores. Tom Calvert, profesor de kinesióloga y ciencias de la computación en la universidad Simon Fraser (Canadá), fue uno de los primeros en usar potenciómetros para rastrear los movimientos del cuerpo humano, con el objetivo de ser utilizados por estudios coreográficos y asistencia clínica para ayudar a pacientes con problemas de locomoción.

Figura 3.2: Rotoscopia en la película Blancanieves y los siete enanitos

Mientras tanto surgen los primeros sistemas de monitoreo visual, centros como el MIT (Massachusetts Institute of Technology) empezaron a realizar experimentos con dispositivos de seguimiento visual aplicados en el cuerpo humano. Mas tarde, empiezan a cobrar importancia los primeros sistemas de seguimiento visual como el Op-Eye y el SelSpot. Estos sistemas normalmente usaban pequeños marcadores adheridos al cuerpo (Leds parpadeantes) con una serie de cámaras alrededor del espacio donde se realizaba la actividad.

En 1988, Brad deGraf y Wahrman desarrollaron *Mike the Talking Head* de Silicon Graphics, capaz de mostrar las capacidades de sus nuevos equipos 4D en tiempo real. Mike estaba dirigido por un controlador que permitía controlar diferentes parámetros de la cara del personaje: como los ojos, boca, expresión y posición de la cabeza. El hardware de Silicon Graphics proporcionaba una interpolación en tiempo real entre las expresiones faciales y la geometría de la cabeza del personaje y del usuario. En el congreso de SIGGRAPH, Mike fue mostrado al público, donde se demostró que la tecnología *mocap* estaba preparada para su explotación.

Tras estos avances, Pacific Data Images desarrolló un exoesqueleto de plástico, de modo que el actor se colocaría el traje con el objetivo de capturar los movimientos corporales: de la cabeza, pecho y brazos, a través de potenciómetros situados en la capa de plástico. De esta manera, los actores podrían controlar los personajes virtuales mimetizando sus movimientos. Pese a que el traje se utilizó en varios proyectos, no se consiguieron los resultados esperados, ya que el ruido de los circuitos y el diseño inapropiado del traje no lo permitían.

En torno a 1992, la empresa SimGraphics desarrolló un sistema de rastreo facial llamado *Face Waldo*. Consiguieron capturar la mayor parte de los

movimientos usando sensores adheridos a la barbilla, labios, mejillas, cejas y en el armazón del casco que llevaba el actor para su posterior aplicación en un personaje virtual en tiempo real. Este sistema logró ser novedoso ya que el actor podía manejar las expresiones faciales de un personaje a través de sus movimientos, logrando unos gestos más naturales que los capturados anteriormente.

Lo que produjo un éxito mayúsculo con este proyecto fue la interpretación en tiempo real de Mario, el personaje principal de la saga de videojuegos *Mario Bros*. Estaba controlado por un actor mediante *Face Waldo*, Mario conseguía dialogar con los miembros de una conferencia, respondiendo a sus preguntas. A partir de ese momento, SimGraphics se centró en la animación en directo, desarrollando personajes para televisión y otros eventos en directo, mejorando la fiabilidad del sistema para el rastreo facial.

De forma gradual, la técnica *mocap* se fue expandiendo entre las empresas desarrolladoras de sistemas de captura de movimientos, con el objetivo de crear productos y métodos que albergasen nuevos sectores empresariales.

A lo largo de la última década, hemos visto como han ido apareciendo largometrajes que iban mostrando la evolución de la captura de movimiento con una gran proyección de futuro. Ha pasado de ser una técnica que se utilizaba de forma esporádica para algunos personajes, ha ser indispensable en cualquier producción. Como por ejemplo en los *films*: Gollum en la trilogía de El Señor de los Anillos y de El Hobbit, Avatar, El planeta de los simios, Los vengadores (véase Figura 3.3) entre otros. De igual modo, esta técnica es muy utilizada actualmente en los videojuegos como, por ejemplo: The Last Of Us (véase Figura 3.4), Fifa 20, Red Dead Redemption, Uncharted 4.

Figura 3.3: Mocap en la película Los vengadores

Figura 3.4: Mocap en el videojuego The Last Of Us

### 3.3. Métodos de captura de movimiento

En la actualidad existen numerosos sistemas de captura de movimiento. Dependiendo de las necesidades de la producción, ya estén relacionados con el presupuesto disponible, así como las posiciones, velocidades, impulsos del actor o el nivel de realismo al que se quiera llegar. Atendiendo a las tecnologías más utilizadas en la actualidad para la captura de movimiento, se desglosan dos grandes grupos: los sistemas ópticos y no ópticos (incluyendo magnéticos, mecánicos, e iniciales).

Los sistemas ópticos funcionan mediante el seguimiento de marcadores físicos, como luces LED, reflectores, adhesivos con apariencia de pelota de ping-pong o incluso simplemente pintura facial.

Los sistemas no ópticos no utilizan ningún tipo de marcador físico. En su lugar, utilizan software de movimiento de fósforos para seguir el movimiento de un actor, pero este software funciona identificando características clave de un humano, como la boca o una prenda de vestir. Los directores de fotografía crean un boceto rápido en gráficos por ordenador de cualquier personaje que quieran dar vida, y mapean el esqueleto del personaje en el metraje de acción en vivo, teniendo en cuenta la posición, la escala, la orientación y el movimiento.

Esto es mucho más asequible ya que se basa principalmente en software y no requiere de tanto hardware. En un plató para grabar una película o un videojuego, primero se crea el personaje animado digitalmente y, mientras se graba, pueden mapear este personaje sobre el actor mediante captura de movimiento, para que puedan ver instantáneamente cómo el movimiento se traduce en el personaje, junto con la iluminación y los ángulos preferidos. Esto se conoce como cinematografía virtual.

Atendiendo a su tecnología, se pueden encontrar los diferentes métodos

que se desarrollan a continuación.

### **3.3.1. Captura de movimiento no óptico**

#### **3.3.1.1. Captura de movimiento magnético**

Los sistemas de captura de movimiento magnéticos están formados por sensores creados por tres espirales ortogonales que miden el flujo magnético, determinando tanto la posición como la orientación del sensor. Un transmisor genera un campo electromagnético de baja frecuencia que los receptores detectan y transmiten a la unidad electrónica de control, donde se filtra y amplifica. A continuación, los datos se envían a un ordenador central, donde se deduce la orientación y posición de todos los sensores del escenario.

Un sistema magnético estándar consta de 18 sensores, una unidad de control electrónica y un software para el procesamiento. En cambio, un sistema de última generación puede tener hasta 90 sensores capaces de capturar hasta 144 muestras por segundo, teniendo un coste medio (en torno a 6.000 €).

Estos sistemas tienen el inconveniente de producir interferencias con materiales metálicos debido a su conductividad, ya que se crean campos magnéticos que interfieren con el campo magnético del emisor. De ese modo, se trata de un sistema difícil de transportar a diferentes escenarios. El proceso de captura no es en tiempo real, aunque se aproxima bastante, pese a que tiene un número de capturas por segundo demasiado bajo. Como punto a favor, es más fácil procesar los datos que en otros sistemas *mocap*, ya que los datos obtenidos están en relación con la posición del actor.

#### **3.3.1.2. Captura de movimiento mecánico**

En el proceso de captura de movimiento mecánica el actor viste unos trajes especiales adaptables al cuerpo humano. En su mayoría, estos trajes están compuestos por estructuras rígidas con barras metálicas o plásticas, unidas mediante potenciómetros colocados en las principales articulaciones. Los potenciómetros se componen de un elemento deslizante acoplado a una resistencia, la cual produce una variación de tensión que puede medirse para conocer el grado de apertura de la articulación donde se encuentre acoplado. Los sensores que recogen la información pueden transmitirla mediante cables, pero normalmente lo hacen con radiofrecuencia.

Estos sistemas tienen el problema de que son incapaces de medir translaciones globales. Pueden medir posiciones relativas de los miembros, pero

no como se desplaza el actor por el escenario. Además, el único valor que utilizan para medir es el grado de apertura, no teniendo en cuenta rotaciones complejas que poseen las articulaciones humanas, como es el caso de los hombros, cadera, tobillos, etc. También tienen el inconveniente de ser pesados, restringir el movimiento del actor y su corto tiempo de vida. En cambio, tienen la ventaja de tener un coste relativamente bajo (en torno a 30.000 €), capaz de registrar los movimientos del actor en tiempo real con un alcance mayúsculo.

### 3.3.1.3. Captura de movimiento inercial

Con el objetivo de capturar el movimiento, los sistemas inerciales se basan en el sistema vestibular humano, el cual regula el sentido de movimiento y del equilibrio, es lo que nos permite situar nuestro cuerpo en el espacio, los desplazamientos y nuestro entorno. El sistema vestibular, ubicado en el oído interno, es un sensor inercial 3D biológico, ya que puede detectar el movimiento angular y la aceleración lineal de la cabeza, permitiendo por su actividad sobre el ojo conservar una imagen estable en la retina.

Un giroscopio de velocidad mide la velocidad angular y, si se integra con el tiempo, proporciona el cambio de ángulo con respecto a un ángulo inicialmente conocido. Un acelerómetro mide las aceleraciones, incluida la aceleración gravitacional  $g$ . Si se conoce el ángulo del sensor con respecto a la vertical, se puede eliminar el componente de gravedad y, mediante integración numérica, se pueden determinar la velocidad y la posición.

De este modo, con la información obtenida de los sensores, esta se transmite a un ordenador, donde se puede observar el movimiento capturado sobre una figura ya animada. Este tipo de sistemas no utiliza mecanismos externos como cámaras; y como en el caso de los sistemas ópticos, cuantos más sensores se utilicen, más real será el movimiento capturado, teniendo unos grandes rangos de captura.

### 3.3.2. Captura de movimiento óptico

La detección de movimiento óptico abarca una amplia y variada colección de tecnologías. Los sistemas basados en imágenes determinan la posición mediante el uso de varias cámaras para rastrear puntos predeterminados (marcadores) en los segmentos del cuerpo del actor, alineados con puntos de referencia óseos específicos. La posición se estima mediante el uso de múltiples imágenes 2D del volumen de trabajo. Las técnicas estereométricas correlacionan puntos de seguimiento comunes en los objetos rastreados en cada imagen y utilizan esta información junto con el conocimiento sobre la

relación entre cada una de las imágenes y los parámetros de la cámara para calcular la posición.

Estos sistemas permiten la grabación en tiempo real, ya que utilizan un ordenador que recibe la entrada de una o más cámaras digitales CCD (*Charge-coupled device*) sincronizadas produciendo proyecciones simultáneas. Habitualmente se utilizan en torno a 4 o 32 cámaras, siempre y cuando no se añadan de forma innecesarias, ya que complicarían el procesamiento de la información.

Las cámaras utilizadas en este tipo de sistemas tienen una velocidad de captura de entre 30 y 1.000 fotogramas por segundo. Estos sistemas se deben calibrar mediante el rastreo de un objeto visible, de modo que se calcule la posición de cada cámara respecto de ese punto, en el caso de que una cámara se mueva, sería necesario recalibrar el sistema. La interferencia de otras fuentes de luz o reflejos también puede ser un problema que puede resultar en los llamados marcadores fantasma.

Existen varias clases de sensores para este tipo de captura de movimiento:

### **3.3.2.1. Marcadores activos**

Este sistema está compuesto por leds que emiten su propia luz para crear un plano de luz que atraviesa la imagen determinando la posición del actor, de esta manera se consigue aumentar la distancia a la que se puede desplazar el artista. La posición de los marcadores se determina iluminando uno o varios indicadores, de manera sincrónica a las cámaras en cada instante de tiempo, a una frecuencia de muestreo muy alta. De modo que los indicadores deben estar sincronizados con todas las cámaras para realizar una sola captura en cada iluminación.

Son más apropiados para aplicaciones de mapeo que para el seguimiento dinámico de movimiento del cuerpo humano. Este tipo de sistemas ópticos sufren problemas de oclusión (línea de visión) cuando se bloquea una trayectoria de luz requerida por el sistema.

### **3.3.2.2. Marcadores pasivos**

Los sistemas ópticos con marcadores pasivos utilizan LED de infrarrojos (IR) montados alrededor de la lente de la cámara, junto con filtros de paso de infrarrojos colocados sobre la lente de la cámara y miden la luz reflejada por los marcadores. De esta forma, la luz que reflejan se origina cerca de las cámaras, siendo recogida por estas.

### **13 3.4. MÉTODOS DE CAPTURA DE MOVIMIENTO EMERGENTES**

Los sistemas ópticos basados en LED de pulso miden la luz infrarroja emitida por los LED colocados en las partes del cuerpo. Además, es posible el seguimiento de objetos naturales mediante una cámara sin la ayuda de marcadores, pero en general es menos preciso. Se basa en gran medida en técnicas de visión por ordenador para el reconocimiento de patrones y, a menudo, requiere grandes recursos computacionales. Este tipo de sistemas pueden capturar un gran número de marcadores a una frecuencia de muestreo de hasta 2000 fotogramas por segundo. Se suelen utilizar principalmente para el registro de movimiento facial.

### **3.4. Métodos de captura de movimiento emergentes**

En la actualidad, el sector cinematográfico y la industria de los videojuegos ha generado la necesidad de crear personajes con gestos y movimientos más realistas, de este modo, se ha conseguido que la captura de movimiento se convierta en una herramienta esencial, ya que permite una mayor inmersión.

Gracias a estos sectores, han surgido nuevas técnicas de *Motion Capture*, en la que cámaras y software de inteligencia artificial pueden realizar un enfoque de captura de movimiento sin la necesidad de marcadores. El software permite que, mediante algoritmos, se analice al usuario que realiza las acciones, identificando formas humanas, movimientos, expresiones y gestos.

#### **3.4.1. Visión artificial**

La visión artificial[3] es una disciplina científica que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica para que puedan ser tratados por un ordenador. Tal y como los humanos usamos nuestros ojos y cerebros para comprender el mundo que nos rodea, la visión artificial trata de producir el mismo efecto para que los ordenadores puedan percibir y comprender una imagen o secuencia de imágenes y actuar según convenga en una determinada situación. Actualmente se está utilizando cada vez más para el análisis y tratamiento de imágenes mediante algoritmos de inteligencia artificial.

Atendiendo a su demanda, se pueden encontrar diferentes tecnologías que se desarrollan a continuación.

### 3.4.1.1. Microsoft Azure Kinect DK

Previamente al actual dispositivo *Azure Kinect DK* (véase Figura 3.5), Microsoft en 2010 desarrolló *Kinect* para *Xbox-One*, un dispositivo pensado para su uso en los videojuegos. Este periférico prescinde de mandos gracias a que dispone de una cámara RGB, un sensor de profundidad, un proyector de luz infrarroja, un micrófono bidireccional y un procesador que utiliza algoritmos para procesar las imágenes tridimensionales.

Figura 3.5: Azure Kinect DK

En marzo de 2020, Microsoft actualizó su dispositivo a la versión *Azure Kinect DK* [4], se trata de un kit para desarrolladores con sensores de inteligencia artificial avanzados que proporcionan modelos sofisticados de visión y voz por ordenador. Este nuevo periférico contiene un sensor de profundidad, una matriz de micrófonos espaciales con una cámara de video y un sensor de orientación como un dispositivo pequeño todo en uno con múltiples modos, opciones y kits de desarrollo de software (SDK) con un gran potencial que se pueden conectar a Azure Cognitive Services, Azure Machine Learning y Azure IoT Edge.

Para localizar los movimientos de los objetos 3D, el periférico, transmite una luz infrarroja a través del escenario, lo que permite conocer el tiempo que tarda la luz en ser reflejada por los objetos. El sistema actúa como un sonar, determinando el tiempo que tarda en reflejarse la luz, estableciendo la distancia a la que se encuentran los objetos en tiempo real.

### 3.4.1.2. OpenCV

*Open Source Computer Vision*[5] comenzó como un proyecto de investigación en Intel. Actualmente es la biblioteca más popular de visión artificial, contando con implementaciones de más de 2500 algoritmos optimizados, que

## 15 3.4. MÉTODOS DE CAPTURA DE MOVIMIENTO EMERGENTES

incluyen un conjunto completo de algoritmos de aprendizaje automático y visión por ordenador clásicos y de última generación. Estos algoritmos se pueden usar para detectar y reconocer rostros (véase Figura 3.6), identificar objetos, clasificar acciones humanas en videos, rastrear movimientos de cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, producir nubes de puntos 3D a partir de cámaras estéreo, etc. Además, está disponible de forma gratuita para fines comerciales y de investigación.

Figura 3.6: Detección Facial con OpenCV

### **3.4.1.3. Deepfake**

Otro de los sistemas emergentes de visión artificial que utilizan el aprendizaje automático y la inteligencia artificial para manipular o generar contenido visual es el *deepfake*. Se trata de una técnica que permite editar videos falsos de personas que aparentemente son reales (véase Figura 3.7), utilizando para ello algoritmos de aprendizaje no supervisado, y videos o imágenes ya existentes.

Para crear un *deepfake*[6], es necesario recopilar cientos de imágenes de las dos personas que queremos sustituir. Inicialmente se crea una codificación de todas las imágenes utilizando una red neuronal con *deep learning*, posteriormente se utiliza un decodificador para reconstruir las imágenes, este proceso tiene un alto coste de procesamiento de GPU, necesitándose aproximadamente 3 días para generar resultados decentes (después de repetir el procesamiento de imágenes más de 10 millones de veces). Con todo ello se extraen las características más importantes para recrear la imagen original sustituyendo al usuario B con las características del usuario A en el video original.

Uno de los ejemplos más destacados del cine utilizando esta técnica es en

A deepfake image of Barack Obama's face superimposed onto a different body, appearing to be in a military uniform.

Figura 3.7: *Deepfake* de Barack Obama

la tercera saga de *Star Wars*, donde en la película *Rogue One una historia de Star Wars* la Princesa Leia aparece con la cara de *Carrie Fisher* cuando era joven, cuando en realidad fue interpretada por la actriz noruega *Ingvild Deila* (véase Figura 3.8). Años más tarde, la actriz falleció poco después de haberse completado el rodaje de la película *Los últimos Jedi* y, contando con ella para aparecer en el *Episodio IX - El ascenso de Skywalker*, fue necesario aplicar nuevamente las mismas técnicas para dar vida de nuevo a *Carrie Fisher* como la Princesa Leia.

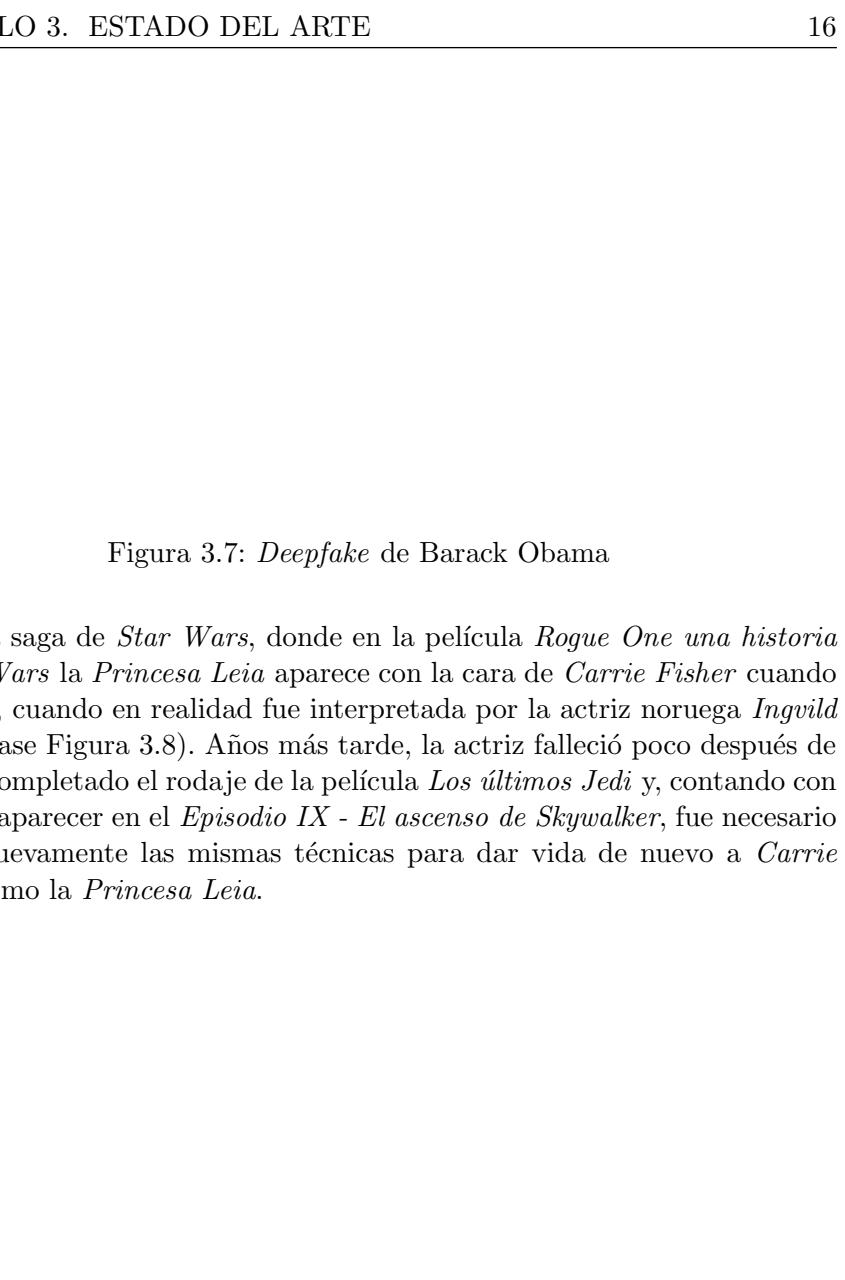
A deepfake image of Carrie Fisher as Princess Leia from Star Wars, with her face replaced by a different woman's face.

Figura 3.8: *Deepfake* de la Princesa Leia en Star Wars

### 3.4.2. Realidad virtual

La realidad virtual es la tecnología que ofrece al usuario la sensación de estar en otro lugar, es decir, la tecnología engaña a tus sentidos, oculta el mundo real a tus ojos y te sumerge en uno nuevo de manera totalmente inmersiva. Se trata de una experiencia sensorial completa, simulando los espacios diversos en los que podemos explorar e interactuar el entorno tal y como si estuviéramos ahí realmente.

### 17 3.4. MÉTODOS DE CAPTURA DE MOVIMIENTO EMERGENTES

Para adentrarse en un entorno virtual, es necesario colocarse unas gafas especiales para poder visualizar la simulación, estas normalmente están conectadas a un ordenador, aunque ya existen modelos *all in one*, aunque tienen la desventaja de no ser tan potentes gráficamente.

Para determinar la posición exacta donde el usuario se encuentra situado tanto en el mundo real, como en el mundo virtual, estos sistemas vienen acompañados de unas estaciones base. Estos sensores son colocados en la habitación de juego permitiendo al sistema determinar la ubicación exacta tanto de las gafas de *VR*, como de los controles y *trackers*.

Estos dispositivos de *VR* disponen de sensores ópticos pasivos que reconocen el movimiento de tu cabeza, de manera que cuando la gires hacia un lado hagas el mismo movimiento dentro del mundo virtual en el que estés.

Algunos modelos *all in one* de *VR* no necesitan estaciones base para determinar la posición en la que se encuentran los dispositivos de *VR* en el entorno. Disponen de una serie de cámaras (véase Figura 3.9) incorporadas en las mismas gafas de *VR*, creando un sistema guardián que determina la zona de juego a utilizar en el mundo virtual.

Figura 3.9: Oculus Quest, modelo *all in one* de *VR*

La producción de contenido de animación ha implicado tradicionalmente configuraciones de captura de movimiento muy caras que utilizan docenas de cámaras. Esta producción de animación de alta calidad estaba limitada a grandes estudios con grandes presupuestos. Pero en los últimos años, los avances en la tecnología de rastreo han traído una producción de animación de excelente calidad a proyectos de menor presupuesto, estudios de cine y juegos independientes.[7]

Sin la necesidad de tener grandes equipos, surgen increíbles resultados con una excelente captura de movimiento (véase Figura 3.10).

Figura 3.10: Ejemplo de *Mocap* con *trackers Vive*

Como base para el estudio de la captura de movimientos y las tecnologías disponibles en este ámbito, se han realizado diferentes consultas a los siguientes proyectos: [2] [1] [8].

### 3.5. Trabajo relacionado

Para la realización de este proyecto se han revisado trabajos anteriores en el área de los *Reactive Virtual Trainers*, además de cuáles son las tecnologías más emergentes en el mercado. En este apartado se describirán los análisis de esos proyectos, y de este modo, poder aclarar y enfocar el desarrollo de este proyecto.

# Capítulo 4

## Tecnologías empleadas

En este capítulo se presentan las tecnologías y dispositivos empleadas para el desarrollo del proyecto. Detallando de forma técnica el *software* y *hardware* utilizado.

### 4.1. Software empleado

Para el desarrollo de este proyecto, el principal software utilizado ha sido **Unity 2019.4.9f1**[9]. Este *Game Engine* hace referencia a una plataforma de desarrollo de software capaz de realizar rutinas de programación que permiten el diseño, la creación y el funcionamiento de un entorno interactivo;

Con el fin de elaborar, compilar y depurar el proyecto de **Unity** se ha utilizado **Visual Studio 2019**[10] con el lenguaje de programación C#, ya que mantienen una sola base de código para todas las plataformas. Respecto a la creación de los personajes, se observó que **Fuse Character Creator de Mixamo**[11] ofrecía una creación de modelos humanos realistas con una alta calidad de las texturas. En cuanto a la configuración previa de las animaciones, se hizo uso de **Mixamo 3D Animation Online Services**[12], ya que es una compañía tecnológica encargada del desarrollo y venta de servicios para la animación de personajes 3D. Como plataforma para el desarrollo colaborativo del proyecto se usó GitHub, utilizando el sistema de control de versiones Git.

#### 4.1.1. Unity

El objetivo de este proyecto trata de como poder realizar una captura de movimiento de calidad sobre el arte marcial afrobrasileño (capoeira) y como representarla gráficamente con las tecnologías más demandadas del mercado.

Un *Game Engine* se especifica como la rutina de programación que permiten el diseño, la creación y la representación de un videojuego. La funcionalidad de un motor de videojuegos consiste en, proveer al desarrollador un motor de renderizado para la creación de gráficos 2D y 3D, un motor físico, sonidos, scripting, animaciones, un escenario gráfico, etc.

Para llevar todo esto a cabo es necesario elegir qué tipo de plataforma de *Game Engine* utilizar en este proyecto. Existe dos grandes plataformas comerciales, por lo que la decisión de elegir entre **Unity**<sup>1</sup> o **Unreal Engine**<sup>2</sup> fue basada en la comunidad que hay detrás de cada uno de ellos, y sobre todo, en lo que se quiere abarcar con este proyecto.

Además de este detalle, es importante conocer el lenguaje de programación con el que trabaja cada uno de ellos. **Unity** está basado en C#, mientras que **Unreal Engine** trabaja con C++.

La versatilidad de **Unity**, ideal para proyectos pequeños, estudiantiles o enfocados al desarrollo de aplicaciones y juegos para móviles, ha conseguido que un 58 % de la población de desarrolladores indies lo elijan como motor de videojuegos. Las facilidades que aporta a la hora de trabajar de una forma visual y escalable lo convierten en un paso muy a tener en cuenta a la hora de realizar desarrollos enfocados a la creación de entornos 3D.

En este caso **Unity**, se adecua correctamente, ya que se pretende realizar un entorno 3D donde los usuarios perciban y aprendan movimientos del arte marcial afrobrasileño.

En un primer momento, **Unity** salió al mercado ofreciendo un *software* de calidad y barato, añadiendo el objetivo de que pequeñas y grandes empresas pudieran utilizarlo por igual, dando además, la posibilidad de que su contenido sea compatible con prácticamente cualquier medio o dispositivo[13]. Gracias a ello, **Unity** creció exponencialmente en el mercado, generando así, una gran comunidad que puede servir de pilar de ayuda para los nuevos desarrolladores.

Como en otras plataformas de desarrollo de software, existen distintas licencias de prueba gratuitas y para estudiantes a las que puedes acceder sin

---

<sup>1</sup><https://unity.com/>

<sup>2</sup><https://www.unrealengine.com/>

complicaciones. Sólo aquél que realmente decida llevar su creación al mercado y supere cierta cantidad de ingresos se verá obligado a pagar la cuota mensual. En la versión personal (licencia gratuita), es necesario promocionar el logotipo de **Unity** cuando se inicia la aplicación una vez compilada.

Otro aspecto importante de **Unity** es el **Asset Store**. Se trata de una biblioteca, que contiene una multitud de paquetes o *Assets* comerciales y gratuitos, creados de forma continua por la comunidad, por **Unity Technologies** o empresas de terceros. Estos paquetes pueden contener modelados 3D, texturas, animaciones y demás contenido que sirva de ayuda para desarrollar un proyecto donde ya sea por tiempo o habilidades técnicas, no sea posible desarrollar cada una de las herramientas que ofrece dicha tienda.

La plataforma para el desarrollo de este proyecto, **Unity**, está compuesta por varios paneles, cada uno de ellos representa información o acciones indispensables para el desarrollo del escenario 3D. La estructuración básica de un juego en **Unity** se realiza mediante escenas. De modo que, para poder tener una previsualización de los elementos que aparecen en una escena, se utiliza el panel denominado *Scene*, este panel representa una dimensión 3D, donde se podrán rotar y desplazar en los ejes de forma tridimensional, para conseguir de esta manera, modificar todos los elementos de la escena en cualquier Ángulo.

El panel que muestra la vista generada por la cámara, es decir, la imagen que se verá cuando se reproduzca el juego se denomina *Game*.

Los elementos incorporados a la escena, se archivan en otro panel, con el nombre *Hierarchy*. Estos pueden ser modelos 3D, cámaras, prefabs(tipo de asset que te permite almacenar un objeto con componentes y propiedades preconfiguradas),sonidos, luces, e incluso objetos de tipo UI(*buttons*, *sliders*, *toggles*) para la implementación de interfaces 2D. Los elementos se muestran de una forma jerárquica, al seleccionar uno, se expondrán sus características y componentes. En el panel, denominado *Inspector*, al seleccionar un componente en el *Hierarchy*, se muestra sus propiedades como el *transform*, el cual muestra la información de su posición y rotación, pudiendo ser modificada.

Existe otro panel denominado *Project*, donde se agrupan en forma de directorio, todos los *scripts*, paquetes, modelos 3D, imágenes y archivos, relacionados con el proyecto.

Para comprobar el funcionamiento desarrollado en **Unity** se necesita activar el *Play mode*. Esta funcionalidad permite poner en funcionamiento el juego, pararlo y detener su ejecución, siendo esto, fundamental para comprobar y depurar un juego[9].

La transición de un script de **Unity** ejecuta una serie de funciones de

eventos en un orden predeterminado. Este diagrama (véase Figura 4.1) describe algunas funciones de eventos y explica cómo encajan en la secuencia de ejecución dentro de la lógica del proyecto.

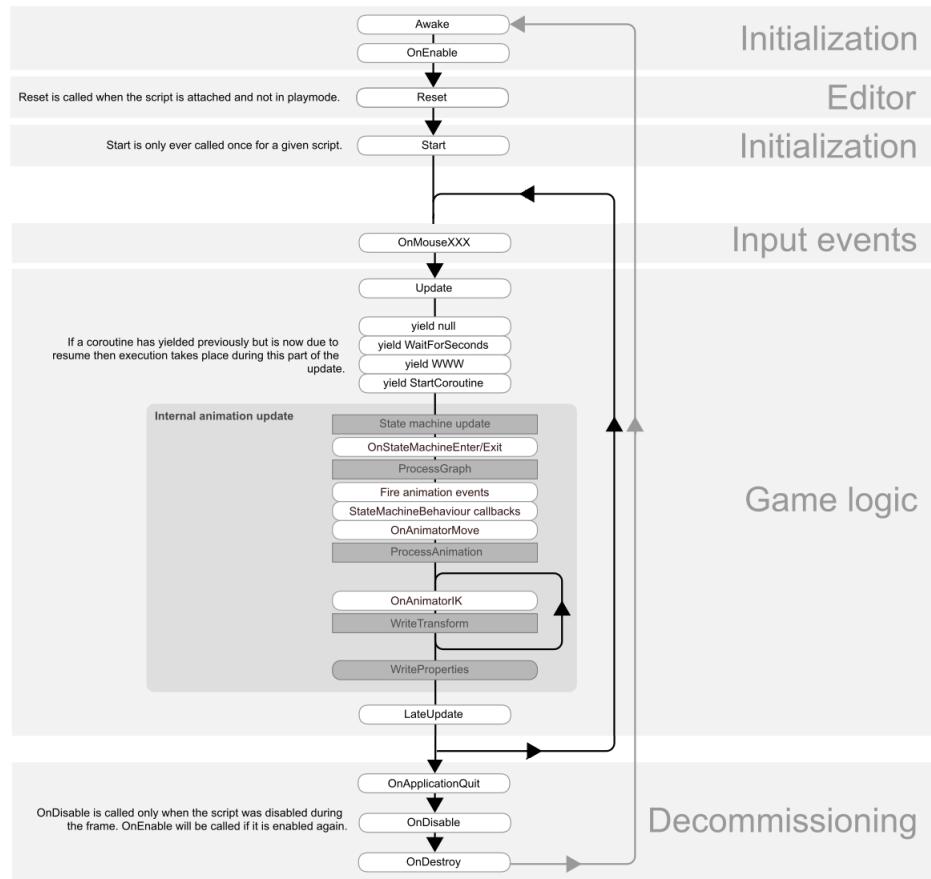


Figura 4.1: Orden de ejecución para los eventos en Unity

Las siguientes funciones, son algunas de las más importantes a la hora de desarrollar un escenario dentro de *Unity*. Estas se llaman cuando una escena se ejecuta (una vez por cada objeto en la escena):

- **Awake:** esta función siempre se llama antes de cualquier función de inicio y también justo después de realizar una instancia de un *prefab*.
- **Start:** se llama al inicio antes de la actualización del primer marco solo si la instancia de secuencia de comandos está habilitada.
- **Update:** la actualización se llama una vez por fotograma. Es la principal función del motor para las actualizaciones de los *frames*.

Para construir todo el proyector de *Unity* se utilizó el potente entorno de desarrollo *Visual Studio 2019*.

*Visual Studio Tools para Unity* es una extensión gratuita de *Visual Studio* que convierte al editor en una poderosa herramienta para desarrollar juegos y aplicaciones multiplataforma con *Unity*.

Si bien el editor de *Unity* es excelente para crear el escenario 3D, pero no puedes escribir el código en él. Con *Visual Studio Tools para Unity*, se pueden usar las funciones familiares de edición, depuración y productividad de código para crear scripts en el proyecto de *Unity* usando C#, pudiendo depurarlos usando las poderosas capacidades de depuración de *Visual Studio*.

Pero *Visual Studio Tools para Unity* es más que eso. También tiene una integración profunda con el editor de *Unity* para que dedique menos tiempo a realizar tareas simples, proporciona mejoras de productividad específicas de *Unity* y pone la documentación de al alcance de su mano.

#### 4.1.2. Adobe Fuse Character Creator

Se trata de una aplicación desarrollada por Mixamo, la cual permite crear personajes 3D únicos mediante una intuitiva interfaz, separando el avatar en diferentes partes, como son la cabeza, el torso, las piernas y los brazos (véase Figura 4.2). Posteriormente a la selección de las diferentes partes del cuerpo ya predefinidas por el sistema, se puede editar cada una de las partes del personaje, ya sean los rasgos de la cara, el tamaño de los brazos, la longitud de las piernas, etc. Al seleccionar la parte del cuerpo que se quiere modificar, simplemente sería necesario pinchar y realizar un *scrool* o deslizamiento hacia la izquierda o derecha para aumentar su tamaño. Por otro lado, para seleccionar el punto exacto donde se ubicarán las diferentes partes del cuerpo como por ejemplo el ombligo, los ojos, las rodillas, etc, se realizará un *scrool* o desplazamiento hacia arriba o abajo.

Al terminar de modelar el personaje, se pasará a la sección de vestuario. Donde existen prendas predefinidas, como camisas, pantalones, zapatos, etc. Al terminar de vestir al avatar, esta aplicación posee la peculiaridad de poder subir al servidor de Mixamo el modelado realizado<sup>3</sup>. Pudiendo completar la configuración de los huesos (*rigger*), con la posibilidad de crear varios esqueletos para un mismo personaje y además incluir las animaciones deseadas.

---

<sup>3</sup><https://www.adobe.com/es/products/fuse.html>

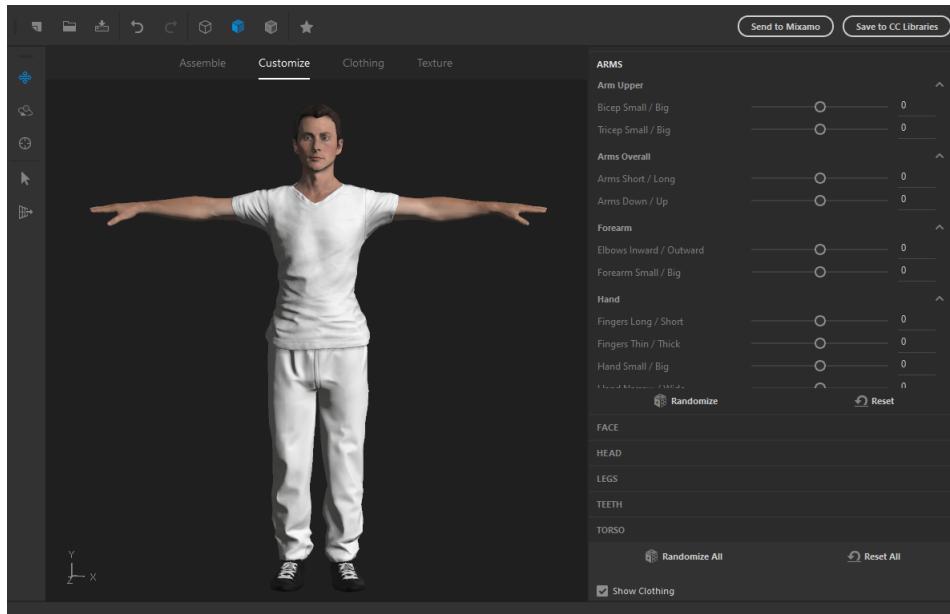


Figura 4.2: Configuración de un avatar con Adobe Fuse Character Creator

#### 4.1.3. Mixamo

Mixamo<sup>4</sup> es una compañía tecnológica (adquirida en 2015 por Adobe System) encargada del desarrollo y venta de servicios para la animación de personajes 3D. Proporcionando un servicio *online*, que permite buscar entre una amplia gama de movimientos, que van desde el combate, pasando por deportes, hasta poder tocar un instrumento (véase Figura 4.3).

Los productos y características de Mixamo son compatibles con todos los paquetes de software 3D, pudiendo realizar una exportación, eligiendo entre varias aplicaciones 3D como: **Unity**, **Unreal Engine**, **Blender** y el conjunto de herramientas de Adobe Creative (entre otros).

Una de las características más sorprendentes de Mixamo, es el *Auto-Rigger*. Ya que ahorras horas de trabajo modelando el personaje y con este sistema, el proceso sucede en cuestión de minutos. Todo lo que se necesita hacer es subir al servidor de Mixamo el personaje 3D (soporta .fbx, .bvh, .zip, .obj), seleccionar el número de huesos que mejor se adapte al esqueleto del personaje y esperar que termine el proceso de *rigger*.

Tras completar la fase de modelado, Mixamo ofrece la posibilidad de animar al personaje con movimientos, que permiten, no solo aplicar cualquier animación, sino también personalizarlas, ya que se pueden modificar

<sup>4</sup><https://www.mixamo.com>

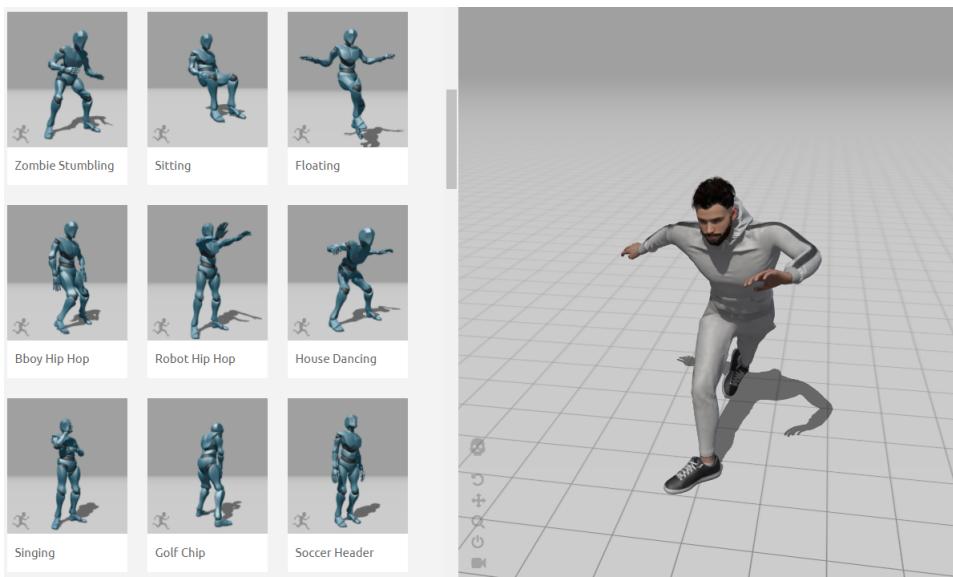


Figura 4.3: Servicio de animaciones de Mixamo

en tiempo real con el personaje, obteniendo el resultado deseado. Por último, descargarlos en cualquier formato que se necesite (.fbx, .bvh, .dae) y fácilmente se podrá combinar de nuevo en las escenas 3D.

Estas animaciones utilizan una base matemática, lo que facilita su uso para los diferentes personajes que se deseen animar. Tradicionalmente era extremadamente lento, costoso y complicado, lograr modelar personajes 3D con resultados de alta calidad. En cambio, mediante esta técnica, Mixamo intenta reducir los tiempos de desarrollo, pudiendo obtener unas animaciones de calidad, sin la necesidad de contar con altos recursos.

## 4.2. Hardware empleado



## Capítulo 5

# Desarrollo del Proyecto

Una vez elegida la tecnología y las plataformas que se van a utilizar se empieza a preparar el desarrollo del proyecto. En este capítulo se explicará que librerías y paquete de **Unity** que se necesitan para tener un entorno de desarrollo potente e innovador. Además, se explicará cuales han sido las metodologías utilizadas para plasmar el proceso seguido para realizar el *Mocap*. El tipo de información guardada, como se ha realizado la grabación de los movimientos captados por los sensores de realidad virtual y cual ha sido la elección para mostrar dicha información.

En la siguiente sección se especificará el proceso realizado para grabar los movimientos capturados, de modo que el profesor obtendrá una visualización directa de cómo ha quedado grabado el movimiento deseado, sin la necesidad de quitarse las gafas de realidad virtual.

Para ofrecer una aplicación dinámica y que el alumno reciba un *feedback* gestual de los movimientos, se ilustra como se diseña e implementan avatares con animaciones en **Unity**.

De forma sucesiva, se describe el diseño de los escenarios 3D para ubicar el avatar principal de la aplicación (profesor). De este modo se han diseñado dos escenarios, uno para grabar y reproducir los movimientos realizados por el profesor dentro de **VR** y otro en **AR** donde el alumno podrá visualizar las veces necesarias los movimientos realizado por el profesor.

Posteriormente se elaboran una serie de escenas en **Unity** que servirán para ilustrar la interfaz para cada una de las aplicaciones (**VR** y **AR**). Se ha creado un entorno diferenciado según las necesidades dadas por las diferentes plataformas a desarrollar en el proyecto. Entre estos escenarios las aplicaciones cuentan con las siguientes escenas:

- VR: Escena principal de VR, escena de grabación de movimientos, escena de reproducción de los movimientos guardados.
- AR: Escena principal de AR (donde el alumno seleccionará el nivel de aprendizaje) y la escena para realizar el entrenamiento.

## 5.1. Investigación para el desarrollo de *Mocap* en Realidad Virtual y Realidad Aumentada

Parar desarrollar cualquier aplicación en **Unity** es necesario conocer el contenido que te puede ofrecer la tienda de este *Game Engine*. El **Asset Store de Unity** es el hogar de una creciente biblioteca de *assets* comerciales y gratuitos creados por **Unity Technologies** y miembros de la comunidad. Hay una gran cantidad de *assets* disponibles, desde texturas, modelos y animaciones hasta ejemplos de proyectos completos, tutoriales y extensiones del editor. Estos *assets* son accesibles desde una interfaz simple dentro de **Unity** y son descargados e importados directamente en el proyecto creado.

### 5.1.1. SteamVR

El primer paquete necesario para poder desarrollar una aplicación de VR en HTC Vice es SteamVR<sup>1</sup>. Con este *Asset*(nombre que se le da a los paquetes de Unity) los desarrolladores podemos dirigir una API a la que se pueden conectar todos los auriculares populares de RV para PC. El moderno **SteamVR Unity Plugin** gestiona tres cosas principales para los desarrolladores: la carga de modelos 3D para los controladores de RV, el manejo de la entrada de esos controladores y la estimación del aspecto de la mano mientras se utilizan esos controladores. Además de gestionar estas cosas, tenemos un ejemplo de Sistema de Interacción para ayudar a poner en marcha una aplicación de VR. Proporcionando ejemplos concretos de interacción con el mundo virtual y las diferentes APIs.

Además, nos permite acceder a los juegos a través de una interfaz que se proyecta en una habitación de nuestra realidad virtual, que podemos aprovechar para jugar a cualquier juego que tengamos en Steam VR.

Este paquete tiene todo lo necesario para reconocer la conectividad de los dispositivos utilizados en VR y para poder utilizar los datos obtenidos de los sensores colocados en la habitación de juego.

El Asset nos ofrece una serie de ejemplos básicos para poder comprender

---

<sup>1</sup><https://www.steamvr.com/es/>

mejor el funcionamiento de VR. A su vez, es necesaria la instalación de **Steam**<sup>2</sup> ya que trae incorporados los drivers imprescindibles para la correcta configuración de los dispositivos utilizados en VR(*controles* y *trackers*).

Los ejemplos que tiene implementado el paquete de **SteamVR Plugin** son variados y pueden servir como un buen punto de partida para desarrollar el proyecto. La escena de ejemplo *Interactions\_Example* incluye todos los componentes principales y es un buen lugar para familiarizarse con el sistema. La escena contiene los siguientes elementos:

- **Player:** este *prefab* es el núcleo de todo el sistema. La mayoría de los demás componentes dependen del jugador para estar presente en la escena.
- **Teleporting:** el *prefab* *Teleporting* maneja toda la lógica de teletransportación del sistema.
- **InteractableExample:** muestra un interacción simple sobre los aspectos básicos para recibir mensajes de las manos y cómo reaccionar ante estas notificaciones.
- **Throwables:** muestra cómo se puede utilizar el sistema para interactuar con los objetos y crear diferentes tipos de objetos tirables.
- **Skeleton:** diferentes objetos de modelos de mano junto con opciones para determinar a qué mano corresponden del esqueleto.
- **Proximity Button:** una tarea común es presionar un botón. Los botones físicos son más satisfactorios de usar que las interfaces planas, pero los sistemas de interacción física pueden volverse complejos rápidamente. Se ha incluido un botón que se puede presionar con solo estar cerca de un controlador.
- **Interesting Interactables:** Estos son ejemplos un poco más complejos del uso del sistema **Skeleton Poser** junto con **Throwables**. Dependiendo del objeto con el que interactuar obtienes diferentes poses de acción.
- **UI:** muestra cómo se manejan las sugerencias en el sistema de interacción y cómo se puede usar para interactuar con los *widgets* de la interfaz, como botones.
- **LinearDrive:** este es un objeto un poco más complejo que combina algunas piezas diferentes para crear un objeto animado que se puede controlar mediante interacciones simples.

---

<sup>2</sup><https://store.steampowered.com>

- **CircularDrive:** esto muestra cómo las interacciones se pueden restringir y mapear de manera diferente para realizar movimientos más complejos.
- **Longbow:** es uno de los objetos más complejos en el *Asset* y muestra cómo se pueden combinar piezas simples para crear una mecánica de juego completa.

Repasando los diferentes objetos en esta escena de ejemplo te das una amplia idea de la amplitud del sistema de interacción y cómo combinar sus diferentes partes para crear acciones complejas.

Una vez explicado los ejemplos, se elige cuál de ellos se podría utilizar para aprovechar su funcionalidad y tener un apoyo base para el desarrollo del proyecto. Los ejemplos seleccionados serán el *Player*, *Skeleton* y *UI*. Más adelante se explica con más detalle que se utiliza de estos ejemplos.

### 5.1.2. Final IK

El uso de la realidad virtual presenta muchos desafíos nuevos para el diseño y desarrollo de captura de movimiento, entre ellos el problema de la cinemática inversa (*Inverse Kinematics*).

La cinemática inversa es la técnica que permite determinar el movimiento de una cadena de articulaciones para lograr que un actuador final se ubique en una posición concreta. El cálculo de la cinemática inversa es un problema complejo que consiste en la resolución de una serie de ecuaciones cuya solución normalmente no es única.[14]

Este concepto es muy importante para el desarrollo de animaciones en 3D, donde se utiliza para conectar físicamente los personajes del juego en el mundo tridimensional, tales como la sujeción rígida de los pies en un terreno. Una figura animada se modela con un esqueleto de segmentos rígidos conectados con las articulaciones. El problema de cinemática inversa radica en calcular los ángulos de las articulaciones para una pose deseada. A menudo es más fácil para los diseñadores, artistas y animadores definir la configuración espacial de un conjunto sobre las partes móviles, como los brazos y las piernas, en lugar de manipular directamente los ángulos de las articulares.

Por lo tanto, la cinemática inversa se utiliza en los sistemas *Mocap* para animar las posiciones de las articulaciones. El conjunto de un esqueleto humano se modela como eslabones rígidos conectados por articulaciones que se definen restricciones geométricas.

Para realizar un movimiento de cualquier extremidad del cuerpo, se re-

quiere el cálculo de los ángulos de las otras articulaciones conectadas con esa parte del cuerpo. Por ejemplo, la cinemática inversa permite mover la mano de un modelo humano 3D con una posición y orientación deseada. Para su correcto funcionamiento es necesario tener un algoritmo que seleccione el ángulo de rotación correcto para cada una de las articulaciones del cuerpo humano, como la muñeca, el codo y los hombros.

Para solventar el problema de la cinemática inversa (véase Figura 5.1) en las articulaciones del cuerpo humano, se estudió una serie de *Assets* capaces de solucionar el problema.

Figura 5.1: Diferencia entre *Forward Kinematics* e *Inverse Kinematics*

La primera elección fue UMotion<sup>3</sup>, ya que se trata de un editor de animación que ofrece potentes herramientas y flujos de trabajo dentro de Unity. La creación de animaciones en la misma aplicación y situación en la que se van a utilizar simplifica todo el flujo de trabajo acelerando el desarrollo del proyecto.

El problema surgió cuando se intentan asignar los diferentes componentes de VR dentro del *Asset*, no era posible realizar dicha acción, no admitía VR, por lo que fue descartada esta opción.

Actualmente no hay muchas soluciones *IK* de cuerpo completo disponibles que cumplan con los requisitos muy específicos del desarrollo en realidad virtual.

Además de la precisión y la calidad general de la cinemática inversa, también es vital que el *Asset* sea altamente eficiente y eficaz, ya que la realidad virtual tiene una gran carga de CPU y GPU.

Por lo tanto, la realidad virtual requiere que *IK* se resuelva no solo en alta frecuencia, sino también en alta calidad: todo se vuelve observable con

---

<sup>3</sup><https://www.soxware.com/umotion/>

gran detalle y en primera persona más aun, ya que incluso debe estar a la altura de la comparación con la realidad.

Teniendo en cuenta todo esto, se optó por la solución **Final IK**<sup>4</sup>, la cual cumplía con todos estos requisitos, incluyendo la compatibilidad con VR.

### 5.1.3. ARCore

En la industria para el desarrollo de aplicaciones AR existe una gran competencia. Esto se debe a la creciente popularidad que grandes empresas tecnológicas, como Apple y Google, están ejerciendo con sus propios *SDKs* de desarrollo de AR. Apple lanzó **ARKit** en 2017, y solo un año después, Google presentó **ARCore**.

Las aplicaciones de AR están desarrolladas para dispositivos móviles que pueden ser dispositivo iOS, Android o auriculares AR más sofisticados como Hololens<sup>5</sup> y Magic Leap<sup>6</sup> utilizados en soluciones empresariales más profesionales.

En la actualidad, existen tres *SDKs* de AR integrados en Unity como *Game Engines*. Se trata de ARkit, ARCore y Vuforia.

ARKit es un conjunto de herramientas creadas por Apple para ayudar a los desarrolladores a crear aplicaciones de realidad aumentada para dispositivos iOS. Con ARKit solo se puede desarrollar aplicaciones AR para iPhones y iPads, más detalladamente desde el iPhone 6s en adelante y iPads a partir del iPad Pro.

ARKit se lanzó con IOS 11 en 2017. En ese entonces, al desarrollar aplicaciones AR, se suponía que debía usar el marco de desarrollo SceneKit de Apple, sobre el cual se construyó la primera versión de ARKit. SceneKit se lanzó en 2012 y recibió pocas actualizaciones hasta la llegada de RealityKit, la tercera versión de ARKit.

ARKit 3 viene con varias características nuevas e innovadoras, como:

- **Oclusión ambiental:** el contenido 3D AR pasa de manera realista detrás y delante de las personas en el mundo real.
- **Seguimiento de caras:** detección de hasta tres rostros a la vez.
- **Captura de movimientos:** utiliza poses y gestos, que interactúan con los movimientos humanos.

<sup>4</sup><http://root-motion.com/>

<sup>5</sup><https://www.microsoft.com/es-es/hololens>

<sup>6</sup><https://www.magicleap.com/en-us>

**Vuforia** es una de las empresas de RA más antiguas del mercado. Después de su adquisición en 2015 por PTC Inc., **Vuforia** ha ampliado su línea de herramientas orientadas a RA. Estas herramientas ahora incluyen productos como Vuforia Engine y Vuforia Studio, que se utilizan en el desarrollo de aplicaciones de RA.

**Vuforia** puede ejecutarse tanto en iOS como en Android e incluso en los modelos más antiguos de iPhone con los que ARKit no es compatible. Además, Vuforia usa ARKit o ARCore cuando el hardware en el que se ejecuta lo admite, de lo contrario, puede utilizar su propia plataforma.

Estas serían algunas de las características más señaladas de este *SDK*:

- **Seguimiento de objetos:** ofrece detección de objetos , de modo que tanto las imágenes como las formas pueden actuar como marcadores.
- **Reconocimiento de textos:** detección de textos, como si se tratase de objetos 3D.
- **VuMark:** este sistema es capaz de detectar tanto imágenes como códigos QR.

El último de los *SDKs* disponibles en el mercado es **ARCore**. Es la respuesta que lanzó Google en 2018 al **ARKit** de Apple. Se trata de una plataforma para el desarrollo de aplicaciones AR en Android (7.0 o superior) e iOS (11 o superior). Además, este kit de herramientas de desarrollo de AR está disponible de forma gratuita tanto para Unity como para Unreal Engine.

**ARCore** ofrece grandes posibilidades para el desarrollo de aplicaciones AR, entre las que caben destacar tres de ellas:

- **Seguimiento del movimiento:** es crucial no solo poner objetos virtuales en el mundo real, sino también asegurarse de que se vean realistas desde todos los ángulos. ARCore garantiza esto alineando la cámara virtual 3D que muestra su contenido 3D con la cámara del dispositivo.
- **Oclusión ambiental:** este sistema detecta planos y puntos característicos para que pueda colocar correctamente objetos virtuales en superficies planas reales. Por ejemplo, objetos en una mesa o paredes.
- **Estimación de la luz:** utilizando la cámara de un teléfono, ARCore puede detectar las posiciones de iluminación actuales en el mundo físico. Por lo tanto, este sistema ilumina los objetos virtuales de la

misma manera que los objetos reales, lo que aumenta la sensación de realismo.

Las tres plataformas son perfectamente capaces de proporcionar las herramientas necesarias para el desarrollo de una aplicación de AR.

La decisión final radicó en utilizar un ecosistema en el cual no existiera una limitación en la plataforma y poder aprovechar la potencia del *SDK*, junto con los numerosos servicios que ofrecía. Por todo ello, se eligió **ARCore** de Google, ya que brinda una mayor flexibilidad en cuanto a los términos donde desplegar la aplicación.

## 5.2. Grabación y desarrollo de *Mocap* en VR

Como el objetivo principal de este proyecto es la captura de movimiento para entrenamiento de actividades físicas en VR, se selecciona como ejemplo de estudio, **VRIK Calibration**, dado su potencial para calibrar las características personales del profesor, y el seguimiento realizado por el avatar virtual.

### 5.2.1. Investigación base

VRIK tiene su propio conjunto de restricciones integradas y los límites de rotación no se pueden utilizar en el proceso de resolución. Sin embargo, es posible aplicar *RotationLimits* encima de VRIK, por ejemplo, para asegurarse de que los huesos de la mano no se dobrén de forma poco natural más allá de los límites razonables. Para hacer esto, tendríamos que deshabilitar los límites de rotación en el inicio para tomar el control de la actualización de las posiciones del avatar y luego actualizarlos usando *VRIK.solver.OnPostUpdate*

```
public VRIK ik;
public RotationLimit[] rotationLimits;
void Start() {
    foreach (RotationLimit limit in rotationLimits) {
        limit.enabled = false;
    }
    ik.solver.OnPostUpdate += AfterVRIK;
}
private void AfterVRIK() {
    foreach (RotationLimit limit in rotationLimits) {
        limit.Apply();
```

```

        }
    }
}
```

Tras revisar las restricciones que se pueden utilizar en las rotaciones de los huesos de un personaje humanoide en Unity, nos adentramos en la escena de calibración de VRIK. Para este proceso, es necesario que el avatar a utilizar tenga definidos los huesos del esqueleto dentro del *mesh* (clase de Unity que permite crear o modificar mallas a partir de scripts) y la relación entre cada uno de ellos. El proceso para realizar este objetivo se denomina *rigging*. Si el avatar a utilizar no tuviera esta característica se podría conseguir usando el *Auto-Rigger* de Mixamo, ya definido en la sección 4.1.3 cuando se describió el software utilizado en el proyecto.

Ya en la escena **VRIK Calibration** se observa una demostración de como usar el calibrador VRIK, de modo que se ayude a calibrar las posiciones donde se encuentran los componentes de VR que harán que el sistema de captura de movimiento tome forma.

Para que un sistema *Mocap* sea de calidad y dé la sensación de realismo al reproducir los movimientos grabados, es necesario como mínimo tener tres puntos de seguimiento, como por ejemplo las gafas de realidad virtual y dos controles, uno para cada mano. Este tema sobre como captura los movimientos los sensores ópticos se trato en la sección 3.4.2.

Visualizando la escena de demostración **VRIK Calibration** y aunque con tres puntos de seguimiento se puede hacer un *Mocap* decente, se decidió utilizar una captura de movimiento con seis puntos de detección. Esta decisión se llevó a cabo ya que el arte marcial afrobrasileño (capoeira) realiza movimientos complejos y utiliza todas las partes del cuerpo, de modo que era indispensable captar toda esa información de la mejor manera posible. Para ello se desarrolló un script(VICIK) específico para este fin, el cual se describirá más adelante.

Al adentrarnos en los componentes que resuelven el problema ocasionado por la cinemática inversa relacionado con modelos humanoides, se observa el script **VRIK** que a continuación se describen en los siguientes puntos:

- **VRIK.fixTransforms:** en el caso de habilitar esta opción, esta solución arreglará todos los *Transforms* (componente de Unity que se utiliza para almacenar y manipular la posición, rotación y escala de un objeto) utilizados a su estado inicial en cada *Update* (descrito en la sección 4.1.1). Evitando posibles problemas ocasionados por huesos no animados. Este problema también ayudamos a solventarlo con la creación del script **VICIK** utilizando las referencias de los seis puntos de seguimiento utilizados en este proyecto.

- **VRIK.references:** se trata del mapeo óseo de un avatar humanoide, en el caso de que el modelo 3D a utilizar contenga las 22 referencias correspondientes a las articulaciones del cuerpo humano esta asignación se hará automáticamente, por el contrario si el avatar a utilizar no contiene estas referencias de forma ordenada, será necesario realizar esta asignación de forma manual.
- **VRIK.solver:** es el encargado de realizar junto con la asignación de los seis componentes que vamos a utilizar para la captura de movimiento, que todo el sistema funcione de forma fluida y estable. Pudiendo ajustar la posición y rotación del avatar virtual, para que coincida con la orientación de cada uno de los huesos del profesor.

### 5.2.2. Desarrollo de la grabación de movimientos

Después de haber realizado la investigación y compresión del material que se puede aprovechar para realizar una captura de movimiento, se plantearon una serie de desafíos e implementaciones que se deben realizar.

En primer lugar se creó el script **VICIK** el cual gestionará el conjunto de la captura de movimiento. Para ello fue necesario determinar un *Game Object* (clase base para todas las entidades en una escena en Unity) de tipo **VRIK**, el cual ya nos permitía acceder a todas las características de este objeto, y por lo tanto modificar las referencias del script **VRIK** de forma ordenada.

Como se describió anteriormente, **VRIK.references**, contiene todas las referencias óseas del cuerpo humanoide a utilizar, pero en el caso de que el modelo 3D(avatar) que vayamos a utilizar tenga una jerarquía de huesos diferente a la utilizada en VRIK será necesario pasar un previo proceso de *rigging*. Para realizar este proceso, tras crear nuestro avatar con **Adobe Fuse Character Creator**, subiremos dicho modelo al sistema de *Auto-Rigger* de Mixamo.

Con el avatar rigueado correctamente, procedemos a automatizar el proceso de asignación de las referencias de los huesos del avatar en el script. El método **AutoReferences()** realizar dicha acción. Previamente se crearan las referencias con los nombres de los huesos que utiliza **VRIK.references**, para posteriormente crear otro método **GetBonesReferences()** que vaya recorriendo todas las referencias y vaya asignando los componentes creados, con los utilizados por **VRIK**. Con todo esto, cualquier avatar que contenga los huesos creados por **Adobe Fuse Character Creator** o cualquier *software similar* tendrá este proceso automatizado, y no será necesario realizar una asignación manual para cada avatar.

El siguiente apartado importante para realizar la captura de movimiento, es como poder asignar los componentes de VR (gafas, controles y *trackers*) a los seis puntos que nuestro avatar iba a seguir para grabar los movimientos del *Mocap*.

Para ello se creó otro script denominado **VICAvatar** el cual se encarga de realizar esta acción. Sería necesario crear una serie de referencias que contuvieran la posición y rotación de los sensores que el profesor iba a tener en el cuerpo. Para desarrollar este *Mocap* se decidió que los seis puntos importantes a capturar serían los siguientes:

- **Head:** como vamos a utilizar gafas de VR, este será el objeto principal del *Mocap*.
- **Left Hand:** para las manos utilizaremos los controles, necesarios a su vez para movernos por diferentes zonas del escenario(*teleporting*).
- **RightHand:** al igual que en la mano izquierda, se utilizará otro de los controles para seguir esta mano.
- **Pelvis:** en este caso utilizaremos el primero de los *trackers*, utilizando un cinturón para ello adherido a nuestra cintura.
- **Left Foot:** para la parte de los pies, al igual que para la cintura, destinaremos otro de los *trackers* sujeto al empeine.
- **Right Foot:** al igual que en el pie izquierdo, destinaremos uno de los *trackers* para seguir sus movimientos.

Para crear el entorno de VR, es necesario la utilización de uno de los componentes de **SteamVR**, la clase **Player** vista en la sección 5.1.1. Este objeto actúa como un **Singleton** (véase la Figura 5.2), lo que significa que solo debe existir un objeto **Player** en la escena. Además del objeto **Player** (componente head de nuestro sistema) que es principalmente la cámara que reproducirá las imágenes en sus dos pantallas, a modo de ojos humanos. Es necesario el uso de otro objeto para cada uno de los controles y los tres *trackers* restantes. Este objeto contiene el script **Steam\_VR\_Behaviour\_Pose** el cual se encarga de realizar el seguimiento de un determinado componente óptico. Como previamente ya hemos creado las referencias a cada uno de nuestros huesos (leftHand, rightHand, pelvis, leftFoot y rightFoot), solo faltaría indicarle a cada objeto que tipo de hueso queremos usar.

Con las referencias de los seis huesos creados, ahora lo que tocaría hacer es asignar el seguimiento de los sensores a cada hueso. Para ello es necesario

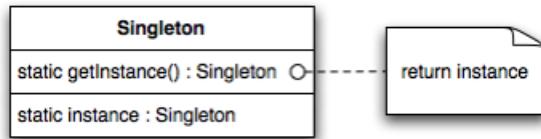


Figura 5.2: Patrón *Singleton* en Unity

crear un objeto de tipo *VICIK* y un método denominado *UpdateBoneReferences()* que realice la asignación para realizar la grabación de la captura de movimiento.

Con todo el escenario ya preparado para realizar la grabación del *Mocap*, se creó la escena principal del proyecto, donde estarán todos estos componentes y scripts mencionados anteriormente.

Para realizar la secuencia de grabación, se creó un panel en VR que permitiese grabar los movimientos a modo de estudio de grabación de música, con los tres botones clásicos (play, pause y stop), además de incluirse un panel para poder escribir el nombre del movimiento que se va a realizar.

El profesor podrá visualizar el movimiento que está realizado a cada instante, ya que verá una imagen de un avatar idéntico a él a modo de espejo, de este modo podrá revisar cada una de las poses que esté realizando mientras graba los movimientos que luego visualizará el alumno en la aplicación de AR.

Cuando el profesor esté grabando un movimiento, podrá repetirlo las veces que consideré oportunas, finalizando la grabación con el botón de stop y volviendo a pulsar el botón de *record* para iniciar de nuevo el proceso. También existe la opción de pausar una grabación, y retomarla instantes después antes de finalizar la grabación, pulsando el botón correspondiente (*pause*). Cuando un movimiento se quiera reproducir antes de finalizar la grabación, existirá la opción de pulsar el botón de *play*, siempre y cuando se haya pulsado el botón de *stop* previamente. Tras toda esta secuencia, se habilitará un botón de finalizar para concluir la grabación del *Mocap* y de nuevo se podría iniciar el proceso de grabación con un nuevo movimiento.

Los movimientos grabados serán guardados como animaciones de Unity, de modo que el profesor podrá visualizar los movimientos que haya grabado hasta el momento, para en el caso deseado, poderlo repetir y borrar el movimiento capturado previamente. Además, estos movimientos capturados serán los que se podrán analizar en el otro escenario del proyecto, el entorno del alumno en realidad aumentada.

### 5.3. Análisis de *Mocap* en AR

Otra parte importante de este proyecto consiste en poder analizar los movimientos previamente grabados por el profesor, dependiendo del nivel del alumno. De esta manera se podrá realizar una corrección del movimiento que necesita practicar el alumno en tiempo real. En esta sección se explica cómo se ha desarrollado esta parte del proyecto con realidad aumentada.

#### 5.3.1. Investigación base

El planteamiento inicial sobre el análisis de los movimientos grabados por el profesor fue muy distinto al que finalmente se desarrolló.

En la mayoría de las artes marciales los movimientos son repetidos miles de veces en cada sesión, es por esto clave optimizar el rendimiento para evitar lesiones y obtener unos resultados eficientes en su ejecución.

La evaluación de los movimientos de capoeira puede ser realizada por el ojo humano de un entrenador, con video análisis o con equipos de biomecánica especializados, como sensores de presión o programas tridimensionales que analicen los movimientos realizados.

En primera instancia se estudió como realizar el análisis de los movimientos capturados por el profesor mediante *Deep Learning* con *OpenCV*. Este sistema de visión artificial (descrito en la sección 3.4.1.2) utiliza una serie de algoritmos basados en modelos (Ejemplos de modelos preentrenados en ImageNet: *Xception*, *VGG19*, *ResNet50*) capaces de seguir la estructura ósea del movimiento del cuerpo humano. Este método se basa en representar las partes del cuerpo humano en segmentos y unirlos mediante puntos, identificando principalmente el torso, la cabeza y las extremidades.

Visualizando la gran cantidad de estudios relacionados con *OpenCV* y el análisis de movimientos, no dejaría de ser un estudio más sobre como analizar y aprender este tipo de arte marcial. Dado que la tecnología a evolucionado en gran medida en estos últimos años, se optó por utilizar una herramienta novedosa y en plena evolución, como es la realidad aumentada.

Para este apartado se observó el desarrollo realizado en VR sobre la grabación de movimiento explicada anteriormente. En esta sección el alumno podrá tener un *feedback* visual inmediato y novedoso, de como poder realizar un movimiento de forma correcta en el arte marcial brasileño (capoeira).

Con el planteamiento de utilizar AR como base para el análisis de los movimientos del profesor y el posterior aprendizaje del alumno, se optó por

desarrollar esta parte del proyecto con **ARCore** (ya se describieron los detalles de esta elección en la sección 5.1.3).

De todo el contenido incluido en el paquete de **ARCore** para Unity, se selecciona como ejemplo de estudio, **HelloAR**, dado el gran potencial que ha incluido **ARCore** en este escenario, un nuevo sistema de oclusión ambiental denominado *Depth API*<sup>7</sup>. Este sistema utiliza la cámara RGB de los dispositivos móviles para crear mapas de profundidad, para posteriormente utilizar esta información y hacer que los objetos virtuales aparezcan con precisión.

Para entender un poco mejor como utiliza **ARCore** el sistema de profundidad vamos a explicar cómo realiza este proceso este *SDK* de realidad aumentada (véase Figura 5.3). En la geometría del mundo real observamos un punto A, y un punto a 2D que representa el mismo punto pero en la imagen con el mapa de profundidad. El valor dado por esta *API* es igual a la longitud de CA proyectada sobre el eje principal. Al trabajar con este sistema es importante destacar que los valores del mapa de profundidad no son la longitud del rayo CA, sino la proyección del mismo.

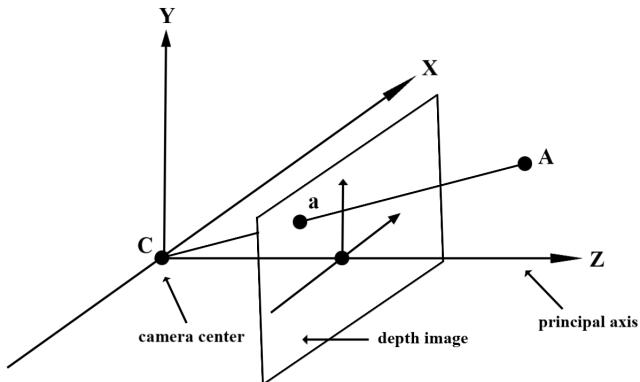


Figura 5.3: Proyección en el plano utilizado en *Depth API*

Ya en la escena de ejemplo **HelloAR** se visualiza como la API necesita tener un mapa del entorno a utilizar para poder colocar los objetos virtuales en el mundo real, permitiendo que **ARCore** establezca un seguimiento completo, detectando la geometría de la superficie con la que se quiere interactuar. Este proceso lo realiza el script **DetectedPlaneVisualizer** necesario en la utilización de este *SDK*.

Otro de los scripts necesarios es **ARCoreSession**, el cual determina cual de las dos cámaras (frontal o trasera) se utilizará en la aplicación a desarrollar. Además serán necesarios dos archivos de configuración denominados *SesionConfig* y *CameraConfigFilter*. Estos archivos con extensión *.asset de*

<sup>7</sup><https://developers.google.com/ar/develop/unity/depth/overview>

*Unity* se utilizan para detallar la configuración de la cámara que **ARCore** utiliza para acceder al sensor de la cámara para una sesión determinada. Estos detalles incluyen, por ejemplo, la velocidad de fotogramas que captura el objetivo y si la cámara a utiliza un sensor de profundidad.

Al crear una nueva sesión de **ARCore**, este utiliza un filtro para determinar la configuración de la cámara que mejor coincida con la lista de configuraciones disponibles. Se puede utilizar el recurso *CameraConfigFilter* para reducir las configuraciones de cámaras disponibles en una serie de dispositivos en tiempo de ejecución, según las necesidades de la aplicación.

Para esta escena de ejemplo, existe un script denominado *HelloARController* el cual controla la instancia de la cámara a utilizar y como colocar los objetos 3D en el mundo real, pudiendo utilizar un plano vertical, uno horizontal o un punto fijo en el entorno. El problema radica en que este script instancia objetos 3D tantas veces como toques la pantalla, por lo tanto desarrollaremos un script propio denominado **ARController** el cual se encargará de realizar una sola instancia del modelo 3D del profesor y otro script denominado **MovementManager** que gestionará de forma automática todos los movimientos grabados previamente en VR para ser asignados a la única instancia de la escena.

### 5.3.2. Desarrollo del análisis de movimientos

Después de haber realizado la investigación y compresión del material necesario para utilizar **ARCore** como sistema de realidad aumentada, se procede a solventar una serie de problemas o necesidades surgidas durante la investigación.

Para crear el entorno de AR, es necesario la utilización de uno de los *prefabs* de **ARCore**, **ARCoreDevices** el cual incluye la configuración de la sesión a utilizar en realidad aumentada (*SessionConfig* y *CameraConfigFilter*). En este escenario donde vamos a desarrollar una aplicación para el análisis visual de los movimientos grabados por el profesor en VR es necesario establecer unas configuraciones avanzadas detalladas a continuación:

- **Config.PlaneFindingMode**: selecciona el comportamiento del sistema de superficie a detectar. En este desarrollo utilizaremos un sistema que solo detecte planos horizontales.
- **Config.CloudAnchorMode**: define una ubicación específica rastreada en el mundo real. De modo que guarda la posición real de objetos 3D para luego ser utilizados. En este desarrollo no utilizaremos esta

opción, ya que no queremos tener un punto fijo para visualizar los movimientos del profesor.

- **Config.FocusMode:** tipo de enfoque de la cámara a utilizar, en los dispositivos compatibles como el utilizado para el desarrollo del proyecto, se utilizará el tipo *Fixed* para optimizar el seguimiento en AR.
- **Config.DepthMode:** modo del mapa de profundidad a utilizar. En los dispositivos compatibles, la mejor profundidad posible se estima en función del hardware y software del dispositivo. Proporciona una estimación de profundidad para cada píxel de la imagen. El inconveniente es que agrega una carga computacional significativa. Para el desarrollo de este proyecto utilizaremos el modo *Automatic* para que pueda ser utilizado en diferentes dispositivos.

Como ocurría en el escenario de VR, es necesario la utilización de un componente que actúa como un **Singleton** (véase la Figura 5.2), de modo que solo puede existir un objeto de tipo **FirstPersonCamera** en la escena. Este objeto como su propio nombre indica, se trata de la cámara que utiliza el sistema para mostrar toda la información desarrollada en realidad aumentada. Este componente estará incluido en el script que se describe a continuación.

Para realizar todo el control de la aplicación de AR, se desarrolló el script **ARController**. Este script controla los siguientes objetos de Unity necesarios para dar forma a esta aplicación:

- **Objeto de tipo DepthMenu:** tipo de Mapa de profundidad que se utilizará, se podrá activar o desactivar en todo momento en tiempo de ejecución (para que los dispositivos no compatibles con este sistema puedan utilizar la aplicación).
- **Objeto de tipo HorizontalPlanePrefab:** se utilizará una instancia única cuando el alumno pulse la pantalla, siendo solamente utilizada para una superficie a detectar de tipo horizontal.
- **Objeto de tipo GameObject:** este **Prefab** utilizará el método **Update()** de Unity (descrito en la sección 4.1.1) para detectar en cada *frame* el instante preciso para crear el avatar del profesor, la posición, rotación y escala que tiene en cada momento dicho modelo 3D.

Para controlar toda la información obtenida en la grabación de movimientos en VR, se creó el script **MovementManager** capaz de gestionar la concurrencia existente entre los diferentes movimientos capturados por el

*Mocap*. En primer lugar, se creará una instancia única del avatar que utilizará el profesor del tipo *Animator*, de este modo el siguiente paso a realizar será la asignación de forma automática de todos los movimientos grabados previamente. Esta asignación se realizará mediante una lista de tipo *Animator* denominada *movements* (`List movements`). Para poder crear una asignación automática de los movimientos, es necesario crear un objeto *AnimatorController* y un componente de Unity del mismo tipo.

Teniendo todo esto, iniciamos el proceso de automatizar la asignación de los movimientos ya incluidos en la lista (ya que han sido importados en el proyecto, en una carpeta específica para tal fin). Para ello se creó un método denominado `AddMovementAnimator()` que recorre la lista y va asignando a modo de diagrama de estados, cada uno de los movimientos, con otro de ellos. De este modo, todos los movimientos podrán ser seleccionados una y otra vez en la interfaz de AR.

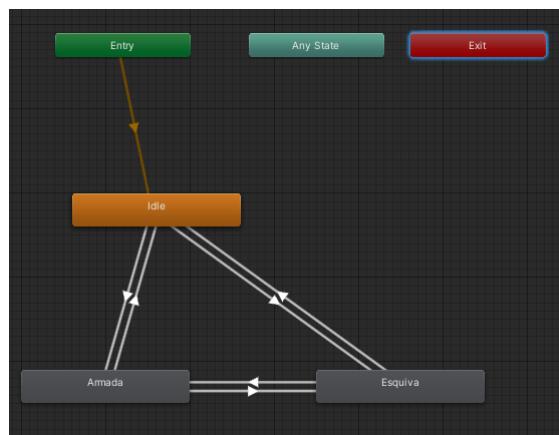


Figura 5.4: Asignación automatizada de los movimientos grabados en el *Mocap*

Además de todo este proceso, se creó un método el cual creará los botones de la interfaz en tiempo de ejecución denominado `MovementChanges()`, de este modo el método recorrerá la lista de movimientos guardados en la lista *movements* y creará tantos botones en la UI como movimientos grabados existan, siempre dependiendo del nivel seleccionado previamente por el alumno.

En el planteamiento de como el alumno analizaría los movimientos grabados por el profesor para su posterior aprendizaje, se decidió que esta parte del proyecto se iba a desarrollar en realidad aumentada (como se explicó en la sección 5.3.1) debido a su gran potencial.

Para este fin, se utilizó un sistema capaz de reproducir, pausar, aumen-

tar y disminuir la velocidad del movimiento realizado por el profesor, para así poder analizar en 360º cada una de las poses que realiza el profesor para ejecutar la transición en cada movimiento del arte marcial brasileño (capoeira). El alumno podrá visualizar diferentes movimientos, dependiendo del nivel elegido al iniciar la aplicación, de modo que el profesor previamente ha grabado movimientos para todos los niveles seleccionables (principiante e intermedio). Todo este sistema ha sido desarrollado en un script denominado *UIController*, creando instancias para cada uno de los botones seleccionables a la hora de reproducir los movimientos, además de los botones de reproducción, pause y el *slider* capaz de aumentar el ritmo que utilizar el profesor en el movimiento capturado en VR.

Otro de los sistemas importantes a desarrollar para dar un efecto más realista es la utilización de la estimación de luz en AR, por defecto viene deshabilitada. Para este tipo de sistemas se pueden utilizar dos tipos de modos de luz ambiental, definidos a continuación:

- **HDR Environmental:** el modo de estimación de luz se establece con o sin reflejos, de modo que el componente de luz ambiental en la escena actualizará la rotación y el color de la luz, además de modificar los componentes de Unity *ambient probe* mediante la propiedad *RenderSettings.ambientProbe* y *reflection probe* mediante la propiedad *RenderSettings.customReflection*.
- **Ambient Intensity:** en este modo de estimación de luz, el componente de luz en la escena de Unity fijará *\_GlobalLightEstimation* propiedad que se utiliza en la propiedad *deARCore DiffuseWithLightEstimation* y *SpecularWithLightEstimation* y otros *shaders* personalizados para ajustar la salida de color final para que coincida con el color de la imagen de la cámara.

En el script denominado *ARController* se asigna mediante el método *SetLightEstimationMode()* el tipo de estimación de luz a utilizar en el proyecto de AR, pasandole al método de tipo *LightEstimationMode* el parámetro *EnvironmentalHDRwithReflections*.

#### 5.4. Diseño y creación de avatares

En esta etapa del proyecto, con el objetivo de dar una sensación más realista al *Mocap*, se han creado una serie de características estéticas en los avatares, siendo esta, similar a los profesionales que practican capoeira. De modo que la experiencia a proporcionar sea más amena para el usuario.

### 5.4.1. Investigación base

En la búsqueda de software dedicado al modelado de personajes 3D se encontraron aplicaciones como **Blender**<sup>8</sup>, **3ds Max**<sup>9</sup>, **ZBrush**<sup>10</sup>, entre otras.

Aunque la idea de esculpir en un escenario 3D puede sonar atractiva, ya que estos sistemas son capaces de crear objetos paramétricos y orgánicos con características de polígono, superficie de subdivisión y modelado basado en *spline* (funciones utilizadas en aplicaciones de modelados 3D que requieren la interpolación de datos, o un suavizado de curvas). Las características interesantes (para los diseñadores en particular) son las herramientas de modelado basadas en NURBS (modelo matemático muy utilizado en programas de modelado 3D para generar y representar curvas y superficies), ya que en estos programas de diseño 3D permiten mallas orgánicas y matemáticamente precisas. Entre las otras técnicas está la capacidad de crear modelos a partir de datos de nube de puntos.

Este tipo de sistemas no son de ninguna manera programas de diseño 3D que puedas dominar intuitivamente. Tienen una curva de aprendizaje empinada, se necesitan muchas horas de práctica para dominar sus muchos pinceles y herramientas, sólo entonces es cuando se producen resultados satisfactorios. De modo que tienen una curva de aprendizaje demasiado larga para las necesidades dadas.

También se examinó una aplicación llamada **MarvelousDesigner**<sup>11</sup> la cual se emplea para desarrollar prendas de vestir. El problema surgía cuando se intentaba importar el avatar creado con **Fuse Character Creator**, ya que no eran compatibles los formatos de las dos aplicaciones y por lo tanto se descartó seguir por esta vía.

En el camino se observó que existía una aplicación para el desarrollo de avatares compatibles con una solución de captura de movimiento y a su vez, un entorno dedicado a la conexión entre la creación de modelos 3D y el ajuste de rigueado que debe tener un avatar para este proyecto. Por este motivo, se tomó la decisión de utilizar **Adobe Fuse Character Creator**<sup>12</sup> como aplicación para el desarrollo de los personajes, ya que se amoldaba perfectamente a las necesidades de este proyecto.

---

<sup>8</sup><https://www.blender.org>

<sup>9</sup><https://www.autodesk.es/products/3ds-max>

<sup>10</sup><https://pixologic.com/>

<sup>11</sup><https://www.marvelousdesigner.com/>

<sup>12</sup><https://www.adobe.com/es/products/fuse.html>

### 5.4.2. Diseño de los avatares

Con el objetivo de crear un entorno más realista, se crearon dos avatares diferentes, cada uno con una estética propia. A la hora de elaborar el vestuario de los personajes, se observó la vestimenta utilizada por los integrantes de escuelas que practican este arte marcial brasileño. Se trata de un pantalón largo y una camiseta o sudadera en color blanco.

Con el propósito de crear una ropa muy similar a la utilizada en capoeira, se usaron prendas prediseñadas en la aplicación **Adobe Fuse Character Creator** como los pantalones, camisetas y sudaderas que utilizarían los avatares del sistema con los retoques apropiados para darle ese color blanco característico de este arte marcial.

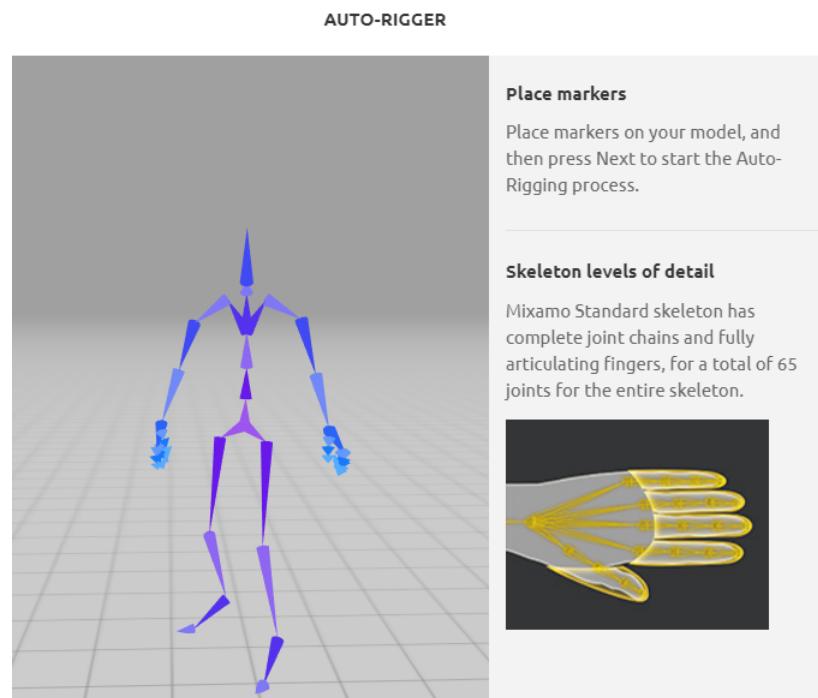


Figura 5.5: Auto-Rig con 65 huesos de Mixamo

Tras completar la estética de los personajes, **Adobe Fuse Character Creator** ofrece la posibilidad de configurar los huesos de los avatares para posteriormente iniciar el proceso de rigueado que suministra **Mixamo**. La misma aplicación **Adobe Fuse Character Creator** proporciona el servicio de conexión con **Mixamo**, de modo que solo sería necesario seleccionar la opción de la aplicación para subir los avatares creados a los servidores de **Mixamo**. Cuando se complete este proceso, la aplicación redirigirá su actividad al navegador web.

**Mixamo** también ofrece la posibilidad de crear varios esqueletos para un mismo avatar, en este caso se presentan cuatro posibles escenarios dependiendo de las necesidades dadas con 25, 41, 49 y 65 huesos respectivamente (véase Figura 5.5). La diferencia mayoritariamente depende de los huesos que queremos que tenga el avatar en las manos.

Una vez investigada la generación y construcción de los huesos de los avatares. El avatar elegido que mejor se adapta a este **Mocap** es el que contiene 65 huesos, ya que demuestra una movilidad correcta y en el futuro podrá ser usado en sistemas de captura de movimiento incluyendo *hand tracking* (véase Figura 5.6) (sistema por el cual, una serie de cámara colocadas en las gafas de VR o AR detectan cualquier movimiento realizado con las manos).

Figura 5.6: *Hand tracking* en Oculus Quest



# Bibliografía

- [1] SARA MÉRIDA MEJÍAS. “Rotoscopia y captura de movimiento. Una aproximación general a través de sus técnicas y procesos en la postproducción”. TFG. 2014.
- [2] Saúl Menéndez Mendoza y Jonathan Rodríguez Marante. “Protocolo de trabajo y banco de pruebas para el uso del equipo de captura de movimientos MOCAP-Optitrack”. TFG. 2015.
- [3] Visión Artificial. *Disciplina científica que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real.* [https://es.wikipedia.org/wiki/Vision\\_artificial](https://es.wikipedia.org/wiki/Vision_artificial). 2020.
- [4] Azure Kinect Dk. *Azure Kinect Dk — Microsoft.* <https://azure.microsoft.com/es-es/services/kinect-dk>. 2020.
- [5] OpenCV. *OpenCV - Biblioteca de software de visión artificial y aprendizaje automático de código abierto.* <https://opencv.org/about>. 2020.
- [6] Jonathan Hui. *Deepfake.* [https://medium.com/@jonathan\\_hui/how-deep-learning-fakes-videos-deepfakes-and-how-to-detect-it-c0b50fbf7cb9](https://medium.com/@jonathan_hui/how-deep-learning-fakes-videos-deepfakes-and-how-to-detect-it-c0b50fbf7cb9). 2018.
- [7] Vive. *VIVE and Motion Workshop Bring Full-Body Interaction To VR.* <https://blog.vive.com/us/2019/06/23/vive-shadow-motion-capture-full-body-vr-tracking>. 2029.
- [8] Víctor Tobes Pérez y Raúl Fernández Pérez. “Uso de Kinect para el entrenamiento de actividades físicas”. TFG. 2017.
- [9] Unity-Technologies. *Unity - Manual: Unity Manual onliner.* English. Ver. Unity 2019.4.9f1. <http://docs.unity3d.com/Manual/index.html>. 2020.
- [10] Microsoft Visual Studio. *Microsoft Visual Studio.* <https://visualstudio.microsoft.com/es/vs/>. 2020.
- [11] AdobeFuseCC. *Adobe Fuse CC - Make customized 3D models to power your designs.* <https://www.mixamo.com/fuse>. 2020.

- [12] Mixamo. *Mixamo 3D Animation Online Services - Animated 3D characters. No 3D knowledge required.* <https://www.mixamo.com/>. 2020.
- [13] Unity-Technologies. *Unity - Game engine, tools and multiplatform.* <https://unity3d.com/es/public-relations>. 2020.
- [14] Wikipedia. *Cinemática Inversa.* [https://es.wikipedia.org/wiki/Cinematica\\_inversa](https://es.wikipedia.org/wiki/Cinematica_inversa). 2020.