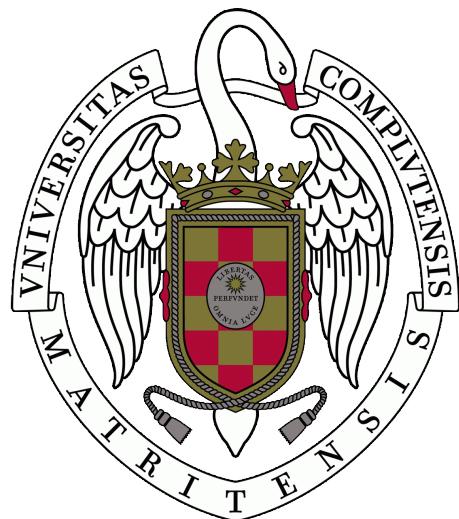


Uso de Realidad Virtual y Realidad Aumentada para el entrenamiento de actividades físicas



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 Realidad Virtual y Realidad Aumentada 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

Dedicatoria	v
Agradecimientos	VII
Resumen	IX
Abstract	XI
1. Introducción	1
1.1. Objetivos del trabajo	1
1.2. Plan de Trabajo	1
1.3. Estructura de la memoria	1
2. Introduction	3
2.1. Objetivos del trabajo	3
2.2. Plan de Trabajo	3
2.3. Estructura de la memoria	3
3. Desarrollo del Proyecto	5
3.1. Investigación para el desarrollo de <i>Mocap</i> en Realidad Virtual y Realidad Aumentada	6
3.1.1. SteamVR	6
3.1.2. Final IK	8
3.1.3. ARCore	10
3.2. Grabación y desarrollo de <i>Mocap</i> en VR	12
3.2.1. Investigación base	12
3.2.2. Desarrollo de la grabación de movimientos	14
3.3. Análisis de <i>Mocap</i> en AR	17
3.3.1. Investigación base	17
3.3.2. Desarrollo del análisis de movimientos	19
3.4. Diseño y creación de avatares	22
3.4.1. Investigación base	23
3.4.2. Diseño de los avatares	24

Índice de figuras

3.1.	Diferencia entre <i>Forward Kinematics</i> e <i>Inverse Kinematics</i>	9
3.2.	Patrón <i>Singleton</i> en Unity	16
3.3.	Proyección en el plano utilizada en <i>Depth API</i>	18
3.4.	Asignación automatizada de los movimientos grabados en el <i>Mocap</i>	21
3.5.	Auto-Rig con 65 huesos de Mixamo	24
3.6.	<i>Hand tracking</i> en Oculus Quest	25

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

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.

3.1. 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.

3.1.1. SteamVR

El primer paquete necesario para poder desarrollar una aplicación de VR en HTC Vice es SteamVR¹. 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

¹<https://www.steamvr.com/es/>

3.1. INVESTIGACIÓN PARA EL DESARROLLO DE *MOCAP* EN REALIDAD VIRTUAL Y REALIDAD AUMENTADA

7 mejor el funcionamiento de VR. A su vez, es necesaria la instalación de **Steam²** 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.

²<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.

3.1.2. 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.[1]

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 3.1) en las articulaciones del cuerpo humano, se estudió una serie de *Assets* capaces de solucionar el problema.

Figura 3.1: Diferencia entre *Forward Kinematics* e *Inverse Kinematics*

La primera elección fue UMotion³, 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

³<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**⁴, la cual cumplía con todos estos requisitos, incluyendo la compatibilidad con VR.

3.1.3. 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 Hololens⁵ y Magic Leap⁶ 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.

⁴<http://root-motion.com/>

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

⁶<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.

3.2. 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.

3.2.1. 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 ?? 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 ??.

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 ??). 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.

3.2.2. 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 3.1.1. Este objeto actúa como un **Singleton** (véase la Figura 3.2), 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

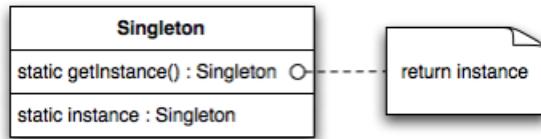


Figura 3.2: Patrón *Singleton* en Unity

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.

3.3. 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.

3.3.1. 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 ??) 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 3.1.3).

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*⁷. 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 3.3). 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.

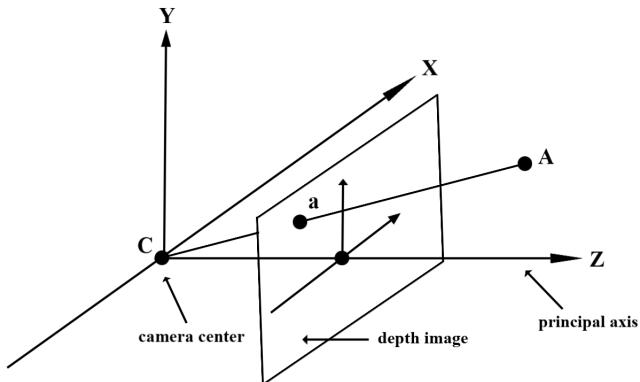


Figura 3.3: Proyección en el plano utilizado en *Depth API*

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*

⁷<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.

3.3.2. 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 Figura3.2), 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 ??) 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.

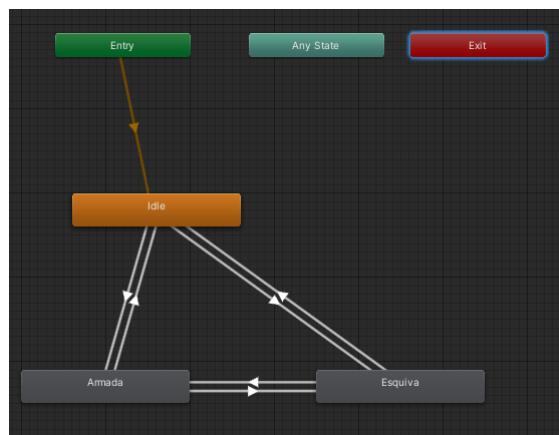


Figura 3.4: Asignación automatizada de los movimientos grabados en el *Mocap*

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 3.3.1) 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*.

3.4. 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.

3.4.1. Investigación base

En la búsqueda de software dedicado al modelado de personajes 3D se encontraron aplicaciones como **Blender**⁸, **3ds Max**⁹, **ZBrush**¹⁰, 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**¹¹ 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**¹² como aplicación para el desarrollo de los personajes, ya que se amoldaba perfectamente a las necesidades de este proyecto.

⁸<https://www.blender.org>

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

¹⁰<https://pixologic.com/>

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

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

3.4.2. 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.

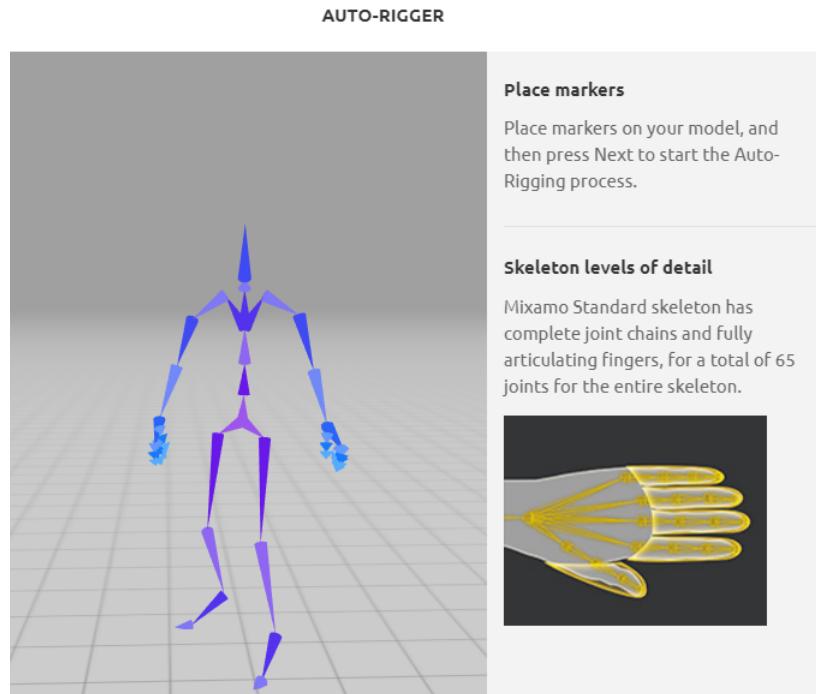


Figura 3.5: Auto-Rig con 65 huesos de Mixamo

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 3.5). 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 3.6) (sistema por el cual, una serie de cámara colocadas en las gafas de VR o AR detectan cualquier movimiento realizado con las manos).

Figura 3.6: *Hand tracking* en Oculus Quest

Bibliografía

- [1] Wikipedia. *Cinemática Inversa*. https://es.wikipedia.org/wiki/Cinematica_inversa. 2020.