
Captura de movimientos para la detección de gestos en entornos virtuales

**Motion capture for gesture detection in
virtual environments**



**Trabajo de Fin de Grado
Curso 2024–2025**

Autor

Alejandro Barrachina Argudo, Pablo Sánchez Martín

Director

Alejandro Romero Hernández, Ismael Sagredo Olivenza

Colaborador

**Grado en Ingeniería Informática y Grado en Desarrollo
de Videojuegos**

Facultad de Informática

Universidad Complutense de Madrid

Captura de movimientos para la detección de gestos en entornos virtuales

Motion capture for gesture detection in virtual environments

**Trabajo de Fin de Grado en Ingeniería Informática y Grado
en Desarrollo de Videojuegos**

Autor

Alejandro Barrachina Argudo, Pablo Sánchez Martín

Director

Alejandro Romero Hernández, Ismael Sagredo Olivenza

Colaborador

Convocatoria: Junio 2025

Calificación: 9.8

**Grado en Ingeniería Informática y Grado en Desarrollo de
Videojuegos**
Facultad de Informática
Universidad Complutense de Madrid

12 de junio de 2025

Dedicatoria

*A toda la gente que nos ha hecho llegar hasta
aquí,
a los que nos han apoyado y a los que nos han
hecho crecer.
A todos los que han estado a nuestro lado.*

Agradecimientos

A mis padres y mi pareja, por estar siempre a mi lado y apoyarme en todo momento. A mi gato Umbreon, por ser mi compañero de programación, y por haber escrito casi más líneas de código que yo sentándome en el teclado. A mis juntas anteriores de LAG, por acompañarme estos años. A la gente que conocí por LAG, por haberme apoyado en los proyectos de la asociación. A Poletti, Pascal y Blanca, por haberme dado la oportunidad de hacer charlas en la facultad, de desarrollar mis conocimientos más allá de las clases y poder ayudar a más personas a buscar más conocimiento. A la gente de cafetería, por cuidarnos tan bien a todos. Y a todos los voluntarios que vinieron a hacer las pruebas para generar datos, sin ellos este trabajo no hubiera obtenido los resultados que se han conseguido.

Alejandro Barrachina Argudo

A mi madre, mi hermana, mi pareja, mi cuñado, mis tíos y mis primos por haberme apoyado y ayudado a seguir adelante. A mi grupo del instituto por estar siempre ahí para lo que necesitase. A la gente que he conocido en la universidad: tanto LAG como mi clase, por hacer que mis días en la facultad hayan sido más llevaderos. A la gente de cafetería, por cuidarme tanto en estos años. Al despacho 409 (y allegados) por soportarme todos los días, ser mis compañeros de cafetería y ayudarme cuando lo he necesitado.

Pablo Sánchez Martín

Resumen

Captura de movimientos para la detección de gestos en entornos virtuales

Este estudio se centra en la comparativa de distintos modelos de IA para la detección de gestos específicos mediante el traje de captura de movimiento Perception Neuron 3 con el fin de crear interacciones con NPCs a través de la comunicación no verbal y la creación de herramientas derivadas de los modelos que ayuden a la comunicación no verbal en entornos virtuales.

Para ello primero se buscaron datasets públicos de gestos, no encontrándose ninguno que se adaptara a las necesidades del estudio. Por ello se creó una herramienta para traducir animaciones de Mixamo a CSVs para que pudieran ser entendidas por los modelos a entrenar. Esta herramienta generó un gran número de animaciones pero muy desbalanceadas en el número de animaciones de cada tipo.

Para solucionar el problema del desbalance, se realizó una serie de pruebas con usuarios ($N=65$) en las cuales se les pidió que realizaran 3 tomas de cada gesto mientras llevaban el traje de captura de movimiento puesto. Esta recolección de datos resultó en un total de 975 animaciones, habiendo un total de 195 gestos humanos de correr, saludar, señalar, pegar y sentarse. Se omitió el gesto de baile en las pruebas por el desbalanceo en el dataset generado por ordenador. Esta omisión sirvió para equilibrar más el número de animaciones, pero no para terminar del todo con el desbalanceo en los datos estandarizados.

Una vez se tuvo el dataset, se implementaron distintos modelos de IA para la detección de gestos bajo una interfaz web para su facilidad de uso. Estos modelos fueron: LSTM, CNN, RNN y Random Forest. Tras realizar los entrenamientos de todos los modelos, se observó que el modelo Random Forest era con diferencia el que mejores datos obtenía, con un 95 % de precisión en entrenamiento y un 86 % en test. Los resultados de los modelos basados en redes neuronales fueron bastante menores, pudiendo ser esto así por la falta de datos, de tiempo de entrenamiento y de hardware suficiente para explotar mejor los modelos.

Finalmente, se exportó el modelo Random Forest a TensorFlow Serving para posteriormente ser utilizado en una aplicación de demostración con un NPC reactivo. Esta aplicación permite al usuario ver como el NPC interpreta distintos gestos y responde de manera simple ante ellos.

Palabras clave

Captura de movimiento, Unity, Inteligencia Artificial, Comunicación no verbal, Perception Neuron, TensorFlow, YDF, Keras, Realidad Virtual

Abstract

Motion capture for gesture detection in virtual environments

This study focuses on comparing different [AI](#) models for detecting specific gestures using the Perception Neuron 3 motion capture suit, aiming to create interactions with [NPCs](#) through non-verbal communication and to develop tools derived from the models that enhance non-verbal communication in virtual environments.

To achieve this, public gesture datasets were first sought, but none were found that met the study's needs. Therefore, a tool was created to convert Mixamo animations into [CSVs](#) format so they could be understood by the models to be trained. This tool generated a large number of animations, but they were highly imbalanced in the number of animations for each type.

To address the imbalance issue, a series of user trials ($N=65$) were conducted, where participants were asked to perform three instances of each gesture while wearing the motion capture suit. This data collection resulted in a total of 975 animations, with 195 human gestures for running, greeting, pointing, punching, and sitting. The gesture of dancing was omitted from the trials due to the imbalance in the dataset generated by computer animations. This omission helped balance the number of animations but did not completely eliminate the imbalance in the standardized data.

Once the dataset was established, various [AI](#) models for gesture detection were implemented under a web interface for ease of use. These models included: [LSTM](#), [CNN](#), [RNN](#), and [Random Forest](#). After training all the models, it was observed that the [Random Forest](#) model significantly outperformed the others, achieving 95% accuracy in training and 86% in testing. The results from the neural network-based models were considerably lower, likely due to insufficient data, limited training time, and hardware constraints that hindered better model performance.

Finally, the [Random Forest](#) model was exported to TensorFlow Serving to be used in a demonstration application featuring a reactive [NPC](#). This application allows users to see how the [NPC](#) interprets different gestures and responds simply to them.

Keywords

Motion capture, Unity, Artificial Intelligence, Non-verbal communication, Perception Neuron, TensorFlow, YDF, Keras, Virtual Reality

Índice

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Plan de trabajo	2
2. Estado de la Cuestión	5
2.1. Entornos virtuales	5
2.2. Comunicación no verbal en entornos virtuales	6
2.3. Captura de movimiento	6
2.3.1. Captura de movimiento óptica	7
2.3.2. Captura de movimiento mecánica	7
2.3.3. Captura de movimiento magnética	8
2.4. Motores de videojuegos	8
2.4.1. Unity	8
2.4.2. Unreal	9
2.5. Modelos de IA	10
2.5.1. Tecnologías de IA	10
2.5.2. Librerías de IA	10
2.5.2.1. Tensorflow	10
2.5.2.2. PyTorch	10
2.5.2.3. Scikit-learn	11
2.5.2.4. YDF	11
2.5.3. Modelos de reconocimiento de gestos y poses	11
2.5.3.1. YoLoV11	11
2.5.3.2. Modelos para la detección del balance de personas andando	12
2.5.3.3. Otros modelos de detección de gestos y poses	12
3. Descripción del Trabajo	13
3.1. Busqueda de un dataset	13
3.1.1. Dataset de la Universidad Carnegie Mellon	13
3.1.2. Kaggle	14

3.2. Dataset Artificial	15
3.2.1. Herramienta para descargarse animaciones de Mixamo de forma automática	16
3.2.2. Carga mediante asset bundles	17
3.2.3. Carga mediante Asset Database	17
3.2.4. Tratamiento de CSV resultantes	20
3.3. Dataset real	22
3.3.1. Traje	22
3.3.1.1. Axis Studio y conexión del traje	22
3.3.1.2. Unity y la conexión del traje	24
3.3.1.3. Limitaciones encontradas con el traje	24
3.3.2. Herramienta de recogida de datos con usuarios reales	25
3.3.3. Recogida de datos	25
3.4. Modelos de IA	28
3.4.1. Interfaz de entrenamiento y seguimiento	28
3.4.2. Carga de datos	31
3.4.3. Modelos usados	31
3.4.3.1. LSTM	31
3.4.3.2. CNN	34
3.4.3.3. RNN	36
3.4.3.4. Random Forest	38
3.4.4. TensorFlow Serving	40
3.4.5. Hardware, complejidad de datos y otras limitaciones	40
3.5. Aplicación final	41
3.5.1. Funcionamiento de la aplicación	41
4. Conclusiones y Trabajo Futuro	45
4.1. Conclusiones	45
4.1.1. Conclusiones de la búsqueda de datasets	45
4.1.2. Conclusiones de la extracción de animaciones de bancos de animaciones	45
4.1.3. Conclusiones de la recolección de animaciones con usuarios	46
4.1.4. Conclusiones de la comparativa entre los modelos de IA	46
4.1.5. Conclusiones de la aplicación final	47
4.1.6. Limitaciones	47
4.2. Trabajo Futuro	47
Introduction	49
4.3. Motivation	49
4.4. Objectives	50
4.5. Work Plan	50
Conclusions and Future Work	53
4.6. Conclusions	53

4.6.1. Dataset Search Conclusions	53
4.6.2. Conclusions of the Animation Extraction Tool	53
4.6.3. Conclusions of the User Data Collection	54
4.6.4. Conclusions of the IA Models Comparison	54
4.6.5. Final Application Conclusions	54
4.6.6. Limitations	54
4.7. Future Work	55
Contribuciones Personales	57
Bibliografía	61
A. Cabecera de los CSVs del dataset	65
B. Carteles para la generación del dataset	69
C. Formulario de recogida de citas	73
D. Formulario de recogida de datos demográficos	75
E. Gráficos de la generación del dataset	77
F. Resultados de los distintos modelos LSTM	81
F.1. Intervalo 0.2s	81
F.2. Intervalo 0.4s	82
F.3. Intervalo 0.6s	83
F.4. Intervalo 0.8s	84
F.5. Intervalo 1.0s	85
G. Resultados de los distintos modelos RNN	87
G.1. Intervalo 0.2s	87
G.2. Intervalo 0.4s	88
G.3. Intervalo 0.6s	89
G.4. Intervalo 0.8s	90
G.5. Intervalo 1.0s	91
H. Resultados de los distintos modelos CNN	93
H.1. Intervalo 0.2s	93
H.2. Intervalo 0.4s	95
H.3. Intervalo 0.6s	96
H.4. Intervalo 0.8s	98
H.5. Intervalo 1.0s	99
I. Resultados de los distintos modelos Random Forest	101
I.1. Intervalo 0.2s	101
I.2. Intervalo 0.4s	102

I.3. Intervalo 0.6s	102
I.4. Intervalo 0.8s	103
I.5. Intervalo 1.0s	103
J. Código de la herramienta MixamoDumper	105
Glosario	109
Licencia de uso del documento	111
Licencia de uso del código fuente	113

Índice de figuras

1.1.	Diagrama de Gantt del proyecto	3
2.1.	Imagen del traje de captura de movimiento XSens	7
3.1.	Imagen del traje de captura de movimiento utilizado para la base de datos de la Universidad Carnegie Mellon (izquierda) y el esqueleto resultante (derecha). Fuente: https://mocap.cs.cmu.edu/info.php	14
3.2.	Herramienta Mixamo Downloader Arreglada	16
3.3.	Frame de un <i>timelapse</i> de la ejecución de la herramienta	18
3.4.	Diagrama de flujo de la herramienta que carga y ejecuta las animaciones de Mixamo en tiempo de ejecución, llamada Mixamo Dumper .	19
3.5.	Número de animaciones conseguidas gracias a la herramienta divididas por tipo de gesto	20
3.6.	Diagrama de flujo del proceso de limpieza de datos (funciones data_cleaner y data_cleaner_aux de la clase DataLoader)	21
3.7.	Diagrama de la clase DataLoader	21
3.8.	Número de animaciones estandarizadas divididas por tipo de gesto .	21
3.9.	Captura de la aplicación Axis Studio antes de conectar un traje . .	22
3.10.	Captura del botón para conectar el traje a Axis Studio	23
3.11.	Captura del botón para calibrar el traje	23
3.12.	Captura del panel de configuración de Axis Studio	24
3.13.	Diagrama de la conexión entre los distintos componentes de la herramienta, llamada Mocap Dumper	25
3.14.	Fotografía de un usuario en las pruebas	26
3.15.	Distribución de los datos brutos	27
3.16.	Distribución de los datos estandarizados	28
3.17.	Interfaz de entrenamiento	29
3.18.	Interfaz de TensorBoard	30
3.19.	Estructura del archivo de la interfaz	31
3.20.	Estructura de la clase LSTMTrainer	32
3.21.	Esquema del modelo LSTM	33
3.22.	Matriz de confusión del modelo LSTM	33
3.23.	Gráfico de entrenamiento del modelo LSTM	34

3.24. Estructura de la clase CNNTrainer	34
3.25. Matriz de confusión del modelo CNN	35
3.26. Esquema del modelo CNN	36
3.27. Gráfico de entrenamiento del modelo CNN	36
3.28. Estructura de la clase RNNTrainer	37
3.29. Esquema del modelo RNN	38
3.30. Matriz de confusión del modelo RNN	38
3.31. Gráfico de entrenamiento del modelo RNN	38
3.32. Estructura de la clase RFTrainer	39
3.33. Matriz de confusión del modelo RandomForest	39
3.34. Ejemplo de petición a TensorFlow Serving	40
3.35. Ejemplo de respuesta de TensorFlow Serving	40
3.36. Captura de la aplicación final desde el editor de Unity	41
3.37. Diagrama de flujo de la aplicación de demostración	43
 4.1. Gantt chart of the project	51
 B.1. Cartel colgado en la facultad de informática	69
B.2. Cartel colgado en redes sociales	70
B.3. Cartel colgado en pantallas de la facultad	71
 C.1. Formulario de recogida de citas (primera parte)	73
C.2. Formulario de recogida de citas (segunda parte)	74
 D.1. Formulario de recogida de datos demográficos (primera parte)	75
D.2. Formulario de recogida de datos demográficos (segunda parte)	76
 E.1. Distribución de género de los encuestados	77
E.2. Distribución de edad de los encuestados	78
E.3. Distribución de nacionalidad de los encuestados	78
E.4. Distribución de idiomas hablados de los encuestados	79
E.5. Distribución de mano dominante de los encuestados	79
 F.1. Esquema del modelo LSTM con 0.2s de intervalo	81
F.2. Matriz de confusión del modelo LSTM con 0.2s de intervalo	81
F.3. Gráfico de entrenamiento del modelo LSTM con 0.2s de intervalo (mejor val_accuracy = 0.5655)	82
F.4. Esquema del modelo LSTM con 0.4s de intervalo	82
F.5. Matriz de confusión del modelo LSTM con 0.4s de intervalo	82
F.6. Gráfico de entrenamiento del modelo LSTM con 0.4s de intervalo (mejor val_accuracy = 0.5462)	82
F.7. Esquema del modelo LSTM con 0.6s de intervalo	83
F.8. Matriz de confusión del modelo LSTM con 0.6s de intervalo	83
F.9. Gráfico de entrenamiento del modelo LSTM con 0.6s de intervalo (mejor val_accuracy = 0.5301)	83

F.10. Esquema del modelo LSTM con 0.8s de intervalo	84
F.11. Matriz de confusión del modelo LSTM con 0.8s de intervalo	84
F.12. Gráfico de entrenamiento del modelo LSTM con 0.8s de intervalo (mejor val_accuracy = 0.5912)	84
F.13. Esquema del modelo LSTM con 1.0s de intervalo	85
F.14. Matriz de confusión del modelo LSTM con 1.0s de intervalo	85
F.15. Gráfico de entrenamiento del modelo LSTM con 1.0s de intervalo (mejor val_accuracy = 0.5318)	85
 G.1. Esquema del modelo RNN con 0.2s de intervalo	87
G.2. Matriz de confusión del modelo RNN con 0.2s de intervalo	87
G.3. Gráfico de entrenamiento del modelo RNN con 0.2s de intervalo (me- jor val_accuracy = 0.3486)	88
G.4. Esquema del modelo RNN con 0.4s de intervalo	88
G.5. Matriz de confusión del modelo RNN con 0.4s de intervalo	88
G.6. Gráfico de entrenamiento del modelo RNN con 0.4s de intervalo (me- jor val_accuracy = 0.4535)	88
G.7. Esquema del modelo RNN con 0.6s de intervalo	89
G.8. Matriz de confusión del modelo RNN con 0.6s de intervalo	89
G.9. Gráfico de entrenamiento del modelo RNN con 0.6s de intervalo (me- jor val_accuracy = 0.41185)	89
G.10. Esquema del modelo RNN con 0.8s de intervalo	90
G.11. Matriz de confusión del modelo RNN con 0.8s de intervalo	90
G.12. Gráfico de entrenamiento del modelo RNN con 0.8s de intervalo (me- jor val_accuracy = 0.47865)	90
G.13. Esquema del modelo RNN con 1.0s de intervalo	91
G.14. Matriz de confusión del modelo RNN con 1.0s de intervalo	91
G.15. Gráfico de entrenamiento del modelo RNN con 1.0s de intervalo (me- jor val_accuracy = 0.39377)	91
 H.1. Esquema del modelo CNN con 0.2s de intervalo	93
H.2. Matriz de confusión del modelo CNN con 0.2s de intervalo	94
H.3. Gráfico de entrenamiento del modelo CNN con 0.2s de intervalo (me- jor val_accuracy = 0.28777)	94
H.4. Esquema del modelo CNN con 0.4s de intervalo	95
H.5. Matriz de confusión del modelo CNN con 0.4s de intervalo	95
H.6. Gráfico de entrenamiento del modelo CNN con 0.4s de intervalo (me- jor val_accuracy = 0.49473)	96
H.7. Esquema del modelo CNN con 0.6s de intervalo	96
H.8. Matriz de confusión del modelo CNN con 0.6s de intervalo	97
H.9. Gráfico de entrenamiento del modelo CNN con 0.6s de intervalo (me- jor val_accuracy = 0.44952)	97
H.10. Esquema del modelo CNN con 0.8s de intervalo	98
H.11. Matriz de confusión del modelo CNN con 0.8s de intervalo	98

H.12.Gráfico de entrenamiento del modelo CNN con 0.8s de intervalo (mejor val_accuracy = 0.40583)	99
H.13.Esquema del modelo CNN con 1.0s de intervalo	99
H.14.Matriz de confusión del modelo CNN con 1.0s de intervalo	100
H.15.Gráfico de entrenamiento del modelo CNN con 1.0s de intervalo (mejor val_accuracy = 0.56002)	100
I.1. Matriz de confusión del modelo Random Forest con 0.2s de intervalo (mejor val_accuracy = 0.85735)	101
I.2. Matriz de confusión del modelo Random Forest con 0.4s de intervalo (mejor val_accuracy = 0.85735)	102
I.3. Matriz de confusión del modelo Random Forest con 0.6s de intervalo (mejor val_accuracy = 0.85233)	102
I.4. Matriz de confusión del modelo Random Forest con 0.8s de intervalo (mejor val_accuracy = 0.85936)	103
I.5. Matriz de confusión del modelo Random Forest con 1.0s de intervalo (mejor val_accuracy = 0.84681)	103

Índice de tablas

2.1. Puntos de referencia del modelo YOLOv11-pose	12
3.1. Cabecera del CSV de cada animación, en órden descendente y de izquierda a derecha (incompleta).	17
3.2. Reacción del NPC al gesto predicho del jugador	42
A.1. Cabecera del CSV de cada animación, en órden descendente y de izquierda a derecha (completa)	65

Capítulo 1

Introducción

En los últimos años el avance de tecnologías inmersivas como la [VR](#) ha proporcionado un gran cambio en la forma en la que los usuarios interactúan en los entornos digitales. Grandes empresas como Meta o Apple han apostado por el desarrollo de hardware para estas tecnologías con el lanzamiento de Meta Quest 3 y Apple Vision Pro, así como con la creación de plataformas de interacción social en el [metaverso](#) con el desarrollo de Meta Horizon Worlds. Esta tecnología ha abierto nuevas oportunidades para explorar modos de comunicación más naturales en espacios virtuales mediante la comunicación no verbal.

La comunicación no verbal es aquella que no se basa en el uso de palabras, sino que utiliza las imágenes y los gestos para transmitir un mensaje. En nuestro día a día representa una parte esencial de las comunicaciones humanas, sin embargo su representación en entornos virtuales es más limitada y dependiente de acciones predefinidas como animaciones o “emotes” ya introducidos en el juego. Es por esto que el desarrollo de herramientas que permitan capturar e interpretar en tiempo real estos gestos constituye un gran paso hacia entornos virtuales más expresivos y realistas.

1.1. Motivación

Como se ha mencionado en la introducción, el uso de la comunicación no verbal en entornos virtuales está limitado y prefijado por el diseño de éstos, limitando una parte fundamental de la forma de expresarse cotidiana de los humanos. Debido a esto y como podemos ver en artículos como [Neverova et al. \(2013\)](#) o [Herbert et al. \(2024\)](#) la detección de gestos es un campo de estudio que actualmente se está explorando para intentar resolver este problema. Por estas razones es interesante investigar formas de ampliar esta representación de la comunicación no verbal para que la comunicación entre usuarios y [NPCs](#) sea más natural.

Es por este motivo que este trabajo se inscribe dentro de la línea de investigación de herramientas que puedan reconocer gestos realizados por un usuario mediante un traje de captura de movimiento con el fin de crear entornos virtuales más inmersivos y naturales para los usuarios.

Por otra parte, es necesaria una gran cantidad de datos para crear modelos de IA que permitan crear las herramientas mencionadas. En el caso de este proyecto, los datos necesarios son una secuencia de coordenadas 3D de posiciones y rotaciones de los huesos del esqueleto a lo largo de las animaciones, por lo que los datos que se necesitan procesar tienen un tamaño relativamente grande. Es por esto que es necesario crear modelos complejos que procesen un número significativo de datos de entrada, pero como desarrollaremos a lo largo del trabajo, no ha sido posible encontrar datasets de este tipo de datos. Como consecuencia de este motivo, una gran aportación de este trabajo en el campo ha sido la creación de un dataset gracias a la extracción de animaciones de un banco especializado en estas y a la captura de movimientos de usuarios.

Vistos estos motivos, se puede concretar que, la carencia de métodos naturales de explotar la comunicación no verbal en entornos virtuales y la falta de datasets de animaciones han motivado los objetivos mencionados y que expandiremos a continuación.

1.2. Objetivos

El objetivo general de este proyecto es la implementación de un modelo de IA que, con baja latencia, permita identificar el gesto que se esté realizando con un traje de captura de movimiento en tiempo real, con la intención de mejorar la comunicación no verbal en entornos virtuales. Para cumplir el objetivo, se han propuesto las siguientes metas durante el desarrollo del trabajo:

1. Búsqueda de datasets de animaciones que se han considerado relevantes a la hora de comunicarse con un NPC en un videojuego. Estas animaciones han sido bailar, saludar, señalar, sentarse, pelear y correr.
2. Extracción automática de las animaciones mencionadas anteriormente de bancos de animaciones y posterior procesamiento.
3. Extracción de las animaciones mediante la captura de movimiento de un grupo heterogéneo de usuarios.
4. Implementación, entrenamiento y comparación de distintos modelos de IA, tanto redes neuronales (LSTM, CNN y RNN) como modelos de clasificación clásicos (Random Forest)
5. Implementación de una aplicación final a modo de demostración para enseñar los resultados en VR para aumentar el grado de inmersión.

Como gran objetivo secundario y debido a la falta de datasets de animaciones se ha propuesto publicar los datasets resultantes, tanto de las animaciones artificiales como de las capturas de usuarios.

1.3. Plan de trabajo

El plan de trabajo consiste en varios pasos:

1. Búsqueda de un dataset: generar un dataset lo suficientemente grande con varios ejemplos de gestos como para poder entrenar de forma adecuada diferentes modelos. Este dataset debe estar compuesto en su mayoría por datos de usuarios reales para poder identificar gestos lo más naturales posibles.
2. Implementación de modelos de IA: implementación de varios modelos de IA para poder hacer una comparativa entre ellos y decidir cuál es el más adecuado teniendo en cuenta su velocidad de predicción y su precisión. La implementación de dichos modelos debe contener: entrenador, forma de sacar las estadísticas del modelo final, búsqueda de hiperparámetros, generador de matriz de confusión y debe ser acoplable a una interfaz de usuario común a todos los entrenamientos.
3. Desarrollo de una interfaz de usuario para el entrenamiento y monitorización de los distintos modelos: desarrollo de una interfaz web para que los usuarios con menos experiencia en programación e IA puedan entrenar modelos capaces de predecir los gestos que necesiten sin necesidad de modificar el código ni tener que saber como funciona internamente la base de código.
4. Desarrollo de una aplicación final: desarrollo de una aplicación para las Oculus Quest a forma de demo en la que se conecte al modelo elegido y se pueda ver en tiempo real su uso.

El desarrollo de las tareas y su planificación se puede ver en la figura 1.1.

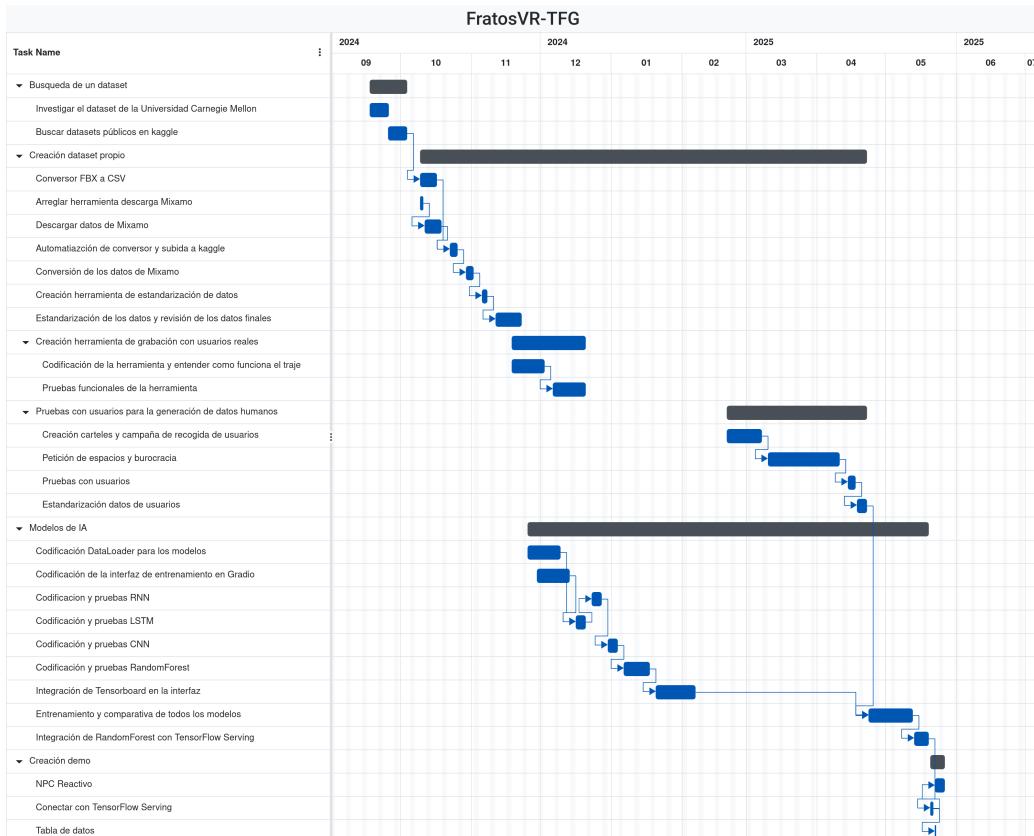


Figura 1.1: Diagrama de Gantt del proyecto

Capítulo 2

Estado de la Cuestión

En este capítulo se presenta el estado del arte de los distintos elementos que componen el trabajo. Estos campos son la comunicación no verbal en entornos virtuales, la captura de movimiento, distintos modelos de IA, motores de videojuegos y el estado actual de los entornos virtuales.

2.1. Entornos virtuales

Como se puede ver en el artículo [Ellis \(1991\)](#), se empezó a hablar en la década de los 90 de entornos virtuales y el [metaverso](#) como nuevas formas de comunicarse con las personas. Estos entornos virtuales al principio eran muy sencillos, con pocas actividades que realizar y el chat escrito como única forma de comunicarse. Algún ejemplo de estos entornos virtuales eran el Second Life o el Habbo Hotel.

Para continuar hablando de los entornos virtuales es necesario introducir los conceptos de “inmersión” y “presencia”: dos términos que, a menudo se usan como sinónimos, pero son ligeramente distintos aunque están muy relacionados. Como lo define el artículo [Wilkinson et al. \(2021\)](#), la presencia es la calidad experimental en entornos virtuales; mientras que la inmersión está asociado con los aspectos técnicos de un sistema virtual que ayudan al usuario a potenciar la sensación de presencia. Estudios recientes como [Van Brakel et al. \(2023\)](#) demuestran como la presencia, tanto la nuestra propia como la social, son potenciadores del apoyo social percibido, lo que se asocia con el bienestar de los usuarios; mientras que el estudio [Pallavicini et al. \(2019\)](#) muestra que, aunque no haya mejoras performáticas entre jugar en entornos inmersivos ([VR](#)) y no inmersivos (un ordenador), los jugadores encontraron más emocionantes y con mayor presencia los entornos interactivos. Es por esto último que con la aparición de la [VR](#) se hicieron populares entornos virtuales focalizados en esta tecnología.

Con todo esto es natural que las empresas inviertan en tecnologías que aumenten la sensación de inmersión para lograr una mayor presencia. Empresas como Apple o Meta están apostando en la tecnología [VR](#), sacando recientemente las gafas Apple Vision Pro (2024) y Meta Quest 3 (2023); mejorando aspectos como la calidad de las cámaras, el sonido, el reconocimiento de gestos y su comodidad que pueden

amplificar la sensación de inmersión cuando se utilizan. Además, Meta está centrada en la idea de [metaverso](#) como un espacio digital en el que se puede socializar.

Con la idea de mejorar la inmersión en entornos virtuales se ideó este proyecto en el que, mediante la captura de movimiento, se puede usar la comunicación no verbal para expresar ciertas acciones y el entorno sea capaz de reconocer lo que estamos haciendo. Todo esto pensado para que se pueda usar en entornos de [VR](#) para llevar la inmersión al máximo

2.2. Comunicación no verbal en entornos virtuales

En los últimos años se han llevado a cabo distintos estudios sobre la comunicación no verbal en entornos virtuales. El estudio [Xenakis et al. \(2023\)](#) se centra en el concepto de presencia en entornos virtuales, enfatizando los roles de inmersión, contenido emocional y fidelidad de avatares en la mejora de la experiencia del usuario. El estudio resalta como el realismo de los avatares y sus comportamientos (movimiento y lenguaje corporal) ayudan a conseguir un mayor sentimiento de presencia social entre los usuarios. También se discute como estos elementos influyen en las dinámicas interpersonales y la percepción del usuario en entornos virtuales.

Estos roles discutidos en el estudio pueden extrapolarse a la experiencia de jugar a videojuegos. En el contexto de los videojuegos tradicionales, las opciones de comunicación no verbal son las dadas por los desarrolladores en forma de gestos, acciones dentro del juego (agacharse, girar su avatar o mover la cabeza al mover la cámara por ejemplo). Dentro de cada videojuego la comunidad va a intentar usar las opciones que se les proporcione para comunicarse entre ellos mismos de la mejor manera posible.

Un ejemplo claro de evolución de la comunicación no verbal en videojuegos y su importancia para los jugadores es el caso de *VR chat*. *VR chat* es un videojuego [MMORPG](#) de [VR](#) donde los jugadores pueden unirse a distintos mundos e interactuar con otros usuarios. Este juego no solo permite el uso de gafas de [VR](#), sino que también permite (de manera experimental) usar distintas formas de captura de movimiento de cuerpo completo ¹.

Los usuarios de esta aplicación han buscado distintas formas de conseguir una mayor expresividad en sus avatares para así poder entablar mejor conversaciones entre ellos o para poder expresarse como les gustaría. Algunos de estos métodos se describen en el siguiente apartado junto a métodos de captura de movimiento no orientados a videojuegos.

2.3. Captura de movimiento

La captura de movimiento es una técnica que permite digitalizar el movimiento de una persona. Esta técnica se usa sobre todo en el ámbito de la animación, el cine y

¹Enlace a la documentación de *VR Chat* para su [SDK](#) de captura de movimiento de cuerpo completo [VRChat \(2024\)](#)

los videojuegos. Para lograr esto se han desarrollado diferentes tipos de tecnologías: óptica, mecánica y magnética.

2.3.1. Captura de movimiento óptica

La captura de movimiento óptica consiste en el uso de cámaras para la captura. Según el artículo [Guerra-Filho \(2005\)](#) esta captura se puede dividir en monocular o multivista dependiendo del número de cámaras, y según si utilizan marcadores o no. En el mismo artículo explica el funcionamiento de la captura óptica multivista con el uso de marcadores, el cual se utiliza en gran medida en la industria audiovisual. Este tipo de captura óptica necesita al menos de un conjunto de cámaras sincronizadas (dos o más), un traje con marcadores en puntos significativos (articulaciones sobre todo) y un sistema de adquisición de varios vídeos simultáneos. La reconstrucción del movimiento se basa en la sincronía de las cámaras y la capacidad de detección de los mismos marcadores desde diferentes puntos de vista, haciendo que se pueda reconstruir en 3D los marcadores mediante la triangulación de los distintos puntos de vista.

Un ejemplo de la captura de movimiento óptica monocular y sin marcadores son los sistemas basados en cámaras 3D que utilizan los [IRs](#) para calcular la profundidad. Según el artículo [Zhang \(2012\)](#) la Kinect (Microsoft, 2010) utilizaba esta tecnología para ello. Consiste en un proyector de [IRs](#) cuyos rayos atraviesan una rejilla de difracción, convirtiéndose en un conjunto de puntos que, como se sabe la geometría relativa entre este proyector y una cámara de [IRs](#) que venía incorporada, se podía triangular la posición de un objeto haciendo coincidir un punto observado con uno de los puntos del proyector.

2.3.2. Captura de movimiento mecánica

La captura de movimiento mecánica se consigue mediante sensores inerciales. Según se puede ver en [Roetenberg et al. \(2009\)](#), el traje Xsens MVN es un ejemplo de ello. No utiliza cámaras o marcadores si no la combinación de giroscopios, acelerómetros y magnetómetros. Con los datos recibidos de estos sensores se puede estimar la rotación de los diferentes puntos, mientras que la posición se puede estimar mediante supuestos de una cadena cinemática. En la figura 2.1 se puede ver el traje visto desde la espalda.

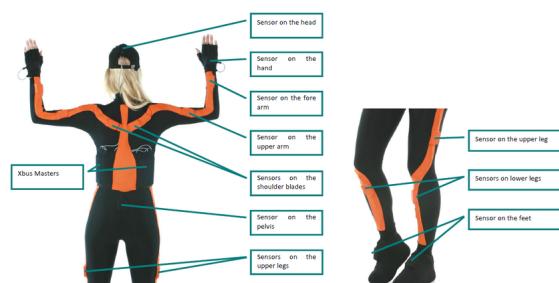


Figura 2.1: Imagen del traje de captura de movimiento Xsens ^{[2](#)}

2.3.3. Captura de movimiento magnética

La captura magnética se consiste mediante sensores electromagnéticos. El artículo [Raab et al. \(1979\)](#) explica que la generación de valores en un campo electromagnético tridimensional es suficiente para determinar la posición y rotación de un objeto emisor frente a un sensor. Para ello se aplican transformaciones lineales a la excitación de la fuente y los vectores recibidos por el receptor para percibir pequeños cambios en los valores a determinar, los cuales se separan mediante combinaciones lineales de los vectores de salida del sensor y se utilizan para actualizar las coordenadas previas.

En esta última tecnología se basa el traje Perception Neuron 3. Este traje fue creado por Noitom, empresa fundada en el 2012 enfocada en el desarrollo de tecnología de captación de movimiento. El traje previamente mencionado se encuentra dentro de la serie de trajes “Perception Neuron”, los cuales son bandas de velcro a las que se pueden acoplar sensores electromagnéticos.

2.4. Motores de videojuegos

Un motor de videojuegos es un software que proporciona herramientas para desarrollar y ejecutar un videojuego. Estos motores están compuestos a su vez de otros motores más específicos que se encargan de funcionalidades básicas para un videojuego para que el desarrollador del juego no tenga que encargarse de ello. Algunos ejemplos de esto último son el motor de render, el de físicas o el de sonido.

Actualmente los motores de videojuegos más usados en el mercado son Unity, Unreal y Godot. Aunque el artículo [Holfeld \(2024\)](#) no se centre en un análisis del mercado en general, si contiene varias estadísticas sobre el uso de los principales motores de videojuegos en el mercado en 2023. Este artículo menciona que, de los juegos publicados en Steam ese año, un 60.04 % fueron hechos en Unity, un 16.42 % fueron hechos en Unreal y el 23.54 % restante fue hecho en motores propios de empresas (Source o CryEngine, por ejemplo).

2.4.1. Unity

Unity es un motor de videojuegos creado por Unity Technologies en 2005. A continuación se listan las principales herramientas que usa el motor:

1. Motor gráfico: Propio de Unity basado mayoritariamente en OpenGL (para depender de qué plataformas puede usar otro, como Direct3D para Windows).
2. Motor físico: PhysX, creado por la empresa Nvidia Corporation.
3. Motor de audio: Propio de Unity.
4. Lenguaje de scripting: C#.

²Fuente de la imagen: https://www.researchgate.net/figure/The-Xsens-MVN-full-body-motion-capture-suit-Picture-source-https-wwwxsenscom_fig1_233926874

5. Lenguaje de scripting de shaders: ShaderLab.

El modelo clásico de Unity se ha basado en la arquitectura Entidad-Componente (EC). Esta arquitectura deja a todo lo que hay en el juego (las entidades, llamadas en Unity “GameObjects”) como contenedores de componentes, que son los que le aportan funcionalidad a éstas. Este modelo supuso una mejoría ante la programación de videojuegos con objetos, ya que conseguía independizar comportamientos.

Desde 2022 mediante el paquete Unity DOTS (Data-Oriented Technology Stack)³ se pueden crear videojuegos siguiendo la arquitectura Entidad-Componente-Sistema (ECS). Esta arquitectura consiste en separar la lógica de los datos, ya que los datos se guardan en componentes mientras que la lógica la ejecutan los sistemas. ECS supone una mejoría a EC gracias a la separación mencionada, ya que mejora la escabilidad del código y la modularidad.

2.4.2. Unreal

Unreal es un motor de videojuegos creado por Epic Games en 1998 con el lanzamiento de Unreal Engine 1. La versión estable actual es Unreal Engine 5.5, lanzada en noviembre de 2024. El código fuente de este motor está escrito en C++ y es accesible desde Github⁴. Las principales herramientas del motor son:

1. Motor gráfico: Propio de Unreal. Este cuenta con tecnologías para optimizar geometría (Nanite), sistema de iluminación dinámica (Lumen), y una técnica de escalado de imágenes (TSR)
2. Motor físico: Chaos Physics, desarrollado por Epic Games para mejorar la integración con las diferentes herramientas del motor.
3. Motor de audio: Propio de Epic Games llamado MetaSounds.
4. Lenguaje de scripting: C++ y Blueprints como scripting visual.
5. Lenguaje de scripting de shaders: High-Level Shading Language (HLSL) y Material Editor como forma de abstracción, permitiendo crear materiales con nodos visuales.

Unreal utiliza una arquitectura orientada a clases jerárquicas más cercana a la programación orientada a objetos con alguna integración con componentes. Los objetos del mundo, llamados “actores”, heredan de la clase AActor y son los componentes quienes le dan alguna de las funcionalidades (mallas, físicas o cámara por ejemplo). Las clases controlables (tanto por el jugador como por IA) son entidades de tipo “Pawn” (clase APawn), de la cual hereda los objetos de tipo “Character” (clase ACharacter), los cuales tienen un movimiento humanoide incorporado. Para introducir movimiento a los objetos Pawn es necesario hacerlo mediante objetos de tipo controller (APlayerController para entrada de input y AAIController para IA).

³Enlace a la documentación del paquete DOTS: <https://docs.unity3d.com/Packages/com.unity.entities@1.3/manual/index.html>

⁴Enlace a la página de Github de Epic: <https://github.com/epicgames>

2.5. Modelos de IA

La inteligencia artificial ha sido un tópico de desarrollo e investigación en los últimos años. Aunque el desarrollo en los últimos años se ha centrado sobre todo en el ámbito de las **IAS** generativas, este estudio se va a centrar sobre todo en el uso de redes neuronales e **IAS** de clasificación más tradicionales.

2.5.1. Tecnologías de IA

En la actualidad, el mundo de la **IA** está centrado en el desarrollo en python mediante el uso de librerías programadas en C++. Las librerías más comunes son Tensorflow, Pytorch y scikit-learn.

Tensorflow tiene un enfoque más orientado a la producción y despliegue de modelos de **IA** a un nivel empresarial, pytorch es más usado en el ámbito académico y de investigación por su capacidad de rápido prototipado. Scikit-learn se usa más para enseñar los principios de la **IA** y desarrollar modelos más ligeros y más tradicionales.

Estas librerías se usan en conjunto con Numpy([Harris et al. \(2020\)](#)) y Pandas([pandas development team \(2020\)](#), [Wes McKinney \(2010\)](#)) para conseguir unos cálculos más efectivos sobre la gran cantidad de datos que se necesitan en el ciclo de vida de un modelo de **IA**.

2.5.2. Librerías de IA

En el apartado anterior ya se han nombrado distintas librerías de **IA** que se usan en la actualidad. En este apartado se van a describir las más relevantes para el desarrollo de este trabajo.

2.5.2.1. Tensorflow

Tensorflow([Abadi et al. \(2015\)](#)) es una librería de **IA** desarrollada y mantenida por el equipo de Google Brain desde 2015. Esta librería permite el desarrollo de modelos de **IA** usando **CPUs**, **GPUs** y **TPUs**. Tensorflow es una plataforma de código abierto que tiene un ecosistema de herramientas y librerías que permite el desarrollo e investigación de modelos de **IA**. Tensorflow tiene un enfoque más orientado a la producción y despliegue de modelos de **IA** a un nivel empresarial, por lo que es más común ver su uso en empresas que en el ámbito académico. Un ejemplo de este enfoque es la presencia de la herramienta Tensorflow Serving [Olston et al. \(2017\)](#), que permite el desplegado de modelos de Tensorflow (o compatibles con el ecosistema) de una forma sencilla.

2.5.2.2. PyTorch

PyTorch([Ansel et al. \(2024\)](#)) fue desarrollado por Facebook AI Research en 2026, se hizo de código abierto en 2017 y en 2022 se unió a la Linux Foundation. Esta librería se centra en la idea de que la búsqueda de usabilidad y velocidad son compatibles. Pytorch permite el desarrollo de modelos de **IA** al igual que Tensorflow, pero su

modelo de código y desarrollo ha hecho que se use más para prototipado rápido y desarrollo de proyectos dinámicos. Pytorch permite la integración con otras librerías como NumPy, SciPy y Cython, lo que permite un desarrollo más rápido y sencillo de modelos de IA.

2.5.2.3. Scikit-learn

Scikit-learn([Pedregosa et al. \(2011\)](#)) es una librería de IA desarrollada en Python que se centra en el aprendizaje automático. Esta librería se basa en NumPy, SciPy y matplotlib, lo que permite un desarrollo más sencillo de modelos de IA. Scikit-learn es una librería más ligera y más tradicional que Tensorflow o Pytorch, por lo que es más común ver su uso en el ámbito académico y de investigación. Scikit-learn se centra más en el análisis de datos predictivo y también es de código abierto como las anteriores.

2.5.2.4. YDF

YDF ([Guillame-Bert et al. \(2023\)](#)) es una librería para entrenar, evaluar, interpretar y desplegar Random Forest, gradient boosted decision trees, CART y isolation forest. Esta librería esta integrada con Tensorflow y keras, centra su desarrollo en la optimización de árboles de decisión y en hacer una API concisa y moderna.

2.5.3. Modelos de reconocimiento de gestos y poses

Muchos modelos de IA se han desarrollado para la detección de gestos y poses. Estos modelos pueden usar imágenes, videos, datos de sensores o esqueletos formados por puntos de referencia. En este apartado se van a destacar los más relevantes para el resto del trabajo.

2.5.3.1. YoLoV11

YoLoV11([Jocher et al. \(2023\)](#)) es un conjunto de modelos YOLO, entre los que se encuentra un modelo de detección de poses. Por defecto este modelo usa 17 puntos de referencia (presentados en la tabla 2.1) y usa imágenes de 640x640 píxeles como entrada. En la página del modelo de pose⁵, Ultralytics incluye los distintos modelos de pose y su rendimiento y requerimientos. Este modelo se ofrece preentrenado en distintos tamaños con la opción de reentrenarse usando un dataset con el formato especificado por Ultralytics⁶

⁵Página de YOLOv11-pose: <https://docs.ultralytics.com/es/tasks/pose/>

⁶Página de especificación del formato del dataset: <https://docs.ultralytics.com/es/datasets/pose/>

Tabla 2.1: Puntos de referencia del modelo YOLOv11-pose

Nariz
Ojo izquierdo
Ojo derecho
Oreja izquierda
Oreja derecha
Hombro izquierdo
Hombro derecho
Codo izquierdo
Codo derecho
Muñeca izquierda
Muñeca derecha
Cadera izquierda
Cadera derecha
Rodilla izquierda
Rodilla derecha
Tobillo izquierdo
Tobillo derecho

2.5.3.2. Modelos para la detección del balance de personas andando

En el artículo [Ma et al. \(2023\)](#) se presenta una comparativa de modelos para la detección del balance de personas andando en distintos niveles de valance y con distintas restricciones con un traje usado. Este artículo presenta una generación de esqueletos en 3D a partir de una kinect y realiza una comparativa de distintos modelos tradicionales y neuronales para ver su efectividad. En las conclusiones podemos ver como, de los distintos modelos utilizados, el que mejor resultados da es un [DCNN](#) que los investigadores proponen. Otros modelos de la comparativa incluyen el [Random Forest](#), el [SVM](#), [LSTM](#) y Naive Bayes en términos de modelos tradicionales. Otra comparativa se realiza con otros modelos basados en [CNNs](#) como AlexNet, VGGNet y ResNet-18.

2.5.3.3. Otros modelos de detección de gestos y poses

En los últimos años se han desarrollado modelos generales de detección de gestos y poses. Estos modelos han tenido distintos enfoques, desde el uso de gafas de [VR](#) para hacer detección de gestos simples con las manos para utilizarlos como método de entrada (Meta Quest 2 y 3 por ejemplo), a modelos de seguimiento de manos y distintos modelos de reconocimiento de Google con Gemini.

Capítulo 3

Descripción del Trabajo

*“In the distant future.
In a land abandoned by man.
Machines wage a continuous war.
Unaware of the futility of their actions.”*
— Nier Automata

3.1. Busqueda de un dataset

Para poder entrenar modelos de IA se requiere una gran cantidad de datos, en este caso de animaciones de los gestos escogidos que se han mencionado en la introducción. Además, estos datos tienen que estar en un formato en el que se pueda extraer la información de los huesos (posición y rotación) para que pueda ser usado por los modelos, y tenían que ser compatibles con los huesos del traje de captura de movimiento.

3.1.1. Dataset de la Universidad Carnegie Mellon

La Universidad privada Carnegie Mellon, ubicada en Pittsburgh, Pensilvania, tiene disponible de forma gratuita un dataset de movimientos recogidos mediante la captura de movimiento.¹ Sin embargo este dataset no ha sido posible utilizarlo por tres motivos.

El primer motivo es la incompatibilidad del esqueleto utilizado por ellos con el utilizado por el traje de captura de movimiento Perception Neuron 3. Como se puede ver en Carnegie Mellon (2003) su traje contiene 41 puntos usados para la captura, mostrados en la figura 3.1, dando a lugar al esqueleto mostrado en la misma figura.

¹Enlace a la página del dataset de la Universidad Carnegie Mellon: <https://mocap.cs.cmu.edu/>

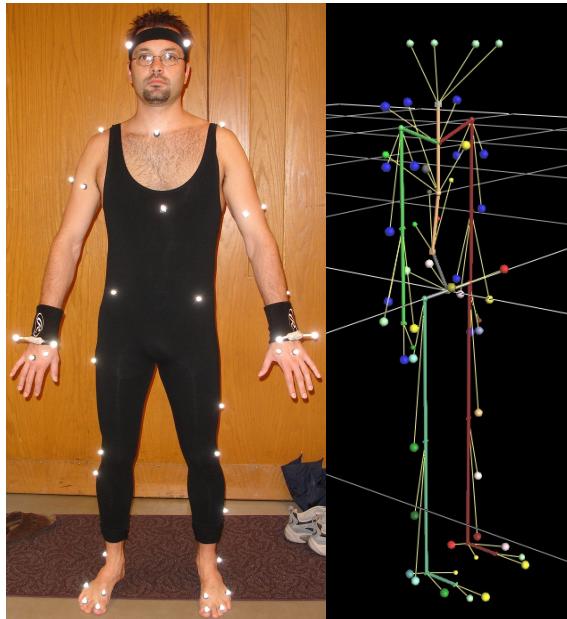


Figura 3.1: Imagen del traje de captura de movimiento utilizado para la base de datos de la Universidad Carnegie Mellon (izquierda) y el esqueleto resultante (derecha). Fuente: <https://mocap.cs.cmu.edu/info.php>

Por otro lado el traje utilizado en este estudio, el Perception Neuron 3, contiene 17 puntos, por lo que debido a las grandes diferencias de los esqueletos decidimos no utilizarlos.

El segundo motivo por el que no se ha usado el dataset de la Universidad Carnegie Mellon es el formato de las animaciones disponibles. Las animaciones presentes en el dataset vienen en tres formatos distintos: c3d, asf (o amc) y vsk (o v). Los formatos c3d, vsk y v son unos formatos en binario usados para objetos en 3D, por lo que no se puede usar de forma sencilla para extraer la información del propio archivo. Por otro lado los archivos asf y amc son archivos en formato ASCII con la información de los huesos. Sin embargo la documentación aportada para poder usar la información es incompleta.

El último motivo por el que no se ha usado este dataset es por la escasez de animaciones de los gestos buscados. Animaciones como “correr” sí están presentes (aunque no en abundancia), pero otras como “pelear” o “saludar” no.

Este último problema también ha estado presente en páginas específicas de bancos de datasets como Kaggle.

3.1.2. Kaggle

Kaggle es una página web dedicada a la ciencia de datos y el aprendizaje automático y la IA. En Kaggle se pueden encontrar datasets, modelos, código y competiciones de IA. Kaggle tiene una gran comunidad de usuarios y una gran selección de datasets para distintas finalidades.

En un primer momento, se pensó en usar datasets de imágenes/vídeo, así que se buscó conjuntos de datos de este formato. La búsqueda no obtuvo los resultados

esperados, ya que no se encontró datos de poses generales como las requeridas, y en concreto no se consiguió mucha cantidad de datos.

Tras analizar las desventajas de usar imágenes y videos, se pensó que era mejor usar un conjunto de datos basado en animaciones (o coordenadas de las mismas). Algunas de estas desventajas consideradas fueron:

- Complejidad de renderizado: las imágenes y/o videos requerirían de renderizar una cámara externa dentro del motor para capturar la presencia del usuario. Esto ya requeriría más potencia de computo, sin contar lo que requeriría el modelo.
- Transferencias de datos más grandes: al plantear un servidor para el modelo, se pensó en el tamaño de las imágenes que habría que mandar para hacer las inferencias. En el caso de tener una red de velocidad limitada sería mejor mandar los puntos concretos del traje que mandar imágenes completas. Las imágenes, incluso al ser de baja calidad, requerirían una cantidad muy grande de píxeles para hacer el modelo del usuario fácilmente reconocible.
- Perdida de contexto: el traje de captura de movimiento te permite obtener datos tridimensionales de cada punto por defecto, sin necesidad de tener que recrear un esqueleto 3D con capas extra en el modelo.

Tras este cambio de enfoque, se buscó datasets de animaciones y modelos 3D. Resultó ser una búsqueda incluso peor que la anterior, ya que por la limitación de gente que tiene acceso a trajes de captura de movimiento, no había datasets públicos en dataset con este tipo de datos.

Ya que no se encontró un gran dataset que cumpliese con nuestros requerimientos se tomó la decisión de buscar en un banco de animaciones los gestos requeridos y transformar esas animaciones en un formato que pudiesen ser procesados.

3.2. Dataset Artificial

El banco de animaciones en el que fueron buscados los gestos necesarios fue Mixamo.²

Mixamo es una página web creada por Adobe en la que se encuentran de manera gratuita modelos con un *rig* humanoide y animaciones para estos rigs, además de una funcionalidad que permite crear el *rig* de un personaje que se suba a la página. En la página hay un buscador en el que puedes buscar tipos de animaciones, pero como no es viable descargarse todas las animaciones de todos los gestos buscados se usó una herramienta para automatizarlo.

²Página web de Mixamo: <https://www.mixamo.com>

3.2.1. Herramienta para descargarse animaciones de Mixamo de forma automática

Esta herramienta se encontró en GitHub hecha por el usuario *juanjo4martinez*³. La herramienta (*mixamo-downloader*⁴) permite buscar y descargar animaciones de Mixamo de manera automática y con el uso de distintos filtros para acomodarse a lo requerido por el usuario.

Al empezar a usarse la herramienta se detectó que al empezar a bajar animaciones, dependiendo del nombre que tuvieran las mismas, el programa era incapaz de guardar dichas animaciones. Esto se debía a la incompatibilidad de distintos sistemas de archivos para usar determinados caracteres en sus nombres de archivos (interrogaciones, barras o dobles barras, dos puntos, etc). Se hizo un *fork* de la herramienta⁵ que efectúa estos arreglos y añade la opción de reescribir animaciones anteriores que tuvieran el mismo nombre. La interfaz completa de la herramienta arreglada se puede ver en la figura 3.2.



Figura 3.2: Herramienta Mixamo Downloader Arreglada

Una vez realizados los cambios, se descargaron las animaciones disponibles en Mixamo para los gestos que se querían usar. El resultado de estas descargas se puede ver en el dataset [Barrachina Argudo y Sánchez Martín \(2025b\)](#).

Las animaciones descargadas tienen un formato **FBX**, por lo que era necesario pasárselas a un formato que puedan usar los modelos de **IA**. Para ello se decidió pasar la información relevante de los huesos en la toda la animación (su posición y rotación en cada frame) a un archivo **CSV**, donde cada columna iba a ser esta información y cada fila un frame de la animación. En la tabla A.1 se puede ver la cabecera para los **CSV**. En la siguiente tabla (3.1) se puede ver un ejemplo de los valores que se guardan de un punto concreto del traje.

³Enlace al perfil del creador de la herramienta <https://github.com/juanjo4martinez>

⁴Enlace al repositorio de la herramienta <https://github.com/juanjo4martinez/mixamo-downloader>

⁵Enlace al repositorio con los arreglos <https://github.com/ALK222/mixamo-downloader>

Tabla 3.1: Cabecera del [CSV](#) de cada animación, en orden descendente y de izquierda a derecha (incompleta).

Robot_Hips_posx	Robot_Hips_posy	Robot_Hips_posz
Robot_Hips_rotx	Robot_Hips_roty	Robot_Hips_rotz
...

Como no es viable ni escalable meter todas las animaciones a un proyecto de Unity ni crear un [Animator](#) con todas estas se ideó una herramienta en Unity que se metiese todas las animaciones y construía el Animator en tiempo de ejecución.

3.2.2. Carga mediante asset bundles

En un principio se pensó en usar los assets bundles para cargar las animaciones. Los assets bundles son archivos de Unity que contienen archivos serializados (cualquier asset para un videojuego menos código) y pueden cargarse en tiempo de ejecución. Finalmente se descartó la idea de usar este tipo de archivos, ya que necesitan construirse en un proyecto de Unity, por lo que no solucionaba el problema de meter las animaciones a mano a un proyecto de Unity.

3.2.3. Carga mediante Asset Database

Finalmente se decidió usar Asset Database para la carga automática de las animaciones. Asset Database es una API de Unity que permiten trabajar con assets, siempre y cuando estén incluidos en el proyecto aunque no necesariamente cargados. Para suplir la condición de que estuviesen incluidos en el proyecto se hizo un script que descargaba las animaciones, separadas por carpetas cuyo nombre era el tipo de gesto, desde la base de datos de Kaggle hacia la carpeta “Resources” del proyecto y ejecutaba desde línea de comandos el editor del proyecto.

Una vez las animaciones estuviesen descargadas en la carpeta de “Resources” del proyecto se busca iterativamente por la carpeta para cargar las animaciones y guardarse la relación entre el tipo de gesto y la animación. Una vez completado empieza la ejecución de las animaciones.

Esta ejecución consiste en cargar la animación en la máquina de estados del [Animator](#) gracias a un componente llamado “Animator Override Controller”, que permite cambiar el clip de animación de una instancia del [Animator](#) sin cambiar la lógica de su máquina de estados. Cuando empieza la animación se crea un [CSV](#) con el formato “Animación_Número de animación.csv” y empieza su ejecución. Una vez finalizada la ejecución de una animación se cierra el [CSV](#) y se busca la siguiente animación. Se puede ver un ejemplo de todo este proceso en un timelapse de la ejecución de la herramienta⁶, también se deja una captura del mismo para ver el funcionamiento de la herramienta en el editor de Unity en la figura 3.3, donde se puede ver al avatar “Remy” en mitad de una animación de baile. Este avatar lleva

⁶Enlace al timelapse completo a x4 de tiempo <https://youtu.be/N0awMoaG98M>

a cabo las animaciones del dataset para, en cada frame de ejecución, convertir sus coordenadas al [CSV](#) correspondiente.



Figura 3.3: Frame de un *timelapse* de la ejecución de la herramienta

En la figura 3.4 se puede ver un diagrama de flujo del funcionamiento de esta herramienta, mientras que el código de la misma es público⁷ y se puede ver en el apéndice J.

⁷Enlace al repositorio de la herramienta: <https://github.com/FratosVR/Mixamo-Animation-Dumper>

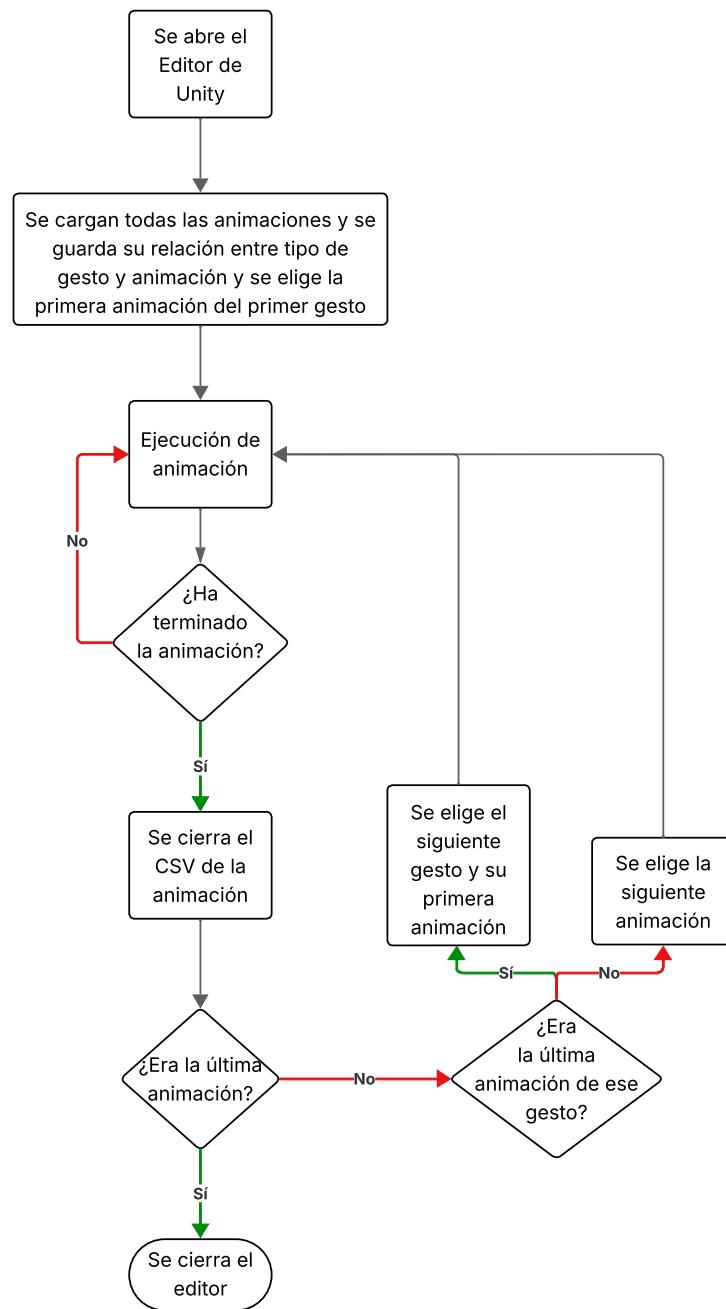


Figura 3.4: Diagrama de flujo de la herramienta que carga y ejecuta las animaciones de Mixamo en tiempo de ejecución, llamada Mixamo Dumper

El número de animaciones que se consiguieron grabar se puede ver en la figura 3.5, donde se puede observar un desbalanceo de algunas animaciones (bailar o pelear) frente a otras (saludar o señalar).

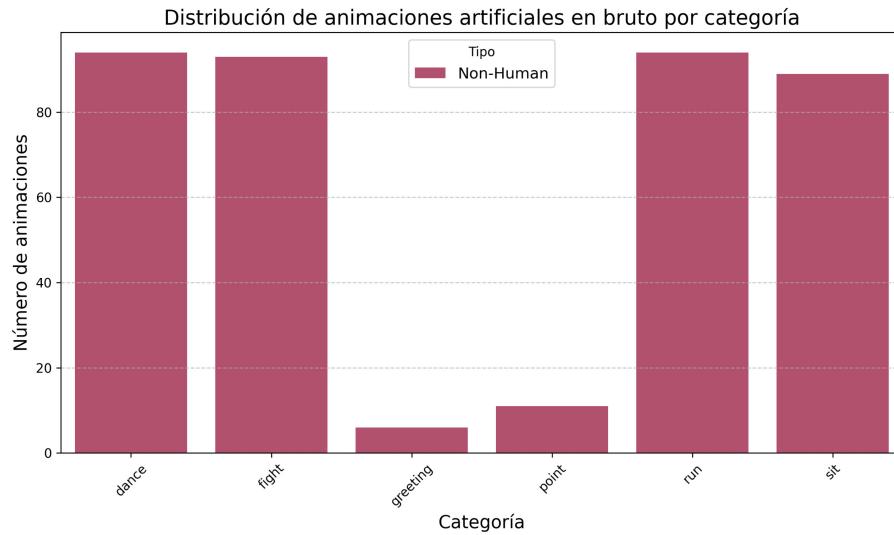


Figura 3.5: Número de animaciones conseguidas gracias a la herramienta divididas por tipo de gesto

Al finalizar todas las animaciones se cierra el editor de Unity y se procesan los CSV resultantes.

La ventaja de hacer esta herramienta es que es independiente del número de gestos y de animaciones por cada gesto, lo que hace que esta herramienta haga escalable el caso de que se quieran introducir o quitar gestos.

3.2.4. Tratamiento de CSV resultantes

Los [CSVs](#) restantes se copiaron en un nuevo *dataset* de Kaggle en una carpeta llamada “full_animations”. Una vez subidos estos datos, se creó una clase en Python para estandarizar estos [CSVs](#). Los datos se estandarizaron con el siguiente criterio:

- Todas las animaciones deben de tener el mismo número de frames. Para ello se estandarizará la duración a 90 frames que es el número de [FPS](#) que usan las gafas de [VR](#) de Meta Quest 2 y la herramienta Mixamo Animation Dumper.
- Las animaciones que duraran más de 90 frames se dividen en varias animaciones de 90 frames o menos.
- Las animaciones que duraran menos de 90 frames se repiten desde el inicio hasta completar los 90 frames.
- Las animaciones que no lleguen a 10 frames se consideran inválidas y se eliminan.

Este proceso se puede ver de manera más visual en la figura 3.6 y se puede ver el diagrama de la clase `DataLoader`⁸ en la figura 3.7, algunas de las funciones de esta clase se verán en los próximos apartados.

⁸Enlace de la herramienta <https://github.com/FratosVR/Models/src/DataLoader.py>

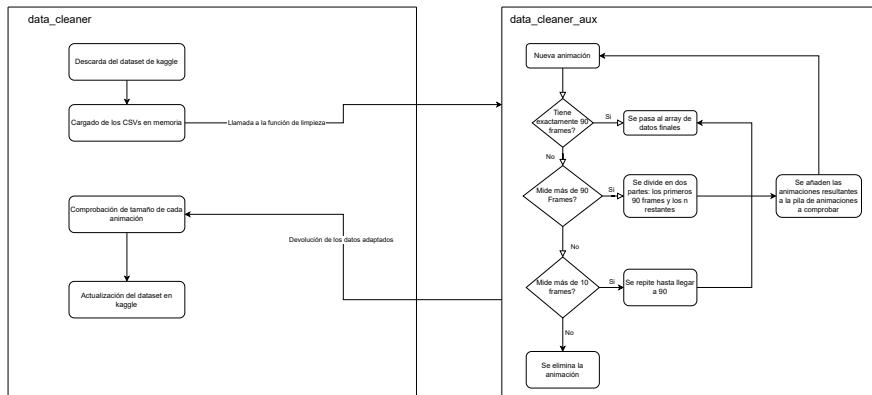


Figura 3.6: Diagrama de flujo del proceso de limpieza de datos (funciones `data_cleaner` y `data_cleaner_aux` de la clase `DataLoader`)

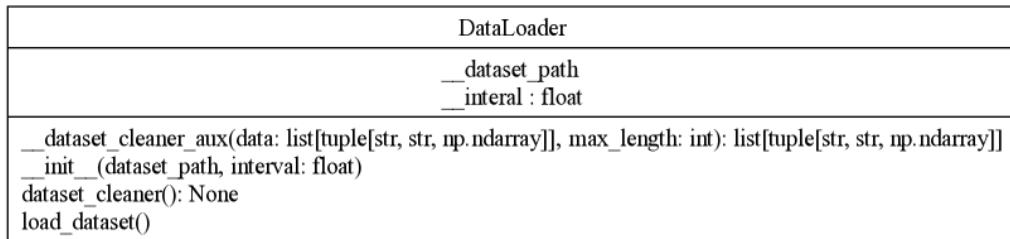


Figura 3.7: Diagrama de la clase `DataLoader`

El resultado de la estandarización se puede ver en la figura 3.8, donde se puede ver una diferencia notable entre las animaciones de bailar frente al resto.

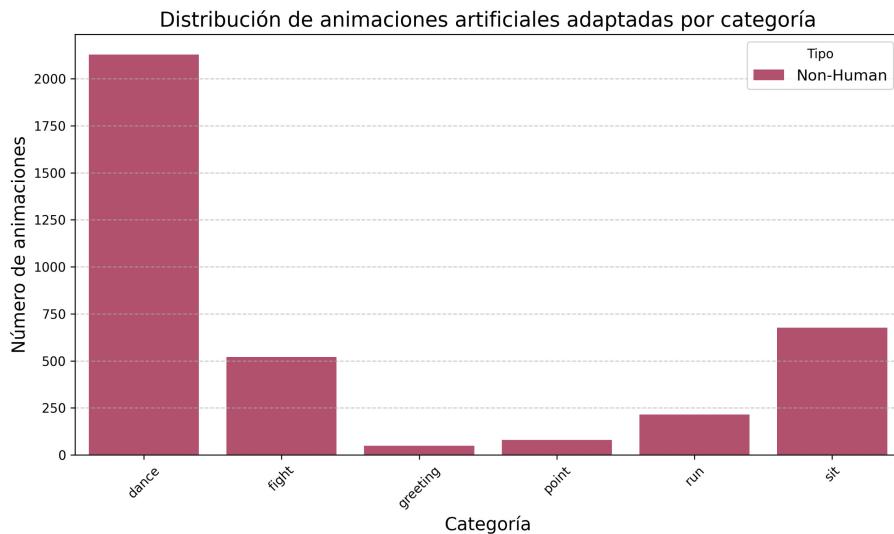


Figura 3.8: Número de animaciones estandarizadas divididas por tipo de gesto

3.3. Dataset real

Debido a que las animaciones encontradas en el banco de animaciones no eran las suficientes y estaban descompensadas, se decidió crear una herramienta para poder recoger los datos de usuarios.

3.3.1. Traje

Para poder recoger el movimiento de los usuarios era necesario configurar el traje y poder usarlo dentro de Unity. El traje usado, Perception Neuron 3, viene con:

- 17 sensores identificados con cada punto.
- 15 Bandas de velcro que se ponen en el cuerpo para colocar los sensores.
- 3 puertos de carga para los sensores junto a 3 cables USB-C para conectarlos.
- 1 USB que contiene la licencia para poder conectar el traje al ordenador.
- 1 USB-C que funciona de receptor de las señales del traje.

Para poder usar el traje se necesita la aplicación Axis Studio.⁹

3.3.1.1. Axis Studio y conexión del traje

Axis Studio es la aplicación creada por Noitom para poder capturar y grabar el movimiento en tiempo real y transmitirlo a aplicaciones de terceros (Unity, Blender o Unreal por ejemplo).

Mientras la aplicación está abierta es necesario tener el USB de la licencia introducido en el equipo. Al abrir la aplicación y crear un proyecto aparece un espacio vacío (como se puede ver en la figura 3.9) en la parte izquierda de la pantalla, que es donde se verá en tiempo real la captación de movimiento; y un panel de la información del traje en la parte derecha.

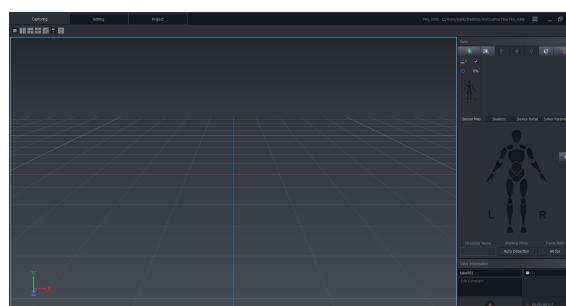


Figura 3.9: Captura de la aplicación Axis Studio antes de conectar un traje

⁹Enlace a la página de descarga de Axis Studio: <https://www.noitom.com/perception-neuron-downloads>

Para conectar el traje se debe tener enchufado en el mismo equipo el USB receptor y los puertos de carga con los sensores en ellos. Posteriormente se deben desenchufar los puertos para que el traje se conecte con el receptor, que se muestra con un parpadeo azul en los sensores.

Una vez los sensores puestos en sus huecos de las bandas se debe pulsar al botón mostrado en la figura 3.10 se debe calibrar el traje en el botón mostrado en la figura 3.11. Esta calibración consiste en tres pasos:

1. Posición de T: cabeza recta, piernas rectas paralelas a los hombros y brazos extendidos hacia los lados.
2. Posición de S: cabeza recta, piernas ligeramente flexionadas, espalda recta y brazos extendidos hacia el frente.
3. Posición de A: cabeza recta, piernas rectas paralelas a los hombros y brazos relajados paralelos al tronco.

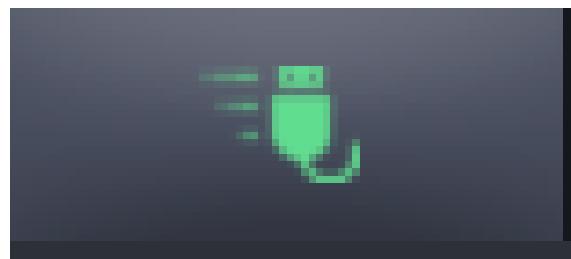


Figura 3.10: Captura del botón para conectar el traje a Axis Studio

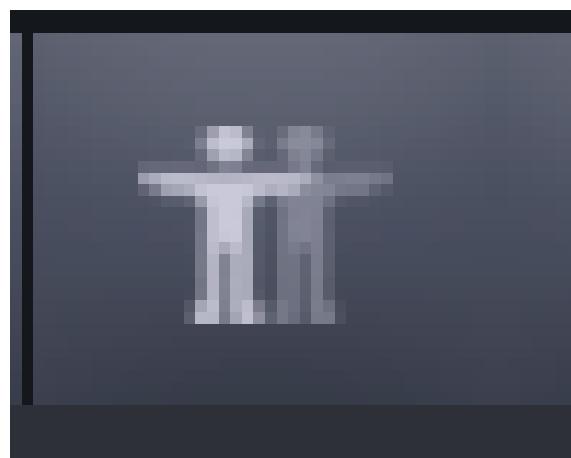


Figura 3.11: Captura del botón para calibrar el traje

Una vez el traje conectado y calibrado ya puede capturar el movimiento en tiempo real y transmitirlo a Unity.

3.3.1.2. Unity y la conexión del traje

Para poder usar la captura en Unity es necesario el plugin Neuron Mocap Live.¹⁰

Una vez metido el plugin en el proyecto se necesita un objeto vacío que contenga el componente “Neuron Source Manager”. Este componente se conecta a la aplicación mediante la dirección IP de la máquina que ejecuta Axis Studio y sockets TCP a un puerto que puedes configurar en la aplicación de Axis Studio mediante el panel que se muestra en la figura 3.12

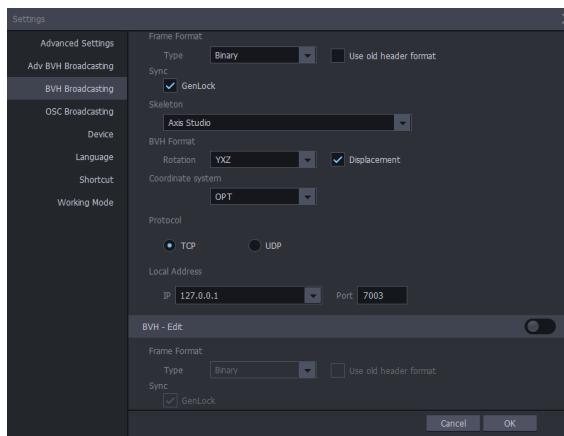


Figura 3.12: Captura del panel de configuración de Axis Studio

Posteriormente todos los objetos que vayan a usar la captura de movimiento deben ser hijos de este objeto vacío y tener el componente “Neuron Transforms Instance”. Este componente tiene una lista de Transforms que usa para mapear los huesos del traje con los del objeto de Unity.

3.3.1.3. Limitaciones encontradas con el traje

Durante el desarrollo del proyecto han habido dos limitaciones con respecto al traje.

La primera de ellos es la ausencia de guantes para capturar los dedos, ya que no vienen incluidos con el traje. A pesar de que esto causa que los dedos estén siempre en la misma posición con respecto a la mano no ha causado ningún problema en la identificación de gestos.

La segunda limitación es la cantidad de ruido electromagnético presente en nuestro espacio de trabajo: la Facultad de Informática de la Universidad Complutense de Madrid. Este ruido causaba interferencias con el traje, haciendo que la señal con el ordenador fuese pobre constantemente y no se capturasen los movimientos correctamente.

Para solventarlo se requirió de un cable alargador de USB 2.0 de 10 metros que se conectaba al ordenador en el que se enchufaba el receptor y se acercaba lo más posible al traje, ampliando la señal al máximo.

¹⁰Enlace a la página de documentación de Neuron Mocap Live: <https://support.neuronmocap.com/hc/en-us/sections/206474008-UNITY-SDK>

Una vez conectado el traje, solventadas limitaciones y sabiendo los puntos que detecta era necesario un dataset que se pudiera acoplar al esqueleto que proporciona el traje.

3.3.2. Herramienta de recogida de datos con usuarios reales

La herramienta se debe conectar con el traje como hemos mencionado anteriormente en el apartado 3.3.1.2 Una vez conectado se le debe pasar al componente MocapDumper el nombre de la animación que se va a grabar, el número de toma de esa animación y un número que sirva como identificador de usuario.

Finalmente cuando la herramienta está ejecutándose y se le da a la tecla “espacio” genera un CSV del formato “animación_User_Número de usuario_Take_Número de toma” (si el fichero ya existía lo sobreescribe), escribe la misma cabecera del CSV que se menciona en la tabla A.1 y pone la aplicación en estado de grabación. En este estado la aplicación escribe en cada frame todos los componentes de la posición y rotación (en forma de vector de ángulos de Euler) de cada hueso. Cuando se le vuelve a dar al espacio la aplicación cierra el fichero y vuelve a un estado de no grabación.

En la figura 3.13 se muestra un diagrama de cómo se conectan los diferentes componentes necesarios en esta herramienta

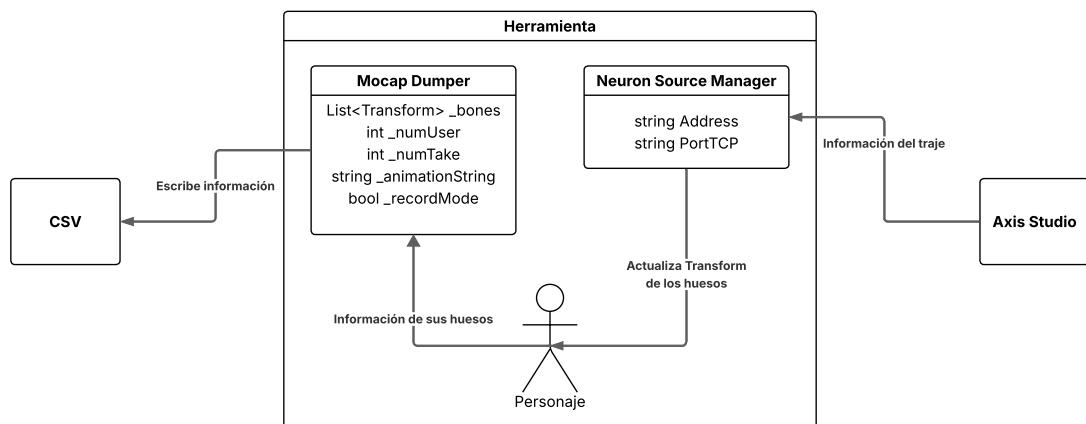


Figura 3.13: Diagrama de la conexión entre los distintos componentes de la herramienta, llamada Mocap Dumper

3.3.3. Recogida de datos

La recogida de datos consistió en ponerle el traje de captura de movimiento a los usuarios y pedirles que realizasen tres tomas de cada uno de los gestos, a excepción del gesto de baile, ya que de ese gesto había bastantes más datos que el resto.

Para ello se creó un formulario para que los usuarios interesados en ello se apuntasen. En el formulario se explicaba el objetivo de la prueba y se numeraban los datos que iban a ser pedidos en la prueba. Los campos a llenar eran: correo, consentimiento informado de la recogida de datos posterior y posibles huecos libres (en

un horario de lunes a viernes, de 9:00 a 20:00 en espacios de una hora).

El formulario se puede ver en las capturas de pantalla del apéndice C. Este formulario y el de demografía fueron comprobados por María del Carmen Fernández Villalba, Responsable de Protección de datos en Vicepresidencia Primera, perteneciente a Presidencia de La Junta de Comunidades de Castilla La Mancha, dado que no se quería vulnerar la Ley de Protección de Datos de ninguna manera.

Una vez creado el formulario se creó una selección de carteles (figuras B.1, B.2, B.3) el cual se colgó en redes y se colgó en distintas facultades, teniendo como resultado que se apuntasen 75 personas en el formulario.

Lo siguiente era tener un sitio en el que hacer las pruebas. Debido a que las pruebas requerían movimiento era preciso un lugar en el que los usuarios se pudiesen mover sin dificultades y que no estuviese a la vista para preservar la intimidad de los mismos.

Una vez se cerró el formulario, se procedió a hacer un algoritmo que asignara la mayor cantidad de citas posibles dadas las restricciones de tiempo de los usuarios y los días disponibles. El algoritmo¹¹ usa un esquema de ramificación y poda para conseguir la combinación con mayor cantidad de citas disponibles sin tardar más de 10 segundos en procesar las combinaciones válidas.

Para poder recolectar los datos, se hizo una herramienta¹² que funciona igual que la herramienta del apartado anterior pero usando los movimientos de los usuarios en vez de los de Mixamo.

Finalmente desde el día 14/04/2025 a las 9:00 hasta el día 17/04/2025 a las 19:30 se pudo grabar en el despacho 216 (Sala de grabaciones) de la Facultad de Informática de la Universidad Complutense de Madrid. De los 75 usuarios apuntados finalmente se presentaron 65, consiguiendo así 975 animaciones en formato CSV, 195 de cada gesto. En la figura 3.14 se muestra una fotografía de un usuario durante el proceso de calibración.



Figura 3.14: Fotografía de un usuario en las pruebas

¹¹Enlace al algoritmo: https://github.com/FratosVR/Models/blob/main/citas_generacion_dataset/Scheduler.py

¹²Enlace del repositorio <https://github.com/FratosVR/Mixamo-Animation-Dumper>

A los usuarios no solo se les pidió que realizaran gestos, sino que también se les pidió una serie de información demográfica de manera anónima para saber como de representativa era la muestra. La información pedida fue: género, edad, nacionalidad (o nacionalidades), idiomas hablados y mano dominante.

Este formulario se puede ver en el apéndice [D](#)

Estos datos se recogieron con la cuenta de Google de las personas que llevaban el formulario, por lo que no se guardó ningún dato identificable de ningún usuario. En la categoría de género([figura E.1](#)) podemos ver que los datos son un 43.1 % femenino, 50.8 % masculino y 6.2 % no binario. La edad([figura E.2](#)) más común es de 21, con un recuento de un 26.2 % de los encuestados. La nacionalidad ([figura E.3](#)) más común es la española, con un 89.23 % de los encuestados. En la figura [E.4](#) se puede ver que el idioma más hablado es el español, con un 100 % de los encuestados, seguido de inglés y francés. En la figura [E.5](#) se puede ver que el 93.85 % de los encuestados son diestros.

Tras las recogida de datos, se siguió el mismo proceso de estandarización de datos que se usó en el dataset artificial. Antes de estandarizar los datos parecía que se había conseguido eliminar el gran desbalance del dataset hacia los gestos de baile, tras la estandarización de los datos, se vió que las animaciones de baile seguían ganando en cantidad por mucho respecto a las otras. La comparativa se puede ver en las figuras [3.15](#) y [3.16](#).

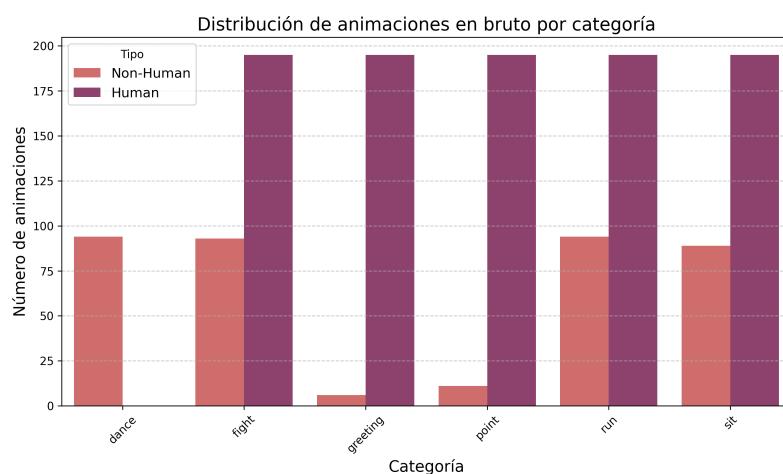


Figura 3.15: Distribución de los datos brutos

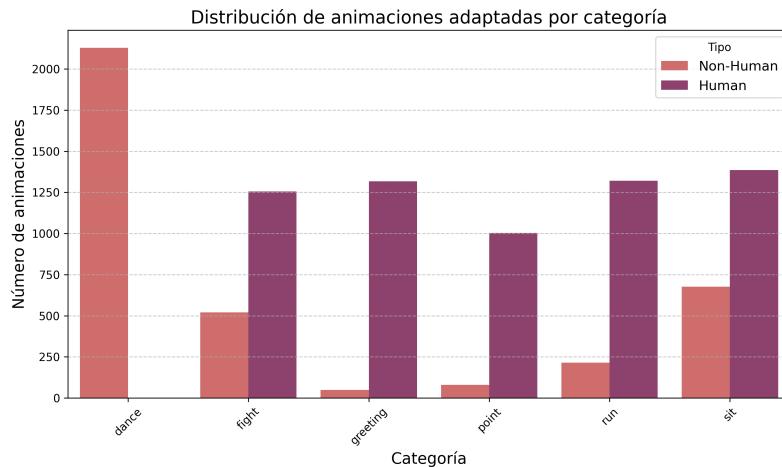


Figura 3.16: Distribución de los datos estandarizados

El resultado final de todos los datos se puede ver en [Barrachina Argudo y Sánchez Martín \(2025a\)](#). Una vez recogidos y procesados los datos recogidos en las pruebas de usuario se procede al entrenamiento de los distintos modelos para su posterior comparativa.

3.4. Modelos de IA

Tras la adaptación y guardado de datos, se procede al desarrollo de los modelos de IA. Este desarrollo no implica solo el modelo, sino también la creación de una interfaz de entrenamiento y visualizado de resultados (para los modelos que lo permiten).

En el repositorio¹³ se puede encontrar el código de todos los modelos y su entrenamiento, así como algunos modelos ya preentrenados. Durante esta sección se explicará el proceso de desarrollo de cada parte del proyecto.

3.4.1. Interfaz de entrenamiento y seguimiento

Al principio del desarrollo, surgió la necesidad de hacer que el entrenamiento de los modelos fuera accesible para gente que nunca hubiera trabajado con IA o Python. Esto ocurre debido a que se ideó una comparativa de modelos, y los ordenadores disponibles no daban la talla para entrenar los modelos en el tiempo necesario. Por esto se pidió como favor a la asociación LAG (asociación de estudiantes de la Facultad de informática de la UCM) el poder usar sus equipos con mejores gráficas para llevar a cabo estos entrenamientos.

Se usó Gradio ([Abid et al. \(2019\)](#)) para crear una interfaz web accesible para cualquier persona. En un primer momento esta interfaz solo contaba con una pantalla en la que había:

- Un selector de tipo de modelo
- Un área de texto para introducir la ruta al conjunto de datos

¹³Enlace al repositorio de GitHub <https://github.com/FratosVR/Models>

- Un *slider* para seleccionar el intervalo de tiempo entre los frames de animación
- Un botón para seleccionar todos los posibles intervalos de tiempo
- Un botón para iniciar el entrenamiento
- Un área de archivo para descargar el modelo entrenado
- Un área de imagen para mostrar la matriz de confusión tras el entrenamiento
- Un área de texto para mostrar el resultado del entrenamiento

Esta interfaz (figura 3.17) se hizo lo mas simple posible para que un usuario estándar pudiera poner a entrenar un modelo sin tener que tocar nada de código. Sólo se tienen que seleccionar el modelo y el intervalo deseado, introducir la ruta al *dataset* y esperar a que termine.

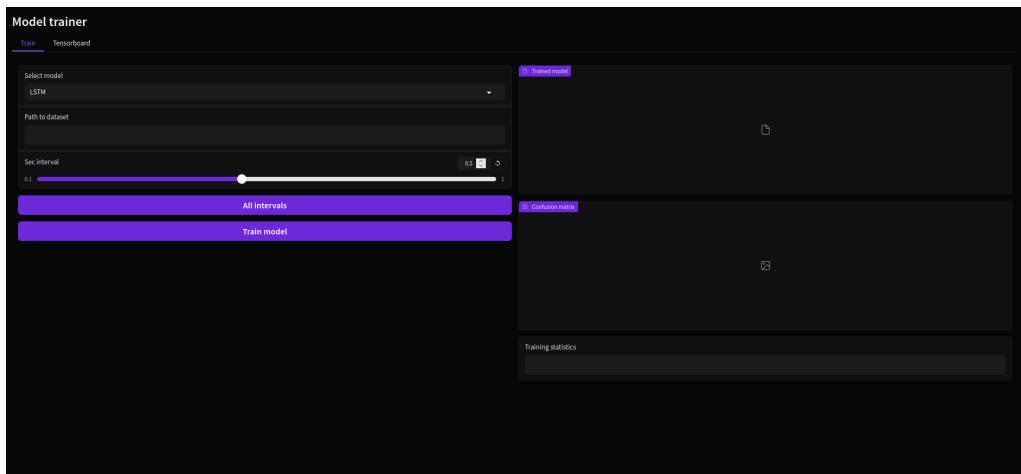


Figura 3.17: Interfaz de entrenamiento

La función de entrenamiento que se ejecuta al pulsar el botón “Train model” hace lo siguiente:

1. Se crea un entrenador del modelo seleccionado
2. Se crea un objeto *DataLoader* que se encargará de cargar los datos con el intervalo de tiempo de separación seleccionado.
3. Se separan los datos cargados en 60 % para entrenamiento, 20 % para validación y 20 % para test. Se consideró hacer cross validation, pero los tiempos de entrenamiento ya eran demasiado largos para los modelos neuronales.
4. Se pasan las categorías con *OneHotEncoder* para que el modelo pueda entenderlas.
5. Se entrena el modelo con los datos de entrenamiento y validación.
6. Se guarda el modelo.

7. Se comprueba cual es el mejor modelo entrenado entre todos los intervalos de tiempo seleccionados.

8. Se muestra la matriz de confusión en la interfaz, el archivo del modelo y sus estadísticas.

En la parte superior izquierda se puede ver dos secciones: “Train” y “TensorBoard”. La sección “Train” es la explicada anteriormente, la sección “TensorBoard” es una sección que permite ver el progreso del entrenamiento en tiempo real. No solo eso, sino que también nos permite ver el rendimiento de modelos ya entrenados y ver como los hiperparámetros han afectado a su tasa de acierto. TensorBoard permite añadir funciones al código de entrenamiento para modificar el comportamiento del método de entrenamiento y dejar constancia de las distintas partes del mismo. Esta interfaz se puede ver en la figura 3.18.

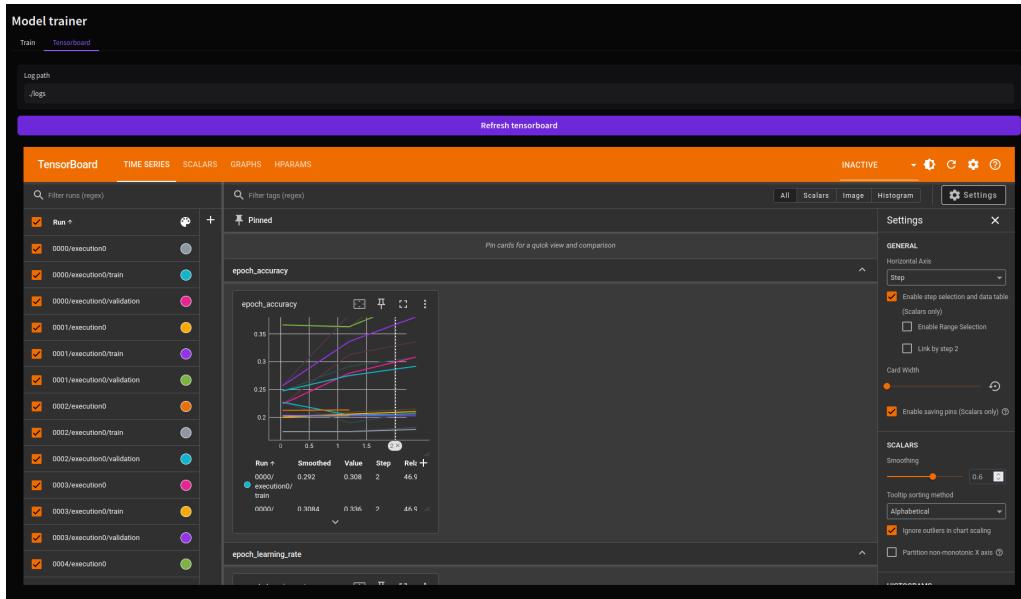


Figura 3.18: Interfaz de TensorBoard

Para todo lo descrito, el archivo “Interface.py” tiene la estructura presentada en la figura 3.19. Las funciones `__change_interval` y `__change_interval_all` cambian los intervalos de tiempo usados dentro del entrenamiento, `__setup_ui` crea el *layout* de la interfaz, `__refresh_tensorboard` cambia la ruta de los *logs* de TensorBoard y recarga el *iframe*. El método `__train` ejecuta todo el proceso de entrenamiento previamente descrito.

Interface
<code>_blocks : Blocks</code> <code>_models : list</code> <code>_models_map : dict</code> <code>_sec_interval : list</code> <code>log_path : NoneType</code> <code>_change_interval(value)</code> <code>_change_interval_all()</code> <code>_init__()</code> <code>_refresh_tensorboard(log_path)</code> <code>_setup_ui()</code> <code>_train(model, path)</code> <code>launch()</code>

Figura 3.19: Estructura del archivo de la interfaz

3.4.2. Carga de datos

La carga de datos se gestiona con la clase *DataLoader*. Esta clase se mencionó en el apartado del dataset artificial ya que es la misma que se encarga de estandarizar los datos. Cuando se llama a la función de inicio de la clase se le pasa un intervalo de tiempo, este intervalo marca la distancia entre frames que se va a cargar. Si le indicamos una distancia de 0.8, cargará el frame 0 y el 72. Esto nos permite entrenar distintos modelos con distintas ventanas de envío de datos para comprobar si hay alguna diferencia en el rendimiento.

Estos datos una vez cargados se envían como una lista de tuplas <tipo_animacion, lista_de_frames>. Tras esto, el objeto que reciba estos datos ya puede transformarlos como sea necesario y separarlos en entrenamiento, test y validación.

3.4.3. Modelos usados

A la hora de escoger modelos para la comparativa, se tuvo en cuenta el estudio [Ma et al. \(2023\)](#) como un referente. Se tuvo siempre en cuenta que se intentaba buscar un modelo con el cómputo más simple posible para que fuera lo más accesible para cualquier tipo de máquina. Valorando también las dificultades de tiempo que conlleva el entrenamiento de modelos al final se consideró probar tres redes neuronales usadas para series temporales y un modelo clásico de clasificación.

En un primer momento se pensó también en modelos como [YOLO](#), pero al ser un modelo basado en imágenes y el estudio plantearse con un traje de captura de movimiento, se decidió que sería una carga muy grande para las aplicaciones que usarán el modelo resultante por tener que renderizar una segunda cámara constantemente para el usuario. Otros modelos como [SVM](#) o [KNN](#) fueron descartados por falta de tiempo de entrenamiento.

3.4.3.1. LSTM

Este primer modelo va a servir para ilustrar como funcionan el resto de modelos basados en redes neuronales en este estudio. La estructura de la clase *LSTMTrainer*

se puede ver en la figura 3.20. La mayoría de los atributos privados de la clase son los hiperparámetros del modelo, los otros atributos se usan para guardar información o para guardar el mecanismo usado para buscar hiperparámetros óptimos.

La función `train_with_hparams` se encarga de generar modelos con distintos hiperparámetros y entrenarlos para buscar el mejor. Para que este mecanismo fuera lo más eficiente posible, se ha hecho uso de la biblioteca keras-tuner ([O’Malley et al. \(2019\)](#)), un *framework* de búsqueda de hiperparámetros con distintos algoritmos de búsqueda ya implementados. Tras leer el artículo [Li et al. \(2018\)](#) se optó por usar el algoritmo Hyperband, dado que los inconvenientes que presenta no representan un problema comparado con las ventajas de velocidad y eficiencia que otorga.

Esta búsqueda de hiperparámetros se centra en los siguientes:

- **activation**: Función de activación de la capa oculta.
- **dropout**: Porcentaje de *dropout* de la capa oculta.
- **recurrent_activation**: Función de activación de la capa recurrente.
- **recurrent_dropout**: Porcentaje de *dropout* de la capa recurrente.
- **unroll**: Si se usa el *unroll* de la capa LSTM.
- **use_bias**: Si se usa el *bias* de la capa LSTM.

El entrenamiento crea un modelo con los hiperparámetros seleccionados dentro de un rango, se entrena durante unas épocas y se comprueban resultados. Se pasa a otro conjunto de hiperparámetros y así hasta completarlos todos. Tras esto, se aumenta el número de épocas y se siguen entrenando los anteriores modelos. Tras acabar de entrenar todos los modelos, se selecciona el que mejor precisión en validación tenga.

De este “mejor modelo” se crea una matriz de confusión, se guarda el modelo en formato keras y se envían las estadísticas a la interfaz de gradio. Todo este proceso queda registrado en TensorBoard gracias a la librería *HParams* ([Petrochuk \(2019\)](#)).

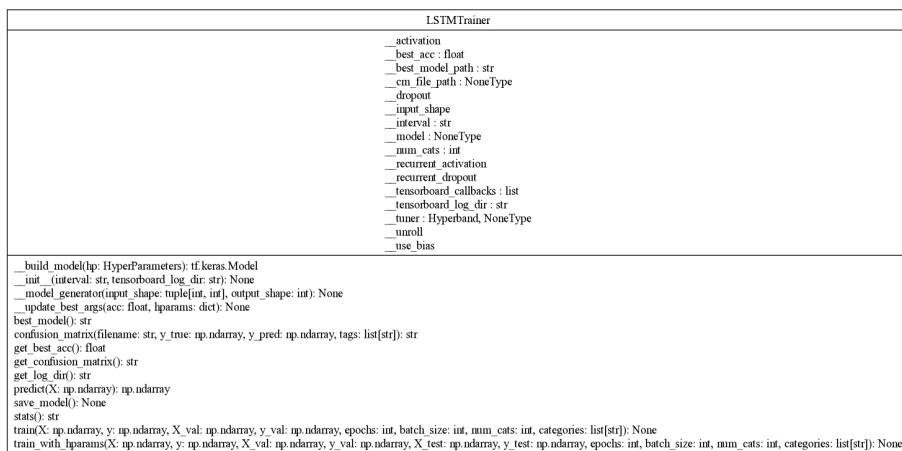


Figura 3.20: Estructura de la clase LSTMTrainer

El modelo **LSTM** usado es un *Sequential* de keras con una capa LSTM y una capa densa. La capa LSTM con los hiperparámetros antes descritos y la capa densa

tiene tamaño el número de categorías de animaciones y una activación softmax. El modelo se compila utilizando el optimizador *Adam*, la perdida se calcula con *categorical_crossentropy* y la métrica de evaluación es la *accuracy*. Durante el proceso de entrenamiento se usan los *callbacks* de *EarlyStopping* para ahorrar recursos en entrenamientos que no parecen mejorar y los *callbacks* de *TensorBoard* para guardar los logs de entrenamiento y poder verlos en la interfaz de TensorBoard.

Entre los modelos de **LSTM** entrenados, el del intervalo de 0.8 fue el que mejor rendimiento tuvo, con una precisión de 0.51025 en entrenamiento y 0.5912 en validación. A continuación se puede ver el esquema del modelo en la figura 3.21, la matriz de confusión en la figura 3.22 y el gráfico de entrenamiento en la figura 3.23. Podemos ver en la matriz de confusión que los gestos menos reconocidos son baile y saludo. Los resultados del resto de intervalos se pueden ver en el apéndice F.

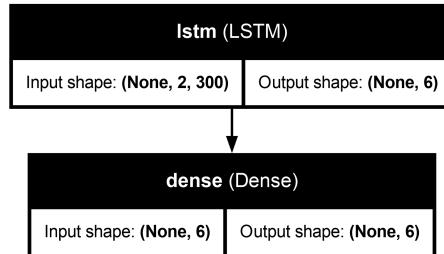


Figura 3.21: Esquema del modelo LSTM

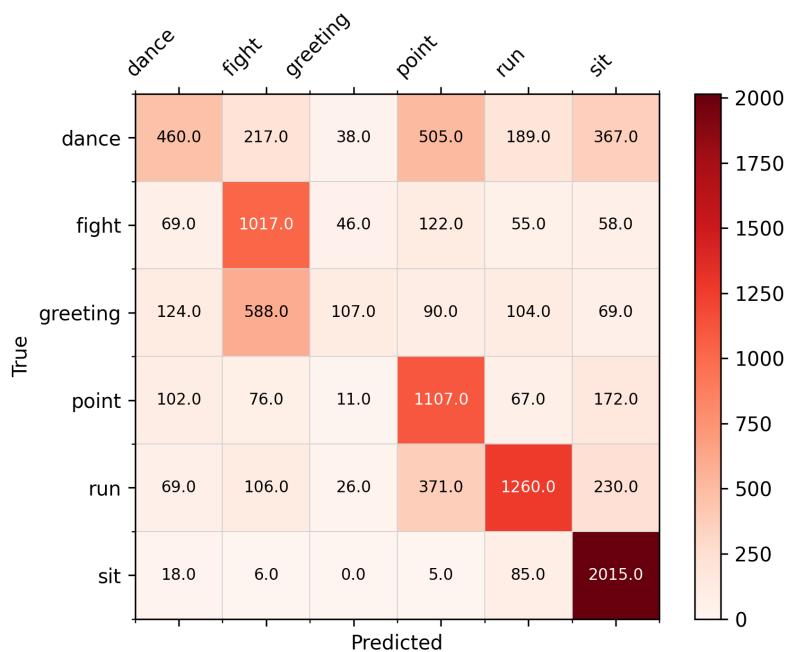


Figura 3.22: Matriz de confusión del modelo LSTM

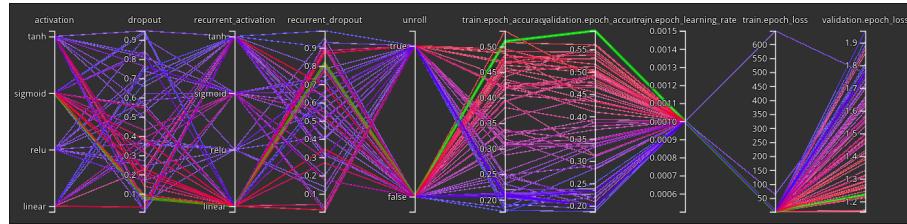


Figura 3.23: Gráfico de entrenamiento del modelo LSTM

3.4.3.2. CNN

El modelo de **CNN** se entrenó de una manera muy similar al de **LSTM**. En el uml de la figura 3.24 se puede ver la estructura de la clase *CNNTrainer*, la cual es muy similar a la de *LSTMTrainer*. La diferencia está en los hiperparámetros buscados. Estos son:

- **activation:** Función de activación de la capa oculta.
- **conv_filters:** Número de filtros de la capa convolucional.
- **dense_units:** Número de neuronas de la capa densa.
- **dropout:** Porcentaje de *dropout* de la capa oculta.
- **kernel_size:** Tamaño del kernel de la capa convolucional.
- **pool_size:** Tamaño del *pooling* de la capa convolucional.

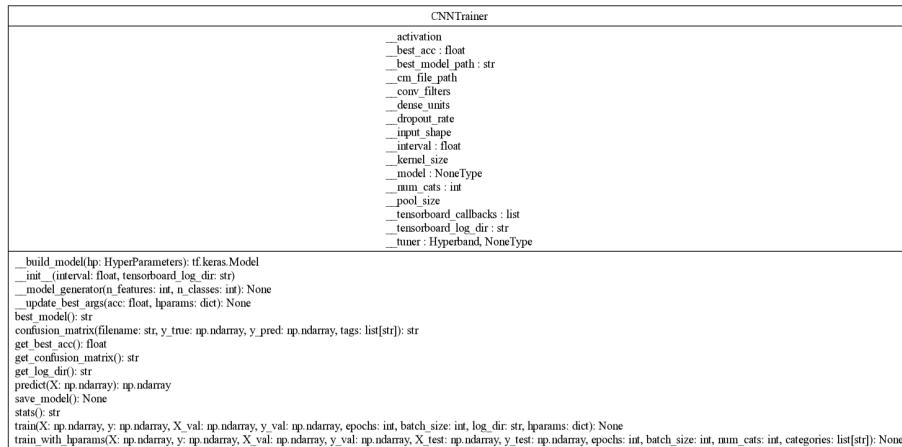


Figura 3.24: Estructura de la clase CNNTrainer

El método de entrenamiento es el mismo que el descrito en el apartado anterior, haciendo uso de keras-tuner para encontrar los mejores hiperparámetros. En el caso de este modelo, el mejor resultado se consigue con el intervalo de 1 segundo, haciendo este modelo más un reconocedor de poses que de gestos, con una precisión de 0.55879 en entrenamiento y 0.56002 en validación. En la figura 3.26 se puede ver el esquema del modelo, en la figura 3.25 la matriz de confusión y en la figura 3.27 el gráfico

de entrenamiento. En este caso podemos ver que los gestos que peor detecta son baile y saludo (siendo saludo muy confundido con pelea). Los resultados del resto de intervalos se pueden ver en el apéndice H.

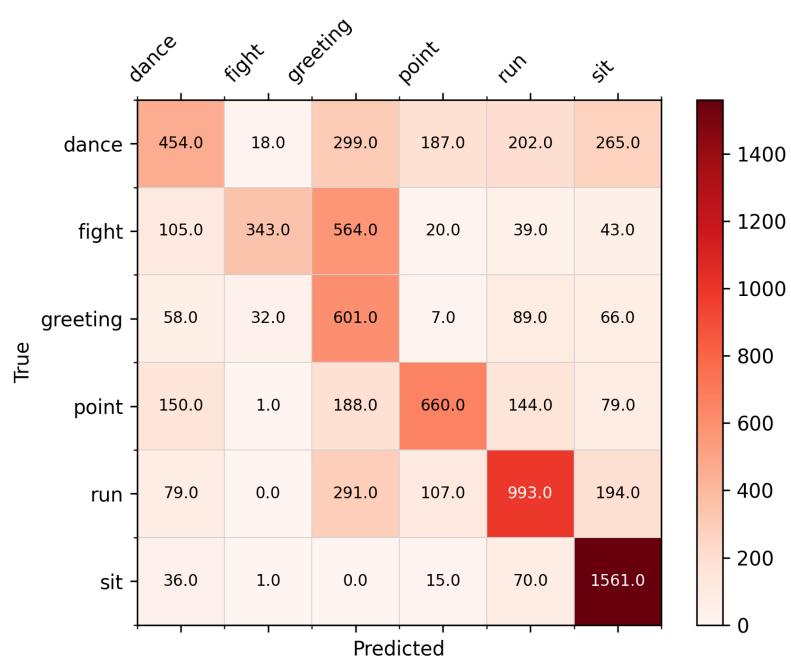


Figura 3.25: Matriz de confusión del modelo CNN

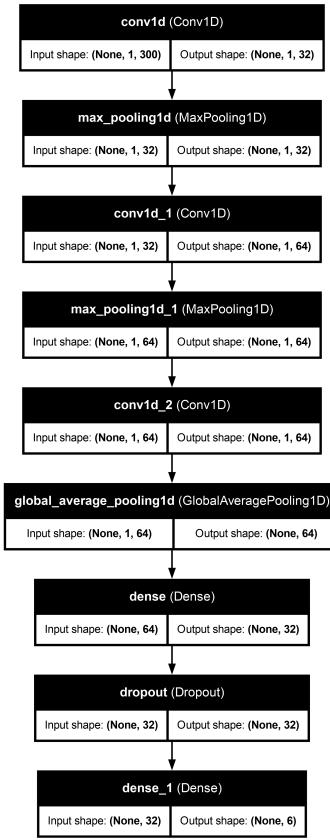


Figura 3.26: Esquema del modelo CNN

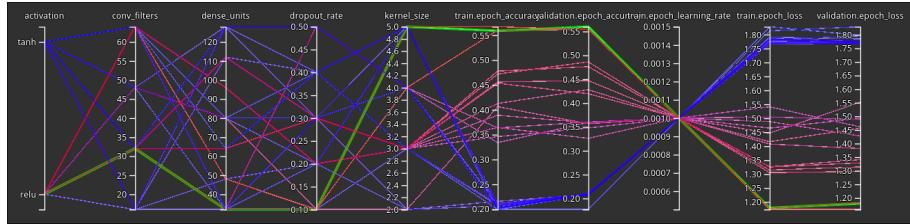


Figura 3.27: Gráfico de entrenamiento del modelo CNN

3.4.3.3. RNN

Con el **RNN** se sigue la misma base que para los dos modelos anteriores. La clase *RNNTrainer* (estructura en la figura 3.28)es muy similar a las anteriores, cambiando una vez más los hiperparámetros a buscar. En este caso son:

- **activation:** Función de activación de la capa oculta.
- **activity regularizer:** Regularizador de la capa oculta.
- **bias constraint:** Restricción del *bias* de la capa oculta.
- **bias initializer:** Inicializador del *bias* de la capa oculta.

- **bias regularizer:** Regularizador del *bias* de la capa oculta.
- **dropout:** Porcentaje de *dropout* de la capa oculta.
- **kernel constraint:** Restricción del kernel de la capa oculta.
- **kernel initializer:** Inicializador del kernel de la capa oculta.
- **kernel regularizer:** Regularizador del kernel de la capa oculta.
- **recurrent constraint:** Restricción de la capa oculta.
- **recurrent dropout:** Porcentaje de *dropout* de la capa oculta.
- **recurrent initializer:** Inicializador de la capa oculta.
- **recurrent regularizer:** Regularizador de la capa oculta.
- **unroll:** Si se usa el *unroll* de la capa oculta.
- **use bias:** Si se usa el *bias* de la capa oculta.

RNNTrainer	
	<pre> __activation __activity_regularizer : NoneType best_acc : float best_model_path : str bias_initializer : NoneType bias_regularizer : NoneType cm_file_path dropout input_shape interval : str kernel_constraint : NoneType kernel_initializer kernel_regularizer : NoneType model : NoneType num_cats : int recurrent_constraint : NoneType recurrent_dropout recurrent_initializer recurrent_regularizer : NoneType tensorboard_callbacks : list tensorboard_log_dir : str tuner : Hyperband, NoneType unroll use_bias </pre> <pre> build_model(hp: HyperParameters) int_(interval: str, tensorboard_log_dir: str) model_generator(input_shape: tuple[int, int], output_shape: int): None update_best_args(acc: float, res: dict): None best_model(): str confusion_matrix(filename: str, y_true: np.ndarray, y_pred: np.ndarray, tags: list[str]): str get_best_acc(): float get_confusion_matrix(): str get_log_dir(): str predict(X: np.ndarray): np.ndarray save_model(): None stats(): str train(X: np.ndarray, y: np.ndarray, X_val: np.ndarray, y_val: np.ndarray, epochs: int, batch_size: int, log_dir: str, hparams: dict): None train_with_hparams(X: np.ndarray, y: np.ndarray, X_val: np.ndarray, y_val: np.ndarray, X_test: np.ndarray, y_test: np.ndarray, epochs: int, batch_size: int, num_cats: int, categories: list[str]): None </pre>

Figura 3.28: Estructura de la clase RNNTrainer

A pesar de ser el modelo con mayor búsqueda de hiperparámetros, está en una reñida competición con el modelo [CNN](#) para ver cual de los dos detecta peor los gestos. Su mejor intervalo ha sido el de 0.8 segundos con una precisión de 0.42714 en entrenamiento y 0.47865 en validación. En la figura [3.29](#) se puede ver el esquema del modelo, en la figura [3.30](#) la matriz de confusión y en la figura [3.31](#) el gráfico de entrenamiento. En este caso podemos ver que los gestos que peor detecta son baile y saludo (siendo pelea muy confundida con saludo). Los resultados del resto de intervalos se pueden ver en el apéndice [G](#).

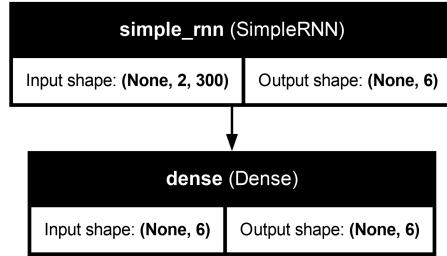


Figura 3.29: Esquema del modelo RNN

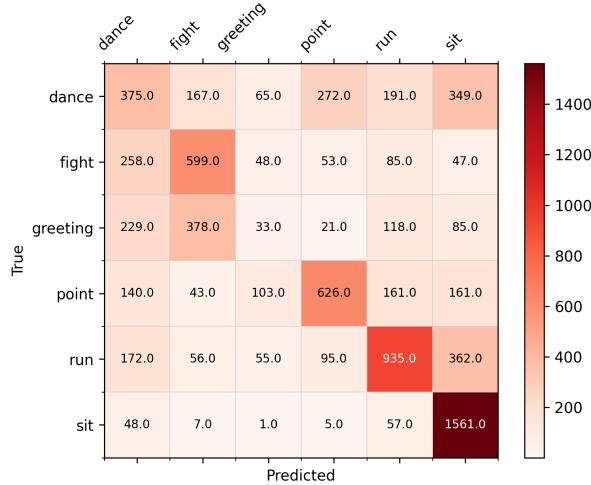


Figura 3.30: Matriz de confusión del modelo RNN

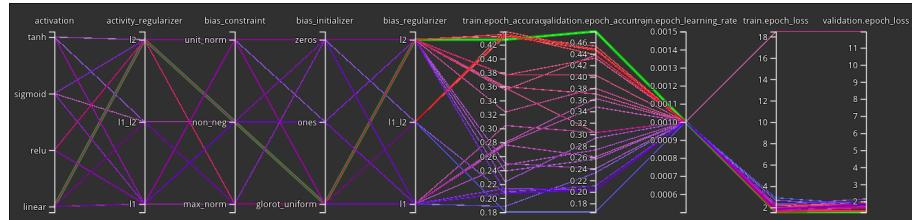


Figura 3.31: Gráfico de entrenamiento del modelo RNN

3.4.3.4. Random Forest

El entrenador del **Random Forest** es el más sencillo y el más distinto de todos. Este modelo no es una red neuronal, sino un modelo clásico de clasificación. La clase *RandomForestTrainer* (estructura en la figura 3.32) sigue utilizando la función *train_with_hparams* como las anteriores aún no habiendo *hparams* como tal, haciendo esto se consigue mantener la igualdad de todas las clases de cara a la interfaz de gradio.

Este modelo no usa tensorflow como tal, dado que YDF es un desarrollo que deja atrás el uso de TensorFlow Decision Forest para implementar algoritmos más eficientes tanto en entrenamiento como en inferencia. A pesar de ser un desarrollo

paralelo, el equipo de YDF decide hacer el modelo compatible con Tensorflow para poder integrarlo en un mayor ecosistema. Estos datos de mayor eficiencia se pueden comprobar en el estudio [Guillame-Bert et al. \(2023\)](#).

A diferencia de los modelos anteriores, este modelo usa DataFrames de pandas para el entrenamiento y la predicción en vez de tensores. En este entrenamiento usamos *RandomSearchTuner* para buscar la mejor combinación de número de árboles, profundidad máxima y mínimo de ejemplos.

RandomForestTrainer						
best_accuracy : float						
best_model_path : str						
cm_file_path : NoneType						
interval : str						
log_dir : str						
model : NoneType						
tensorboard_callbacks : list						
init(interval: str, log_dir: str): None						
best_model(): str						
get_best_acc(): float						
get_confusion_matrix(): str						
get_log_dir(): str						
plot_confusion_matrix(filename: str, y_true: np.ndarray, y_pred: np.ndarray, tags: list[str]): str						
predict(X: pd.DataFrame): np.ndarray						
save_model(): None						
stats(): str						
train(X: np.ndarray, y: np.ndarray, X_val: np.ndarray, y_val: np.ndarray, categories: list[str]): None						
train_with_hparams(X: np.ndarray, y: np.ndarray, X_val: np.ndarray, y_val: np.ndarray, X_test: np.ndarray, y_test: np.ndarray, epochs: int, batch_size: int, num_cats: int, categories: list[str]): None						

Figura 3.32: Estructura de la clase RFTrainer

Por desgracia, este modelo no deja una constancia tan rigurosa como los anteriores dado que no usa TensorBoard ni tiene un equivalente. A pesar de esto, este es el modelo con mayor rendimiento de lejos. Todos los intervalos tienen un rendimiento más o menos equivalente, ganando el de 0.8 segundos por muy poco.

En la figura 3.33 se puede ver la matriz de confusión del modelo y en el apendice I se pueden ver los resultados de todos los intervalos. En este caso podemos ver que los gestos que peor detecta son pelea y saludo. La precisión en validación fue de 0.85936 y de 0.97487 en entrenamiento, lo que lo convierte en el mejor modelo entrenado.

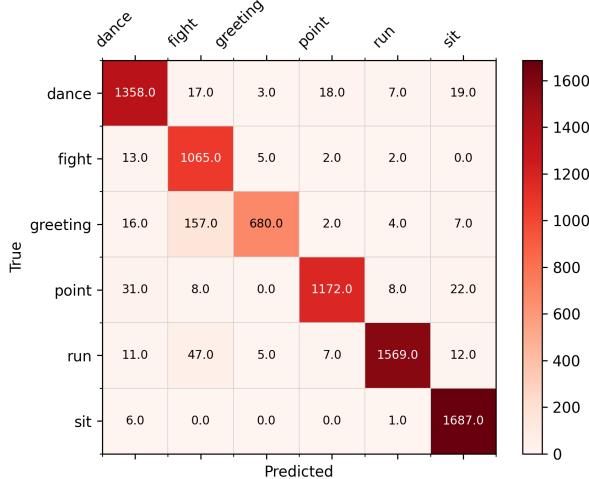


Figura 3.33: Matriz de confusión del modelo RandomForest

Este modelo finalmente es exportable a TensorFlow, siendo esto muy útil para su despliegado y compatibilización con otras herramientas. Dentro del repo, en la

carpeta de “notebooks” se puede ver un resumen de las características más decisivas del modelo y su importancia en la diferenciación de dos gestos cuales sean.

3.4.4. TensorFlow Serving

Que todos los modelos sean exportables a TensorFlow permite el uso de TensorFlow Serving ([Olston et al. \(2017\)](#)) para desplegar modelos. Serving nos proporciona una interfaz sencilla mediante [API REST](#) para re entrenar los modelos y poder hacer inferencia sobre ellos desde cualquier máquina conectada a la red.

Este método de despliegado es especialmente útil a la hora de integrar los modelos con otras tecnologías sin necesidad de hace traducciones a otros lenguajes o formatos. El servidor se despliega usando docker y cogiendo el modelo de la última *release* del repositorio. Para hacer una inferencia solo hay que hacer una petición *POST* con el formato visto en la figura 3.34 y el servidor devuelve la respuesta vista en la figura 3.35.

```

1 curl <model_server>/v1/models/<model_name>:predict \
2 -X POST \
3 -d '{"instances": [
4     {"feature_0":0.24,
5     "feature_1":0.12,
6     ...
7     "feature_599":0.56
8 ]}'
```

Figura 3.34: Ejemplo de petición a TensorFlow Serving

```

1 {
2     "predictions": [
3         "scores": [0.981395662, 0.0186043456, 0.1920000, 0.0112349,
4         0.0001234, 0.0001234],
5         "classes": ["dance", "fight", "greeting", "point", "run",
6         "sit"]
7     ]
8 }
```

Figura 3.35: Ejemplo de respuesta de TensorFlow Serving

En el repositorio se pueden encontrar dos scripts, uno para windows y otro para linux, que permiten desplegar el servidor de manera automática siempre que docker ya esté instalado en el sistema.

3.4.5. Hardware, complejidad de datos y otras limitaciones

Este estudio ha estado muy limitado por el hardware disponible para el mismo. El equipo usado para el modelo tenía gráficas de consumidor de hace más de 5 años,

haciendo que los modelos con redes neuronales tardaran bastante en entrenarse. Esto no fue un problema en el caso del **Random Forest**, ya que se entrena en **CPU** y permite unos tiempos de entrenamiento mucho menores.

Estas limitaciones de hardware también han afectado a la visión de los modelos a desarrollar, ya que en un principio se pensó en hacer modelos lo más simples posibles para poder utilizarlos incluso en equipos más limitados (gafas de **VR** o equipos más antiguos).

Así mismo, la falta de espacios y de tiempo han resultado en que, junto a la falta de datasets ya formados, el número de datos fuera limitado para lo que requeriría el entrenamiento de redes neuronales.

3.5. Aplicación final

Una vez los modelos estaban preparados era necesaria una aplicación a modo de demostración que se pueda ejecutar en las gafas de **VR** y con el traje de caputra de movimiento. La aplicación consta de un panel en el que introduces la IP de la máquina que tiene el servidor de Tensor ejecutándose con el modelo y la aplicación de Axis Studio para poder utilizar el traje. Además, tiene un panel de texto en el que aparecen los resultados de la predicción y un NPC que reacciona a tus gestos. En la figura 3.36 se puede ver una captura de esa aplicación.



Figura 3.36: Captura de la aplicación final desde el editor de Unity

3.5.1. Funcionamiento de la aplicación

Al introducir la IP en el cuadro de texto la aplicación se conecta con el traje y el servidor de Tensor. La conexión con el traje es la misma que la del apartado 3.3.1.2, mientras que para el servidor se utiliza una **API REST** mediante **UnityWebRequest**,¹⁴ una API de Unity para comunicarse con servidores web. Esta ejecución se hace mediante una corutina, la cual al principio espera la cantidad de tiempo óptima entre frame y frame que se determinó en el entrenamiento del modelo usado (en nuestro caso el Random Forest con 0.8 segundos).

¹⁴Página de la documentación de Unity: <https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.html>

La información de los huesos se guarda en una cola, donde se guarda la información actual y, una vez enviado al servidor, se elimina la más antigua, siendo siempre el tamaño de la cola de máximo dos elementos. Al pasar estos segundos la aplicación manda al servidor la información de los huesos en el frame actual y en el frame de hace 0.8 segundos (previamente guardado) en formato JSON mediante POST con protocolo HTTP, a lo que el servidor le responde con dos listas en formato JSON: una con los gestos y otra con el score que ha conseguido cada gesto en la predicción, coincidiendo el índice del score de un gesto con el índice del gesto de la anterior lista. Para hacer todas las conversiones entre cadenas de texto y el formato JSON se ha usado la clase JsonUtility nativa en Unity.

Una vez se tenga la respuesta del servidor empieza de nuevo la corriputina, se ordena de mayor a menor score las predicciones mediante bubble sort para escribirlas en el panel correspondiente y se ejecuta la animación de respuesta del [NPC](#). Las animaciones de respuesta que tiene a los gestos se puede ver en la tabla 3.2

Gesto	Animación de reacción
Bailar	Aplaudir
Saludar	Saludar
Señalar	Mirar
Sentarse	Sentarse
Pelear	Pose defensiva
Correr	Correr

Tabla 3.2: Reacción del [NPC](#) al gesto predicho del jugador

Si hay un error de conexión en mitad de la ejecución se deberá ingresar de nuevo la IP de la máquina host en el campo de texto para reintentarlo.

En la figura 3.37 se puede ver el diagrama de flujo de esta aplicación.

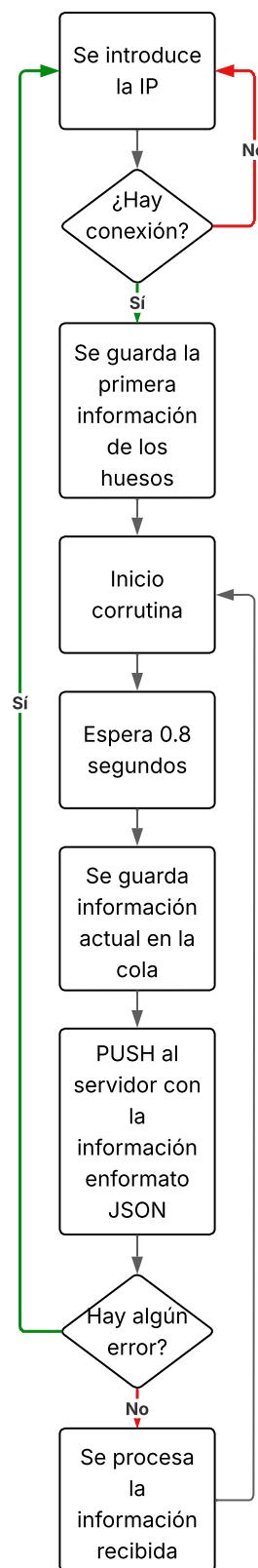


Figura 3.37: Diagrama de flujo de la aplicación de demostración

Capítulo 4

Conclusiones y Trabajo Futuro

4.1. Conclusiones

El objetivo de este estudio era la creación de un modelo de IA que pudiera detectar una serie de gestos de usuarios con un traje de captura de movimiento. Para ello, se buscaron datasets de gestos que concordaran con las restricciones del trabajo. Al no encontrar nada fácilmente adaptable o con los gestos necesitados, se creó un dataset de animaciones, con datos tanto generados por ordenador como recopilados de usuarios reales.

Las animaciones generadas por ordenador se convirtieron a CSV por una herramienta creada por los autores del trabajo, al igual que las recolectadas por usuarios reales se hicieron con otra herramienta de los mismos creadores.

Tras los entrenamientos de los distintos modelos de IA, se llegó a la conclusión de que el Random Forest es el mejor modelo por diferentes razones. Tras esto, se creó una aplicación a modo de demostración para mostrar el resultado final.

Las conclusiones de cada una de las partes del trabajo se presentan en las siguientes secciones de manera más detalladas.

4.1.1. Conclusiones de la búsqueda de datasets

Tras la búsqueda de datasets de animaciones se ha llegado a la conclusión de que no existen datasets públicos suficientes para el objetivo de este estudio.

Estas conclusiones nos llevaron a la necesidad de crear un dataset propio recaudando animaciones de bancos de animaciones y mediante pruebas con usuarios.

4.1.2. Conclusiones de la extracción de animaciones de bancos de animaciones

Para la conversión de animaciones de Mixamo se ha creado una herramienta para hacerlo de forma automática y se ha modificado otra herramienta para hacer una descarga en masa de estas animaciones.

La herramienta de conversión de gestos cumple su objetivo completamente, ya

que tan solo indicando el nombre de la carpeta que contiene los gestos descargados coge los gestos, se procesan en Unity y se suben a Kaggle de forma automática. Además la herramienta es independiente al número de gestos que se quieren convertir y a su vez del número de animaciones que hay de esos gestos, consiguiendo una gran escabilidad de esta forma haciendo posible la introducción de nuevos gestos sin tener que cambiar nada de la herramienta.

4.1.3. Conclusiones de la recolección de animaciones con usuarios

Para la recolección de datos con usuarios se anunció de forma insistente en las redes sociales de los autores del estudio así como por las distintas facultades de la Universidad Complutense de Madrid. Los anuncios cumplieron su objetivo, consiguiendo 75 respuestas de los cuales se presentaron 65 participantes, consiguiendo un 86,67 % de asistencia.

Como se puede ver en las figuras del apéndice D el grupo de participantes, con excepción del género no binario, es heterogéneo en cuanto al género con una pequeña diferencia de porcentaje entre hombres y mujeres. Donde se puede ver mayor diferencia es entre la mano dominante de los usuarios, siendo los zurdos tan solo un 6.15 % de la población total del grupo.

Gracias a la prueba se obtuvieron 975 animaciones (195 animaciones de saludar, señalar, sentarse, pelear y correr).

4.1.4. Conclusiones de la comparativa entre los modelos de IA

En cuanto a los modelos de IA, se ha visto que los modelos de redes neuronales no han obtenido los resultados esperados, siendo esto seguramente por una mezcla de falta de datos y simplificación de las redes neuronales para asegurar un tiempo de entrenamiento menor y un computo más rápido a la hora de realizar inferencias.

Sin embargo, el Random Forest ha sido el que ha obtenido mejores resultados, siendo el modelo más rápido (tanto en inferencia como en entrenamiento), el modelo más ligero y el modelo más adaptable. Este modelo se podría usar en sistemas con recursos limitados como pueden ser gafas de VR independientes como las Meta Quest o en ordenadores más potentes, e incluso en ecosistemas de IA por su posible traducción a TensorFlow ya implementada.

En el análisis de los resultados del Random Forest se ha visto que el modelo le da una gran importancia a la coordenada y de la pierna izquierda, a la coordenada y del pie izquierdo en el segundo intervalo de tiempo. Esto puede indicar que los gestos dependen mucho de la posición de las piernas para discriminar si se está corriendo, bailando, saludando o señalando.

También se puede ver una ligera confusión entre los gestos de pelea y saludar. Esto puede ser debido a los movimientos rápidos de las manos en ambos tipos de gesto, que al no tener dedos por limitación del traje, puede dar lugar a confusión entre un puñetazo y un saludo.

4.1.5. Conclusiones de la aplicación final

La aplicación final es una demo que se conecta a un servidor para mostrar los resultados.

La parte negativa de esta aplicación es la forma de conectarse con el servidor en el que está el modelo, ya que hace peticiones web al servidor de Tensor Serving directamente, lo que lo hace dependiente de éste.

Finalmente la aplicación final cumple su propósito de enseñar al usuario la animación predicha con una latencia mínima.

4.1.6. Limitaciones

Este estudio ha tenido una serie de limitaciones, desde escasez de datasets públicos, desbalanceo de tipos de animaciones, falta de más datos de usuarios reales, problemas de entrenamiento con los modelos de redes neuronales y problemas de falta de datos para estos mismos modelos.

La limitación de escasez de datasets se debe no solo a la falta en número de datasets, sino también a la falta de gestos referentes a comunicación no verbal, habiendo algunos sobre deportes o temáticas muy concretas y pocos de carácter general. Otra limitación de estos datasets ha sido la facilidad de adecuación o conversión a un formato compatible con el traje de captura de movimiento usado en el estudio.

En cuanto al desbalanceo de los tipos de animaciones, la problemática viene dada por la duración de las animaciones de baile de Mixamo. Al seguir el proceso de estandarizado, las distintas animaciones de baile se multiplican en un número bastante mayor que los del resto de tipos. Esto no se solucionó con los datos recopilados de usuarios reales, ya que aún que se obtuvo un gran número de animaciones nuevas, tras la estandarización las de baile seguían prevaleciendo por bastante. Esto se puede ver en las figuras 3.15 y 3.16.

Los modelos de redes neuronales no han dado los resultados esperados, esto puede ser por la falta de datos, que aunque haya un número que a simple vista parece suficiente no lo es para entrenar redes neuronales, por las limitaciones de hardware para entrenar modelos más complejos y por los tiempos requeridos para la realización de este estudio no ser lo suficientemente largos como para llevar a cabo entrenamientos más extensos y con mayor búsqueda de hiperparámetros.

Todas estas limitaciones, han llevado al siguiente plan de trabajo a futuro.

4.2. Trabajo Futuro

El trabajo a futuro de este estudio se puede centrar en varios esfuerzos:

- Mejorar el dataset, incluyendo gente con diversidad funcional, aumentando el número de muestras por gesto, teniendo en cuenta los dedos de las manos y añadiendo nuevas categorías. Esto no solo puede llegar a mejorar el rendimiento de modelos de redes neuronales, sino que puede dotar de mayores diferencias a los modelos para poder generalizar mejor.

- Investigar otros modelos de IA que puedan mejorar la precisión de la clasificación y hacer clasificación no solo de gestos, sino también de emociones. Esto podría ayudar a que los NPCs puedan reaccionar de una manera más natural a los gestos del usuario, haciendo la interacción más inmersiva.
- Estandarizar el servidor para que pueda ser usado con otras tecnologías de IA y no solo con TensorFlow. Esto permitiría integrar tecnologías que vayan surgiendo en el futuro a aplicaciones ya existentes.
- Implementar otro modelo que no solo tenga en cuenta el gesto predicho actual sino también todo el contexto para poder avanzar en el uso de la comunicación no verbal con NPCs.
- Evaluación del resultado final con usuarios.

Introduction

In recent years, the advancement of immersive technologies such as VR has brought about a significant change in how users interact in digital environments. Major companies like Meta and Apple have invested in the development of hardware for these technologies, launching devices like the Meta Quest 3 and Apple Vision Pro, and creating social interaction platforms in the metaverse with the development of Meta Horizon Worlds. This technology has opened new opportunities to explore more natural modes of communication in virtual spaces through non-verbal communication.

Non-verbal communication refers to the transmission of messages without the use of words, relying instead on images and gestures. In our daily lives, it plays a crucial role in human interactions, yet its representation in virtual environments is often limited and dependent on predefined actions such as animations or "emotes" already integrated into the game. Therefore, the development of tools that can capture and interpret these gestures in real-time represents a significant step towards creating more expressive and realistic virtual environments.

4.3. Motivation

As mentioned in the introduction, the use of non-verbal communication in virtual environments is limited and predefined by their design, restricting a fundamental part of everyday human expression. Due to this, and as seen in articles like [Neverova et al. \(2013\)](#) or [Herbert et al. \(2024\)](#), gesture detection is a field of study currently being explored to address this issue. For these reasons, it is interesting to investigate ways to expand this representation of non-verbal communication so that interaction between users and [NPCs](#) becomes more natural.

This work is therefore part of the research line focused on developing tools that can recognize gestures performed by a user using a motion capture suit, with the aim of creating more immersive and natural virtual environments for users. On the other hand, a large amount of data is required to create AI models that enable the development of the mentioned tools. In the case of this project, the necessary data consists of a sequence of 3D coordinates representing the positions and rotations of the skeleton's bones throughout the animations, resulting in relatively large data sizes. This necessitates the creation of complex models capable of processing a

significant amount of input data. However, as we will discuss throughout the work, it has not been possible to find datasets containing this type of data. As a result, a significant contribution of this work to the field has been the creation of a dataset through the extraction of animations from a specialized bank and the motion capture of users.

Given these reasons, it can be concluded that the lack of natural methods to leverage non-verbal communication in virtual environments and the scarcity of animation datasets have motivated the objectives mentioned, which we will expand upon below.

4.4. Objectives

The general objective of this project is to implement an [AI](#) model that, with low latency, allows real-time identification of gestures performed with a motion capture suit, aiming to enhance non-verbal communication in virtual environments. To achieve this objective, the following goals have been proposed during the development of the work:

1. Search for animation datasets that are considered relevant for communicating with an [NPC](#) in a video game. These animations include dancing, greeting, pointing, sitting, fighting, and running.
2. Automatic extraction of the aforementioned animations from animation banks and subsequent processing.
3. Extraction of animations through motion capture from a heterogeneous group of users.
4. Implementation, training, and comparison of different [AI](#) models, including neural networks (LSTM, CNN, and RNN) as well as classical classification models (Random Forest).
5. Development of a final application as a demonstration to showcase the results in [VR](#) to enhance immersion.

As a significant secondary objective, due to the lack of animation datasets, it has been proposed to publish the resulting datasets, both from artificial animations and user captures.

4.5. Work Plan

The work plan consists of several steps:

1. Search for a dataset: generate a sufficiently large dataset with multiple examples of gestures to adequately train different models. This dataset should primarily consist of data from real users to identify the most natural gestures possible.

2. Implementation of AI models: implement various AI models to compare them and determine which one is most suitable based on prediction speed and accuracy. The implementation of these models should include: training, a way to extract final model statistics, hyperparameter search, confusion matrix generator, and compatibility with a common user interface for all trainings.
3. Development of a user interface for training and monitoring different models: develop a web interface so that users with less programming and AI experience can train models capable of predicting the gestures they need without modifying the code or needing to understand the internal workings of the codebase.
4. Development of a final application: create an application for Oculus Quest as a demo that connects to the chosen model and allows real-time usage demonstration.

The development of tasks and their planning can be seen in Figure 4.1.

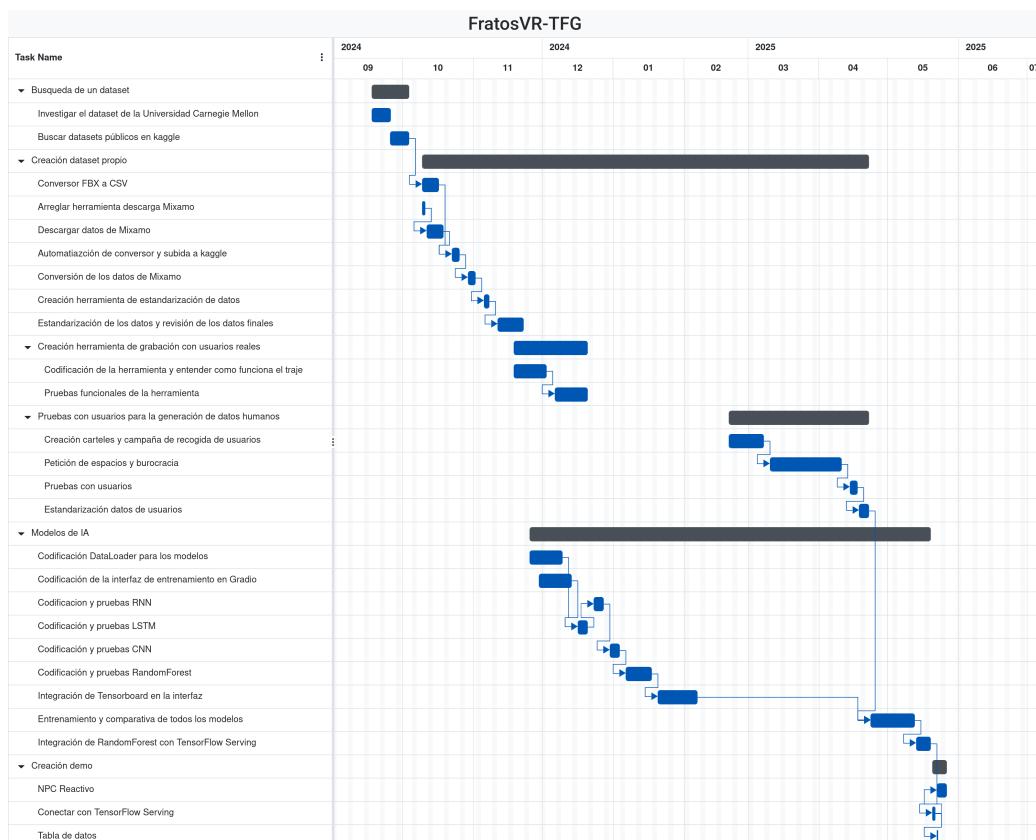


Figure 4.1: Gantt chart of the project

Conclusions and Future Work

4.6. Conclusions

The objective of this study was to create an [AI](#) model capable of detecting a series of gestures from users wearing a motion capture suit. To achieve this, datasets of gestures that matched the study's constraints were sought. Since no easily adaptable datasets with the required gestures were found, an animation dataset was created, consisting of both computer-generated data and data collected from real users. The computer-generated animations were converted to [CSV](#) format using a tool created by the authors of the study, while the animations collected from real users were processed with another tool developed by the same authors. After training various [AI](#) models, it was concluded that the [Random Forest](#) model is the best choice for several reasons. Following this, a demonstration application was created to showcase the final results.

The conclusions of each part of the work are presented in the following sections in more detail.

4.6.1. Dataset Search Conclusions

After the search for animation datasets, it was concluded that there are not enough public datasets available for the objective of this study.

These conclusions led us to the need to create our own dataset by collecting animations from animation banks and through user trials.

4.6.2. Conclusions of the Animation Extraction Tool

The gesture conversion tool fully meets its objective, as it only requires specifying the name of the folder containing the downloaded gestures. It automatically processes the gestures in Unity and uploads them to Kaggle. Additionally, the tool is independent of the number of gestures to be converted and the number of animations available for those gestures, achieving great scalability. This allows for the introduction of new gestures without requiring any changes to the tool.

4.6.3. Conclusions of the User Data Collection

As can be seen in the figures in Appendix D, the participant group except for the non-binary gender category is heterogeneous in terms of gender, with only a small percentage difference between men and women. The most notable difference can be seen in users' dominant hand, with left-handed individuals making up only 6.15% of the groups total population.

Thanks to the test, a total of 975 animations were obtained (195 animations each for waving, pointing, sitting, fighting, and running).

4.6.4. Conclusions of the IA Models Comparison

In terms of the AI models, it has been observed that neural network models did not achieve the expected results, likely due to a combination of insufficient data and simplification of the neural networks to ensure shorter training times and faster inference computations.

However, the Random Forest model has yielded the best results, being the fastest model (both in inference and training), the lightest model, and the most adaptable. This model can be used in systems with limited resources, such as standalone VR glasses like Meta Quest or more powerful computers, and even in AI ecosystems due to its possible translation to TensorFlow already implemented.

In the analysis of the Random Forest results, it has been observed that the model places significant importance on the y-coordinate of the left leg and the y-coordinate of the left foot in the second time interval. This may indicate that gestures heavily depend on leg positions to distinguish between running, dancing, waving, or pointing.

Also, there is a slight confusion between the gestures of fighting and waving. This may be due to the rapid hand movements in both types of gestures, which, due to the lack of fingers in the suit, can lead to confusion between a punch and a wave.

4.6.5. Final Application Conclusions

The final application is a demo that connects to a server to display the results.

The negative aspect of this application is the way it connects to the server where the model is hosted, as it makes direct web requests to the Tensor Serving server, making it dependent on that server.

Finally, the final application fulfills its purpose of showing the user the predicted animation with minimal latency.

4.6.6. Limitations

This study has faced a series of limitations, including the scarcity of public datasets, imbalance in types of animations, lack of more data from real users, training issues with neural network models, and data shortages for those same models.

The limitation of dataset scarcity is not only due to the lack of datasets in number but also to the lack of gestures related to non-verbal communication. Most

available datasets focus on sports or very specific themes, with few general-purpose datasets. Another limitation of these datasets has been the ease of adaptation or conversion to a format compatible with the motion capture suit used in the study.

Regarding the imbalance of animation types, the issue arises from the duration of Mixamo’s dance animations. Following the standardization process, the various dance animations are multiplied by a significantly larger number than those of other types. This was not resolved with the data collected from real users, as even though a large number of new animations were obtained, after standardization, dance animations still predominated significantly. This can be seen in Figures 3.15 and 3.16.

The neural network models did not yield the expected results, which may be due to the lack of data. Although there appears to be a sufficient number of samples at first glance, it is not enough for training neural networks. Additionally, hardware limitations for training more complex models and the time constraints of this study did not allow for longer training sessions or more extensive hyperparameter searches.

All this limitations have led to the following future work plan.

4.7. Future Work

The future work of this study can focus on several efforts:

- Improve the dataset by including people with functional diversity, increasing the number of samples per gesture, considering finger movements, and adding new categories. This could not only enhance the performance of neural network models but also provide greater differentiation for better generalization.
- Investigate other [AI](#) models that could improve classification accuracy and enable classification not only of gestures but also of emotions. This could help [NPCs](#) react more naturally to user gestures, making interactions more immersive.
- Standardize the server to be compatible with other [AI](#) technologies beyond TensorFlow. This would allow for the integration of emerging technologies into existing applications.
- Implement another model that considers not only the current predicted gesture but also the entire context to advance the use of non-verbal communication with [NPCs](#).
- Evaluation of the final result with users.

Contribuciones Personales

Alejandro Barrachina Argudo

Alejandro Barrachina, estudiante del Grado de Ingeniería Informática, se ha involucrado en el desarrollo del estudio desde el momento en que se le propuso la idea. Desde el principio, ha estado en comunicación constante con su compañero y sus tutores para comunicar los avances y dudas del proyecto. A lo largo del proyecto, estas son las tareas que ha desempeñado:

En un primer momento, se dedicó a estudiar el funcionamiento de Unity en gafas de VR y su posible integración con el traje de captura de movimiento y los modelos de IA pensados. Esto implicó aprender sobre Unity, C#, y servidores en Unity.

Una vez tuvo claro el comportamiento y funcionamiento de Unity, junto a su compañero, se dedicó a buscar posibles datasets para entrenar los modelos de IA. Tras varias búsquedas, cuando se optó por crear un dataset a mano, se dedicó a buscar herramientas para automatizar el proceso.

Al encontrar la herramienta Mixamo Downloader, la analizó hasta encontrar los errores de funcionamiento de la misma. Tras arreglarlos y añadir nuevas funcionalidades, se dedicó a crear la primera versión del dataset con los datos descargados, decidiendo la estructura de carpetas y su método de guardado.

Para este método de guardado, se decidió usar Kaggle para evitar problemas con Github y su límite de tamaño por archivo. Alejandro se ha encargado también de la estructura de repositorios y herramientas en la organización creada para este TFG, *FratosVR*, para así facilitar el desarrollo e interoperabilidad de las distintas herramientas con un desarrollo paralelo.

Tras investigar distintos estudios de IA para la detección de gestos, Alejandro decide hacer una comparativa entre los cuatro modelos expuestos en el estudio, reservándose otros por si hubiera más tiempo o recursos.

Mientras Pablo Sánchez desarrollaba la herramienta de conversión de datos, Alejandro investigó como hacer un *pipeline* de descarga del dataset, conversión al formato deseado usando Unity y su posterior actualización en el nuevo conjunto de datos de Kaggle. Para ello tuvo que investigar las funcionalidades de línea de comando de Unity, así como el funcionamiento del editor de Unity y sus funcionalidades expuestas en su API de C#.

Una vez creados estos datos, Alejandro discutió con sus tutores cual sería la

mejor manera de estandarizar los datos para su uso en los entrenamientos de los modelos. Cuando se llegó a un formato concreto, Alejandro se dedicó a programar la clase DataLoader para procesar todos los datos, estandarizarlos y actualizarlos en la nube para su posterior uso.

Tras esto, Alejandro modeló los cuatro modelos a probar y comparar. Esto implicó el uso de Gradio y TensorBoard para facilitar el entrenamiento para usuarios ajenos al proyecto. Esta idea se desarrolló por la ayuda cedida por la asociación LAG, que consistió en dejarnos equipos con gráficas más potentes que las que teníamos disponibles para así acelerar el entrenamiento de los modelos.

Mientras que Pablo se dedicaba a la adaptación de la herramienta para su uso con personas reales, Alejandro hizo dos formularios para las pruebas con usuarios. Uno de ellos para que las personas interesadas dejaran su correo y las fechas que tenían disponibles, y otro para la posterior recogida de datos demográficos. Estos formularios se pasaron a María del Carmen Fernández Villalba, Responsable de Protección de datos en Vicepresidencia Primera, perteneciente a Presidencia de La Junta de Comunidades de Castilla La Mancha, para su revisión y aprobación. Esto se hizo para asegurar que ninguno de los formularios incumplía la ley de protección de datos y así asegurar el anonimato de los datos recogidos y de los usuarios participantes. Tras sus sugerencias, se realizaron los cambios pertinentes y se abrieron para respuesta.

De manera adicional, empezó a plantear los diferentes modelos a entrenar y su representación gráfica en la web de entrenamiento. Para ello, investigó YDF, ya que era una herramienta nueva que no había usado, y desarrolló los modelos con TensorFlow de manera que fueran todos iguales de cara a la interfaz. Para ello, tuvo que investigar también keras-tuner para hacer que los entrenamientos fueran lo más eficientes posibles teniendo en cuenta las restricciones de hardware.

Una vez Pablo adaptó la herramienta para las pruebas con usuarios, Alejandro creó parte de los carteles para la difusión de las pruebas. Junto a Pablo hicieron una ruta por todas las facultades de la UCM de Ciudad Universitaria para colgarlos. También inició una campaña por sus redes sociales junto a las asociaciones de estudiantes de la facultad de informática para atraer a más gente.

Previo a las pruebas con usuarios, Alejandro junto a su compañero habló con los tutores y con distinto personal de la facultad para conseguir un espacio reservado el mayor tiempo posible para realizar dichas pruebas.

Durante las pruebas, tanto Alejandro como Pablo estuvieron presentes en todas para ayudar a los usuarios a ponerse el traje, realizar el calibrado del mismo y recoger las preguntas del cuestionario. Durante este proceso de pruebas, tanto Alejandro como Pablo se encargaron de atraer usuarios de prueba nuevos en caso de que fallaran los que tenían prueba asignada, resultando en que solo 5 plazas no fueran completadas. También se encargaron de gestionar horarios para maximizar el número de pruebas realizadas y gestionar la espera de los usuarios que se solaparan.

Tras esto, Alejandro se encargó de los entrenamientos de los modelos y los problemas que surgieron durante el proceso (Falta de VRAM, problemas de ingestión de datos, etc). También se encargó de desplegar los mecanismos de entrenamiento en distintos sistemas para no ocasionar inconvenientes a los usuarios de la asociación. Esto consistió en montar [WSL](#) en los ordenadores de la asociación para compatibilizar

lizarlos con las últimas versiones de TensorFlow.

Una vez se acabaron los entrenamientos y se decidió el mejor modelo, Alejandro investigó una manera de hacer que estos modelos fueran multiplataforma y pudieran ser usados en cualquier dispositivo. Esto resultó en el descubrimiento e investigación de TensorFlow Serving. Alejandro posteriormente hizo scripts para Windows y Linux para facilitar el despliegado del modelo en docker usando las *releases* de Github.

Tras finalizar todo esto, Alejandro se dedicó a la redacción de este documento junto a su compañero, garantizando así cohesión durante todo el texto y ayuda con L^AT_EXen los momentos necesarios. También se dedicó a hacer las gráficas explicativas de los modelos y los datos demográficos para asegurar un entendimiento más sencillo de todos los resultados.

Pablo Sánchez Martín

Pablo Sánchez Martín, estudiante del Grado de Desarrollo de Videojuegos, se ha involucrado en el proyecto desde el momento en el que salió la idea manteniendo una comunicación constante con su compañero y tutores.

Desde el año 2023 mantuvo contacto con uno de los tutores, Alejandro Romero, para hacer un proyecto con un traje de captura de movimiento. Tras varias reuniones debatiendo lo que podría ser un trabajo interesante los dos decidieron el estudio actual. Para ello Pablo contactó al otro tutor del proyecto, Ismael Sagredo, e involucró a su compañero Alejandro Barrachina.

Durante el desarrollo estas son las tareas que ha realizado:

La primera tarea que tuvo que realizar junto a su compañero Alejandro era establecer qué gestos se querían detectar, qué modelos de IA se iban a utilizar y si se iba a detectar el movimiento de los dedos mediante las gafas de VR (handtracking). Finalmente los dos acordaron los cinco gestos en los que se basa el proyecto, los cuatro modelos con los que se hace la comparación y la decisión de no tener el handtracking por posibles problemas a la hora de no estar viendo las manos en todo momento.

Lo siguiente que hizo fue investigar el funcionamiento del traje de captura de movimiento junto al software necesario, tanto Axis Studio como el plugin de Unity. Para ello estuvo investigando las distintas funcionalidades dentro de un proyecto de Axis Studio que podría interesar para el estudio y se descargó un proyecto cedido por el tutor Alejandro Romero en el que se usaba el plugin para poder analizarlo y saber cómo usarlo.

Una vez entendió el funcionamiento del software necesario para el funcionamiento del traje empezó a buscar distintos datasets de animaciones para entrenar los modelos junto a su compañero. Al ver la escasez de estos datasets se optó por sacarlos de un banco de animaciones, por lo que empezó el desarrollo de la herramienta Mixamo Dumper.

En un primer momento investigó sobre el uso de los Assets Bundles de Unity para la carga automática de recursos externos a la aplicación, pero al final decidió no usar ese método. Finalmente para esa aplicación decidió usar el Asset Database de Unity, por lo que se dedicó a la investigación de esta API y finalmente a la

implementación de la herramienta. Una vez funcionaba la implementación de la carga de animaciones en tiempo de ejecución investigó como poder ejecutar todas estas animaciones seguidas en tiempo de ejecución, hasta que finalmente encontró el componente “Animator Override Controller”, lo que le permitió cumplir el objetivo de la herramienta.

Debido a los pocos datos encontrados en Mixamo se decidió grabar a usuarios con el traje de captura de movimiento, por lo que hizo en Unity una herramienta sencilla para crear [CSVs](#) en tiempo real con el traje de captura de movimiento.

Para las pruebas con usuarios se encargó de la comunicación con sus tutores y personal de la Facultad de Informática para conseguir un lugar en el que poder grabar. Con el fin de capar usuarios junto a su compañero hizo carteles y ayudó en la campaña en redes sociales y difusión por el resto de facultades de Ciudad Universitaria.

En la prueba explicó junto a su compañero a todos los usuarios el propósito de las pruebas, les ayudó a ponerse el traje y a explicarles el proceso de calibración. También le fue dando indicaciones a los usuarios sobre los gestos a realizar, grabó las animaciones y verificó que no hubiese ningún problema con estas. Adicionalmente se encargó junto a su compañero de buscar nuevos usuarios para cubrir huecos que habían quedado libres con el fin de maximizar el número de datos recogidos y gestionar a las personas que solapaban con otros usuarios.

Una vez recogidos los datos investigó diferentes formas de conectarse con el servidor que hostee el modelo elegido. Investigó el uso de “Unity NetCode”, un paquete propio de Unity para la comunicación en red en videojuegos y también investigó la posibilidad de realizar la comunicación mediante las llamadas nativas de C#. Finalmente se decidió usar la arquitectura [API REST](#), por lo que investigó su funcionamiento mediante el paquete nativo “UnityWebRequest” y empezó la aplicación final con la implementación de ésta en Unity.

Una vez estaba hecha la comunicación con el servidor diseñó e implementó una aplicación final con soporte para las gafas de [VR](#) y el traje de captura de movimiento en el que se conectase con el servidor y se pueda ver el resultado del modelo elegido en tiempo real y como un [NPC](#) reacciona a los gestos predichos por éste.

Finalmente ha contribuído a la redacción de este documento junto a su compañero, así como la constante comunicación con los tutores para comprobar que el formato de éste era adecuado y a la realización de distintos diagramas para que las herramientas creadas durante el proceso del proyecto sean más sencillas de entender.

Bibliografía

The sciences, each straining in its own direction, have hitherto harmed us little; but some day the piecing together of dissociated knowledge will open up such terrifying vistas of reality, and of our frightful position therein, that we shall either go mad from the revelation or flee from the deadly light into the peace and safety of a new dark age.

H.P. Lovecraft, *The Call of Cthulhu*

ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I., HARP, A., IRVING, G., ISARD, M., JIA, Y., JOZEFOWICZ, R., KAISER, L., KUDLUR, M., LEVENBERG, J., MANÉ, D., MONGA, R., MOORE, S., MURRAY, D., OLAH, C., SCHUSTER, M., SHLENS, J., STEINER, B., SUTSKEVER, I., TALWAR, K., TUCKER, P., VANHOUCKE, V., VASUDEVAN, V., VIÉGAS, F., VINYALS, O., WARDEN, P., WATTENBERG, M., WICKE, M., YU, Y. y ZHENG, X. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. Software available from tensorflow.org.

ABID, A., ABDALLA, A., ABID, A., KHAN, D., ALFOZAN, A. y ZOU, J. Gradio: Hassle-free sharing and testing of ML models in the wild. 2019.

ANSEL, J., YANG, E., HE, H., GIMELSHEIN, N., JAIN, A., VOZNESENSKY, M., BAO, B., BELL, P., BERARD, D., BUROVSKI, E., CHAUHAN, G., CHOUDRIA, A., CONSTABLE, W., DESMAISON, A., DEVITO, Z., ELLISON, E., FENG, W., GONG, J., GSCHWIND, M., HIRSH, B., HUANG, S., KALAMBARKAR, K., KIRSCH, L., LAZOS, M., LEZCANO, M., LIANG, Y., LIANG, J., LU, Y., LUK, C., MAHER, B., PAN, Y., PUHRSCH, C., RESO, M., SAROUFIM, M., SIRACHI, M. Y., SUK, H., SUO, M., TILLET, P., WANG, E., WANG, X., WEN, W., ZHANG, S., ZHAO, X., ZHOU, K., ZOU, R., MATHEWS, A., CHANAN, G., WU, P. y CHINTALA, S. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. En *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, 2024.

- BARRACHINA ARGUDO, A. y SÁNCHEZ MARTÍN, P. Csv-pose-animations. <https://www.kaggle.com/dsv/11405888>, 2025a.
- BARRACHINA ARGUDO, A. y SÁNCHEZ MARTÍN, P. Raw mixamo animations. <https://www.kaggle.com/dsv/10687872>, 2025b.
- CARNEGIE MELLON, U. Cmu graphics lab motion capture database. <https://mocap.cs.cmu.edu/info.php>, 2003. Accessed: (04/05/2025).
- ELLIS, S. R. Nature and origins of virtual environments: a bibliographical essay. *Computing Systems in Engineering*, vol. 2, páginas 321–347, 1991.
- GUERRA-FILHO, G. Optical motion capture: Theory and implementation. *Rita*, vol. 12(2), páginas 61–90, 2005.
- GUILLAME-BERT, M., BRUCH, S., STOTZ, R. y PFEIFER, J. Yggdrasil decision forests: A fast and extensible decision forests library. En *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, páginas 4068–4077. 2023.
- HARRIS, C. R., MILLMAN, K. J., VAN DER WALT, S. J., GOMMERS, R., VIRTANEN, P., COURNAPEAU, D., WIESER, E., TAYLOR, J., BERG, S., SMITH, N. J., KERN, R., PICUS, M., HOYER, S., VAN KERKWIJK, M. H., BRETT, M., HALDANE, A., DEL RÍO, J. F., WIEBE, M., PETERSON, P., GÉRARD-MARCHANT, P., SHEPPARD, K., REDDY, T., WECKESSER, W., ABBASI, H., GOHLKE, C. y OLIPHANT, T. E. Array programming with NumPy. *Nature*, vol. 585(7825), páginas 357–362, 2020.
- HERBERT, O. M., PÉREZ-GRANADOS, D., RUIZ, M. A. O., CADENA MARTÍNEZ, R., GUTIÉRREZ, C. A. G. y ANTUÑANO, M. A. Z. Static and dynamic hand gestures: A review of techniques of virtual reality manipulation. *Sensors*, vol. 24(12), página 3760, 2024.
- HOLFELD, J. On the relevance of the godot engine in the indie game development industry. <https://arxiv.org/abs/2401.01909>, 2024.
- JOCHER, G., QIU, J. y CHAURASIA, A. Ultralytics YOLO. 2023.
- LI, L., JAMIESON, K., DESALVO, G., ROSTAMIZADEH, A. y TALWALKAR, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. 2018.
- MA, X., ZENG, B. y XING, Y. Combining 3d skeleton data and deep convolutional neural network for balance assessment during walking. *Frontiers in Bioengineering and Biotechnology*, vol. 11, 2023.
- WES MCKINNEY. Data Structures for Statistical Computing in Python. En *Proceedings of the 9th Python in Science Conference* (editado por Stéfan van der Walt y Jarrod Millman), páginas 56 – 61. 2010.
- NEVEROVA, N., WOLF, C., PACI, G., SOMMAVILLA, G., TAYLOR, G. W. y NEBOUT, F. A multi-scale approach to gesture detection and recognition. 2013.

- OLSTON, C., FIEDEL, N., GOROVOY, K., HARMSEN, J., LAO, L., LI, F., RASHEKHAR, V., RAMESH, S. y SOYKE, J. Tensorflow-serving: Flexible, high-performance ml serving. <https://arxiv.org/abs/1712.06139>, 2017.
- O'MALLEY, T., BURSZTEIN, E., LONG, J., CHOLLET, F., JIN, H., INVERNIZZI, L. ET AL. Keras Tuner. <https://github.com/keras-team/keras-tuner>, 2019.
- PALLAVICINI, F., PEPE, A. y MINISSI, M. E. Gaming in virtual reality: What changes in terms of usability, emotional response and sense of presence compared to non-immersive video games? *SIMULATION AND GAMING*, vol. 50, páginas 136–159, 2019.
- PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M. y DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, vol. 12, páginas 2825–2830, 2011.
- PETROCHUK, M. Hparams: Hyperparameter management solution. <https://github.com/PetrochukM/HParams>, 2019.
- RAAB, F. H., BLOOD, E. B., STEINER, T. O. y JONES, H. R. Magnetic position and orientation tracking system. *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-15(5), páginas 709–718, 1979.
- ROETENBERG, D., LUINGE, H., SLYCKE, P. ET AL. Xsens mvn: Full 6dof human motion tracking using miniature inertial sensors. *Xsens Motion Technologies BV, Tech. Rep*, vol. 1(2009), páginas 1–7, 2009.
- PANDAS DEVELOPMENT TEAM, T. pandas-dev/pandas: Pandas. 2020.
- VAN BRAKEL, V., BARREDA-ÁNGELES, M. y HARTMANN, T. Feelings of presence and perceived social support in social virtual reality platforms. *Computers in Human Behavior*, vol. 139, 2023.
- VRCHAT, I. Vr chat full-body tracking. <https://docs.vrchat.com/docs/full-body-tracking>, 2024. Accessed: (03/05/2025).
- WILKINSON, M., BRANTLEY, S. y FENG, J. A mini review of presence and immersion in virtual reality. *Human Factors and Ergonomics Society*, 2021.
- XENAKIS, I., GAVALAS, D., KASAPAKIS, V., DZARDANOVA, E. y VOSINAKIS, S. Non-verbal communication in immersive virtual reality through the lens of presence: A critical review. *PRESENCE: Virtual and Augmented Reality*, vol. 31, páginas 1–71, 2023.
- ZHANG, Z. Microsoft kinect sensor and its effect. *IEEE MultiMedia*, vol. 19(2), páginas 4–10, 2012.

Apndice A

Cabecera de los CSVs del dataset

En esta apéndice se presenta la cabecera completa de los CSVs del dataset.

Tabla A.1: Cabecera del CSV de cada animación, en orden descendente y de izquierda a derecha (completa)

Robot_Hips_posx	Robot_Hips_posy	Robot_Hips_posz
Robot_Hips_rotx	Robot_Hips_roty	Robot_Hips_rotz
Robot_LeftUpLeg_posx	Robot_LeftUpLeg_posy	Robot_LeftUpLeg_posz
Robot_LeftUpLeg_rotx	Robot_LeftUpLeg_roty	Robot_LeftUpLeg_rotz
Robot_LeftLeg_posx	Robot_LeftLeg_posy	Robot_LeftLeg_posz
Robot_LeftLeg_rotx	Robot_LeftLeg_roty	Robot_LeftLeg_rotz
Robot_LeftFoot_posx	Robot_LeftFoot_posy	Robot_LeftFoot_posz
Robot_LeftFoot_rotx	Robot_LeftFoot_roty	Robot_LeftFoot_rotz
Robot_RightUpLeg_posx	Robot_RightUpLeg_posy	Robot_RightUpLeg_posz
Robot_RightUpLeg_rotx	Robot_RightUpLeg_roty	Robot_RightUpLeg_rotz
Robot_RightLeg_posx	Robot_RightLeg_posy	Robot_RightLeg_posz
Robot_RightLeg_rotx	Robot_RightLeg_roty	Robot_RightLeg_rotz
Robot_RightFoot_posx	Robot_RightFoot_posy	Robot_RightFoot_posz
Robot_RightFoot_rotx	Robot_RightFoot_roty	Robot_RightFoot_rotz
Robot_Spine_posx	Robot_Spine_posy	Robot_Spine_posz
Robot_Spine_rotx	Robot_Spine_roty	Robot_Spine_rotz
Robot_Spine1_posx	Robot_Spine1_posy	Robot_Spine1_posz
Robot_Spine1_rotx	Robot_Spine1_roty	Robot_Spine1_rotz
Robot_Spine2_posx	Robot_Spine2_posy	Robot_Spine2_posz
Robot_Spine2_rotx	Robot_Spine2_roty	Robot_Spine2_rotz

Continua en la siguiente página

Tabla A.1: Cabecera del CSV de cada animación, en órden descendente y de izquierda a derecha (completa) (Continuado)

Robot_Hips_posx	Robot_Hips_posy	Robot_Hips_posz
Robot_LeftShoulder_posx	Robot_LeftShoulder_posy	Robot_LeftShoulder_posz
Robot_LeftShoulder_rotx	Robot_LeftShoulder_roty	Robot_LeftShoulder_rotz
Robot_LeftArm_posx	Robot_LeftArm_posy	Robot_LeftArm_posz
Robot_LeftArm_rotx	Robot_LeftArm_roty	Robot_LeftArm_rotz
Robot_LeftForeArm_posx	Robot_LeftForeArm_posy	Robot_LeftForeArm_posz
Robot_LeftForeArm_rotx	Robot_LeftForeArm_roty	Robot_LeftForeArm_rotz
Robot_LeftHand_posx	Robot_LeftHand_posy	Robot_LeftHand_posz
Robot_LeftHand_rotx	Robot_LeftHand_roty	Robot_LeftHand_rotz
Robot_LeftHandIndex1_posx	Robot_LeftHandIndex1_posy	Robot_LeftHandIndex1_posz
Robot_LeftHandIndex1_rotx	Robot_LeftHandIndex1_roty	Robot_LeftHandIndex1_rotz
Robot_LeftHandIndex2_posx	Robot_LeftHandIndex2_posy	Robot_LeftHandIndex2_posz
Robot_LeftHandIndex2_rotx	Robot_LeftHandIndex2_roty	Robot_LeftHandIndex2_rotz
Robot_LeftHandIndex3_posx	Robot_LeftHandIndex3_posy	Robot_LeftHandIndex3_posz
Robot_LeftHandIndex3_rotx	Robot_LeftHandIndex3_roty	Robot_LeftHandIndex3_rotz
Robot_LeftHandMiddle1_posx	Robot_LeftHandMiddle1_posy	Robot_LeftHandMiddle1_posz
Robot_LeftHandMiddle1_rotx	Robot_LeftHandMiddle1_roty	Robot_LeftHandMiddle1_rotz
Robot_LeftHandMiddle2_posx	Robot_LeftHandMiddle2_posy	Robot_LeftHandMiddle2_posz
Robot_LeftHandMiddle2_rotx	Robot_LeftHandMiddle2_roty	Robot_LeftHandMiddle2_rotz
Robot_LeftHandMiddle3_posx	Robot_LeftHandMiddle3_posy	Robot_LeftHandMiddle3_posz
Robot_LeftHandMiddle3_rotx	Robot_LeftHandMiddle3_roty	Robot_LeftHandMiddle3_rotz
Robot_LeftHandPinky1_posx	Robot_LeftHandPinky1_posy	Robot_LeftHandPinky1_posz
Robot_LeftHandPinky1_rotx	Robot_LeftHandPinky1_roty	Robot_LeftHandPinky1_rotz
Robot_LeftHandPinky2_posx	Robot_LeftHandPinky2_posy	Robot_LeftHandPinky2_posz
Robot_LeftHandPinky2_rotx	Robot_LeftHandPinky2_roty	Robot_LeftHandPinky2_rotz
Robot_LeftHandPinky3_posx	Robot_LeftHandPinky3_posy	Robot_LeftHandPinky3_posz
Robot_LeftHandPinky3_rotx	Robot_LeftHandPinky3_roty	Robot_LeftHandPinky3_rotz
Robot_LeftHandRing1_posx	Robot_LeftHandRing1_posy	Robot_LeftHandRing1_posz
Robot_LeftHandRing1_rotx	Robot_LeftHandRing1_roty	Robot_LeftHandRing1_rotz
Robot_LeftHandRing2_posx	Robot_LeftHandRing2_posy	Robot_LeftHandRing2_posz
Robot_LeftHandRing2_rotx	Robot_LeftHandRing2_roty	Robot_LeftHandRing2_rotz
Robot_LeftHandRing3_posx	Robot_LeftHandRing3_posy	Robot_LeftHandRing3_posz
Robot_LeftHandRing3_rotx	Robot_LeftHandRing3_roty	Robot_LeftHandRing3_rotz

Continua en la siguiente página

Tabla A.1: Cabecera del CSV de cada animación, en órden descendente y de izquierda a derecha (completa) (Continuado)

Robot_Hips_posx	Robot_Hips_posy	Robot_Hips_posz
Robot_LeftHandThumb1_posx	Robot_LeftHandThumb1_posy	Robot_LeftHandThumb1_posz
Robot_LeftHandThumb1_rotx	Robot_LeftHandThumb1_roty	Robot_LeftHandThumb1_rotz
Robot_LeftHandThumb2_posx	Robot_LeftHandThumb2_posy	Robot_LeftHandThumb2_posz
Robot_LeftHandThumb2_rotx	Robot_LeftHandThumb2_roty	Robot_LeftHandThumb2_rotz
Robot_LeftHandThumb3_posx	Robot_LeftHandThumb3_posy	Robot_LeftHandThumb3_posz
Robot_LeftHandThumb3_rotx	Robot_LeftHandThumb3_roty	Robot_LeftHandThumb3_rotz
Robot_Neck_posx	Robot_Neck_posy	Robot_Neck_posz
Robot_Neck_rotx	Robot_Neck_roty	Robot_Neck_rotz
Robot_Head_posx	Robot_Head_posy	Robot_Head_posz
Robot_Head_rotx	Robot_Head_roty	Robot_Head_rotz
Robot_RightShoulder_posx	Robot_RightShoulder_posy	Robot_RightShoulder_posz
Robot_RightShoulder_rotx	Robot_RightShoulder_roty	Robot_RightShoulder_rotz
Robot_RightArm_posx	Robot_RightArm_posy	Robot_RightArm_posz
Robot_RightArm_rotx	Robot_RightArm_roty	Robot_RightArm_rotz
Robot_RightForeArm_posx	Robot_RightForeArm_posy	Robot_RightForeArm_posz
Robot_RightForeArm_rotx	Robot_RightForeArm_roty	Robot_RightForeArm_rotz
Robot_RightHand_posx	Robot_RightHand_posy	Robot_RightHand_posz
Robot_RightHand_rotx	Robot_RightHand_roty	Robot_RightHand_rotz
Robot_RightHandIndex1_posx	Robot_RightHandIndex1_posy	Robot_RightHandIndex1_posz
Robot_RightHandIndex1_rotx	Robot_RightHandIndex1_roty	Robot_RightHandIndex1_rotz
Robot_RightHandIndex2_posx	Robot_RightHandIndex2_posy	Robot_RightHandIndex2_posz
Robot_RightHandIndex2_rotx	Robot_RightHandIndex2_roty	Robot_RightHandIndex2_rotz
Robot_RightHandIndex3_posx	Robot_RightHandIndex3_posy	Robot_RightHandIndex3_posz
Robot_RightHandIndex3_rotx	Robot_RightHandIndex3_roty	Robot_RightHandIndex3_rotz
Robot_RightHandMiddle1_posx	Robot_RightHandMiddle1_posy	Robot_RightHandMiddle1_posz
Robot_RightHandMiddle1_rotx	Robot_RightHandMiddle1_roty	Robot_RightHandMiddle1_rotz
Robot_RightHandMiddle2_posx	Robot_RightHandMiddle2_posy	Robot_RightHandMiddle2_posz
Robot_RightHandMiddle2_rotx	Robot_RightHandMiddle2_roty	Robot_RightHandMiddle2_rotz
Robot_RightHandMiddle3_posx	Robot_RightHandMiddle3_posy	Robot_RightHandMiddle3_posz
Robot_RightHandMiddle3_rotx	Robot_RightHandMiddle3_roty	Robot_RightHandMiddle3_rotz
Robot_RightHandPinky1_posx	Robot_RightHandPinky1_posy	Robot_RightHandPinky1_posz
Robot_RightHandPinky1_rotx	Robot_RightHandPinky1_roty	Robot_RightHandPinky1_rotz

Continua en la siguiente página

Tabla A.1: Cabecera del [CSV](#) de cada animación, en órden descendente y de izquierda a derecha (completa) (Continuado)

Robot_Hips_posx	Robot_Hips_posy	Robot_Hips_posz
Robot_RightHandPinky2_posx	Robot_RightHandPinky2_posy	Robot_RightHandPinky2_posz
Robot_RightHandPinky2_rotx	Robot_RightHandPinky2_roty	Robot_RightHandPinky2_rotz
Robot_RightHandPinky3_posx	Robot_RightHandPinky3_posy	Robot_RightHandPinky3_posz
Robot_RightHandPinky3_rotx	Robot_RightHandPinky3_roty	Robot_RightHandPinky3_rotz
Robot_RightHandRing1_posx	Robot_RightHandRing1_posy	Robot_RightHandRing1_posz
Robot_RightHandRing1_rotx	Robot_RightHandRing1_roty	Robot_RightHandRing1_rotz
Robot_RightHandRing2_posx	Robot_RightHandRing2_posy	Robot_RightHandRing2_posz
Robot_RightHandRing2_rotx	Robot_RightHandRing2_roty	Robot_RightHandRing2_rotz
Robot_RightHandRing3_posx	Robot_RightHandRing3_posy	Robot_RightHandRing3_posz
Robot_RightHandRing3_rotx	Robot_RightHandRing3_roty	Robot_RightHandRing3_rotz
Robot_RightHandThumb1_posx	Robot_RightHandThumb1_posy	Robot_RightHandThumb1_posz
Robot_RightHandThumb1_rotx	Robot_RightHandThumb1_roty	Robot_RightHandThumb1_rotz
Robot_RightHandThumb2_posx	Robot_RightHandThumb2_posy	Robot_RightHandThumb2_posz
Robot_RightHandThumb2_rotx	Robot_RightHandThumb2_roty	Robot_RightHandThumb2_rotz
Robot_RightHandThumb3_posx	Robot_RightHandThumb3_posy	Robot_RightHandThumb3_posz
Robot_RightHandThumb3_rotx	Robot_RightHandThumb3_roty	Robot_RightHandThumb3_rotz

Carteles para la generación del dataset



Figura B.1: Cartel colgado en la facultad de informática



Figura B.2: Cartel colgado en redes sociales



Figura B.3: Cartel colgado en pantallas de la facultad

C

Apndice

Formulario de recogida de citas

Participación Dataset para detección de poses

Necesitamos datos para un TFG de la Facultad de Informática de la UCM. Lo importante, ¿qué datos recopilamos? La siguiente lista de datos son los datos que vamos a recopilar y hacer públicos en el propio estudio del tfg o en un dataset público en [kaggle](#):

- Edad
- Sexo
- Nacionalidad
- Datos de coordenadas de distintas partes del cuerpo dadas por un traje de captura de movimiento

Estos datos se recogerán en un formulario a parte el día de la prueba, siendo hechos desde un correo de los organizadores y permaneciendo totalmente anónimos sin correlación a la persona que ha realizado la prueba.

OBJETIVO DE LOS DATOS: los datos de edad, nacionalidad y sexo son meramente estadísticos, solo se usarán para demostrar la heterogeneidad (o falta de) de los datos tomados. Los datos de coordenadas se usarán para entrenar distintos modelos de clasificación de pose con la intención de poder decir con acierto el gesto o pose que está haciendo una persona mientras lleva el traje puesto.

LOCALIZACIÓN DE LA PRUEBA: [Facultad de Informática \(UCM\)](#), Calle del Prof. José García Santesmases, 9, Moncloa - Aravaca, 28040 Madrid

alejba02@ucm.es [Cambiar de cuenta](#) 

* Indica que la pregunta es obligatoria

Correo *

Tu dirección de correo electrónico 

¿Aceptas la recogida de datos y la participación en el muestreo? *

Si
 No

[Siguiente](#)  Página 1 de 2 [Borrar formulario](#) 

Este formulario se creó en Universidad Complutense de Madrid.
¿Parece sospechoso este formulario? [Informe](#)

Figura C.1: Formulario de recogida de citas (primera parte)

Participación Dataset para detección de poses

alejba02@ucm.es [Cambiar de cuenta](#)

Horario disponible

A continuación marca las horas en las podrías estar disponible. Con la respuesta que proporciones te mandaremos un correo con un día y hora determinados.

Lunes

9:00-10:00
 10:00-11:00
 11:00-12:00
 12:00-13:00
 13:00-14:00
 14:00-15:00
 15:00-16:00
 16:00-17:00
 17:00-18:00
 18:00-19:00
 19:00-20:00

!

!

Figura C.2: Formulario de recogida de citas (segunda parte)

Apndice D

Formulario de recogida de datos demográficos

Diversidad TFG

alejba02@ucm.es [Cambiar de cuenta](#)  

No compartido

Género

Masculino
 Femenino
 No binario
 Prefiero no decirlo
 Otro: _____

Edad

Tu respuesta _____

Nacionalidad(sin espacios, separado por comas)

Tu respuesta _____

Idiomas hablados(sin espacios, separado por comas)

Tu respuesta _____

Diestro o Zurdo

Diestro
 Zurdo

Figura D.1: Formulario de recogida de datos demográficos (primera parte)

The screenshot shows the header and footer sections of a Google Form. At the top left is a purple "Enviar" button, and at the top right is a "Borrar formulario" link. Below these are two small text links: "Nunca envíes contraseñas a través de Formularios de Google." and "Este formulario se creó en Universidad Complutense de Madrid." A "Parece sospechoso este formulario? Informe" link is also present. The footer features the "Google Formularios" logo and a circular icon with a pencil.

Figura D.2: Formulario de recogida de datos demográficos (segunda parte)

Apndice E

Gráficos de la generación del dataset

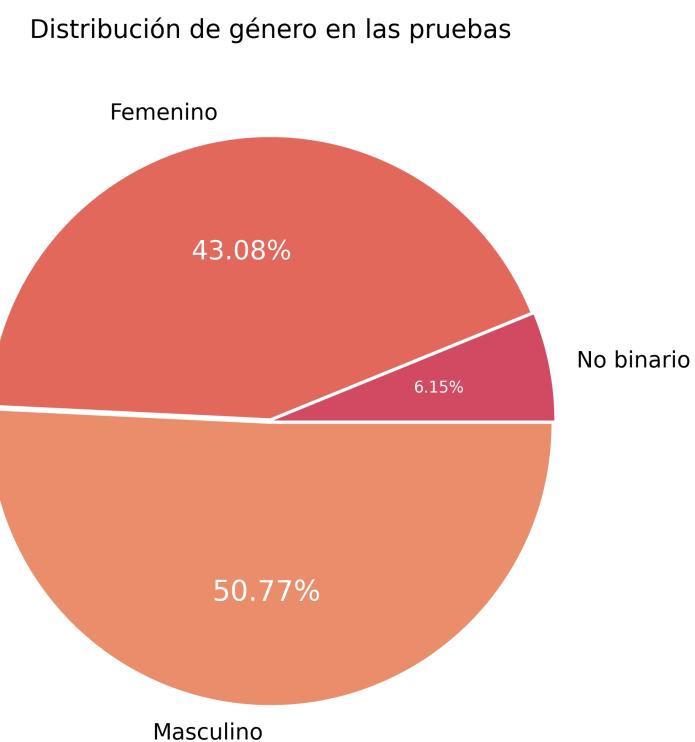


Figura E.1: Distribución de género de los encuestados

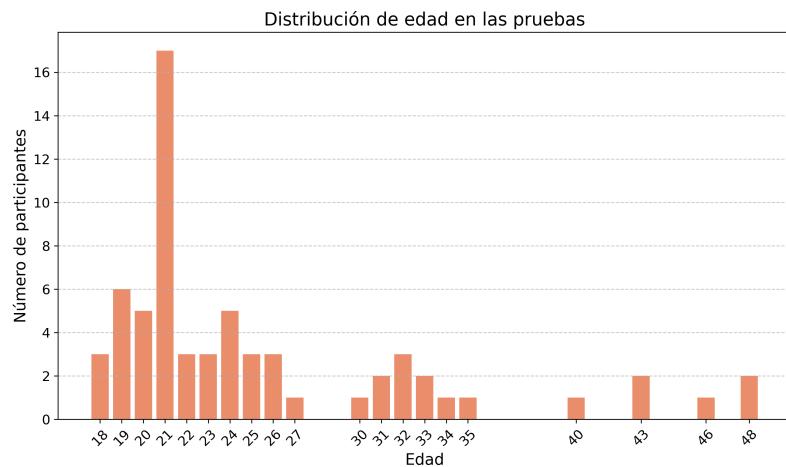


Figura E.2: Distribución de edad de los encuestados

Distribución de nacionalidades en las pruebas

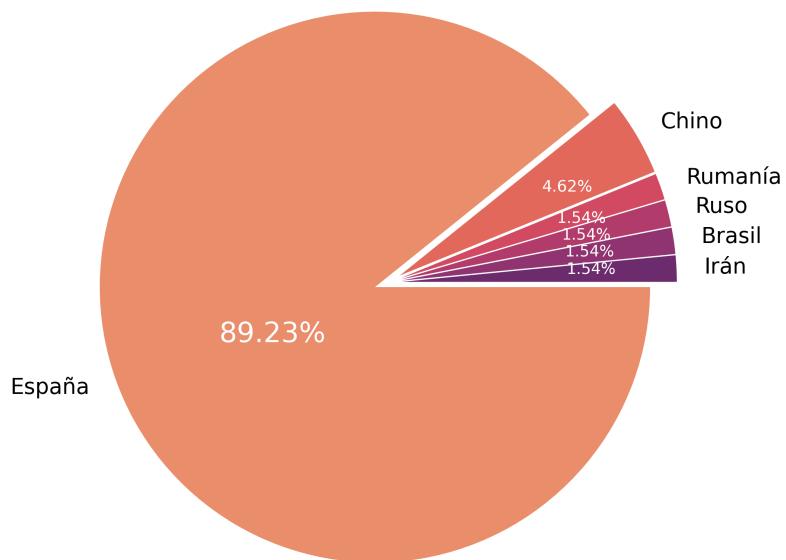


Figura E.3: Distribución de nacionalidad de los encuestados

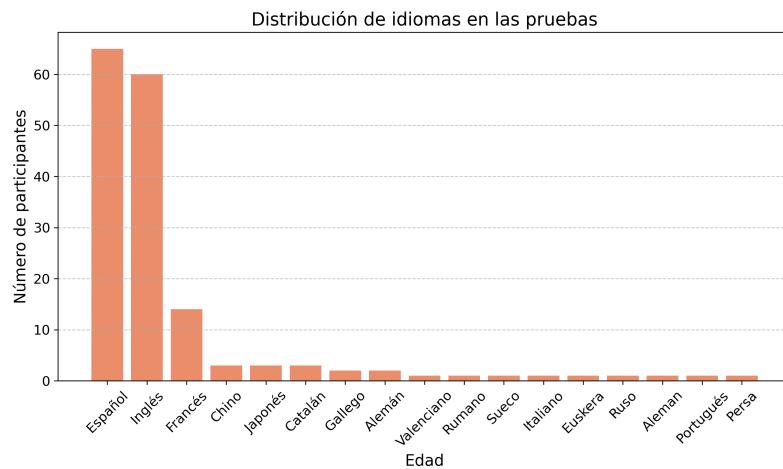


Figura E.4: Distribución de idiomas hablados de los encuestados

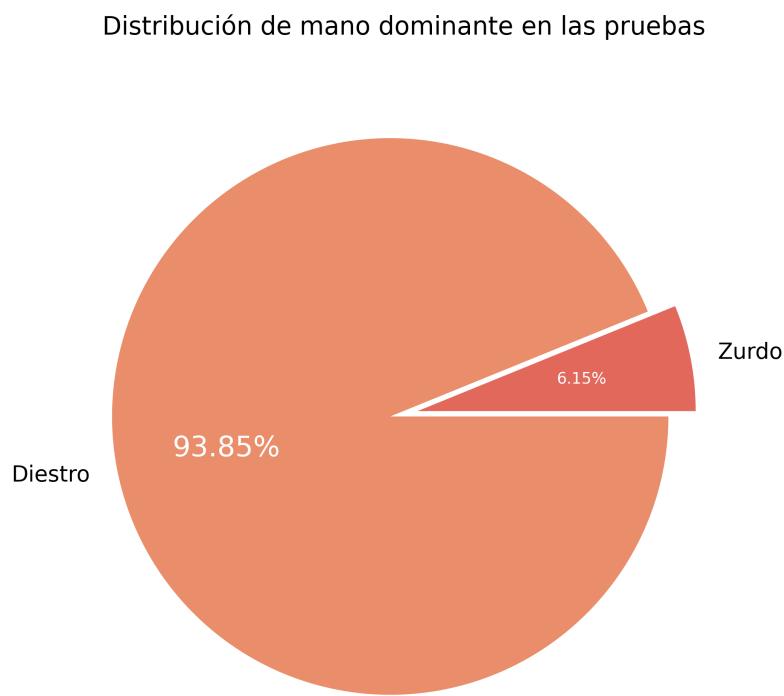


Figura E.5: Distribución de mano dominante de los encuestados

Apndice F

Resultados de los distintos modelos LSTM

La línea verde en los gráficos de tensorboard indican la mejor combinación de hiperparámetros encontrada en cada caso

F.1. Intervalo 0.2s

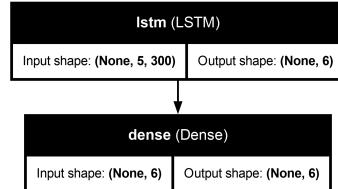


Figura F.1: Esquema del modelo LSTM con 0.2s de intervalo



Figura F.2: Matriz de confusión del modelo LSTM con 0.2s de intervalo

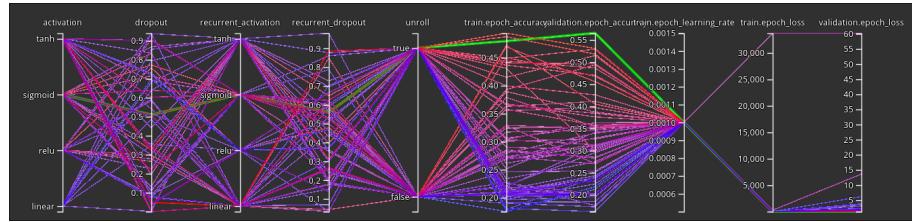


Figura F.3: Gráfico de entrenamiento del modelo LSTM con 0.2s de intervalo (mejor val_accuracy = 0.5655)

F.2. Intervalo 0.4s

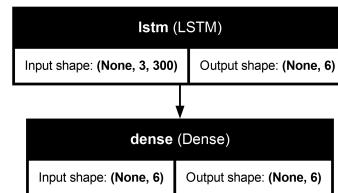


Figura F.4: Esquema del modelo LSTM con 0.4s de intervalo

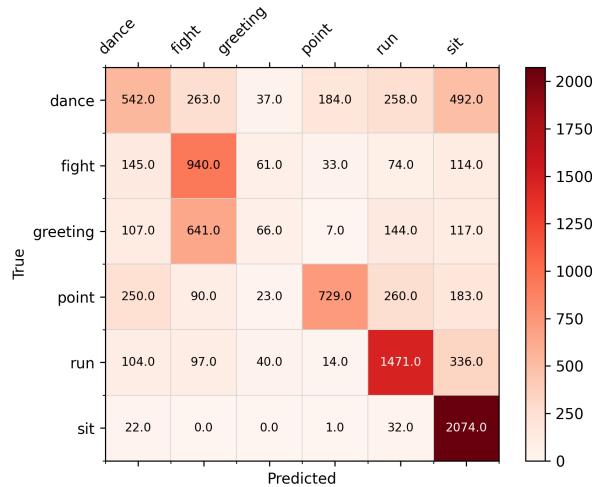


Figura F.5: Matriz de confusión del modelo LSTM con 0.4s de intervalo

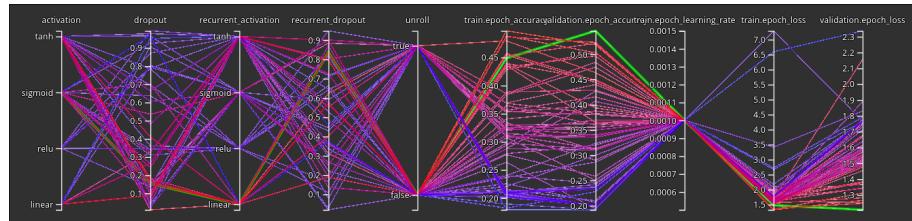


Figura F.6: Gráfico de entrenamiento del modelo LSTM con 0.4s de intervalo (mejor val_accuracy = 0.5462)

F.3. Intervalo 0.6s

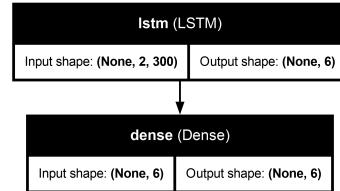


Figura F.7: Esquema del modelo LSTM con 0.6s de intervalo

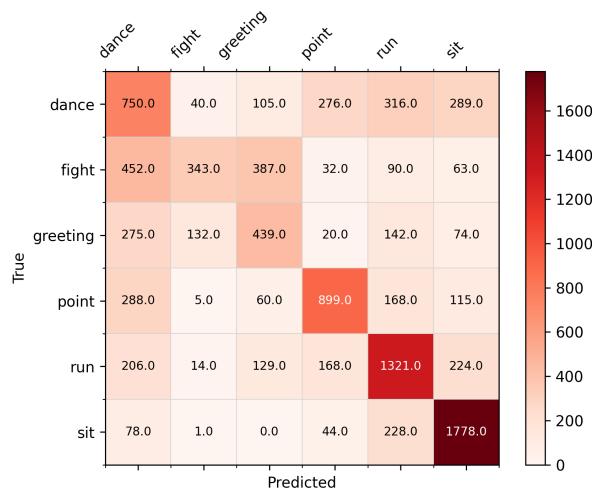


Figura F.8: Matriz de confusión del modelo LSTM con 0.6s de intervalo

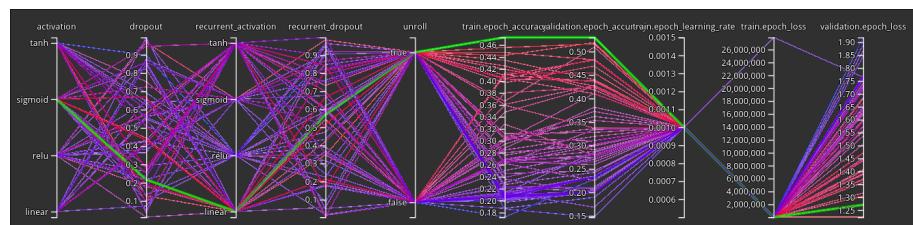


Figura F.9: Gráfico de entrenamiento del modelo LSTM con 0.6s de intervalo (mejor val_accuracy = 0.5301)

F.4. Intervalo 0.8s

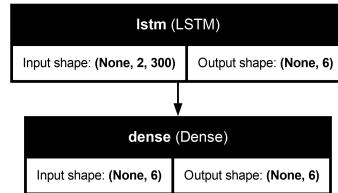


Figura F.10: Esquema del modelo LSTM con 0.8s de intervalo

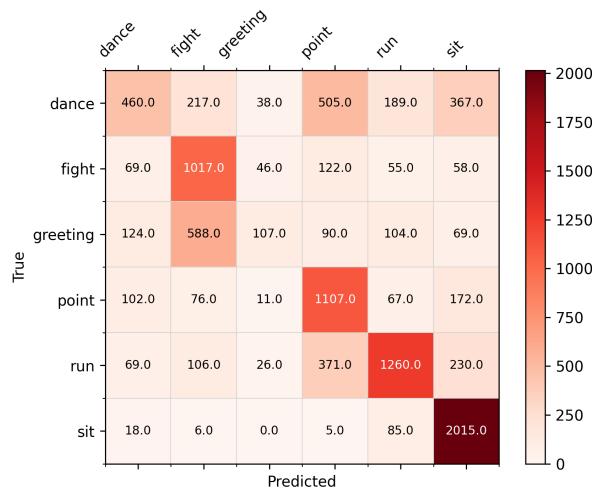


Figura F.11: Matriz de confusión del modelo LSTM con 0.8s de intervalo

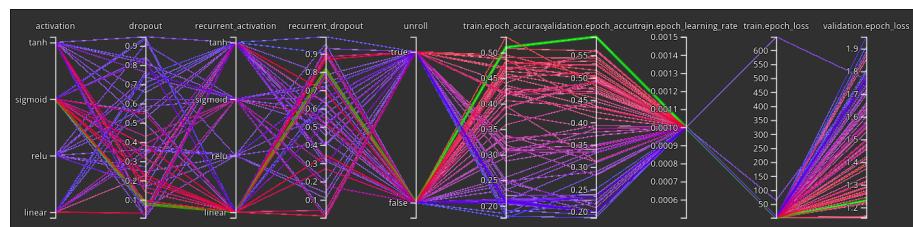


Figura F.12: Gráfico de entrenamiento del modelo LSTM con 0.8s de intervalo (mejor val_accuracy = 0.5912)

F.5. Intervalo 1.0s

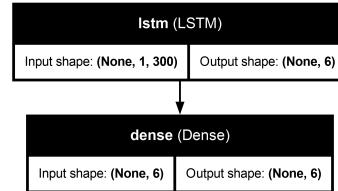


Figura F.13: Esquema del modelo LSTM con 1.0s de intervalo

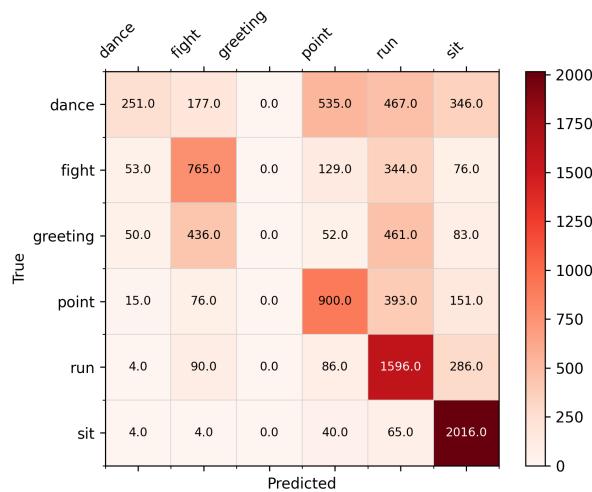


Figura F.14: Matriz de confusión del modelo LSTM con 1.0s de intervalo

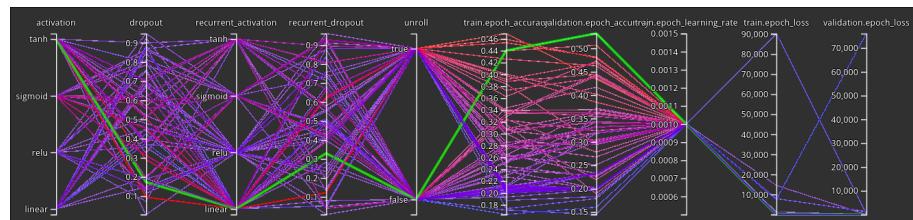


Figura F.15: Gráfico de entrenamiento del modelo LSTM con 1.0s de intervalo (mejor val_accuracy = 0.5318)

Resultados de los distintos modelos RNN

La línea verde en los gráficos de tensorboard indican la mejor combinación de hiperparámetros encontrada en cada caso

G.1. Intervalo 0.2s

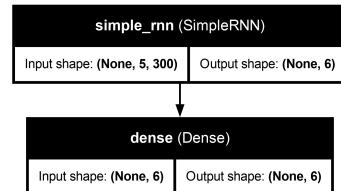


Figura G.1: Esquema del modelo RNN con 0.2s de intervalo

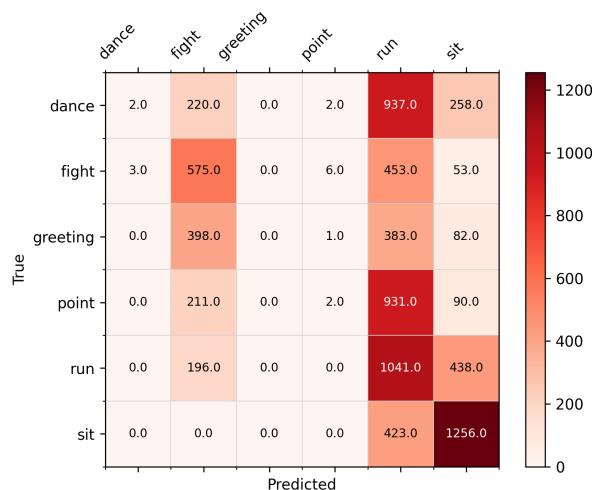


Figura G.2: Matriz de confusión del modelo RNN con 0.2s de intervalo

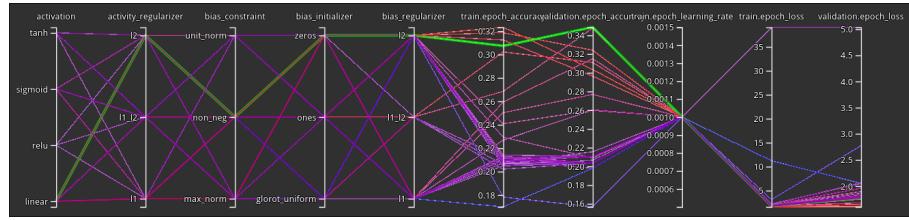


Figura G.3: Gráfico de entrenamiento del modelo RNN con 0.2s de intervalo (mejor val_accuracy = 0.3486)

G.2. Intervalo 0.4s

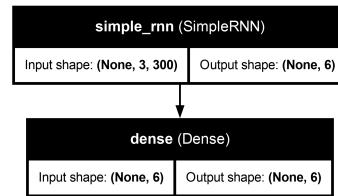


Figura G.4: Esquema del modelo RNN con 0.4s de intervalo

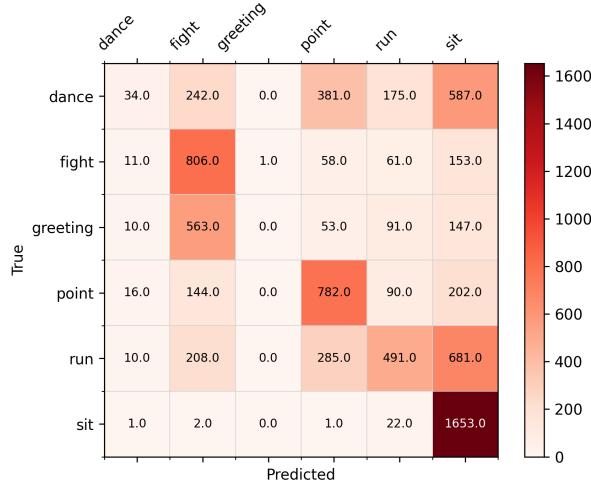


Figura G.5: Matriz de confusión del modelo RNN con 0.4s de intervalo

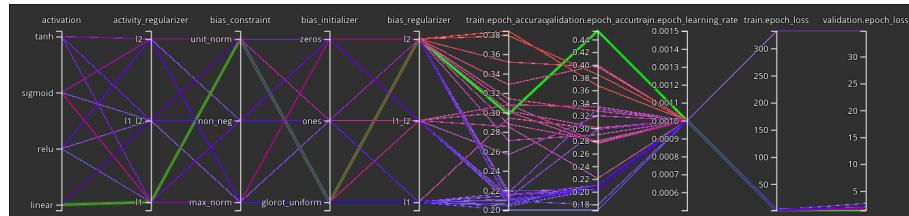


Figura G.6: Gráfico de entrenamiento del modelo RNN con 0.4s de intervalo (mejor val_accuracy = 0.4535)

G.3. Intervalo 0.6s

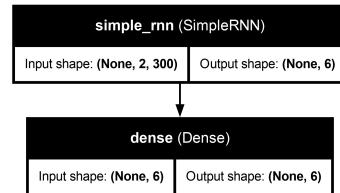


Figura G.7: Esquema del modelo RNN con 0.6s de intervalo

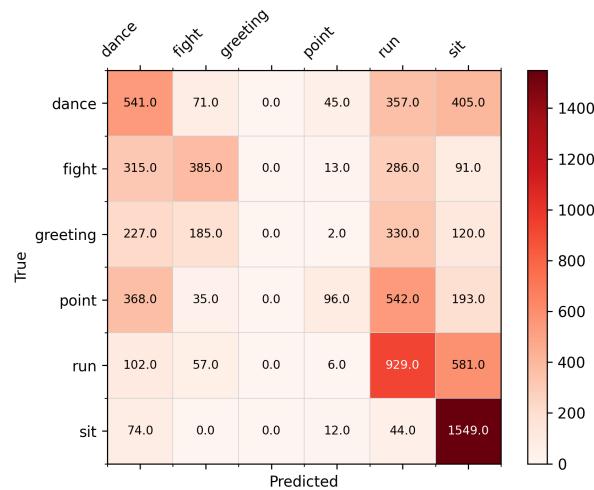


Figura G.8: Matriz de confusión del modelo RNN con 0.6s de intervalo

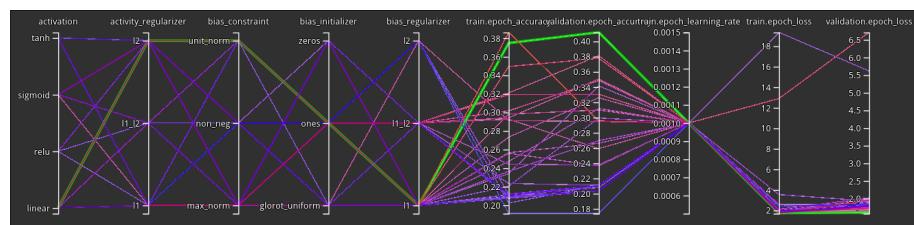


Figura G.9: Gráfico de entrenamiento del modelo RNN con 0.6s de intervalo (mejor val_accuracy = 0.41185)

G.4. Intervalo 0.8s

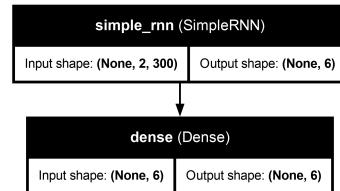


Figura G.10: Esquema del modelo RNN con 0.8s de intervalo

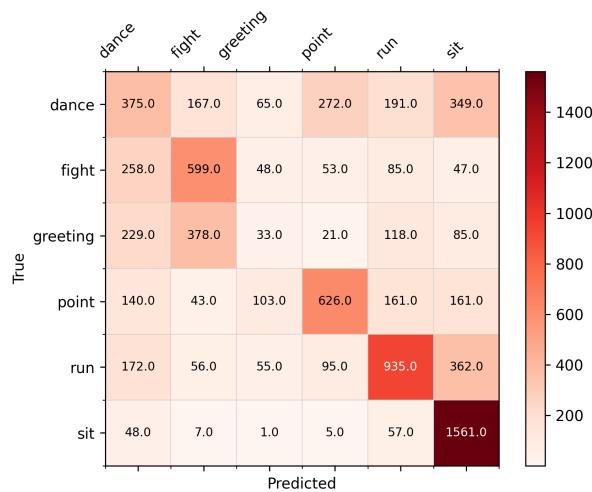


Figura G.11: Matriz de confusión del modelo RNN con 0.8s de intervalo

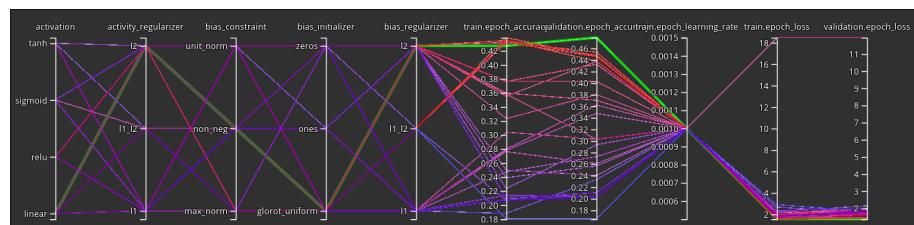


Figura G.12: Gráfico de entrenamiento del modelo RNN con 0.8s de intervalo (mejor val_accuracy = 0.47865)

G.5. Intervalo 1.0s

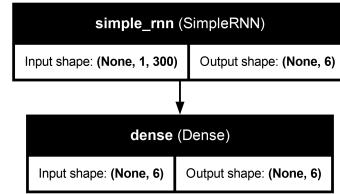


Figura G.13: Esquema del modelo RNN con 1.0s de intervalo

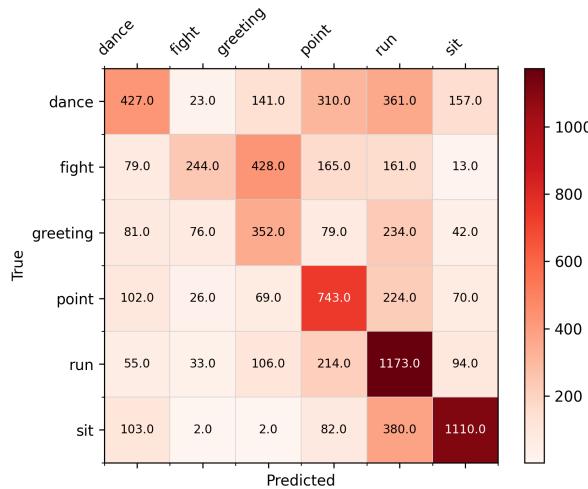


Figura G.14: Matriz de confusión del modelo RNN con 1.0s de intervalo

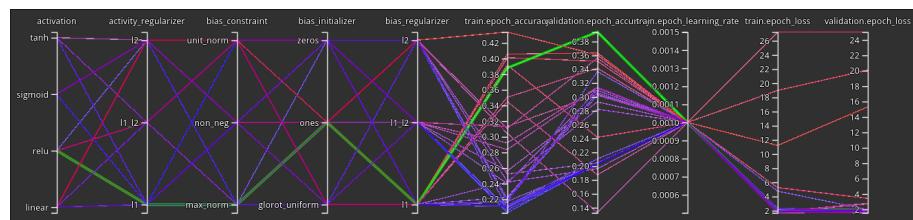


Figura G.15: Gráfico de entrenamiento del modelo RNN con 1.0s de intervalo (mejor val_accuracy = 0.39377)

Apndice H

Resultados de los distintos modelos CNN

La línea verde en los gráficos de tensorboard indican la mejor combinación de hiperparámetros encontrada en cada caso

H.1. Intervalo 0.2s

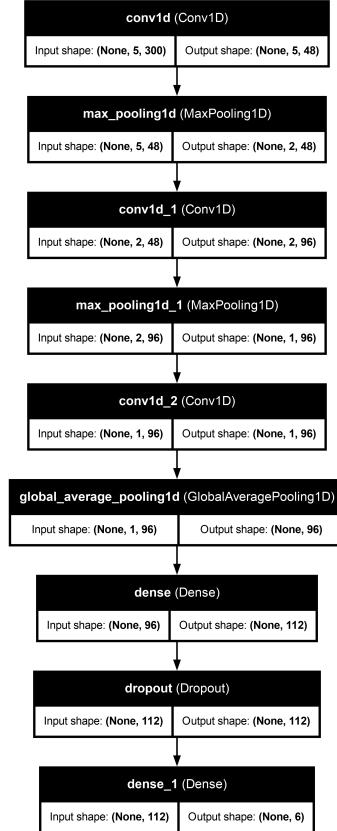


Figura H.1: Esquema del modelo CNN con 0.2s de intervalo

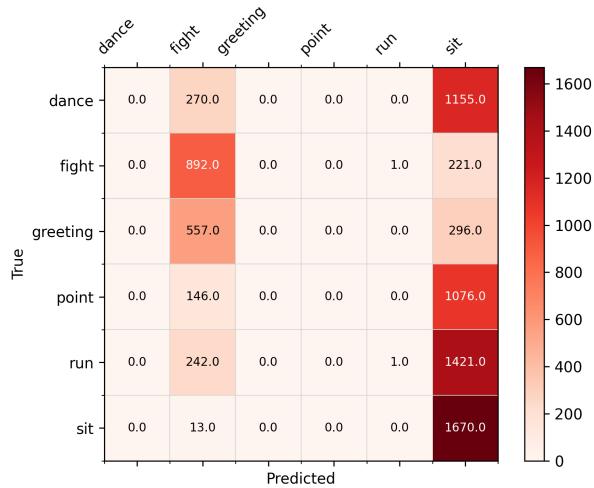


Figura H.2: Matriz de confusión del modelo CNN con 0.2s de intervalo

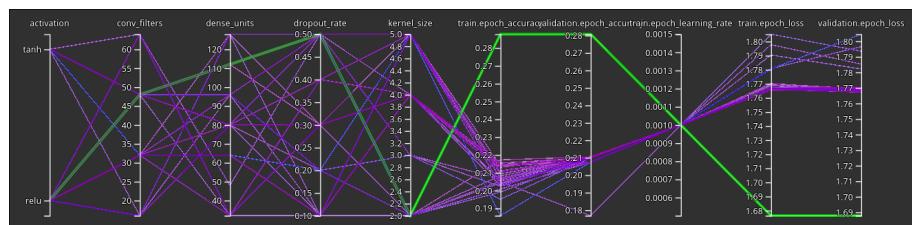


Figura H.3: Gráfico de entrenamiento del modelo CNN con 0.2s de intervalo (mejor val_accuracy = 0.28777)

H.2. Intervalo 0.4s

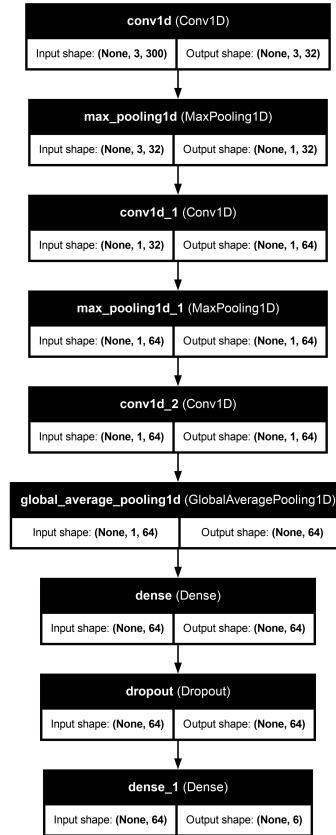


Figura H.4: Esquema del modelo CNN con 0.4s de intervalo

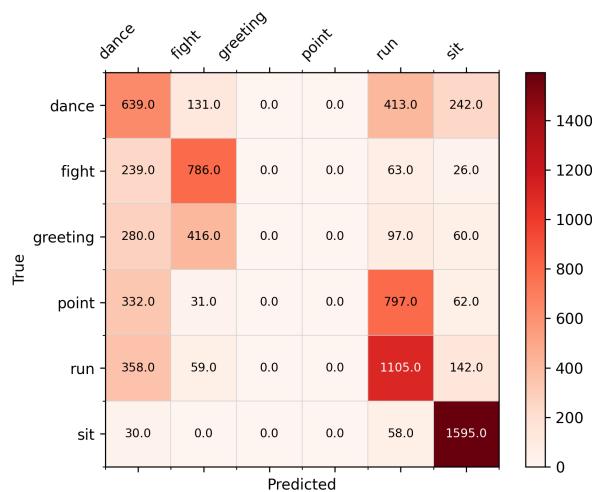


Figura H.5: Matriz de confusión del modelo CNN con 0.4s de intervalo

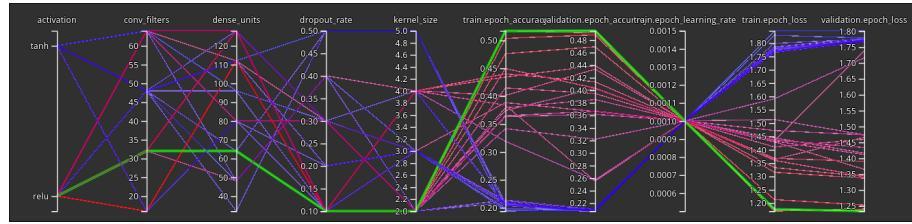


Figura H.6: Gráfico de entrenamiento del modelo CNN con 0.4s de intervalo (mejor val_accuracy = 0.49473)

H.3. Intervalo 0.6s

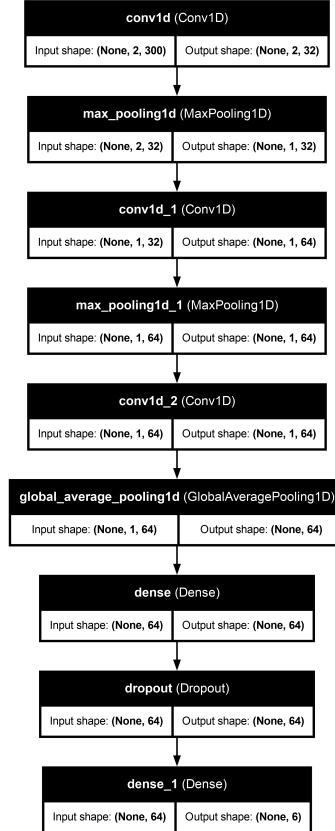


Figura H.7: Esquema del modelo CNN con 0.6s de intervalo

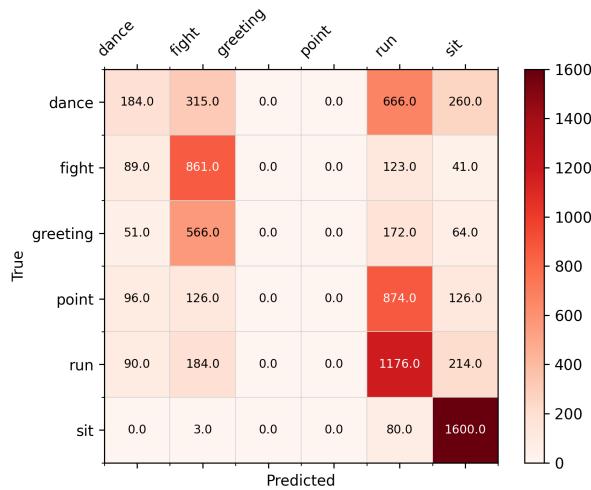


Figura H.8: Matriz de confusión del modelo CNN con 0.6s de intervalo

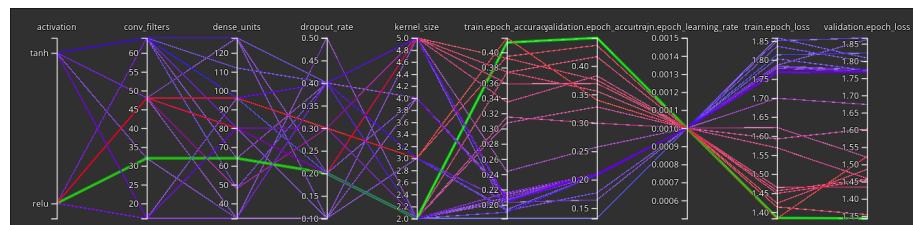


Figura H.9: Gráfico de entrenamiento del modelo CNN con 0.6s de intervalo (mejor val_accuracy = 0.44952)

H.4. Intervalo 0.8s

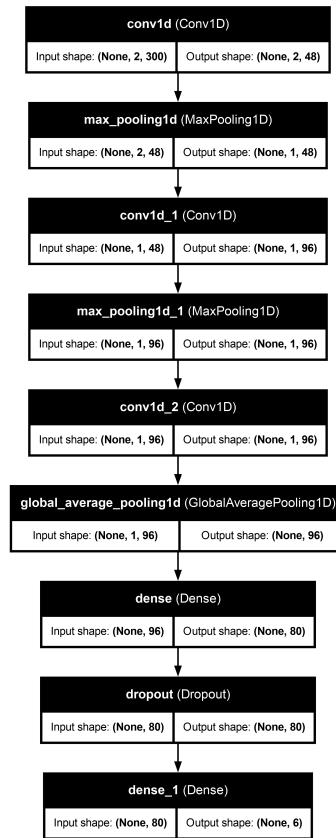


Figura H.10: Esquema del modelo CNN con 0.8s de intervalo



Figura H.11: Matriz de confusión del modelo CNN con 0.8s de intervalo

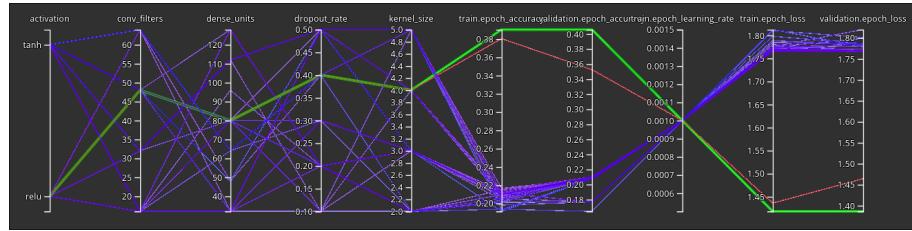


Figura H.12: Gráfico de entrenamiento del modelo CNN con 0.8s de intervalo (mejor val_accuracy = 0.40583)

H.5. Intervalo 1.0s

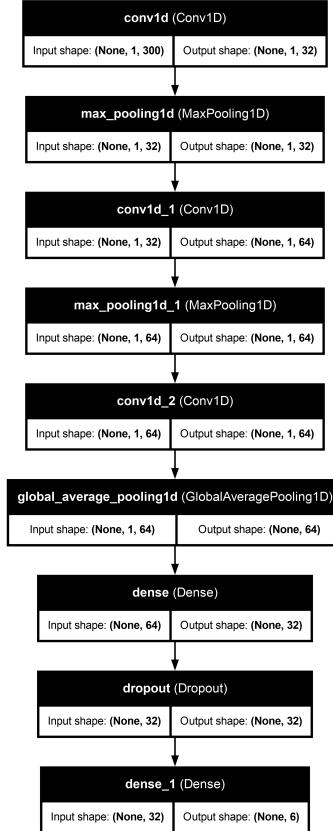


Figura H.13: Esquema del modelo CNN con 1.0s de intervalo

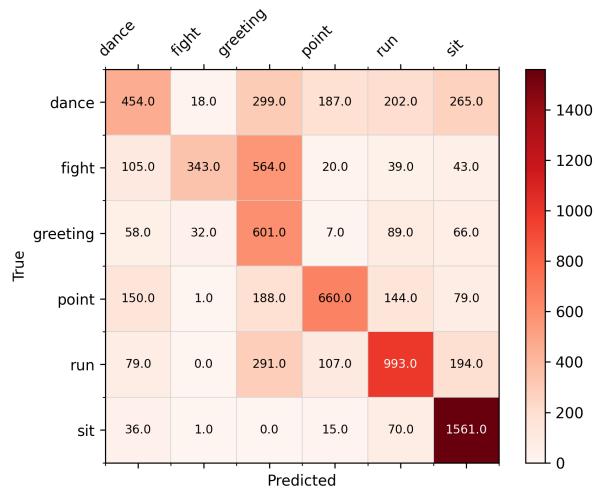


Figura H.14: Matriz de confusión del modelo CNN con 1.0s de intervalo

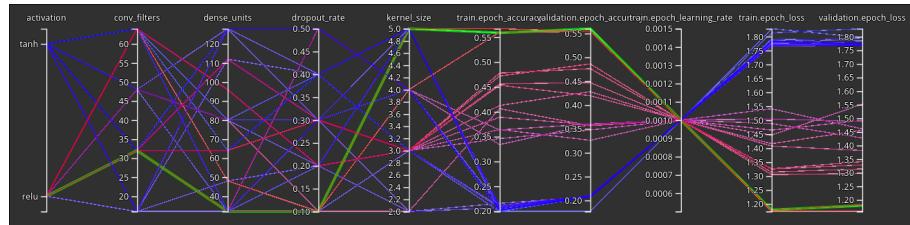


Figura H.15: Gráfico de entrenamiento del modelo CNN con 1.0s de intervalo (mejor val_accuracy = 0.56002)

Apndice I

Resultados de los distintos modelos Random Forest

I.1. Intervalo 0.2s

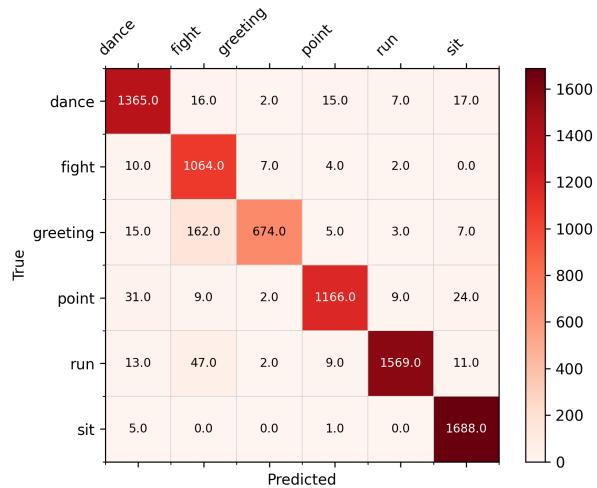


Figura I.1: Matriz de confusión del modelo Random Forest con 0.2s de intervalo (mejor val_accuracy = 0.85735)

I.2. Intervalo 0.4s

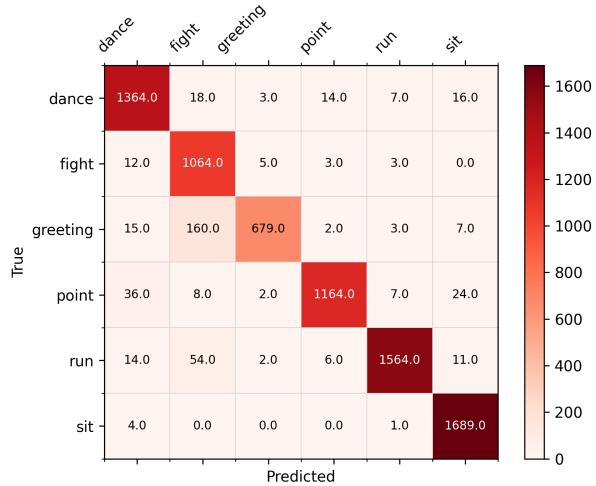


Figura I.2: Matriz de confusión del modelo Random Forest con 0.4s de intervalo (mejor val_accuracy = 0.85735)

I.3. Intervalo 0.6s

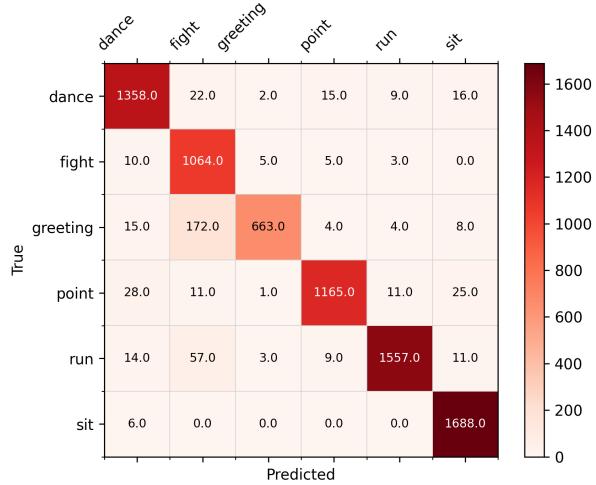


Figura I.3: Matriz de confusión del modelo Random Forest con 0.6s de intervalo (mejor val_accuracy = 0.85233)

I.4. Intervalo 0.8s

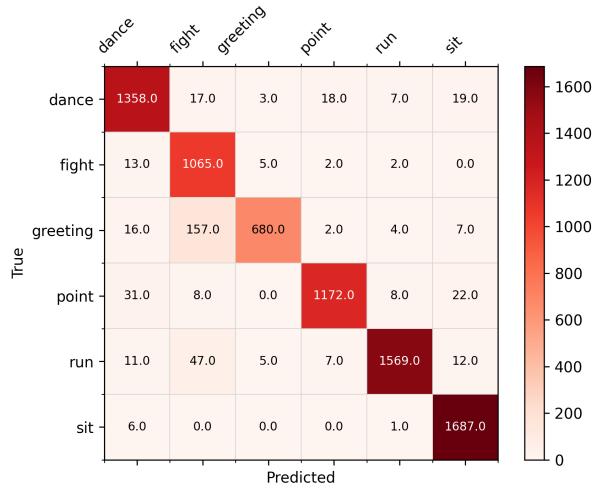


Figura I.4: Matriz de confusión del modelo Random Forest con 0.8s de intervalo (mejor val_accuracy = 0.85936)

I.5. Intervalo 1.0s

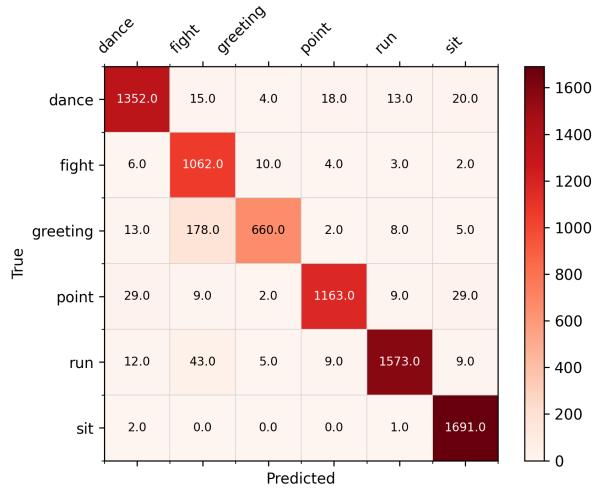


Figura I.5: Matriz de confusión del modelo Random Forest con 1.0s de intervalo (mejor val_accuracy = 0.84681)

Código de la herramienta MixamoDumper

```
1 public class AnimImport : MonoBehaviour{
2     private Animator anim;
3     private string path;
4     private string header;
5     private string csv_path;
6     private StreamWriter csv_file;
7
8     private Dictionary<string, List<AnimationClip>> animacionMap
9     ;
10    protected AnimatorOverrideController
11    animatorOverrideController;
12
13    private int indexAnim;
14    private int index;
15    private float timer;
16
17    [SerializeField]
18    private List<Transform> _bones;
19    private NumberFormatInfo nfi;
20
21    // Start is called before the first frame update
22    void Start()
23    {
24        nfi = new NumberFormatInfo();
25        nfi.NumberDecimalSeparator = ".";
26        animacionMap =
27            new Dictionary<string, List<AnimationClip>>();
28        anim = GetComponent<Animator>();
29        animatorOverrideController =
30            new AnimatorOverrideController(anim.
31            runtimeAnimatorController);
31        anim.runtimeAnimatorController =
32        animatorOverrideController;
33        index = 0;
34        indexAnim = -1;
35        timer = 0.0f;
36        path = Application.dataPath + "/Resources";
```

```

36         Directory.CreateDirectory(Path.Combine(Application.
37             dataPath, "CSV"));
38         header = createHeader();
39         loadClips();
40         setClip();
41     }
42
43     // Update is called once per frame
44     void Update()
45     {
46         timer += Time.deltaTime;
47         string data = "";
48         foreach (Transform t in _bones)
49         {
50             data += t.position.x.ToString(nfi) + ","
51             + t.position.y.ToString(nfi) + ","
52             + t.position.z.ToString(nfi) + "," +
53             t.rotation.eulerAngles.x.ToString(nfi) + "," +
54             t.rotation.eulerAngles.y.ToString(nfi) + "," +
55             t.rotation.eulerAngles.z.ToString(nfi) + ",";
56         }
57         data = data.Remove(data.Length - 1);
58         csv_file.WriteLine(data + "\n");
59
60         if (timer >= anim.GetCurrentAnimatorClipInfo(0)[0].clip.
61             length)
62         {
63             timer = 0.0f;
64             csv_file.Close();
65             setClip(); //nextAnimation
66         }
67
68     /// <summary>
69     /// Sets the animation clip that will be executed.
70     /// </summary>
71     private void setClip()
72     {
73         KeyValuePair<string, List<AnimationClip>> val =
74             animacionMap.ElementAt(index);
75         indexAnim++;
76         if (indexAnim == val.Value.Count)
77         {
78             index++;
79             if (index == animacionMap.Count)
80                 EditorApplication.Exit(0);
81             val = animacionMap.ElementAt(index);
82             indexAnim = 0;
83         }
84         csv_path = Path.Combine(Application.dataPath,
85             "CSV",
86             val.Key.ToLower() + "_" + indexAnim.ToString() + ".csv");
87         ;
88         csv_file = new StreamWriter(csv_path, false);
89         csv_file.WriteLine(header);

```

```

89         animatorOverrideController[val.Value[indexAnim].name] =
90             val.Value[indexAnim];
91         anim.runtimeAnimatorController =
92             animatorOverrideController;
93         anim.Play("Default", 0, 0f);
94     }
95
96     /// <summary>
97     /// Loads all the animation clips from the "Resources"
98     /// directory.
99     Each animation must be in a directory with the same name
100    as the type of gesture.
101    /// </summary>
102    private void loadClips()
103    {
104        try
105        {
106            string[] dir = Directory.GetDirectories(path);
107            foreach (string dirName in dir)
108            {
109                string[] animsPath = Directory.GetFiles(dirName)
110                ;
111                string[] tmp = dirName.Split('\\');
112                animacionMap.Add(tmp[tmp.Length - 1],
113                    new List<AnimationClip>());
114                foreach (string animPath in animsPath)
115                {
116
117                    string[] tmp2 = animPath.Split('\\');
118                    string a = Path.Combine(resourcePath,
119                        tmp2[tmp2.Length - 2],
120                        tmp2[tmp2.Length - 1]);
121                    var allAsset =
122                        AssetDatabase.LoadAllAssetsAtPath(
123                            Path.Combine(resourcePath,
124                                tmp2[tmp2.Length - 2],
125                                tmp2[tmp2.Length - 1]));
126                    foreach (var asset in allAsset)
127                    {
128                        if (asset is AnimationClip &&
129                            asset.name != "__preview__mixamo.com")
130                        {
131                            string tag = tmp[tmp.Length - 1];
132                            animacionMap[tag].Add(asset as
133                                AnimationClip);
134                            break;
135                        }
136                    }
137                catch (Exception e)
138                {
139                    Debug.Log($"Error: {e.Message}");
140                }
141            }
142        }
143    }
144
145    /// <summary>
146    /// Loads all the animation clips from the "Resources"
147    /// directory.
148    Each animation must be in a directory with the same name
149    as the type of gesture.
150    /// </summary>
151    private void loadClips()
152    {
153        try
154        {
155            string[] dir = Directory.GetDirectories(path);
156            foreach (string dirName in dir)
157            {
158                string[] animsPath = Directory.GetFiles(dirName)
159                ;
160                string[] tmp = dirName.Split('\\');
161                animacionMap.Add(tmp[tmp.Length - 1],
162                    new List<AnimationClip>());
163                foreach (string animPath in animsPath)
164                {
165
166                    string[] tmp2 = animPath.Split('\\');
167                    string a = Path.Combine(resourcePath,
168                        tmp2[tmp2.Length - 2],
169                        tmp2[tmp2.Length - 1]);
170                    var allAsset =
171                        AssetDatabase.LoadAllAssetsAtPath(
172                            Path.Combine(resourcePath,
173                                tmp2[tmp2.Length - 2],
174                                tmp2[tmp2.Length - 1]));
175                    foreach (var asset in allAsset)
176                    {
177                        if (asset is AnimationClip &&
178                            asset.name != "__preview__mixamo.com")
179                        {
180                            string tag = tmp[tmp.Length - 1];
181                            animacionMap[tag].Add(asset as
182                                AnimationClip);
183                            break;
184                        }
185                    }
186                }
187            }
188        }
189    }
190
191    /// <summary>
192    /// Loads all the animation clips from the "Resources"
193    /// directory.
194    Each animation must be in a directory with the same name
195    as the type of gesture.
196    /// </summary>
197    private void loadClips()
198    {
199        try
200        {
201            string[] dir = Directory.GetDirectories(path);
202            foreach (string dirName in dir)
203            {
204                string[] animsPath = Directory.GetFiles(dirName)
205                ;
206                string[] tmp = dirName.Split('\\');
207                animacionMap.Add(tmp[tmp.Length - 1],
208                    new List<AnimationClip>());
209                foreach (string animPath in animsPath)
210                {
211
212                    string[] tmp2 = animPath.Split('\\');
213                    string a = Path.Combine(resourcePath,
214                        tmp2[tmp2.Length - 2],
215                        tmp2[tmp2.Length - 1]);
216                    var allAsset =
217                        AssetDatabase.LoadAllAssetsAtPath(
218                            Path.Combine(resourcePath,
219                                tmp2[tmp2.Length - 2],
220                                tmp2[tmp2.Length - 1]));
221                    foreach (var asset in allAsset)
222                    {
223                        if (asset is AnimationClip &&
224                            asset.name != "__preview__mixamo.com")
225                        {
226                            string tag = tmp[tmp.Length - 1];
227                            animacionMap[tag].Add(asset as
228                                AnimationClip);
229                            break;
230                        }
231                    }
232                }
233            }
234        }
235    }
236
237    /// <summary>
238    /// Loads all the animation clips from the "Resources"
239    /// directory.
240    Each animation must be in a directory with the same name
241    as the type of gesture.
242    /// </summary>
243    private void loadClips()
244    {
245        try
246        {
247            string[] dir = Directory.GetDirectories(path);
248            foreach (string dirName in dir)
249            {
250                string[] animsPath = Directory.GetFiles(dirName)
251                ;
252                string[] tmp = dirName.Split('\\');
253                animacionMap.Add(tmp[tmp.Length - 1],
254                    new List<AnimationClip>());
255                foreach (string animPath in animsPath)
256                {
257
258                    string[] tmp2 = animPath.Split('\\');
259                    string a = Path.Combine(resourcePath,
260                        tmp2[tmp2.Length - 2],
261                        tmp2[tmp2.Length - 1]);
262                    var allAsset =
263                        AssetDatabase.LoadAllAssetsAtPath(
264                            Path.Combine(resourcePath,
265                                tmp2[tmp2.Length - 2],
266                                tmp2[tmp2.Length - 1]));
267                    foreach (var asset in allAsset)
268                    {
269                        if (asset is AnimationClip &&
270                            asset.name != "__preview__mixamo.com")
271                        {
272                            string tag = tmp[tmp.Length - 1];
273                            animacionMap[tag].Add(asset as
274                                AnimationClip);
275                            break;
276                        }
277                    }
278                }
279            }
280        }
281    }
282
283    /// <summary>
284    /// Loads all the animation clips from the "Resources"
285    /// directory.
286    Each animation must be in a directory with the same name
287    as the type of gesture.
288    /// </summary>
289    private void loadClips()
290    {
291        try
292        {
293            string[] dir = Directory.GetDirectories(path);
294            foreach (string dirName in dir)
295            {
296                string[] animsPath = Directory.GetFiles(dirName)
297                ;
298                string[] tmp = dirName.Split('\\');
299                animacionMap.Add(tmp[tmp.Length - 1],
300                    new List<AnimationClip>());
301                foreach (string animPath in animsPath)
302                {
303
304                    string[] tmp2 = animPath.Split('\\');
305                    string a = Path.Combine(resourcePath,
306                        tmp2[tmp2.Length - 2],
307                        tmp2[tmp2.Length - 1]);
308                    var allAsset =
309                        AssetDatabase.LoadAllAssetsAtPath(
310                            Path.Combine(resourcePath,
311                                tmp2[tmp2.Length - 2],
312                                tmp2[tmp2.Length - 1]));
313                    foreach (var asset in allAsset)
314                    {
315                        if (asset is AnimationClip &&
316                            asset.name != "__preview__mixamo.com")
317                        {
318                            string tag = tmp[tmp.Length - 1];
319                            animacionMap[tag].Add(asset as
320                                AnimationClip);
321                            break;
322                        }
323                    }
324                }
325            }
326        }
327    }
328
329    /// <summary>
330    /// Loads all the animation clips from the "Resources"
331    /// directory.
332    Each animation must be in a directory with the same name
333    as the type of gesture.
334    /// </summary>
335    private void loadClips()
336    {
337        try
338        {
339            string[] dir = Directory.GetDirectories(path);
340            foreach (string dirName in dir)
341            {
342                string[] animsPath = Directory.GetFiles(dirName)
343                ;
344                string[] tmp = dirName.Split('\\');
345                animacionMap.Add(tmp[tmp.Length - 1],
346                    new List<AnimationClip>());
347                foreach (string animPath in animsPath)
348                {
349
350                    string[] tmp2 = animPath.Split('\\');
351                    string a = Path.Combine(resourcePath,
352                        tmp2[tmp2.Length - 2],
353                        tmp2[tmp2.Length - 1]);
354                    var allAsset =
355                        AssetDatabase.LoadAllAssetsAtPath(
356                            Path.Combine(resourcePath,
357                                tmp2[tmp2.Length - 2],
358                                tmp2[tmp2.Length - 1]));
359                    foreach (var asset in allAsset)
360                    {
361                        if (asset is AnimationClip &&
362                            asset.name != "__preview__mixamo.com")
363                        {
364                            string tag = tmp[tmp.Length - 1];
365                            animacionMap[tag].Add(asset as
366                                AnimationClip);
367                            break;
368                        }
369                    }
370                }
371            }
372        }
373    }
374
375    /// <summary>
376    /// Loads all the animation clips from the "Resources"
377    /// directory.
378    Each animation must be in a directory with the same name
379    as the type of gesture.
380    /// </summary>
381    private void loadClips()
382    {
383        try
384        {
385            string[] dir = Directory.GetDirectories(path);
386            foreach (string dirName in dir)
387            {
388                string[] animsPath = Directory.GetFiles(dirName)
389                ;
390                string[] tmp = dirName.Split('\\');
391                animacionMap.Add(tmp[tmp.Length - 1],
392                    new List<AnimationClip>());
393                foreach (string animPath in animsPath)
394                {
395
396                    string[] tmp2 = animPath.Split('\\');
397                    string a = Path.Combine(resourcePath,
398                        tmp2[tmp2.Length - 2],
399                        tmp2[tmp2.Length - 1]);
400                    var allAsset =
401                        AssetDatabase.LoadAllAssetsAtPath(
402                            Path.Combine(resourcePath,
403                                tmp2[tmp2.Length - 2],
404                                tmp2[tmp2.Length - 1]));
405                    foreach (var asset in allAsset)
406                    {
407                        if (asset is AnimationClip &&
408                            asset.name != "__preview__mixamo.com")
409                        {
410                            string tag = tmp[tmp.Length - 1];
411                            animacionMap[tag].Add(asset as
412                                AnimationClip);
413                            break;
414                        }
415                    }
416                }
417            }
418        }
419    }
420
421    /// <summary>
422    /// Loads all the animation clips from the "Resources"
423    /// directory.
424    Each animation must be in a directory with the same name
425    as the type of gesture.
426    /// </summary>
427    private void loadClips()
428    {
429        try
430        {
431            string[] dir = Directory.GetDirectories(path);
432            foreach (string dirName in dir)
433            {
434                string[] animsPath = Directory.GetFiles(dirName)
435                ;
436                string[] tmp = dirName.Split('\\');
437                animacionMap.Add(tmp[tmp.Length - 1],
438                    new List<AnimationClip>());
439                foreach (string animPath in animsPath)
440                {
441
442                    string[] tmp2 = animPath.Split('\\');
443                    string a = Path.Combine(resourcePath,
444                        tmp2[tmp2.Length - 2],
445                        tmp2[tmp2.Length - 1]);
446                    var allAsset =
447                        AssetDatabase.LoadAllAssetsAtPath(
448                            Path.Combine(resourcePath,
449                                tmp2[tmp2.Length - 2],
450                                tmp2[tmp2.Length - 1]));
451                    foreach (var asset in allAsset)
452                    {
453                        if (asset is AnimationClip &&
454                            asset.name != "__preview__mixamo.com")
455                        {
456                            string tag = tmp[tmp.Length - 1];
457                            animacionMap[tag].Add(asset as
458                                AnimationClip);
459                            break;
460                        }
461                    }
462                }
463            }
464        }
465    }
466
467    /// <summary>
468    /// Loads all the animation clips from the "Resources"
469    /// directory.
470    Each animation must be in a directory with the same name
471    as the type of gesture.
472    /// </summary>
473    private void loadClips()
474    {
475        try
476        {
477            string[] dir = Directory.GetDirectories(path);
478            foreach (string dirName in dir)
479            {
480                string[] animsPath = Directory.GetFiles(dirName)
481                ;
482                string[] tmp = dirName.Split('\\');
483                animacionMap.Add(tmp[tmp.Length - 1],
484                    new List<AnimationClip>());
485                foreach (string animPath in animsPath)
486                {
487
488                    string[] tmp2 = animPath.Split('\\');
489                    string a = Path.Combine(resourcePath,
490                        tmp2[tmp2.Length - 2],
491                        tmp2[tmp2.Length - 1]);
492                    var allAsset =
493                        AssetDatabase.LoadAllAssetsAtPath(
494                            Path.Combine(resourcePath,
495                                tmp2[tmp2.Length - 2],
496                                tmp2[tmp2.Length - 1]));
497                    foreach (var asset in allAsset)
498                    {
499                        if (asset is AnimationClip &&
500                            asset.name != "__preview__mixamo.com")
501                        {
502                            string tag = tmp[tmp.Length - 1];
503                            animacionMap[tag].Add(asset as
504                                AnimationClip);
505                            break;
506                        }
507                    }
508                }
509            }
510        }
511    }
512
513    /// <summary>
514    /// Loads all the animation clips from the "Resources"
515    /// directory.
516    Each animation must be in a directory with the same name
517    as the type of gesture.
518    /// </summary>
519    private void loadClips()
520    {
521        try
522        {
523            string[] dir = Directory.GetDirectories(path);
524            foreach (string dirName in dir)
525            {
526                string[] animsPath = Directory.GetFiles(dirName)
527                ;
528                string[] tmp = dirName.Split('\\');
529                animacionMap.Add(tmp[tmp.Length - 1],
530                    new List<AnimationClip>());
531                foreach (string animPath in animsPath)
532                {
533
534                    string[] tmp2 = animPath.Split('\\');
535                    string a = Path.Combine(resourcePath,
536                        tmp2[tmp2.Length - 2],
537                        tmp2[tmp2.Length - 1]);
538                    var allAsset =
539                        AssetDatabase.LoadAllAssetsAtPath(
540                            Path.Combine(resourcePath,
541                                tmp2[tmp2.Length - 2],
542                                tmp2[tmp2.Length - 1]));
543                    foreach (var asset in allAsset)
544                    {
545                        if (asset is AnimationClip &&
546                            asset.name != "__preview__mixamo.com")
547                        {
548                            string tag = tmp[tmp.Length - 1];
549                            animacionMap[tag].Add(asset as
550                                AnimationClip);
551                            break;
552                        }
553                    }
554                }
555            }
556        }
557    }
558
559    /// <summary>
560    /// Loads all the animation clips from the "Resources"
561    /// directory.
562    Each animation must be in a directory with the same name
563    as the type of gesture.
564    /// </summary>
565    private void loadClips()
566    {
567        try
568        {
569            string[] dir = Directory.GetDirectories(path);
570            foreach (string dirName in dir)
571            {
572                string[] animsPath = Directory.GetFiles(dirName)
573                ;
574                string[] tmp = dirName.Split('\\');
575                animacionMap.Add(tmp[tmp.Length - 1],
576                    new List<AnimationClip>());
577                foreach (string animPath in animsPath)
578                {
579
580                    string[] tmp2 = animPath.Split('\\');
581                    string a = Path.Combine(resourcePath,
582                        tmp2[tmp2.Length - 2],
583                        tmp2[tmp2.Length - 1]);
584                    var allAsset =
585                        AssetDatabase.LoadAllAssetsAtPath(
586                            Path.Combine(resourcePath,
587                                tmp2[tmp2.Length - 2],
588                                tmp2[tmp2.Length - 1]));
589                    foreach (var asset in allAsset)
590                    {
591                        if (asset is AnimationClip &&
592                            asset.name != "__preview__mixamo.com")
593                        {
594                            string tag = tmp[tmp.Length - 1];
595                            animacionMap[tag].Add(asset as
596                                AnimationClip);
597                            break;
598                        }
599                    }
600                }
601            }
602        }
603    }
604
605    /// <summary>
606    /// Loads all the animation clips from the "Resources"
607    /// directory.
608    Each animation must be in a directory with the same name
609    as the type of gesture.
610    /// </summary>
611    private void loadClips()
612    {
613        try
614        {
615            string[] dir = Directory.GetDirectories(path);
616            foreach (string dirName in dir)
617            {
618                string[] animsPath = Directory.GetFiles(dirName)
619                ;
620                string[] tmp = dirName.Split('\\');
621                animacionMap.Add(tmp[tmp.Length - 1],
622                    new List<AnimationClip>());
623                foreach (string animPath in animsPath)
624                {
625
626                    string[] tmp2 = animPath.Split('\\');
627                    string a = Path.Combine(resourcePath,
628                        tmp2[tmp2.Length - 2],
629                        tmp2[tmp2.Length - 1]);
630                    var allAsset =
631                        AssetDatabase.LoadAllAssetsAtPath(
632                            Path.Combine(resourcePath,
633                                tmp2[tmp2.Length - 2],
634                                tmp2[tmp2.Length - 1]));
635                    foreach (var asset in allAsset)
636                    {
637                        if (asset is AnimationClip &&
638                            asset.name != "__preview__mixamo.com")
639                        {
640                            string tag = tmp[tmp.Length - 1];
641                            animacionMap[tag].Add(asset as
642                                AnimationClip);
643                            break;
644                        }
645                    }
646                }
647            }
648        }
649    }
650
651    /// <summary>
652    /// Loads all the animation clips from the "Resources"
653    /// directory.
654    Each animation must be in a directory with the same name
655    as the type of gesture.
656    /// </summary>
657    private void loadClips()
658    {
659        try
660        {
661            string[] dir = Directory.GetDirectories(path);
662            foreach (string dirName in dir)
663            {
664                string[] animsPath = Directory.GetFiles(dirName)
665                ;
666                string[] tmp = dirName.Split('\\');
667                animacionMap.Add(tmp[tmp.Length - 1],
668                    new List<AnimationClip>());
669                foreach (string animPath in animsPath)
670                {
671
672                    string[] tmp2 = animPath.Split('\\');
673                    string a = Path.Combine(resourcePath,
674                        tmp2[tmp2.Length - 2],
675                        tmp2[tmp2.Length - 1]);
676                    var allAsset =
677                        AssetDatabase.LoadAllAssetsAtPath(
678                            Path.Combine(resourcePath,
679                                tmp2[tmp2.Length - 2],
680                                tmp2[tmp2.Length - 1]));
681                    foreach (var asset in allAsset)
682                    {
683                        if (asset is AnimationClip &&
684                            asset.name != "__preview__mixamo.com")
685                        {
686                            string tag = tmp[tmp.Length - 1];
687                            animacionMap[tag].Add(asset as
688                                AnimationClip);
689                            break;
690                        }
691                    }
692                }
693            }
694        }
695    }
696
697    /// <summary>
698    /// Loads all the animation clips from the "Resources"
699    /// directory.
700    Each animation must be in a directory with the same name
701    as the type of gesture.
702    /// </summary>
703    private void loadClips()
704    {
705        try
706        {
707            string[] dir = Directory.GetDirectories(path);
708            foreach (string dirName in dir)
709            {
710                string[] animsPath = Directory.GetFiles(dirName)
711                ;
712                string[] tmp = dirName.Split('\\');
713                animacionMap.Add(tmp[tmp.Length - 1],
714                    new List<AnimationClip>());
715                foreach (string animPath in animsPath)
716                {
717
718                    string[] tmp2 = animPath.Split('\\');
719                    string a = Path.Combine(resourcePath,
720                        tmp2[tmp2.Length - 2],
721                        tmp2[tmp2.Length - 1]);
722                    var allAsset =
723                        AssetDatabase.LoadAllAssetsAtPath(
724                            Path.Combine(resourcePath,
725                                tmp2[tmp2.Length - 2],
726                                tmp2[tmp2.Length - 1]));
727                    foreach (var asset in allAsset)
728                    {
729                        if (asset is AnimationClip &&
730                            asset.name != "__preview__mixamo.com")
731                        {
732                            string tag = tmp[tmp.Length - 1];
733                            animacionMap[tag].Add(asset as
734                                AnimationClip);
735                            break;
736                        }
737                    }
738                }
739            }
740        }
741    }
742
743    /// <summary>
744    /// Loads all the animation clips from the "Resources"
745    /// directory.
746    Each animation must be in a directory with the same name
747    as the type of gesture.
748    /// </summary>
749    private void loadClips()
750    {
751        try
752        {
753            string[] dir = Directory.GetDirectories(path);
754            foreach (string dirName in dir)
755            {
756                string[] animsPath = Directory.GetFiles(dirName)
757                ;
758                string[] tmp = dirName.Split('\\');
759                animacionMap.Add(tmp[tmp.Length - 1],
760                    new List<AnimationClip>());
761                foreach (string animPath in animsPath)
762                {
763
764                    string[] tmp2 = animPath.Split('\\');
765                    string a = Path.Combine(resourcePath,
766                        tmp2[tmp2.Length - 2],
767                        tmp2[tmp2.Length - 1]);
768                    var allAsset =
769                        AssetDatabase.LoadAllAssetsAtPath(
770                            Path.Combine(resourcePath,
771                                tmp2[tmp2.Length - 2],
772                                tmp2[tmp2.Length - 1]));
773                    foreach (var asset in allAsset)
774                    {
775                        if (asset is AnimationClip &&
776                            asset.name != "__preview__mixamo.com")
777                        {
778                            string tag = tmp[tmp.Length - 1];
779                            animacionMap[tag].Add(asset as
780                                AnimationClip);
781                            break;
782                        }
783                    }
784                }
785            }
786        }
787    }
788
789    /// <summary>
790    /// Loads all the animation clips from the "Resources"
791    /// directory.
792    Each animation must be in a directory with the same name
793    as the type of gesture.
794    /// </summary>
795    private void loadClips()
796    {
797        try
798        {
799            string[] dir = Directory.GetDirectories(path);
800            foreach (string dirName in dir)
801            {
802                string[] animsPath = Directory.GetFiles(dirName)
803                ;
804                string[] tmp = dirName.Split('\\');
805                animacionMap.Add(tmp[tmp.Length - 1],
806                    new List<AnimationClip>());
807                foreach (string animPath in animsPath)
808                {
809
810                    string[] tmp2 = animPath.Split('\\');
811                    string a = Path.Combine(resourcePath,
812                        tmp2[tmp2.Length - 2],
813                        tmp2[tmp2.Length - 1]);
814                    var allAsset =
815                        AssetDatabase.LoadAllAssetsAtPath(
816                            Path.Combine(resourcePath,
817                                tmp2[tmp2.Length - 2],
818                                tmp2[tmp2.Length - 1]));
819                    foreach (var asset in allAsset)
820                    {
821                        if (asset is AnimationClip &&
822                            asset.name != "__preview__mixamo.com")
823                        {
824                            string tag = tmp[tmp.Length - 1];
825                            animacionMap[tag].Add(asset as
826                                AnimationClip);
827                            break;
828                        }
829                    }
830                }
831            }
832        }
833    }
834
835    /// <summary>
836    /// Loads all the animation clips from the "Resources"
837    /// directory.
838    Each animation must be in a directory with the same name
839    as the type of gesture.
840    /// </summary>
841    private void loadClips()
842    {
843        try
844        {
845            string[] dir = Directory.GetDirectories(path);
846            foreach (string dirName in dir)
847            {
848                string[] animsPath = Directory.GetFiles(dirName)
849                ;
850                string[] tmp = dirName.Split('\\');
851                animacionMap.Add(tmp[tmp.Length - 1],
852                    new List<AnimationClip>());
853                foreach (string animPath in animsPath)
854                {
855
856                    string[] tmp2 = animPath.Split('\\');
857                    string a = Path.Combine(resourcePath,
858                        tmp2[tmp2.Length - 2],
859                        tmp2[tmp2.Length - 1]);
860                    var allAsset =
861                        AssetDatabase.LoadAllAssetsAtPath(
862                            Path.Combine(resourcePath,
863                                tmp2[tmp2.Length - 2],
864                                tmp2[tmp2.Length - 1]));
865                    foreach (var asset in allAsset)
866                    {
867                        if (asset is AnimationClip &&
868                            asset.name != "__preview__mixamo.com")
869                        {
870                            string tag = tmp[tmp.Length - 1];
871                            animacionMap[tag].Add(asset as
872                                AnimationClip);
873                            break;
874                        }
875                    }
876                }
877            }
878        }
879    }
880
881    /// <summary>
882    /// Loads all the animation clips from the "Resources"
883    /// directory.
884    Each animation must be in a directory with the same name
885    as the type of gesture.
886    /// </summary>
887    private void loadClips()
888    {
889        try
890        {
891            string[] dir = Directory.GetDirectories(path);
892            foreach (string dirName in dir)
893            {
894                string[] animsPath = Directory.GetFiles(dirName)
895                ;
896                string[] tmp = dirName.Split('\\');
897                animacionMap.Add(tmp[tmp.Length - 1],
898                    new List<AnimationClip>());
899                foreach (string animPath in animsPath)
900                {
901
902                    string[] tmp2 = animPath.Split('\\');
903                    string a = Path.Combine(resourcePath,
904                        tmp2[tmp2.Length - 2],
905                        tmp2[tmp2.Length - 1]);
906                    var allAsset =
907                        AssetDatabase.LoadAllAssetsAtPath(
908                            Path.Combine(resourcePath,
909                                tmp2[tmp2.Length - 2],
910                                tmp2[tmp2.Length - 1]));
911                    foreach (var asset in allAsset)
912                    {
913                        if (asset is AnimationClip &&
914                            asset.name != "__preview__mixamo.com")
915                        {
916                            string tag = tmp[tmp.Length - 1];
917                            animacionMap[tag].Add(asset as
918                                AnimationClip);
919                            break;
920                        }
921                    }
922                }
923            }
924        }
925    }
926
927    /// <summary>
928    /// Loads all the animation clips from the "Resources"
929    /// directory.
930    Each animation must be in a directory with the same name
931    as the type of gesture.
932    /// </summary>
933    private void loadClips()
934    {
935        try
936        {
937            string[] dir = Directory.GetDirectories(path);
938            foreach (string dirName in dir)
939            {
940                string[] animsPath = Directory.GetFiles(dirName)
941                ;
942                string[] tmp = dirName.Split('\\');
943                animacionMap.Add(tmp[tmp.Length - 1],
944                    new List<AnimationClip>());
945                foreach (string animPath in animsPath)
946                {
947
948                    string[] tmp2 = animPath.Split('\\');
949                    string a = Path.Combine(resourcePath,
950                        tmp2[tmp2.Length - 2],
951                        tmp2[tmp2.Length - 1]);
952                    var allAsset =
953                        AssetDatabase.LoadAllAssetsAtPath(
954                            Path.Combine(resourcePath,
955                                tmp2[tmp2.Length - 2],
956                                tmp2[tmp2.Length - 1]));
957                    foreach (var asset in allAsset)
958                    {
959                        if (asset is AnimationClip &&
960                            asset.name != "__preview__mixamo.com")
961                        {
962                            string tag = tmp[tmp.Length - 1];
963                            animacionMap[tag].Add(asset as
964                                AnimationClip);
965                            break;
966                        }
967                    }
968                }
969            }
970        }
971    }
972
973    /// <summary>
974    /// Loads all the animation clips from the "Resources"
975    /// directory.
976    Each animation must be in a directory with the same name
977    as the type of gesture.
978    /// </summary>
979    private void loadClips()
980    {
981        try
982        {
983            string[] dir = Directory.GetDirectories(path);
984            foreach (string dirName in dir)
985            {
986                string[] animsPath = Directory.GetFiles(dirName)
987                ;
988                string[] tmp = dirName.Split('\\');
989                animacionMap.Add(tmp[tmp.Length - 1],
990                    new List<AnimationClip>());
991                foreach (string animPath in animsPath)
992                {
993
994                    string[] tmp2 = animPath.Split('\\');
995                    string a = Path.Combine(resourcePath,
996                        tmp2[tmp2.Length - 2],
997                        tmp2[tmp2.Length - 1]);
998                    var allAsset =
999                        AssetDatabase.LoadAllAssetsAtPath(
1000                            Path.Combine(resourcePath,
1001                                tmp2[tmp2.Length - 2],
1002                                tmp2[tmp2.Length - 1]));
1003                    foreach (var asset in allAsset)
1004                    {
1005                        if (asset is AnimationClip &&
1006                            asset.name != "__preview__mixamo.com")
1007                        {
1008                            string tag = tmp[tmp.Length - 1];
1009                            animacionMap[tag].Add(asset as
1010                                AnimationClip);
1011                            break;
1012                        }
1013                    }
1014                }
1015            }
1016        }
1017    }
1018
1019    /// <summary>
1020    /// Loads all the animation clips from the "Resources"
1021    /// directory.
1022    Each animation must be in a directory with the same name
1023    as the type of gesture.
1024    /// </summary>
1025    private void loadClips()
1026    {
1027        try
1028        {
1029            string[] dir = Directory.GetDirectories(path);
1030            foreach (string dirName in dir)
1031            {
1032                string[] animsPath = Directory.GetFiles(dirName)
1033                ;
1034                string[] tmp = dirName.Split('\\');
1035                animacionMap.Add(tmp[tmp.Length - 1],
1036                    new List<AnimationClip>());
1037                foreach (string animPath in animsPath)
1038                {
1039
1040                    string[] tmp2 = animPath.Split('\\');
1041                    string a = Path.Combine(resourcePath,
1042                        tmp2[tmp2.Length - 2],
1043                        tmp2[tmp2.Length - 1]);
1044                    var allAsset =
1045                        AssetDatabase.LoadAllAssetsAtPath(
1046                            Path.Combine(resourcePath,
1047                                tmp2[tmp2.Length - 2],
1048                                tmp2[tmp2.Length - 1]));
1049                    foreach (var asset in allAsset)
1050                    {
1051                        if (asset is AnimationClip &&
1052                            asset.name != "__preview__mixamo.com")
1053                        {
1054                            string tag = tmp[tmp.Length - 1];
1055                            animacionMap[tag].Add(asset as
1056                                AnimationClip);
1057                            break;
1058                        }
1059                    }
1060                }
1061            }
1062        }
1063    }
1064
1065    /// <summary>
1066    /// Loads all the animation clips from the "Resources"
1067    /// directory.
1068    Each animation must be in a directory with the same name
1069    as the type of gesture.
1070    /// </summary>
1071    private void loadClips()
1072    {
1073        try
1074        {
1075            string[] dir = Directory.GetDirectories(path);
1076            foreach (string dirName in dir)
1077            {
1078                string[] animsPath = Directory.GetFiles(dirName)
1079                ;
1080                string[] tmp = dirName.Split('\\');
1081                animacionMap.Add(tmp[tmp.Length - 1],
1082                    new List<AnimationClip>());
1083                foreach (string animPath in animsPath)
1084                {
1085
1086                    string[] tmp2 = animPath.Split('\\');
1087                    string a = Path.Combine(resourcePath,
1088                        tmp2[tmp2.Length - 2],
1089                        tmp2[tmp2.Length - 1]);
1090                    var allAsset =
1091                        AssetDatabase.LoadAllAssetsAtPath(
1092                            Path.Combine(resourcePath,
1093                                tmp2[tmp2.Length - 2],
1094                                tmp2[tmp2.Length - 1]));
1095                    foreach (var asset in allAsset)
1096                    {
1097                        if (asset is AnimationClip &&
1098                            asset.name != "__preview__mixamo.com")
1099                        {
1100                            string tag = tmp[tmp.Length - 1];
1101                            animacionMap[tag].Add(asset as
1102                                AnimationClip);
1103                            break;
1104                        }
1105                    }
1106                }
1107            }
1108        }
1109    }
1110
1111    /// <summary>
1112    /// Loads all the animation clips from the "Resources"
1113    /// directory.
1114    Each animation must be in a directory with the same name
1115    as the type of gesture.
1116    /// </summary>
1117    private void loadClips()
1118    {
1119        try
1120        {
1121            string[] dir = Directory.GetDirectories(path);
1122            foreach (string dirName in dir)
1123            {
1124                string[] animsPath = Directory.GetFiles(dirName)
1125                ;
1126                string[] tmp = dirName.Split('\\');
1127                animacionMap.Add(tmp[tmp.Length - 1],
1128                    new List<AnimationClip>());
1129                foreach (string animPath in animsPath)
1130                {
1131
1132                    string[] tmp2 = animPath.Split('\\');
1133                    string a = Path.Combine(resourcePath,
1134                        tmp2[tmp2.Length - 2],
1135                        tmp2[tmp2.Length - 1]);
1136                    var allAsset =
1137                        AssetDatabase.LoadAllAssetsAtPath(
1138                            Path.Combine(resourcePath,
1139                                tmp2[tmp2.Length - 2],
1140                                tmp2[tmp2.Length - 1]));
1141                    foreach (var asset in allAsset)
1142                    {
1143                        if (asset is AnimationClip &&
1144                            asset.name != "__preview__mixamo.com")
1145                        {
1146                            string tag = tmp[tmp.Length - 1];
1147                            animacionMap[tag].Add(asset as
1148                                AnimationClip);
1149                            break;
1150                        }
1151                    }
1152                }
1153            }
1154        }
1155    }
1156
1157    /// <summary>
1158    /// Loads all the animation clips from the "Resources"
1159    /// directory.
1160    Each animation must be in a directory with the same name
1161    as the type of gesture.
1162    /// </summary>
1163    private void loadClips()
1164    {
1165        try
1166        {
1167            string[] dir = Directory.GetDirectories(path);
1168            foreach (string dirName in dir)
1169            {
1170                string[] animsPath = Directory.GetFiles(dirName)
1171                ;
1172                string[] tmp = dirName.Split('\\');
1173                animacionMap.Add(tmp[tmp.Length - 1],
1174                    new List<AnimationClip>());
1175                foreach (string animPath in animsPath)
1176                {
1177
1178                    string[] tmp2 = animPath.Split('\\');
1179                    string a = Path.Combine(resourcePath,
1180                        tmp2[tmp2.Length - 2],
1181                        tmp2[tmp2.Length - 1]);
1182                    var allAsset =
1183                        AssetDatabase.LoadAllAssetsAtPath(
1184                            Path.Combine(resourcePath,
1185                                tmp2[tmp2.Length - 2],
1186                                tmp2[tmp2.Length - 1]));
1187                    foreach (var asset in allAsset)
1188                    {
1189                        if (asset is AnimationClip &&
1190                            asset.name != "__preview__mixamo.com")
1191                        {
1192                            string tag = tmp[tmp.Length - 1];
1193                            animacionMap[tag].Add(asset as
1194                                AnimationClip);
1195                            break;
1196                        }
1197                    }
1198                }
1199            }
1200        }
1201    }
1202
1203    /// <summary>
1204    /// Loads all the animation clips from the "Resources"
1205    /// directory.
1206    Each animation must be in a directory with the same name
1207    as the type of gesture.
1208    /// </summary>
1209    private void loadClips()
1210    {
1211        try
1212        {
1213            string[] dir = Directory.GetDirectories(path);
1214            foreach (string dirName in dir)
1215            {
1216                string[] animsPath = Directory
```

```
141
142     /// <summary>
143     /// Creates the CSV header.
144     /// </summary>
145     /// <returns>CSV header.</returns>
146     private string createHeader()
147     {
148         string s = "";
149         string[] coordinates = { "x", "y", "z" };
150
151         foreach (Transform bone in _bones)
152         {
153             foreach (string coordinate in coordinates)
154             {
155                 s += bone.name + "_pos" + coordinate + ",";
156             }
157             foreach (string coordinate in coordinates)
158             {
159                 s += bone.name + "_rot" + coordinate + ",";
160             }
161         }
162
163         s = s.Remove(s.Length - 1);
164
165         return s;
166     }
167 }
```

Listing J.1: Código de la herramienta MixamoDumper

Glosario

AI	Artificial Intelligence
Animator	Componente de Unity que funciona como una máquina de estados para la ejecución de animaciones
API	Application Programming Interface
API REST	Arquitectura de diseño utilizada para crear servicios web y comunicar diferentes sistemas
CART	Un árbol de decisión CART (Classification and Regression Trees) es un algoritmo de aprendizaje automático que utiliza árboles binarios para clasificar o predecir valores continuos
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Comma Separated Values
DCNN	Deep Convolutional Neural Network
FBX	El formato FBX es un tipo de archivo que contiene datos de objetos en 3D así como de animaciones
fork	Copia de un proyecto, generalmente software, que se hace con la intención de ser modificado y desarrollado, con posibilidad de independencia del proyecto original o con afán de mejorar el original
FPS	Frames Per Second
GPU	Graphics Processing Unit
gradient boosted decision tree	Un gradient boosted decision tree es un algoritmo de aprendizaje automático que utiliza múltiples árboles de decisión y optimización por gradiente para mejorar la precisión de las predicciones
IA	Inteligencia Artificial

iframe	Un iframe es un elemento HTML que permite incluir otro documento HTML dentro de una página web
IRs	Rayos Infrarrojos
isolation forest	Un isolation forest es un algoritmo de aprendizaje automático que utiliza árboles de decisión para detectar anomalías en los datos
KNN	K-Nearest Neighbors
LSTM	Long Short-Term Memory
metaverse	The metaverse is a set of digital spaces to socialize, learn, play and do other activities. (Definition by the company Meta)
metaverso	El metaverso es un conjunto de espacios digitales para socializar, aprender, jugar y realizar otras actividades. (Definición de la empresa Meta)
MMORPG	Massively Multiplayer Online Role-Playing Game
NPC	Non Playable Character
Random Forest	Un random forest es un algoritmo de aprendizaje automático que utiliza múltiples árboles de decisión para mejorar la precisión y la robustez de las predicciones
rig	Un rig es el resultado del rigging, una técnica usada en animación digital que permite crear estructuras alrededor de una malla para poder animar de forma más sencilla
RNN	Recurrent Neural Network
SDK	Software Development Kit
SVM	Support Vector Machine
TPU	Tensor Processing Unit
VR	Realidad Virtual
WSL	Windows Subsystem for Linux
YOLO	You Only Look Once

Licencia de uso del documento

Esta obra está bajo una licencia Creative Commons «Atribución-NoComercial-CompartirIgual 4.0 Internacional».

<http://creativecommons.org/licenses/by-sa/4.0/legalcode>.



Licencia de uso del código fuente

Los archivos de código fuente para generar este documento se encuentran en <https://github.com/FratosVR/Memoria-TFG> bajo la licencia [GPL-3.0](#)