



Universidad Rey Juan Carlos

Detección de objetos en escenas de fútbol con imágenes sintéticas

Memoria del Trabajo Fin de Máster
en Visión Artificial

Autor:

Luis Rosario Tremoulet

Tutor: José F. Vélez Serrano

Marzo 2023

Agradecimientos

Me gustaría aprovechar esta oportunidad para expresar mi gratitud a quienes me han apoyado a lo largo de este proyecto.

En primer lugar, quiero agradecer sinceramente a mi familia por su apoyo y ánimo.

También me gustaría dar las gracias a mi tutor, José F. Vélez, por su paciencia y sus comentarios. Sus puntos de vista han sido inestimables para dar forma a mi investigación y han permitido mejorar mis competencias en visión artificial.

Me gustaría agradecer sus contribuciones a todos los desarrolladores de software libre que han hecho posible este proyecto, como Blender, Python y YOLOv8.

Y, por supuesto, no puedo olvidarme de dar las gracias a mi cafetera por proporcionarme el combustible esencial para seguir adelante. Su dedicación a preparar siempre una taza de café perfecta me ha salvado la vida, y prometo mantenerla bien cuidada.

Resumen

La detección de objetos es una tarea fundamental de la visión por ordenador que consiste en identificar y localizar objetos dentro de una imagen o vídeo. Tiene una amplia gama de aplicaciones, como la vigilancia, los vehículos autónomos, la robótica ... Sin embargo, el entrenamiento de los modelos de detección de objetos requiere una gran cantidad de datos etiquetados, cuya recopilación puede llevar mucho tiempo y resultar costosa, sobre todo en dominios especializados como el deporte. Para hacer frente a este reto, los datos sintéticos han surgido como una alternativa prometedora a los datos reales.

Este documento pretende presentar un modelo de detección de objetos en escenas de fútbol entrenado solamente con imágenes sintéticas. El modelo tiene como objetivo de detectar: los jugadores de cada equipo, los porteros, el árbitro y la pelota. Se ha utilizado Blender, un software de modelización 3D para la creación de esas imágenes sintéticas, con el objetivo que se parezcan lo máximo a escenarios futbolísticos reales. Se ha utilizado un *script* personalizado en Python para generar automáticamente esas imágenes etiquetadas. Este *script* permite un control sobre diversas propiedades de la imagen, como la iluminación, las texturas y el ángulo de la cámara. El uso de datos sintéticos permite evitar problemas de privacidad, derechos de autor y cuestiones éticas que surgen cuando se utilizan imágenes reales. Por último, se ha utilizado un modelo preentrenado denominado YOLOV8 y se ha perfeccionado con el conjunto de datos sintéticos para adaptarlo a la tarea específica de detectar objetos relacionados con el fútbol.

Se ha conseguido crear un modelo de detección de objetos sobre escenas de fútbol que alcanza una elevada tasa de precisión sobre imágenes reales, superior al 92 %. Los resultados indican que los datos sintéticos generados con Blender pueden ser eficaces para entrenar modelos de detección de objetos y pueden constituir una alternativa rentable a los métodos tradicionales de recopilación de datos.

Índice general

1	Introducción	1
1.1	Motivación	1
1.2	Marco teórico	2
1.2.1	El aprendizaje profundo	3
1.2.2	La detección de objetos	3
1.2.3	El groundtruth y las anotaciones en detección de objetos	3
1.3	Estado del arte	4
1.3.1	Bases de datos	4
1.3.2	Modelos	5
1.4	Objetivos	6
1.4.1	Objetivos generales	6
1.4.2	Objetivos específicos	6
1.5	Estructura de la memoria	6
2	Análisis	8
2.1	Requisitos	8
2.1.1	Cualidad de las imágenes	8
2.1.2	Adaptabilidad del software	8
2.1.3	El groundtruth	9
2.1.4	Rapidez de generación	9
2.1.5	Resultados del modelo	10
2.2	Metodología	10
2.3	Blender	10
3	Diseño	13
3.1	Herramientas utilizadas	13
3.1.1	Python	13
3.1.2	Blender y su API	13
3.1.3	YoloV8	14
3.1.4	Conda	14
3.1.5	Paquetes	14
3.1.6	Notebook	15
3.2	Arquitectura del add-on	15
3.2.1	Selección de los modelos	17
3.2.2	Tomar medidas	18
3.2.3	Outlier del Blender	19
3.2.4	Creación de los objetos	21
3.2.5	Posición de los objetos	21

3.2.6	Orientación de los objetos	23
3.2.7	Morfología de los personajes	24
3.2.8	Texturas	24
3.2.9	Rigging	31
3.2.10	Iluminación	33
3.2.11	Cámara	33
3.2.12	Rendering y groundtruth	34
3.2.13	Limpiar la escena	36
3.2.14	Optimización	36
3.2.15	Interfaz gráfica	37
3.3	Anotación y modificación de las imágenes	38
3.3.1	Modificación de la resolución de las imágenes	38
3.3.2	Anotación de las imágenes	39
3.3.3	Verificación de las anotaciones	40
4	Experimentos y pruebas del modelo	41
4.0.1	Parámetros de un entrenamiento	41
4.0.2	Métricas de evaluación del modelo	42
4.0.3	Modelo con 24 clases	43
4.0.4	Modelo con 6 clases	44
4.0.5	Modelo con 4 clases	46
4.0.6	Modelo personalizado	48
5	Métricas del desarrollo	54
5.1	Tiempo dedicado	54
5.2	Métricas técnicas	55
6	Conclusiones	56
6.1	Objetivos cumplidos	56
6.2	Futuros trabajos	56
A	Guía de instalación y utilización	58
	Bibliografía	59

Índice de tablas

1	Tiempo de generación del <i>dataset</i> por el número de imágenes deseadas.	10
2	Probabilidades de selección del método de colocación de objetos.	22
3	Probabilidades de selección del método de rotación.	23
4	Probabilidades del color de pelo de cada personaje.	31
5	Probabilidades de poses para cada grupo de personajes.	32
6	Modelos propuestos por YoloV8.	41
7	Parámetros del modelo “Modelo_24clases”	44
8	Parámetros del modelo “Modelo_6clases_20epochs” (1) y “Modelo_6clases_30epochs” (2).	45
9	Parámetros del modelo “Val-Real-6clases”	49
10	Resultados del modelo “Val-Real-6clases” sobre imágenes reales.	52

Índice de figuras

1	Gráfico de la proporción de datos utilizados por la AI a lo largo del tiempo [10] [11].	2
2	Ejemplo de detección de objetos con cuadro delimitadores, etiqueta y confianza utilizando Yolo [18].	4
3	Renders propuestos por TRICTRAC y Google.	5
4	Ejemplo de segmentación semántica.	9
5	Resultado del tutorial del donut.	11
6	Resultado del tutorial del cuarto.	12
7	Diagrama de flujo del funcionamiento general de la solución.	16
8	Ejemplo de segmentación semántica.	17
9	Modelos Blender utilizado para los deportistas.	18
10	Coordenadas de los puntos claves del campo de fútbol.	18
11	<i>Outlier</i> del proyecto Blender.	20
12	Modelo virgen de un jugador de fútbol.	21
13	Método para obtener la posición del árbitro.	22
14	Métodos de colocación para los jugadores y pelota.	23
15	Resultado de la generación después haber colocado los objetos.	23
16	Resultado de la generación después haber aplicado la rotación a los objetos.	24
17	Textura final de la pelota.	25
18	Jugador de fútbol con sur fichero de textura.	26
19	Explicación de “TextureHorizontalNb.png”.	27
20	Probabilidades de colores y forma de las camisetas de fútbol La Liga 2022/2023.	28
21	Probabilidades de colores y forma de las camisetas de porteros y árbitros de la Liga 2022/2023.	30
22	Ejemplos de texturas generadas aleatoriamente.	31
23	Todas las poses que se pueden aplicar.	32
24	3 ejemplos de iluminaciones diferentes.	33
25	Localizaciones posibles de la cámara.	34
26	Resultados finales con sus imágenes verdaderas.	35
27	Interfaz gráfica de la generación de imagen.	38
28	Ejemplo de anotación de una imagen para YoloV8.	39
29	Resultado de “squares.py”.	40
30	Explicación del IOU (<i>Intersection over union</i>).	42
31	Explicación del <i>recall</i> y precisión.	43
32	Matriz de confusión del modelo “Modelo_24clases”.	44
33	Matriz de confusión del modelo “Modelo_6clases_20epochs”.	45

ÍNDICE DE FIGURAS

34 Matriz de confusión de “Modelo_6clases_20epochs” y “Modelo_6clases_30epochs”.	46
35 Matriz de confusión del modelo “Modelo_4clases”.	47
36 Curva de precisión de “Modelo_4clases”.	47
37 Curva de <i>recall</i> de “Modelo_4clases”.	48
38 Ejemplo de detección del modelo “Modelo_4clases” sobre una imagen real.	48
39 Comparación imagen sintética / real.	49
40 Matriz de confusión del modelo “Val-Real-6clases”.	50
41 Curva de precisión de “Val-Real-6clases”.	51
42 Curva de <i>recall</i> de “Val-Real-6clases”.	51
43 Gráfico que compara la puntuación F1 con el umbral de confianza del modelo “Val-Real-6clases”.	52
44 Resultado del modelo “Val-Real-6clases” sobre una imagen real.	53
45 Resultado del modelo “Val-Real-6clases” sobre una imagen real (2).	53

Algoritmos

1 Algoritmo para elegir el color del pantalón de los jugadores.	29
2 Algoritmo para elegir el color de los calcetines de los jugadores.	29

Nomenclatura

AI	Artificial intelligence - Inteligencia artificial.
API	Application program interface - Interfaz de programación de aplicaciones.
CNN	Convolutional neural network - Red neuronal convolucional.
CPU	Central processing unit - Unidad central de proceso.
DL	Deep learning - Aprendizaje profundo.
FN	False negatives - Falsos negativos.
FP	False positives - Falsos positivos.
GPU	Graphics processing unit - Unidad de procesamiento gráfico.
IOU	Intersection over union.
ML	Machine learning - Aprendizaje automático.
PNG	Portable Network Graphics.
RGB	Red green blue - Rojo verde azul.
TN	True negatives - Verdaderos negativos.
TP	True positives - Verdaderos positivos.
XML	Extensible Markup Language - Lenguaje de Marcado Extensible.

Capítulo 1

Introducción

En este capítulo se va a introducir un proyecto que explora el uso de datos sintéticos para entrenar modelos de detección de objetos en escenas de fútbol. Se explicará la motivación de este proyecto, el estado del arte de ese problema y objetivos del mismo.

Los capítulos siguientes tratarán con más detalle el método de trabajo utilizado, el software, los experimentos, sus métricas y los resultados.

1.1. Motivación

Los datos son un componente fundamental de los algoritmos de aprendizaje automático, ya que se utilizan para entrenar modelos que permiten realizar predicciones o tomar decisiones precisas [1] [2]. Sin datos suficientes, los modelos de ML pueden no funcionar bien y la calidad de las predicciones o decisiones pueden verse afectadas [3]. Por ello, la recopilación y el procesamiento de datos se han convertido en una parte fundamental de muchas aplicaciones de ML [4]. Esto es especialmente cierto en la visión artificial, donde se necesitan grandes volúmenes de datos etiquetados para entrenar modelos precisos para tareas como la detección de objetos o la clasificación de imágenes [5] [6].

Para el entrenamiento de los modelos de visión artificial, clásicamente, los datos reales han sido el tipo de datos más utilizado debido a su autenticidad y relevancia (ver Figura 1). Sin embargo, los datos reales tienen algunos puntos negativos, como por ejemplo la dificultad de adquirirlos y etiquetarlos en grandes cantidades. Por eso, han surgido como alternativa los datos sintéticos. Actualmente, las posibilidades gráficas de los ordenadores permiten generar imágenes sintéticas muy similares a las reales. Dichas imágenes tienen diversas muchas ventajas, como:

- Control y reproducibilidad: La generación de datos sintéticos permite a los investigadores tener un control total sobre el conjunto de datos y sus características, lo que posibilita experimentos más reproducibles [7].
- Privacidad y seguridad: Los datos sintéticos se pueden utilizar como una solución para preservar la privacidad cuando los datos reales no se pueden compartir debido a problemas de privacidad o seguridad [8].

- Flexibilidad y escalabilidad: Los datos sintéticos se pueden generar y modificar fácilmente para adaptarse a escenarios o casos de uso específicos, lo que los convierte en una solución escalable para diversas aplicaciones de ML [9].

By 2030, Synthetic Data Will Completely Overshadow Real Data in AI Models

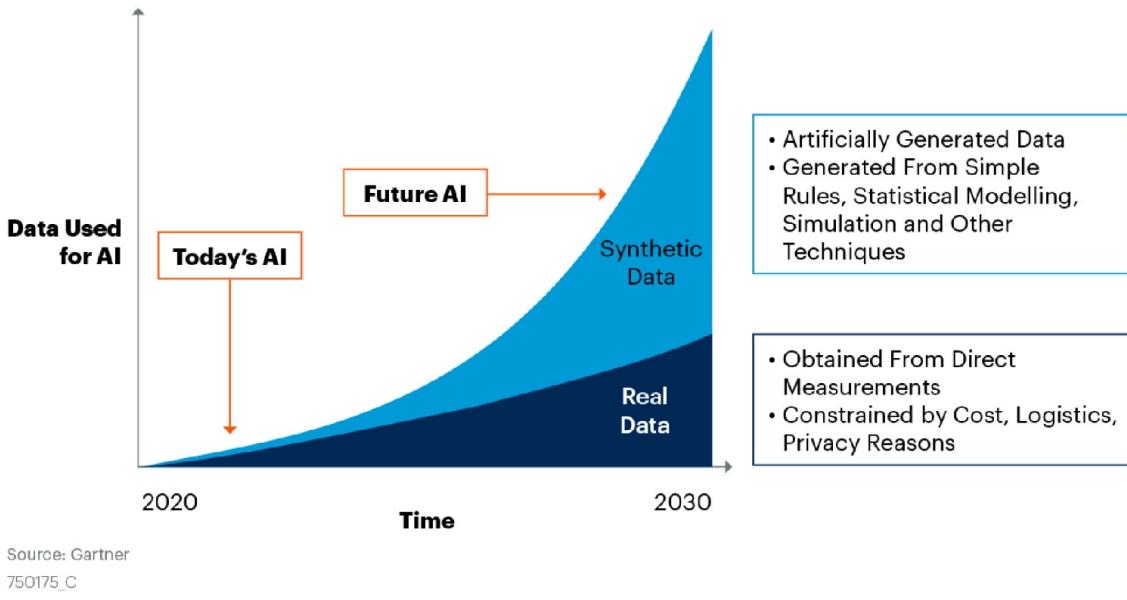


Figura 1: Gráfico de la proporción de datos utilizados por la AI a lo largo del tiempo [10] [11].

Además, según un informe publicado por Gartner en 2021 (ver Figura 1), los datos sintéticos deberían ser más utilizados que los datos reales a partir de 2030. Por eso se ha decidido investigar sobre la posibilidad y eficacia de un modelo de detección de objetos entrenado con imágenes sintéticas complejas. De hecho, se ha optado trabajar con imágenes de fútbol porque:

- Proceden de una situación real.
- Dependen de muchos parámetros (cambio de iluminación, ángulo de la cámara, muchos objetos en la imagen ...).
- Existen buenos modelos de entrenamiento con imágenes reales con los que se puede comparar el modelo de este proyecto [12] [13].

1.2. Marco teórico

Para poder entender el resto del trabajo hay algunos conceptos técnicos que son necesarios de conocer.

1.2.1. El aprendizaje profundo

En primer lugar, es crucial saber cómo funciona un modelo de DL. Un modelo de DL es una red neuronal artificial que puede aprender y hacer predicciones basadas en datos de entrada. Se compone de múltiples capas de nodos interconectados que procesan y transforman los datos a medida que fluyen por la red. Cada capa puede extraer diferentes características de los datos, lo que permite al modelo realizar predicciones cada vez más complejas y precisas [2] [14] [15].

Como se ha dicho anteriormente, para entrenar un modelo de DL, se necesita un gran conjunto de datos etiquetados para que el modelo pueda aprender. El modelo ajusta sus pesos y sesgos internos a través de un proceso llamado aprendizaje, que mejora iterativamente su capacidad para hacer predicciones precisas [16].

1.2.2. La detección de objetos

En un segundo lugar, se explicará cómo se suele realizar la detección de objetos usando DL. Un modelo de detección de objetos permite identificar y localizar objetos dentro de una imagen o vídeo (secuencia de imágenes). Así, usualmente, los modelos de DL dividen una imagen de entrada en regiones más pequeñas y predicen la probabilidad de que un objeto esté presente en cada región, junto con su ubicación y forma. Estas predicciones se mejoran a través de múltiples rondas de procesamiento y filtrado, dando como resultado un conjunto final de objetos detectados [17] [18]. Los modelos de detección de objetos como las R-CNN [19] o CornetNet [20] se utilizan habitualmente en aplicaciones como los coches autónomos [21], los sistemas de vigilancia [22] y los motores de búsqueda de imágenes [23].

1.2.3. El groundtruth y las anotaciones en detección de objetos

En último lugar, para poder entender mejor la detección de objetos en visión artificial se necesita saber cómo funciona la anotación de datos y el *groundtruth* (imagen verdadera). En la detección de objetos, la anotación de datos es el proceso de marcar una imagen o vídeo para indicar la presencia y ubicación de objetos dentro de la imagen. Suele hacerse dibujando un cuadro alrededor del objeto, asignándole una etiqueta y una puntuación de confianza [5] [24]. El proceso de anotación puede realizarse manualmente o con la ayuda de herramientas informáticas. Aunque el cuadro delimitador (ver Figura 2) sea la manera más utilizada para anotar las imágenes, existen otras maneras de anotar una imagen, como por ejemplo con polígonos, círculos, puntos ...

El *groundtruth*, también conocido como imagen verdadera, es la imagen resultante de la anotación. Es la imagen que se utiliza para entrenar y probar modelos de detección de objetos, ya que proporciona una representación clara y precisa de los objetos que se espera que el modelo detecte. El *groundtruth* también sirve de referencia para medir el rendimiento del modelo.

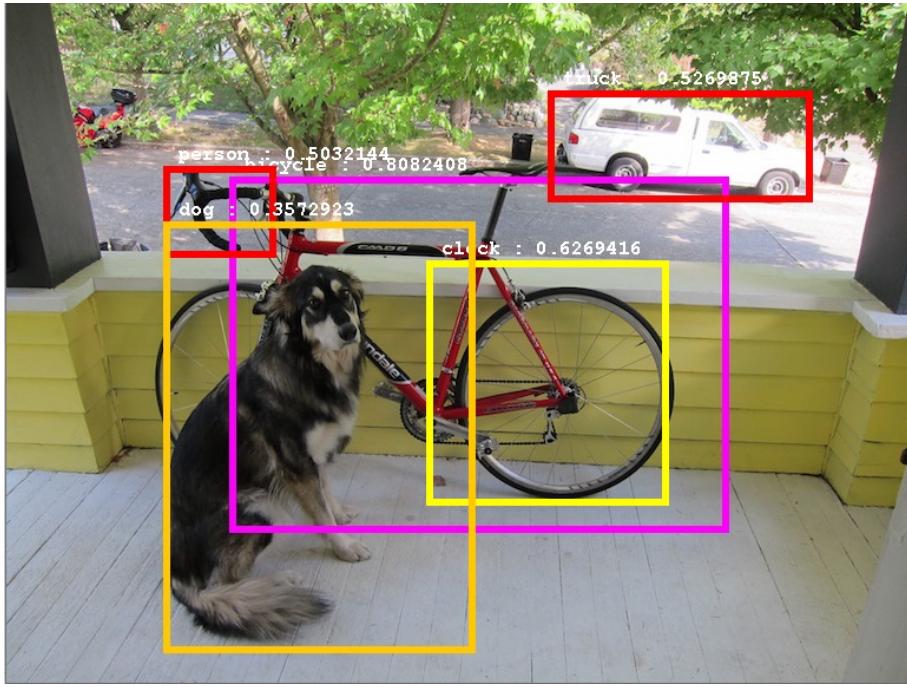


Figura 2: Ejemplo de detección de objetos con cuadro delimitadores, etiqueta y confianza utilizando Yolo [18].

1.3. Estado del arte

Se acaba de ver que la detección de objetos mediante el entrenamiento de datos sintéticos en el mundo de la visión por ordenador es bastante nueva. Esto es aún más cierto si se quiere trabajar en un entorno específico como el fútbol. Se han encontrado pocos trabajos sobre este tema. Además, las bases de datos sintéticas sobre el fútbol también son bastante difícil de encontrar.

1.3.1. Bases de datos

A parte de las bases de datos con imágenes de videojuegos como FIFA y PES [25], solo se han encontrado dos conjuntos de datos sintéticos sobre el tema del fútbol.

El primer *dataset* [26], propone un software que genera vídeos sintéticos de partidos de fútbol (ver Figura 3). Además de un vídeo generado, también se genera un *groundtruth* (fichero XML) donde se puede encontrar varias informaciones sobre la imagen generada. Por ejemplo, se puede encontrar en el fichero XML: las posiciones de las cámaras, las posiciones de los jugadores, la hora a la cual la imagen se ha generado ...

El segundo conjunto de datos procede de Google [27]. El objetivo principal de este proyecto de Google no es de ser un *dataset*, sino más bien un software que permite probar su modelo para jugar al fútbol. Este software permite controlar a los jugadores y poder jugar contra otros modelos (y también contra los de Google). El resultado es muy similar a un videojuego, pero las imágenes generadas por este software se han utilizado para entrenar algunos modelos de detección de objetos (como [28] [29]).

El problema con los dos *datasets* presentados (o los que provienen de imágenes de videojuego) es que no son flexibles. Es decir, que no se ha encordado un software publico

capaz de generar imágenes de fútbol donde se pueda modificar parámetros. De hecho, no se puede en estos ejemplos, cambiar la iluminación, el color de las camisetas de los jugadores, sus posiciones o el ángulo de la cámara. Esto permitiría poder generar imágenes adaptándose a escenarios específicos.

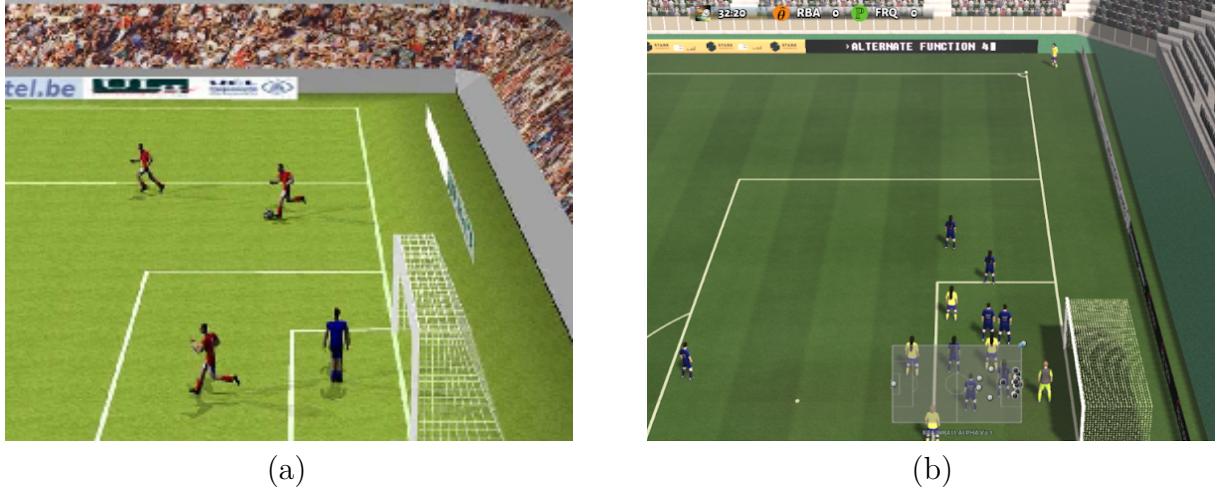


Figura 3: En (a) un render generado por el proyecto TRICTRAC [26]. En (b) un render generado por el software de Google [27].

1.3.2. Modelos

Como ya se ha dicho, de momento no existen muchos modelos de detección de objetos en el fútbol entrenados con imágenes sintéticas. Solo se han encontrado cuatro proyectos más o menos similares.

Los dos primeros estudios [30] y [31], publicados en 2019 y 2022, tratan de localizar el terreno de fútbol. Evidentemente, los dos modelos se han entrenado con imágenes sintéticas ([31] propone su propio generador de imágenes). El objetivo principal del primer estudio [30] es detectar los puntos claves del terreno. En cuando al segundo estudio [31], el objetivo es detectar el terreno para poder calibrar la cámara. Los dos proyectos obtienen muy buenos resultados sobre imágenes sintéticas y reales. Por desgracia, solo intentan detectar el terreno y no los objetos de la imagen como los jugadores o la pelota.

Los dos estudios [28] [29], crean un modelo cada uno capaz de detectar objetos. Más precisamente, el estudio [29] detecta los jugadores y las acciones y el estudio [28] hace un tracking de los jugadores. En cuanto a la base de datos, ambos estudios utilizan en mayor parte las imágenes de Google [27] que se ha visto anteriormente. Para los dos estudios, los resultados sobre imágenes sintéticas son casi excelentes. Por desgracia, ambos no dan mucho detalles sobre la detección con imágenes reales, incluso admiten que los resultados no son muy buenos.

Los cuatro proyectos concluyen con la misma idea: los datos sintéticos serán, dentro de poco, una mejor alternativa que los datos reales en algunas situaciones.

También, se habría podido analizar los proyectos [32] [33] [34] [35], pero no se ha encontrado bastante similitud con este estudio.

1.4. Objetivos

En esta sección se hablarán de los objetivos de esta memoria. Se dividirán en dos partes, los objetivos generales y los objetivos específicos.

1.4.1. Objetivos generales

El objetivo general de este trabajo es: obtener un modelo capaz de detectar objetos en escenas de fútbol. Esos objetos se dividirán en varias categorías como: los jugadores del equipo a domicilio, los jugadores del equipo exterior, el árbitro, el portero del equipo a domicilio, el portero del equipo exterior y la pelota. Las detecciones deberán de ser el resultado de un entrenamiento con únicamente imágenes sintéticas. Para obtener esas imágenes, se construirá un software.

1.4.2. Objetivos específicos

Para la parte de generación de imágenes, se deberá cumplir algunos objetivos. El software deberá ser capaz de generar ilimitadas imágenes. Las imágenes deberán ser lo más realistas posible. Algunos parámetros deberán ser configurables, como:

- La iluminación.
- El tamaño y forma de los personajes (jugadores, árbitro ...).
- Las texturas (por ejemplo, las camisetas de los jugadores).
- La localización de los objetos (pelota, personajes ...).
- La localización y ángulo de la cámara.
- La posición de los personajes (corriendo, tumbado, andando ...).

Además, el software deberá ser fácil de utilización (posibilidad de integrar una interfaz gráfica). Las imágenes generadas deberán cubrir un máximo de situaciones reales. Un objetivo será también el de generar esas imágenes lo más rápido posible. Y por último, cada generación de imagen deberá ser acompañada de una imagen verdadera (*groundtruth*) configurable que servirá para entrenar o evaluar el modelo.

1.5. Estructura de la memoria

Esta memoria se divide en siete capítulos, que se describen brevemente en los siguientes párrafos.

En el próximo capítulo llamado “Análisis” se hará una descripción completa del sistema que se desea construir. El capítulo será dividido en dos partes, los requisitos (que sean funcionales o no) y la metodología utilizada durante el proyecto.

El tercer capítulo llamado “Diseño”, será el capítulo principal de esta memoria. Tiene como objetivo describir la solución que se ha creado para conseguir los objetivos y los requisitos que se han planteado previamente. Se dividirá en tres secciones, donde se hablará de las herramientas utilizadas, del diseño del software y de su despliegue.

El cuarto capítulo, llamado “Experimentos y pruebas del modelo”, será el capítulo donde se incluirá descripciones de los experimentos que se han realizado y se discutirá sobre los resultados obtenidos. Además, se hablará del modelo utilizando y de la noción de aprendizaje por transferencia.

El quinto capítulo, llamado “Métricas del desarrollo”, se explicarán las medidas que se han acumulado durante el desarrollo del proyecto. Como, por ejemplo, cuánto tiempo se ha empleado en cada una de las fases del proyecto o de la complejidad informática del proyecto realizado.

Y en el último capítulo, llamado “Conclusiones”, se hablará de cómo se han cumplido los objetivos planteados al principio de la memoria. También se explicará cuáles serían los próximos pasos y trabajos futuros que podrían abordarse.

Además de esos capítulos, se incluirá una parte anexa que explicará cómo instalar y utilizar el código del proyecto.

Capítulo 2

Análisis

2.1. Requisitos

En esta sección se abordarán los requisitos necesarios del proyecto.

2.1.1. Cualidad de las imágenes

El primer requisito que deberá cumplir el proyecto está relacionado con la calidad de las imágenes generadas. Efectivamente, las imágenes generadas por el software deberán ser lo más realistas posible. En efecto, la calidad de los modelos (o objetos) deberá de ser lo más alta posible. Esto significa, también, que habrá que respetar el tamaño, la forma y las texturas de los objetos. De hecho, no sería realista utilizar, por ejemplo, un color rosa para las porterías.

2.1.2. Adaptabilidad del software

El segundo requisito de este proyecto será la adaptabilidad del software de generación de imágenes. Para poder representar el máximo número de escenas reales con el conjunto de datos sintéticos, será necesario que cada imagen sea única. En efecto, para que cada imagen de entrenamiento sea diferente del resto, debería haber un algoritmo que genere de manera aleatoria los diferentes parámetros necesarios. Estos parámetros podrían ser:

- La intensidad de la iluminación.
- El tipo de iluminación (sol, foco ...).
- La dirección de la iluminación (generación de sombra).
- El tamaño y forma de los personajes (jugadores, árbitro ...).
- La orientación de los personajes (mirando a la pelota, a un compañero ...).
- El color de las camisetas de los jugadores.
- El color de piel de los personajes.

- La posición de los personajes (corriendo, tumbado, andando ...).
- La localización de los objetos (pelota, personajes ...).
- La localización y ángulo de la cámara.

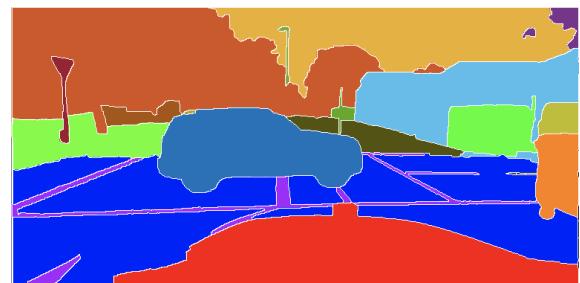
El requisito no es realizar una lista exacta de parámetros que generen una buena escena, sino encontrar esta lista y realizarla.

2.1.3. El groundtruth

Además de un conjunto de datos, que debe cubrir el mayor número posible de escenas con imágenes de buena calidad, el programa informático también deberá generar una imagen verdadera (*groundtruth*), para cada imagen. Como se ha visto antes, el *groundtruth* permitirá saber con precisión dónde están los objetos y a qué clase pertenecen. Esta imagen de referencia será similar a una segmentación semántica (ver Figura 4). Donde, cada color representará una clase (árbitro, jugador, pelota, etc.).



(a)



(b)

Figura 4: En (a) la imagen original. En (b) su segmentación semántica [36].

2.1.4. Rapidez de generación

Uno de los requisitos más importantes será la rapidez de generación de una imagen (y de su *groundtruth*). En efecto, el tiempo que empleara el software en generar una sola imagen es crucial. Si una imagen se genera en una hora, será casi imposible generar una base de datos de un tamaño adecuado para el entrenamiento en un tiempo razonable. Dependiendo del tamaño de la base de datos que se necesitara, no se tardará lo mismo en generar todo el *dataset*. Como se puede ver en la Tabla 1, si una imagen se genera en un minuto, se tardaría casi 2 semanas sin interrupción en generar un *dataset* de 20.000 imágenes.

2.2. METODOLOGÍA

1 imagen	500	1.000	5.000	20.000	100.000
1 segundo	8m20s	16m40s	1h23m20s	5h33m20s	1d3h46m40s
5 segundos	41m40s	1h23m20s	6h56m40s	1d3h46m40s	5d18h53m20s
10 segundos	1h23m20s	2h46m40s	13h53m20s	2d7h33m20s	11d13h46m40s
20 segundos	2h46m40s	5h33m20s	1d3h46m40s	4d15h6m40s	23d3h33m20s
1 minuto	8h20m	16h40m	3d11h20m	13d21h20m	69d10h40m
2 minutos	16h40m	1d9h20m	6d22h40m	27d18h40m	138d21h20m

Tabla 1: Tiempo de generación del *dataset* por el número de imágenes deseadas.

Además, el tiempo de ejecución depende del hardware utilizado, no se tardará lo mismo si se utiliza una CPU o una GPU [37]. Por eso, la generación de imagen debería de ser bastante rápida para que funcione en la mayoría de los ordenadores.

2.1.5. Resultados del modelo

Uno de los requisitos más importante, el que permitirá concluir ese trabajo, es el resultado. Es por ello que se fija como requisito obtener una tasa de verdaderos positivos superior al 80 % y una tasa de falsos positivos inferior al 20 %. En la detección de objetos, los verdaderos positivos representan los números de casos positivos correctamente identificados. Los falsos positivos representan las instancias que se predijeron positivas, pero que en realidad fueron negativas. Los falsos positivos permiten verificar que no se hagan muchas más detecciones que el número de objetos en toda la imagen.

2.2. Metodología

Durante la elaboración de este proyecto, se ha utilizado una metodología de sprints inspirada en la metodología ágil Scrum. Esta metodología permite gestionar un proyecto implicando ciclos de desarrollo iterativos de duración determinada (aquí semanales). Cada lunes se definían sobre papel las tareas a realizar durante el periodo de sprint (*to do list*). Al final de cada sprint, se organizaba una revisión para evaluar los progresos y garantizar que los objetivos del proyecto seguían alineados con la visión general del proyecto. Al final de cada parte importante del proyecto, se organizaba una reunión con el tutor para que pueda aconsejar y dar su opinión sobre el progreso del proyecto. Se ha elegido esta metodología porque se centra en el desarrollo iterativo, lo que permite responder rápidamente a los cambios y adaptarse a la evolución de los requisitos.

2.3. Blender

Para generar las imágenes se decidió utilizar el software libre Blender¹. Es un software de creación 3D gratuito y de código abierto que permite a los usuarios modelar, animar y renderizar gráficos 3D. Considerado como uno de los mejores softwares de su categoría [38] [39], Blender está disponible para todos los sistemas (Windows, MacOS y Linux).

¹Blender:<https://www.blender.org/>

Se utiliza ampliamente para crear imágenes generadas por ordenador, efectos visuales y animaciones para películas, juegos y otros medios. Blender admite diversas funciones, como simulación, composición y seguimiento del movimiento, lo que lo convierte en una herramienta versátil y muy completa para artistas y diseñadores 3D.

Se ha decidido utilizar Blender porque es un software muy completo, y sobre todo gratis y de código abierto. Además, se puede utilizar Blender desde el código, lo que permite una automatización de generación de imágenes.

Blender es un software bastante difícil de usar. Por eso existen tantos tutoriales para familiarizarse con el programa [40] [41]. Después de haber hablado con expertos del software (diseñadores 3D), se ha decidido hacer el famoso tutorial del donut² para empezar a manejar el software. Se puede ver el resultado del tutorial en la Figura 5. Para mejorar las bases de Blender se ha decidido continuar con otro tutorial, el del cuarto 3D³, donde se puede ver el resultado en la Figura 6.

Después de haber dedicado más o menos una semana para comprender cómo funcionaba Blender, se ha comenzado el diseño del generador de imágenes. Este generador se explicará en detalle en el próximo capítulo.

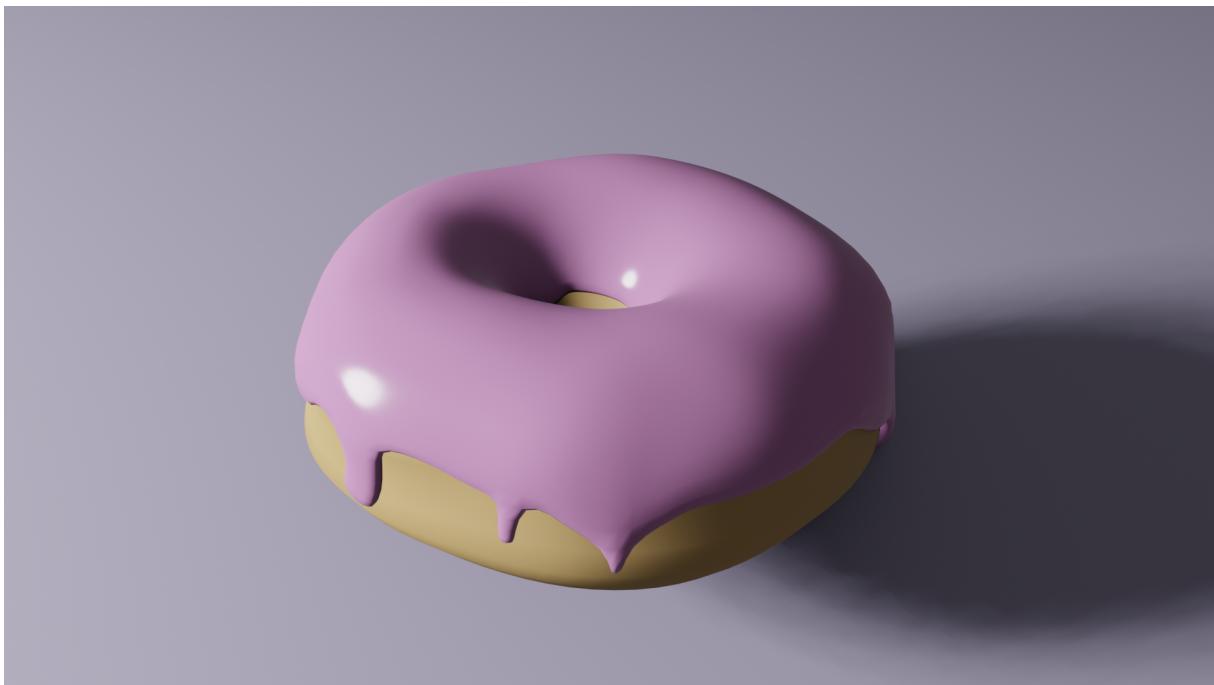


Figura 5: Resultado del tutorial del donut.

²Donut:https://www.youtube.com/watch?v=nIoX0plUvAw&list=PLjEaoINr3zgFX8ZsChQVQsuDSjEqdWMAD&ab_channel=BlenderGuru

³Cuarto 3D:https://www.youtube.com/watch?v=yCHT23A6aJA&ab_channel=3DGreenthorn



Figura 6: Resultado del tutorial del cuarto.

Capítulo 3

Diseño

En este capítulo se hablará, en un primer lugar, de las descripciones de las herramientas que se han utilizado. En un segundo lugar, se verá en detalle la arquitectura del software de generación de imágenes.

3.1. Herramientas utilizadas

3.1.1. Python

El único lenguaje de programación que se ha utilizado para este proyecto ha sido Python¹. Python3 es un lenguaje de programación de alto nivel muy utilizado para el desarrollo web, la informática científica, el análisis de datos, la inteligencia artificial y otras aplicaciones. Es un lenguaje interpretado, lo que significa que el código se ejecuta línea a línea en lugar de compilarse previamente. Python3 es la última versión del lenguaje de programación Python y se publicó en 2008. Incluye muchas nuevas características y mejoras con respecto a Python2, como una sintaxis mejorada, una mejor gestión de la memoria y bibliotecas estándar mejoradas.

Python3 no solamente se ha utilizado para automatizar la generación de imágenes con Blender. También, se ha utilizado para anotar las imágenes automáticamente a partir del *groundtruth* y para crear y entrenar el modelo de detección de objetos. Python3 ha sido con Blender una de las herramientas claves de este proyecto.

3.1.2. Blender y su API

Como ya se ha hablado en el capítulo anterior, Blender ha servido para generar las imágenes. Junto con Python3 se ha elaborado un *script* (fichero) que automatice la generación de imágenes. Para desarrollar esta automatización se ha tenido que trabajar con un paquete esencial para el uso de Python3 con Blender: el API de Blender nominado “BPY”.

BPY significa Blender Python API, es un módulo de Python3 que permite automatizar tareas e interactuar con la funcionalidad de Blender a través de *scripts* de Python.

¹Python3:<https://www.python.org/downloads/>

3.1. HERRAMIENTAS UTILIZADAS

Proporciona acceso a las estructuras de datos subyacentes de Blender, permitiendo a manipular objetos 3D, materiales, texturas, animaciones y mucho más. BPY también permite la creación de *scripts* personalizados y complementos, que pueden ampliar la funcionalidad de Blender más allá de sus características incorporadas. Los usuarios avanzados y los desarrolladores suelen utilizar esta API para crear flujos de trabajo automatizados o herramientas personalizadas para tareas específicas.

3.1.3. YoloV8

Para el modelo, se ha decidido utilizar hacer entrenamiento por transferencia con YoloV8². YoloV8 [18] es un modelo de detección de objetos (ver Figura 2) basado en DL que utiliza una única CNN para identificar y clasificar objetos en imágenes o videos. Se introdujo el 10 de enero 2023 como actualización de las versiones anteriores de Yolo. La versión 8 de Yolo se presentó como más rápida y precisa para una mejor detección de objetos, segmentación y clasificación de imágenes.

El aprendizaje por transferencia es una técnica que permite adaptar un modelo preentrenado a una nueva tarea, ajustando sus parámetros en un nuevo conjunto de datos. Mediante el aprendizaje por transferencia, YoloV8 puede adaptarse rápidamente a nuevas tareas de detección de objetos, ajustando con precisión sus pesos preentrenados en el nuevo conjunto de datos. Esto reduce significativamente el tiempo y los recursos (tamaño de la base de datos) necesarios para entrenar un nuevo modelo desde cero. Para intentar conseguir buenos resultados con pocas imágenes se ha decidido hacer entrenamiento por transferencia con YoloV8.

3.1.4. Conda

Para poder preparar el modelo con mucha más rapidez, se ha utilizado Conda³. Conda es un popular gestor de paquetes y entornos para Python que permite instalar y gestionar fácilmente paquetes y dependencias. Se utiliza a menudo en proyectos de ciencia de datos y ML. Además, Conda se puede utilizar para crear y gestionar entornos virtuales para diferentes proyectos, lo que ayuda a mantener la coherencia entre diferentes entornos. El uso de Conda es especialmente útil cuando se entrena un modelo de ML en una GPU, ya que permite instalar y configurar fácilmente los controladores y las dependencias de la GPU.

3.1.5. Paquetes

Como se ha visto en la subsección que habla de Conda, se ha necesitado paquetes de Python para poder realizar el proyecto. Un paquete es una forma de organizar módulos relacionados entre sí, lo que facilita la gestión y reutilización del código. Los más importantes han sido:

- Random⁴: Para generar variables aleatorias.

²[YoloV8:https://github.com/ultralytics/ultralytics](https://github.com/ultralytics/ultralytics)

³[Conda:https://docs.conda.io/en/latest/](https://docs.conda.io/en/latest/)

⁴[Random:https://docs.python.org/3/library/random.html](https://docs.python.org/3/library/random.html)

- Numpy⁵: Para la computación numérica y para poder trabajar con matrices.
- Pillow⁶: Permite abrir, manipular y guardar muchos formatos de archivo de imagen diferentes.
- OpenCV⁷: Para el procesamiento de imágenes y vídeo (para el análisis y anotación de las imágenes verdaderas).
- PyTorch⁸: Proporciona soporte para el cálculo tensorial con aceleración por GPU (para el entrenamiento con YoloV8).

3.1.6. Notebook

Finalmente, la última herramienta utilizada ha sido Notebook. Un Notebook, o un Jupyter Notebook⁹, es un entorno informático interactivo basado en la web que permite crear y compartir documentos que contienen código en vivo, ecuaciones, visualizaciones y texto narrativo. Es compatible con una amplia gama de lenguajes de programación (como Python3), y proporciona una plataforma flexible para el análisis de datos, la computación científica y el ML. Los Notebooks pueden ejecutarse localmente o en servicios basados en la nube, y permiten la creación rápida de prototipos y la colaboración entre investigadores y científicos de datos.

La principal ventaja de utilizar Notebook, es la posibilidad de ejecutar solamente fragmentos de código, o ejecutar el código fragmento por fragmento. Esto es muy útil cuando se quiere entrenar un modelo.

3.2. Arquitectura del add-on

En esta sección se explicará en detalle la arquitectura del *add-on* (módulo complementario) que se ha realizado. Además, para entender mejor la estructura del *add-on* se ha realizado un *flowchart* o diagrama de flujo (ver Figura 7). Un diagrama de flujo es una representación gráfica de un proceso o algoritmo. Se puede utilizar para visualizar los pasos y el flujo de un programa Python, facilitando su comprensión y comunicación a los demás. Este *flowchart* tiene solamente como objetivo de explicar de manera general la estructura. Se verá en detalle cada etapa de la arquitectura del módulo complementario en las subsecciones futuras.

⁵Numpy:<https://numpy.org/>

⁶Pillow:<https://pillow.readthedocs.io/en/stable/>

⁷OpenCV:<https://opencv.org/>

⁸PyTorch:<https://pytorch.org/>

⁹Jupyter Notebook:<https://jupyter.org/>

3.2. ARQUITECTURA DEL ADD-ON

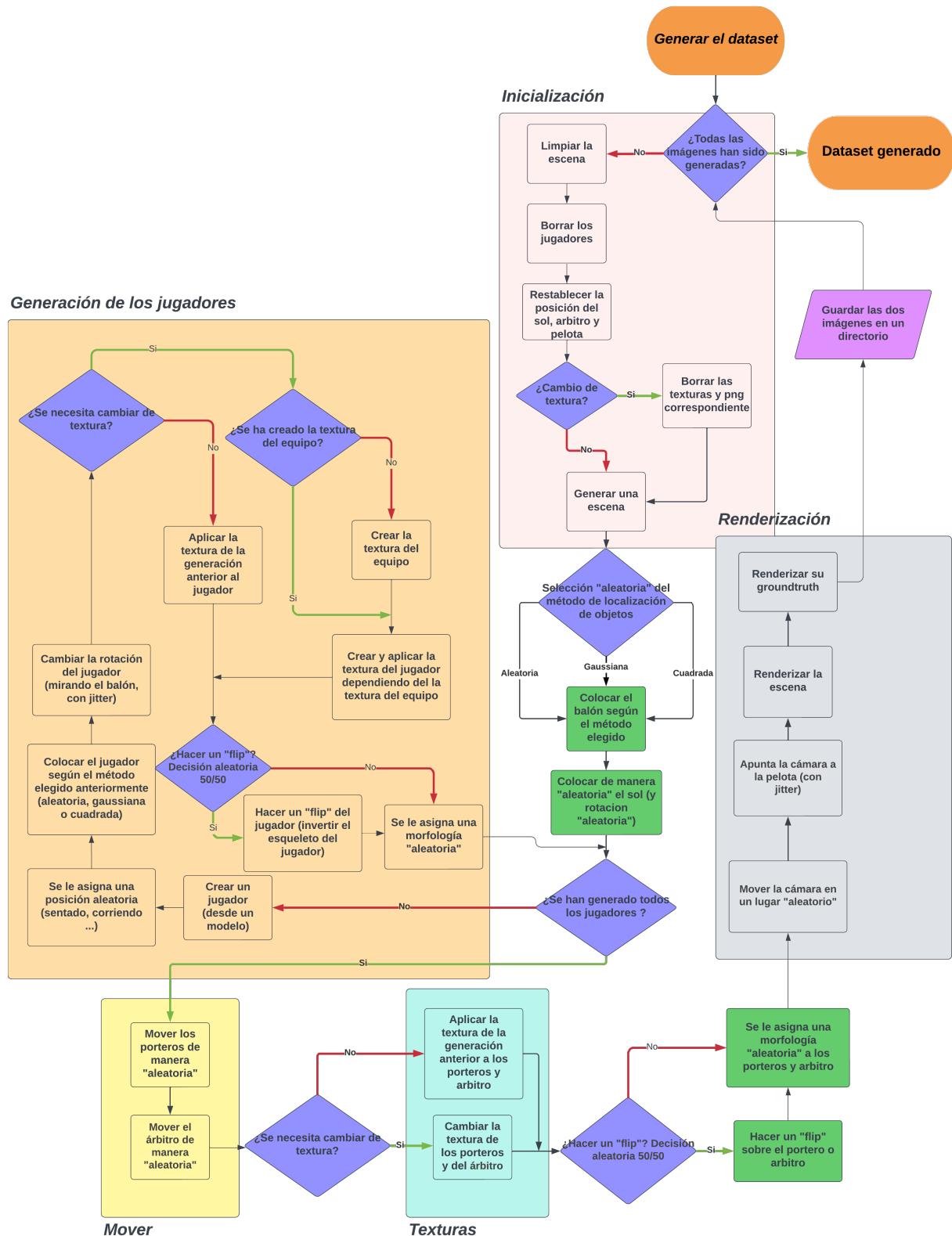


Figura 7: Diagrama de flujo del funcionamiento general de la solución.

3.2.1. Selección de los modelos

Antes de empezar la programación del módulo complementario, se han elegido los modelos con los cuales se iba a trabajar. Se han elegidos modelos “*low poly*” de campo de fútbol, de pelota y de personajes (que servirán como jugadores, árbitro y portero). Un modelo *low poly* es un modelo 3D que utiliza un número reducido de polígonos para representar un objeto complejo. Se utiliza a menudo en videojuegos, simulaciones y otras aplicaciones en tiempo real porque requiere menos potencia de cálculo para renderizar.

Cuando se generan imágenes, el uso de un modelo de baja poligonización puede ser beneficioso porque puede reducir drásticamente la cantidad de computación necesaria para renderizar una imagen. Sin embargo, cabe señalar que el uso de modelos de baja poligonización también puede dar lugar a una pérdida de detalle y realismo, sobre todo en el caso de formas complejas o escenas con muchos objetos. Los modelos de baja poligonización harán que la generación de la base de datos requiera mucho menos tiempo.

También es importante elegir modelos con proporciones reales. Por ejemplo, es importante elegir un estadio de fútbol con un tamaño de campo proporcional a las porterías.

Entonces, era necesario encontrar modelos *low poly* de buena calidad y con dimensiones proporcionales. Para el estadio de fútbol se ha encontrado un modelo muy realista y gratuito¹⁰ (ver Figura 8). Desgraciadamente, no se ha encontrado ningún modelo de personajes gratuito de buena calidad, por lo que se recurrió a un modelo de pago. En efecto, era necesario encontrar un modelo de deportista con el que fuera fácil cambiar su textura. Además, debía estar equipado con un *rigging* sencillo pero completo. El *rigging* es el proceso de añadir un esqueleto digital a un modelo 3D para que pueda moverse y posar de forma realista, lo cual es importante para crear modelos de personajes donde se pueda mover las partes del cuerpo. Entonces, se ha encontrado un total de seis modelos de deportista en un solo fichero¹¹ (ver Figura 9).

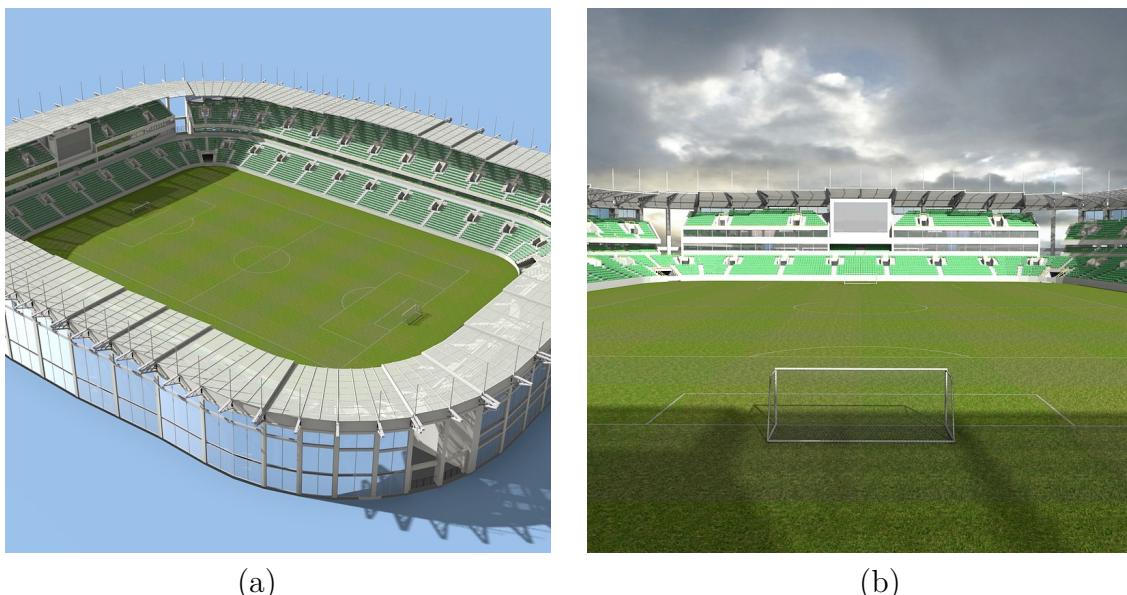


Figura 8: Dos puntos de vista del modelo de campo de fútbol utilizado.

¹⁰Modelo campo de fútbol:<https://creazilla.com/nodes/67469-football-stadium-3d-model>

¹¹Modelo deportistas:<https://sketchfab.com/3d-models/football-soccer-players-animated-rigged-14a393bdace245718c8e172c1b31628b>

3.2. ARQUITECTURA DEL ADD-ON

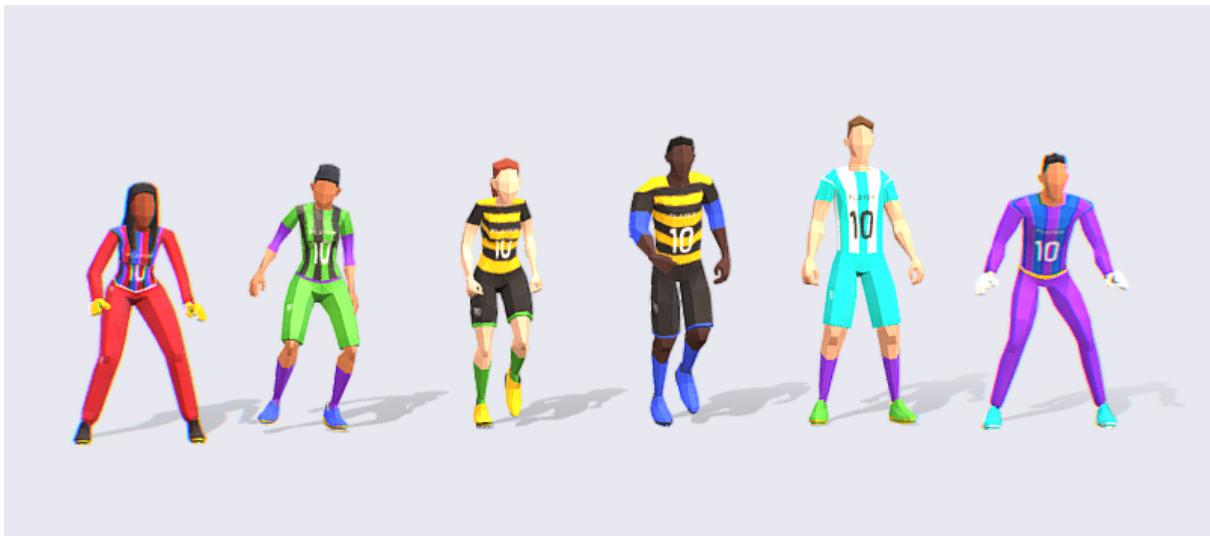


Figura 9: Modelos Blender utilizado para los deportistas.

3.2.2. Tomar medidas

Tras importar los modelos, se ha tenido que verificar que las proporciones de las dimensiones eran correctas. Se ha tenido que medir el tamaño de los personajes y compararlos con el estadio de fútbol para redimensionarlos proporcionalmente. Además, se ha tomado las coordenadas de cada punto importante del campo, como el centro o las esquinas (ver Figura 10). El hecho de haber tomado las dimensiones nos permitirá más tarde colocar los objetos en el campo de forma precisa.

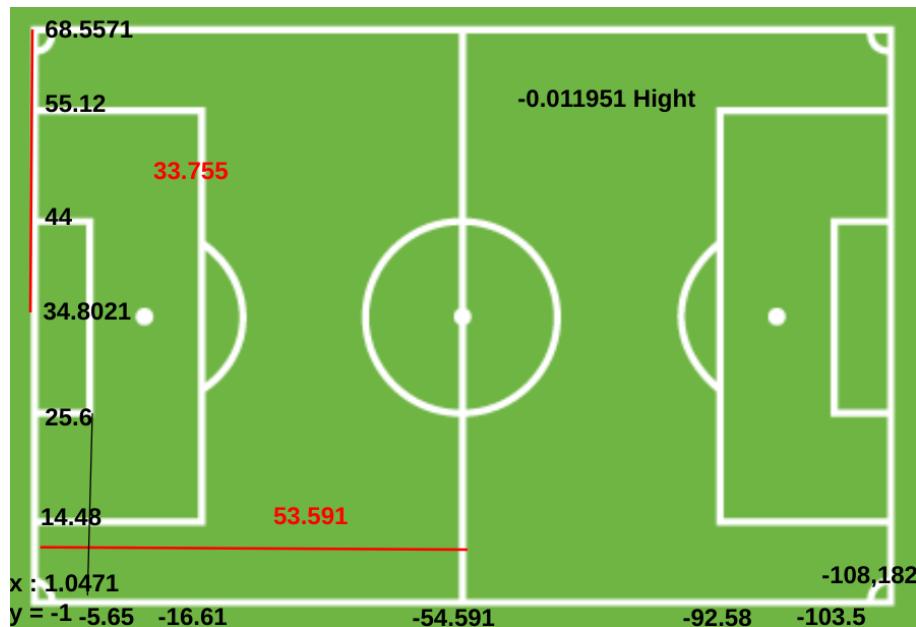


Figura 10: Coordenadas de los puntos claves del campo de fútbol.

3.2.3. Outlier del Blender

Para empezar con el módulo complementario, se ha decidido organizar el proyecto Blender. De hecho, Blender cuenta con un sistema llamado *Outlier*, que enumera todos los objetos y datos de una escena, permitiendo a los usuarios navegar, seleccionar y manipularlos fácilmente. Proporciona una vista jerárquica de los elementos, como cámaras, luces, mallas, etc., y ofrece diversas opciones de filtrado y visualización para una gestión eficaz del flujo de trabajo. Entonces, se ha decidido organizar el proyecto Blender en varias colecciones. Las colecciones corresponden a carpetas en lenguaje Blender.

Para una mejor organización se han separado los objetos en diferentes colecciones principales llamadas (azul en la Figura 11):

- “*Cameras*”: Donde se guardan las cámaras.
- “*Lights*”: Donde se guardan las luces.
- “*Stadium*”: Donde se guardan los objetos que construyen el estadio de fútbol.
- “*ModelPlayer*”: Donde se guardan los modelos de los personajes.
- “*Players*”: Donde se generan los personajes a cada generación.

Esta organización es muy importante para poder añadir, encontrar o modificar objetos fácilmente. De hecho, como se puede ver en la Figura 11, es esencial tener una organización clara y bien estructurada porque uno puede perderse rápidamente.

3.2. ARQUITECTURA DEL ADD-ON

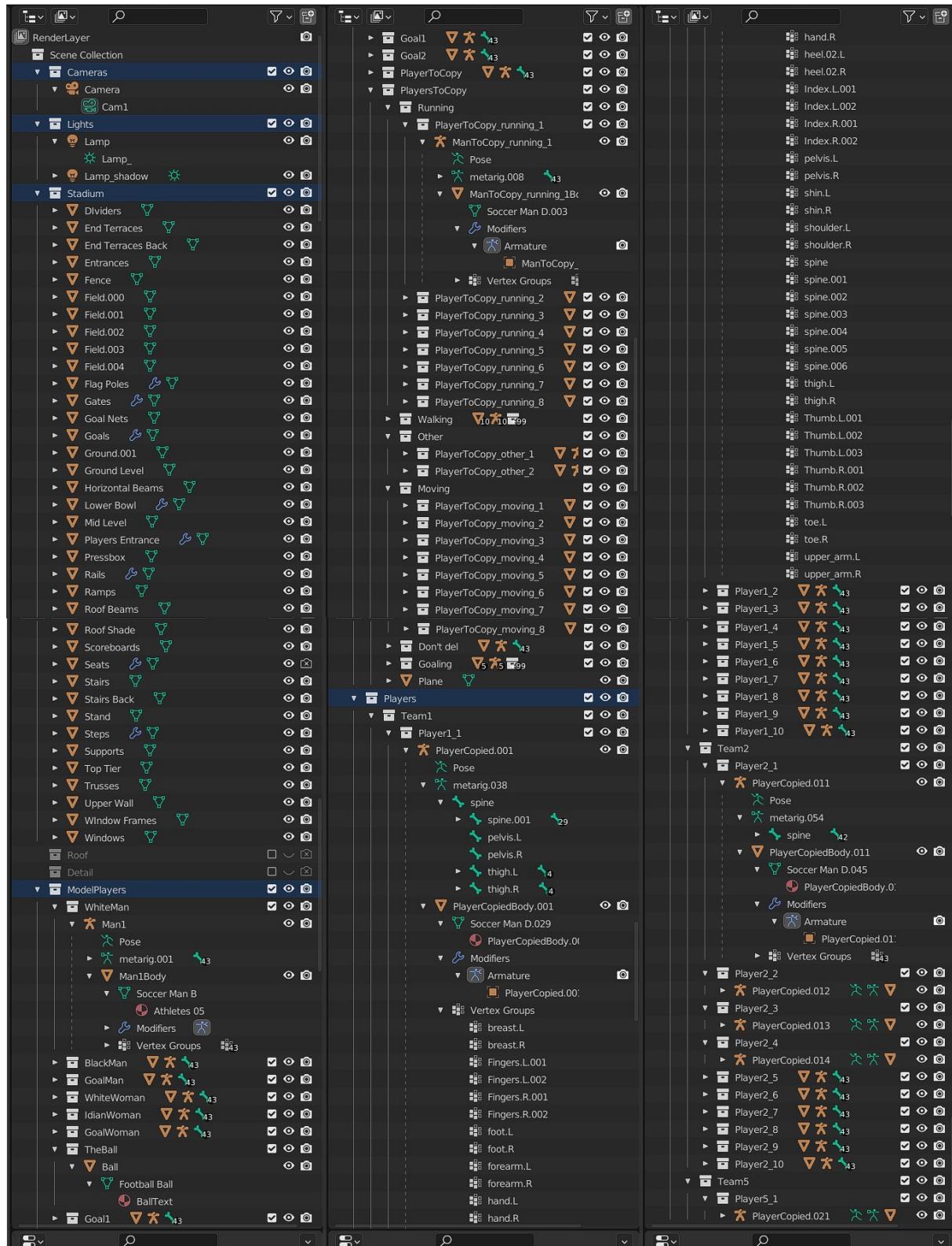


Figura 11: *Outlier* del proyecto Blender.

3.2.4. Creación de los objetos

Se han dedicado las primeras líneas de código a duplicar los jugadores de fútbol. Para cada generación de imagen se generan 20 jugadores (10 para cada equipo). Esos jugadores se duplican desde un modelo virgen. La Figura 12 muestra el modelo básico que se ha utilizado para la duplicación. En las próximas subsecciones, se verá como aplicar una textura, cambiar la morfología, cambiar el esqueleto ... de cada jugador.

Cada jugador duplicado se almacena en la colección adecuada y se le atribuye un nombre como: "Jugador_1_7". Donde, el primer dígito del nombre representa el equipo al que el jugador pertenece y el segundo dígito a su número de jugador. Para una buena organización se ha decidido almacenar todos los jugadores duplicados en una colección llamada "*Players*" que se separa en dos colecciones llamadas "*Team1*" o "*Team2*".

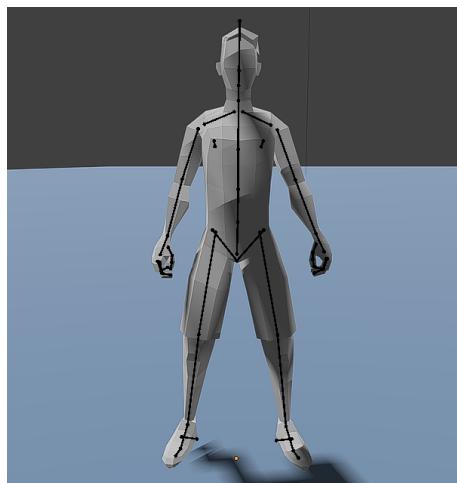


Figura 12: Modelo virgen de un jugador de fútbol.

Una vez la generación acabada y la imagen generada, cada jugador será destruido para poder ser generado a la próxima iteración. El árbitro funciona de la misma manera que los jugadores, pero se almacena en la colección "*Team5*". Al contrario, la pelota y los porteros no serán creados y destruidos, sino solamente desplazado y reinicializados a cada generación. Esto se explicará en la subsección de optimización.

3.2.5. Posición de los objetos

Una vez los jugadores generados, se necesita desplazar los objetos en el campo de fútbol. La localización de los objetos es muy importante, permite, si está bien echo, cubrir un máximo de situaciones reales. Tras reflexionar y analizar varios partidos de fútbol, se ha decidido desarrollar distintos métodos para colocar los objetos. Los jugadores y la pelota se colocarán según el mismo método, mientras que el árbitro y los porteros tendrán su propio algoritmo de movimiento.

A partir de ahora se considera que el eje X corresponde a la longitud del campo y que el eje Y corresponde a su anchura.

Empecemos por el árbitro, se ha decidido colocar el árbitro en el campo de fútbol de manera probabilística (gracias a dos gaussianas). De hecho, se ha observado que el árbitro

3.2. ARQUITECTURA DEL ADD-ON

(en situaciones reales) se sitúa casi siempre de la misma manera. El árbitro siempre intenta mantenerse cerca del centro para el eje X, y para el eje Y cerca del balón. Como se puede ver en la Figura 13, la gaussiana del eje X (gaussiana azul) está centrada sobre la pelota y tiene como anchura un valor de $1/8.93$ (≈ 0.11) veces el tamaño del terreno. Para la gaussiana del eje Y (roja), está simplemente centrada en el centro del terreno y tiene como anchura un valor de $1/6.751$ (≈ 0.15) veces el tamaño del terreno. Además, las gaussianas tienen como límite el principio y final los valores del tamaño del terreno de fútbol. Este método permite, según las observaciones realizadas, tener una posición del árbitro lo más realista posible.

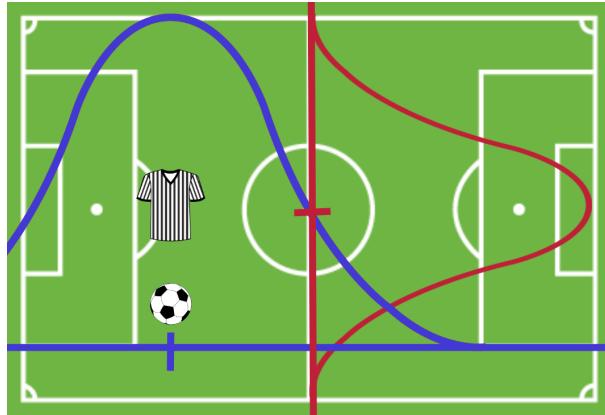


Figura 13: Método para obtener la posición del árbitro.

Para la localización de los porteros, se ha empleado otro método. Este método, mucho más sencillo que el del árbitro, consiste en colocar los porteros en su área de penalti (más algunos metros) al azar. Sin embargo, para la pelota y los jugadores el método de colocación es bastante más complejo. Efectivamente, en un primer lugar, se escoge de manera aleatoria entre tres métodos (ver Tabla 2 para ver las probabilidades). Dependiendo del método escogido, la pelota y los jugadores no serán colocados de la misma manera. De nuevo, esto se ha hecho para intentar abarcar el mayor número de escenas posibles.

Gaussiana	Rectangular	Aleatoria
66.5 %	28.5 %	5 %

Tabla 2: Probabilidades de selección del método de colocación de objetos.

Cuando el método aleatorio se ha escogido, los jugadores y la pelota se colocan de manera totalmente aleatoria en el terreno de fútbol. El método rectangular funciona más o menos de la misma manera, los objetos se colocan aleatoriamente dentro de un rectángulo delimitado. Este rectángulo tiene una longitud y una anchura aleatoria (con dimensiones mínimas y máximas) y se centra también aleatoriamente. La última manera de colocar los jugadores y la pelota, es la manera gaussiana. Este método funciona casi igualmente a el del árbitro. Se basa sobre dos gaussianas (eje X e Y) centradas aleatoriamente y con una anchura aleatoria. Por supuesto, para cada método existen valores mínimos y máximos, de modo que no sea posible que el balón o que un jugador se sitúe fuera del campo. Los tres métodos se explican de manera simple y gráfica en la Figura 14, donde los puntos rojos y azules representan los jugadores (o pelota).

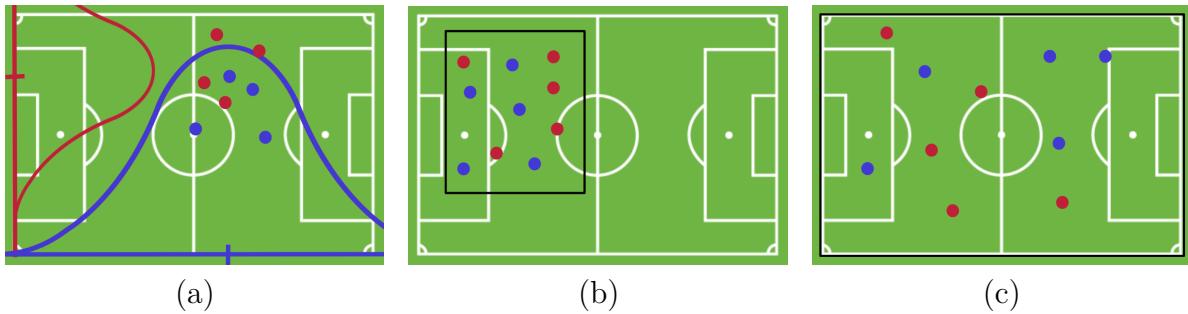


Figura 14: En (a) el método gaussiano. En (b) el método cuadrado. En (c) el método aleatorio.

Una vez todos los objetos colocados se obtiene un resultado similar al de la Figura 15.



Figura 15: Resultado de la generación después haber colocado los objetos.

3.2.6. Orientación de los objetos

Después de haber colocado todos los objetos, se ha decidido ocuparse de su rotación. La rotación solo se aplica a los personajes. Efectivamente, no importa la rotación de la pelota porque no cambia su aspecto. Se ha decidido orientar los personajes con tres métodos diferentes. Para cada personaje se elige un método donde se pueden ver las probabilidades en la Tabla 3.

Método 1 (Aleatorio)	Método 2 (Con <i>jitter</i>)	Método 3 (Con <i>jitter</i> / 2)
5 %	47.5 %	47.5 %

Tabla 3: Probabilidades de selección del método de rotación.

El método aleatorio, como se puede adivinar, aplica una rotación totalmente aleatoria al personaje. El segundo método, hace mirar el personaje hacia la pelota con una variación o ruido llamado “*jitter*”. El objetivo aquí es que no todos los personajes miren exactamente a la pelota, sino aproximadamente. El *jitter* tiene un valor por defecto de 60 grados (se puede cambiar). El ruido aplicado será, entonces, una variación aleatoria entre 0 y el valor

3.2. ARQUITECTURA DEL ADD-ON

del *jitter*. Antes de aplicar este ruido se necesita calcular el ángulo entre el personaje y la pelota. Este ángulo se ha obtenido mediante la fórmula (1). Donde $a=(xa, ya)$ es la localización de un jugador y $b=(xb, yb)$ la localización de la pelota.

$$\alpha = \arccos \left[(xa \cdot xb + ya \cdot yb) / \left(\sqrt{(xa^2 + ya^2)} \cdot \sqrt{(xb^2 + yb^2)} \right) \right] \quad (1)$$

El tercer método es el mismo que el segundo sino que el valor del *jitter* se divide por 2. Esto permite tener una diversidad de ángulos, siempre con el objetivo de cubrir un máximo de escenas posibles. Una vez aplicado esas rotaciones a todos los personajes, se obtiene un resultado como el de la Figura 16.



Figura 16: Resultado de la generación después haber aplicado la rotación a los objetos.

3.2.7. Morfología de los personajes

Una vez la rotación aplicada a todos los personajes, se ha decidido modificar aleatoriamente la altura y la anchura de cada personaje. Esta modificación se hace dentro de un cierto rango. Esto se aplica tanto a los jugadores de fútbol como al árbitro y los porteros.

3.2.8. Texturas

Esta parte llamada “Texturas” es seguramente la más importante de esta sección. Primero, se ha decidido aplicar una textura clásica a la pelota de fútbol y no cambiarla a cada generación (ver Figura 17).



Figura 17: Textura final de la pelota.

Al contrario, se ha decidido que los personajes cambiarán de textura a cada generación. Se explicará esto en dos partes. Primeramente, se explicará como se generan las texturas de la ropa, y segundamente como se modifican las texturas propias al personaje (color de piel, color del pelo ...).

Para empezar, es esencial entender cómo funcionan las texturas en Blender. Se utilizan para añadir detalles a los objetos 3D mediante la aplicación de imágenes o patrones. Esas imágenes que se aplican a los modelos 3D, suelen ser en formato PNG. Las imágenes se mapean en el modelo 3D basándose en las coordenadas UV de la malla del modelo. En este caso, como se trata de modelos, “*low poly*” es bastante fácil aplicar un color sobre la ropa o la parte del cuerpo deseada. Para ello, se ha tenido que encontrar qué parte del modelo estaba asociada a qué parte del archivo PNG. Como se puede ver en la Figura 18, la parte de color violeta (tercera sección) está asociada al color de los calcetines, la parte verde (penúltima sección) a los zapatos... Así pues, se han numerado todas las secciones, teniendo:

- 1: El pelo.
- 2: La piel.
- 3: Los calcetines.
- 4: Camiseta espalda.
- 5: Camiseta frente 1 (rayas).
- 6: Camisera frente 2.
- 7: Pantalón.
- 8: Zapatos.
- 9: Clavos de los zapatos.

3.2. ARQUITECTURA DEL ADD-ON

- 10, 11 y 12: Número + nombre.

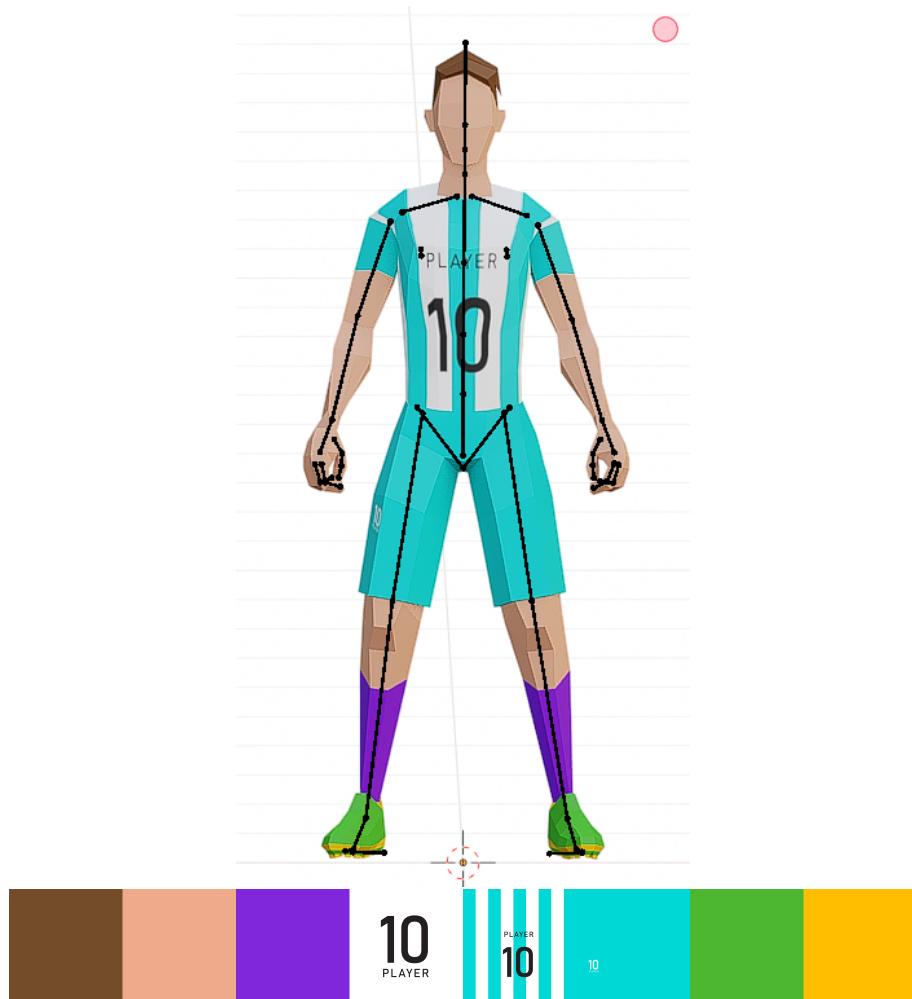


Figura 18: Un jugador de fútbol con su fichero PNG correspondiente.

Antes de pasar al siguiente apartado, es necesario explicar qué es él RGB. RGB son las siglas rojo, verde y azul, es un modelo de color utilizado en imagen digital. Representa los colores como combinaciones de valores de rojo, verde y azul en un rango de 0-255. La notación [0, 0, 0] en RGB representa la ausencia de color, donde los tres canales de color tienen el valor mínimo de 0. Esto significa que el píxel de la imagen no tiene contribución de ninguno de los colores rojo, verde y azul, lo que resulta en una apariencia de color negro.

Una vez cada parte enumerada, se han creado dos ficheros PNG que servirán de base a todas las otras texturas. Esos ficheros llamados “TextureHorizontalNb.png” y “Texture-VerticalNb.png”, son copias del fichero PNG que se ve en la Figura 18 sino que se han modificado los colores. Por ejemplo, la primera sección (la del pelo) se ha cambiado por un valor RGB de [0, 0, 1], la segunda sección (piel) se ha cambiado por un valor RGB de [0, 0, 2] ... Esto permitirá, más tarde, poder generar texturas cambiando con código Python el color de una sección en particular muy fácilmente. Se puede ver el resultado de “TextureHorizontalNb.png” en la Figura 19. Aunque parezca una imagen totalmente negra, cada sección tiene un color diferente (pero invisible al ojo humano). Se ha creado dos ficheros, para poder generar texturas con camisetas rayadas horizontalmente y verticalmente.

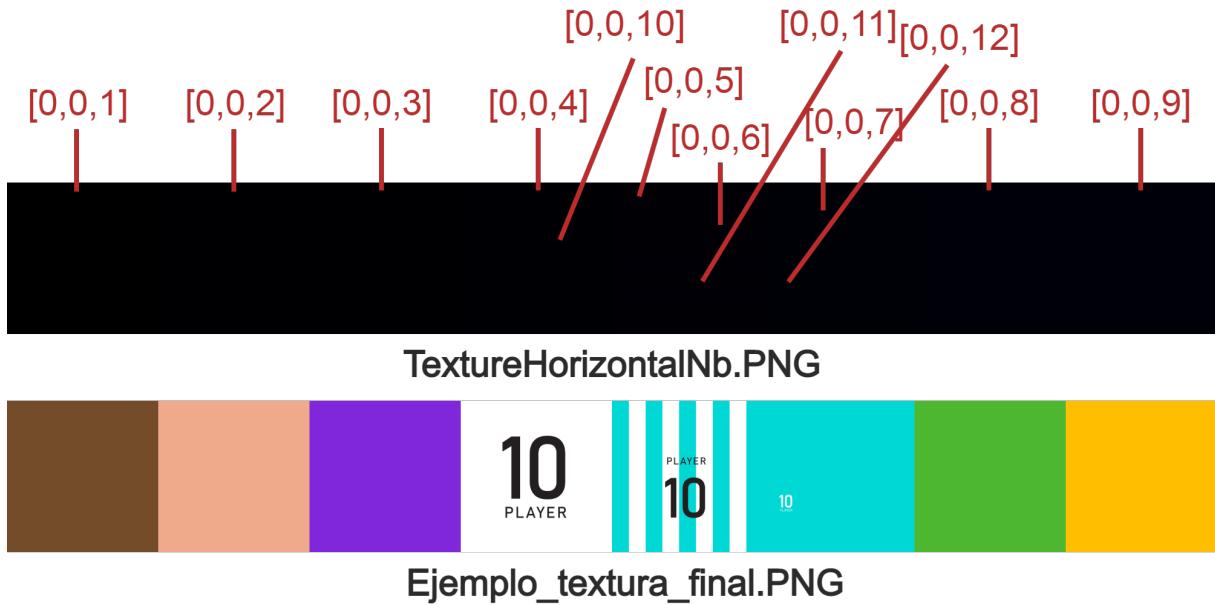


Figura 19: Explicación de “TextureHorizontalNb.png”.

Una vez los archivos “TextureHorizontalNb.png” y “TextureVerticalNb.png” creados, se ha decidido el método de selección de colores para la ropa. Para la creación de la ropa, los personajes se han dividido en cinco grupos:

- Grupo 1: Los jugadores del equipo local (portero excluido).
- Grupo 2: Los jugadores del equipo exterior (portero excluido).
- Grupo 3: Portero del equipo local.
- Grupo 4: Portero del equipo exterior.
- Grupo 5: Árbitro.

A cada generación de imagen, se crea una textura de ropa diferente para cada grupo. Esta selección incluye el color de: la camiseta, del pantalón, de los calcetines y del número y nombre sobre la camiseta. Cada grupo tendrá un método diferente de selección de colores.

Para el grupo uno y dos (jugadores, porteros excluidos), se ha analizado el color de la ropa de los equipos de fútbol de la Liga¹² del año 2022/2023. Los resultados que se han obtenido (gracias a La Liga¹³) se pueden observar en la Figura 20.

¹²La Liga:<https://www.laliga.com/en-GB>

¹³Camisetas de la Liga:<https://planetafobal.com/notas/camisetas-de-la-liga-espanola-2022-2023/>



Figura 20: Probabilidades de colores y forma de las camisetas de fútbol La Liga 2022/2023.

Para la Figura 20, se han analizado las 53 camisetas del año 2022/2023 de La Liga. Como se puede observar, se han contabilizado 106 colores principales. Por ejemplo, el color verde se ha visto 7 veces sobre 53 camisetas. Con esas estadísticas se ha podido programar un método de selección de color para la camiseta de los jugadores. Para cada generación de imágenes, se generan 2 texturas de camisetas (uno para cada equipo), teniendo una forma y un color aleatorio, con probabilidades iguales a las estadísticas obtenidas en la Figura 20. Por ejemplo, una camiseta tiene una probabilidad de $\frac{14}{53}$ de ser rayada y una probabilidad de $\frac{78}{2809}$ de ser blanca y azul ($\frac{24}{106} * \frac{13}{106} = \frac{78}{2809}$).

Para el color del pantalón y calcetines de los jugadores, se ha analizado también los uniformes de la Liga del año 2022/2023. Se puede observar los resultados en los algoritmos 1 y 2. Por ejemplo, para el Algoritmo 1 se puede ver que si el color de la camiseta es uniforme, el pantalón tendrá una probabilidad de $\frac{1}{4}$ de ser del mismo color (sino, se elige con el color con las mismas probabilidades de las camisetas).

Algoritmo 1 Algoritmo para elegir el color del pantalón de los jugadores.

```
def elegir color pantalón():
    if camiseta uniforme:
        if 25%:
            pantalón mismo color
        else 75%:
            pantalón color "aleatorio"
    else (rayada):
        if 90%:
            if 50%:
                pantalón mismo color que raya 1
            else 50%:
                pantalón mismo color que raya 2
        else 10%:
            pantalón color "aleatorio"
```

Algoritmo 2 Algoritmo para elegir el color de los calcetines de los jugadores.

```
def elegir color calcetines():
    if 75%:
        if camiseta uniforme:
            calcetines mismo color
        else (rayada):
            if 50%:
                calcetines mismo color que raya 1
            else 50%:
                calcetines mismo color que raya 2
    else 25%:
        calcetines color "aleatorio"
```

Para el grupo 3 y 4 (porteros), el método es casi el mismo que el de los jugadores. Sino que, los colores elegidos tienen otras probabilidades (ver Figura 21). Además, las camisetas de los porteros siempre se generan con un solo color (no tienen camisetas rayadas). Para el pantalón, primeramente, su modelo es diferente, es largo y no corto, segundamente, el pantalón tiene una probabilidad de 90 % de ser del mismo color que la camiseta.

Para el último grupo (el árbitro), el método sigue el mismo sino que también tiene probabilidades de selección de colores diferentes (ver Figura 21). El árbitro también tendrá solamente camisetas uniformes. Además, el color de su pantalón y de sus calcetines (y zapatos) serán siempre negros.

3.2. ARQUITECTURA DEL ADD-ON



(a)

(b)

Figura 21: Probabilidades de colores y forma de las camisetas de porteros y árbitros de la Liga 2022/2023. En (a) los de los porteros. En (b) los de los árbitros.

Para acabar con las texturas, se atribuye un color para el número y nombre de la camiseta de los porteros y jugadores (grupo 1, 2, 3 y 4). Dependiendo del color principal de la camiseta, el número y nombre serán de color blanco o negro. Para saber qué color se elige, se ha utilizado la luminancia relativa. Permite calcular el brillo que tiene un color para el ojo humano, basado en la intensidad de luz percibida. Se calcula mediante la fórmula (2), donde R, G y B son los valores RGB de un color. Si el resultado del color principal de la camiseta del personaje es superior a 128, su número y nombre tendrán un color blanco. Al contrario, si ese valor es inferior o igual a 128, su número y nombre tendrán un color negro. Esto se hace para que el número y nombre se contrasten con el color principal de la camiseta.

$$y = 0,2126 * R + 0,7152 * G + 0,0722 * B \quad (2)$$

Una vez la textura del grupo generado, se crea una textura única para cada personaje. Para cada personaje se genera un fichero PNG a partir del de su grupo. Una vez copiado, se cambia el color de las secciones que influyen sobre la piel, el pelo, los zapatos y los clavos de los zapatos.

Para esta etapa, se ha analizado más de 200 jugadores y árbitros de ligas de fútbol europeas¹⁴¹⁵. Se ha concluido que la piel de los jugadores y porteros era más o menos de

¹⁴Todos los árbitros FIFA: https://en.wikipedia.org/wiki/List_of_FIFA_international_referees

¹⁵Todos los jugadores de ligue 1: https://www.worldfootball.net/players_list/fra-ligue-1-2022-2023/nach-name/1/

color clara a un 50 % y oscura a otro 50 %. Para los árbitros, se ha concluido que tenían la piel clara a un 95 % y oscura a un 5 %.

Para el color del pelo se ha decidido usar las mismas probabilidades para todos los personajes. Se pueden ver esas probabilidades en la Tabla 4.

Piel	Cabello negro	Cabello castaño	Cabello rubio	Cabello pelirrojo
Piel clara	50 %	50 %	0 %	0 %
Piel oscura	18 %	36 %	36 %	10 %

Tabla 4: Probabilidades del color de pelo de cada personaje.

Una vez que se ha decidido de los colores de cada sección, se atribuye a cada color un valor RGB. Ese valor RGB será seleccionado aleatoriamente de una lista. Por ejemplo, si se ha decidido que la camiseta de un equipo sea azul, un tipo de azul será elegido entre 10 valores como [176,196,222] o [6,72,120]. Una vez que cada sección tiene su valor RGB correspondiente, se cambian esos valores sobre un fichero PNG y se guarda ese fichero. Para cada generación de imágenes, 28 ficheros PNG se crean (20 para los jugadores, 2 para los porteros, 1 para el árbitro y 5 para los modelos de grupo). Se puede ver algunos resultados en la Figura 22 (primera columna son los jugadores, segunda los árbitros y tercera los porteros).



Figura 22: Ejemplos de texturas generadas aleatoriamente.

3.2.9. Rigging

Para que las escenas parezcan más realistas todavía, se ha decidido añadir una última modificación a los personajes. Para cada generación de imágenes se modifica el esqueleto de todos los personajes. Efectivamente, los modelos que se han elegido pueden cambiar

3.2. ARQUITECTURA DEL ADD-ON

de pose (*rigging* en Blender). Se han creado 33 poses diferentes (sentado, corriendo con la mano derecha, levantada, corriendo, mirando a la izquierda ...). Las 33 poses se han categorizado en cinco grupos:

- Grupo “*Walking*”: Poses de personajes andando
- Grupo “*Running*”: Poses de personajes corriendo
- Grupo “*Moving*”: Poses de personajes estáticos
- Grupo “*Goaling*”: Poses clásicas de porteros
- Grupo “*Other*”: Otras poses que no entran en los otros grupos (por ejemplo, alguien tumbado)

Se pueden ver todas las poses en la Figura 23, también se puede observar los grupos en esas figuras. Por ejemplo, el grupo de poses en la parte superior derecha es el grupo “*Goaling*”.

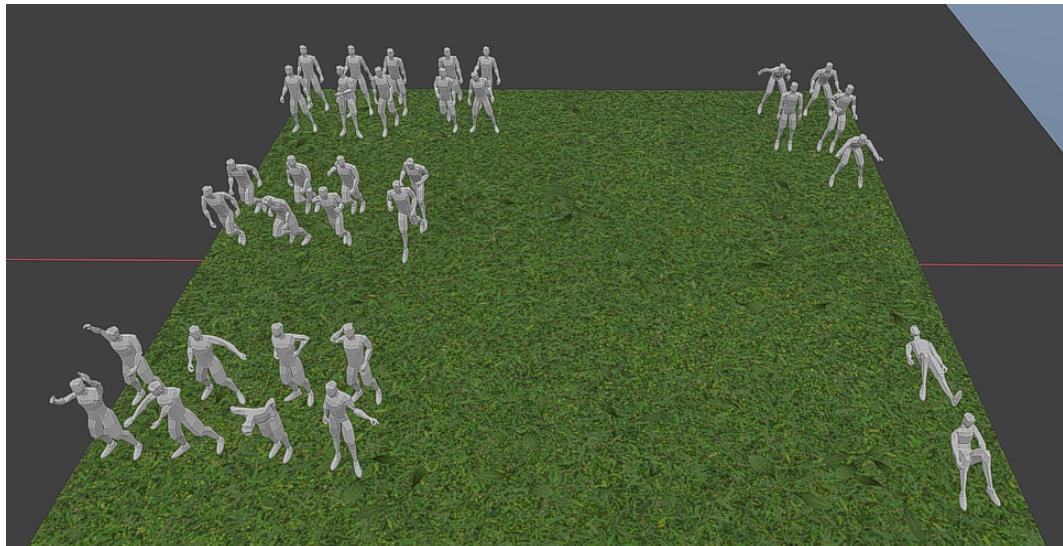


Figura 23: Todas las poses que se pueden aplicar.

Como para las texturas, los personajes se han dividido en grupos (aquí tres: jugadores, porteros y el árbitro). Cada grupo de personajes tiene su probabilidad de ser afectada a un grupo de pose. Por ejemplo, un portero y un jugador no tendrán la misma probabilidad de estar corriendo. Esas probabilidades se pueden observar en la Tabla 5.

Grupo de pose	Jugadores	Porteros	Árbitro
Walking	entre 10 % y 74.83 %	20 %	55 %
Running	entre 10 % y 74.83 %	5 %	30 %
Moving	15 %	20 %	15 %
Goaling	0 %	50 %	0 %
Other	0.17 %	5 %	0 %

Tabla 5: Probabilidades de poses para cada grupo de personajes.

Como se puede observar en la Tabla 5, los jugadores tienen entre 10 % y 74.83 % de estar corriendo o andando. Esto se ha hecho para que se generen escenas donde una mayoría de jugadores estén o andado o corriendo (siempre con el objetivo de cubrir el máximo número de escenas posibles). Además, para cada esqueleto copiado, se aplica una vez sobre dos un volteo axial sobre el modelo. Esto nos permiten pasar simplemente de 33 a 66 poses diferentes.

3.2.10. Iluminación

Una vez los modelos listos (bien colocados, texturados y con un esqueleto “aleatorio”), el algoritmo se ocupa de la parte de la iluminación. Para iluminar la escena, se han creado dos objetos (comparables a dos soles). El primer sol pretende aumentar la iluminación general y el segundo sirve para generar las sombras. Por supuesto, se ha aplicado también un factor aleatorio a la iluminación. Para cada generación, la iluminación tiene una potencia y orientación aleatoria. La potencia tiene un 50 % de probabilidad de ser “normal” y un 50 % de probabilidad de ser aleatoria, lo que puede crear escenas más o menos iluminadas. La orientación es totalmente aleatoria, lo que genera sombras diferentes en cada escena. En la Figura 24, se puede observar varias escenas con diferentes iluminaciones.



Figura 24: 3 ejemplos de iluminaciones diferentes.

3.2.11. Cámara

Una vez que toda la escena se ha generada, solo falta colocar y orientar la cámara para poder renderizar la escena. Se ha decidido renderizar las imágenes con el punto de vista más utilizado en los partidos europeos de 2022/2023. Este punto de vista consiste en colocar la cámara justo arriba de las tribunas presidenciales. Como no todos los estadios tienen el mismo tamaño, el punto de vista de la cámara es diferente según cada uno. Por eso no se ha decidido colocar la cámara en un punto fijo, pero en un cubo de manera aleatoria (ver Figura 25). Una vez la cámara colocada, se calcula el ángulo que forma con la pelota (gracias a la fórmula (1)). Ese ángulo sirve para poder enfocar la cámara sobre la pelota. Finalmente, se añade una perturbación (*jitter*) a ese ángulo.

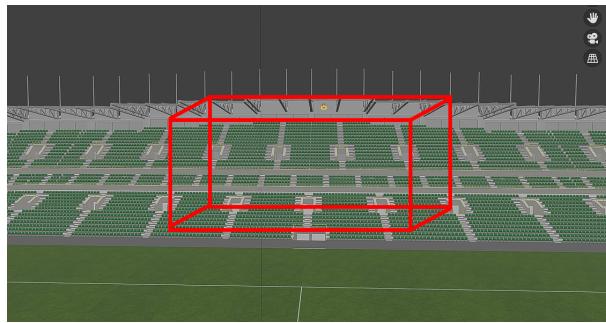


Figura 25: Localizaciones posibles de la cámara.

3.2.12. Rendering y groundtruth

Cuando todos los objetos de la escena están bien colocados y configurados, se puede renderizar la escena. En menos de 10 segundos, Blender genera una imagen de nuestra escena. Esta imagen se guarda en una carpeta en formato PNG para conservar la máxima información posible. De hecho, los archivos PNG utilizan la compresión sin pérdidas, lo que significa que se conserva todos los datos originales de la imagen sin comprimirlos (al contrario de JPEG, por ejemplo).

Para cada generación se guarda también una imagen verdadera (*groundtruth*). Esa imagen se genera mediante el “Solid mode” de Blender. El modo sólido en Blender muestra objetos 3D con su sombreado e iluminación básicos. Es una forma más rápida de ver modelos sin utilizar materiales o texturas complejas. Con este modo, se ha podido remplazar la textura de cada objeto por un color sólido:

- Objetos de color blanco [255, 255, 255]: Objetos inútiles para la detección.
- Objetos de color azul claro [124, 208, 190]: Césped del terreno.
- Objetos de color rojo [190, 1 a 10, 0]: Jugadores del equipo local (portero excluido).
- Objetos de color verde [1 a 10, 130, 0]: Jugadores del equipo exterior (portero excluido).
- Objetos de color naranja [204, 70, 0]: Portero del equipo local.
- Objetos de color azul [8, 14, 204]: Portero del equipo exterior.
- Objetos de color gris [119, 86, 92]: Árbitro.
- Objetos de color rosa [204, 0, 112]: Pelota.
- Objetos de color negro [15, 7, 0]: Líneas del campo de fútbol.

Se pueden observar algunos resultados de renderización con sus imágenes verdaderas en la Figura 26.

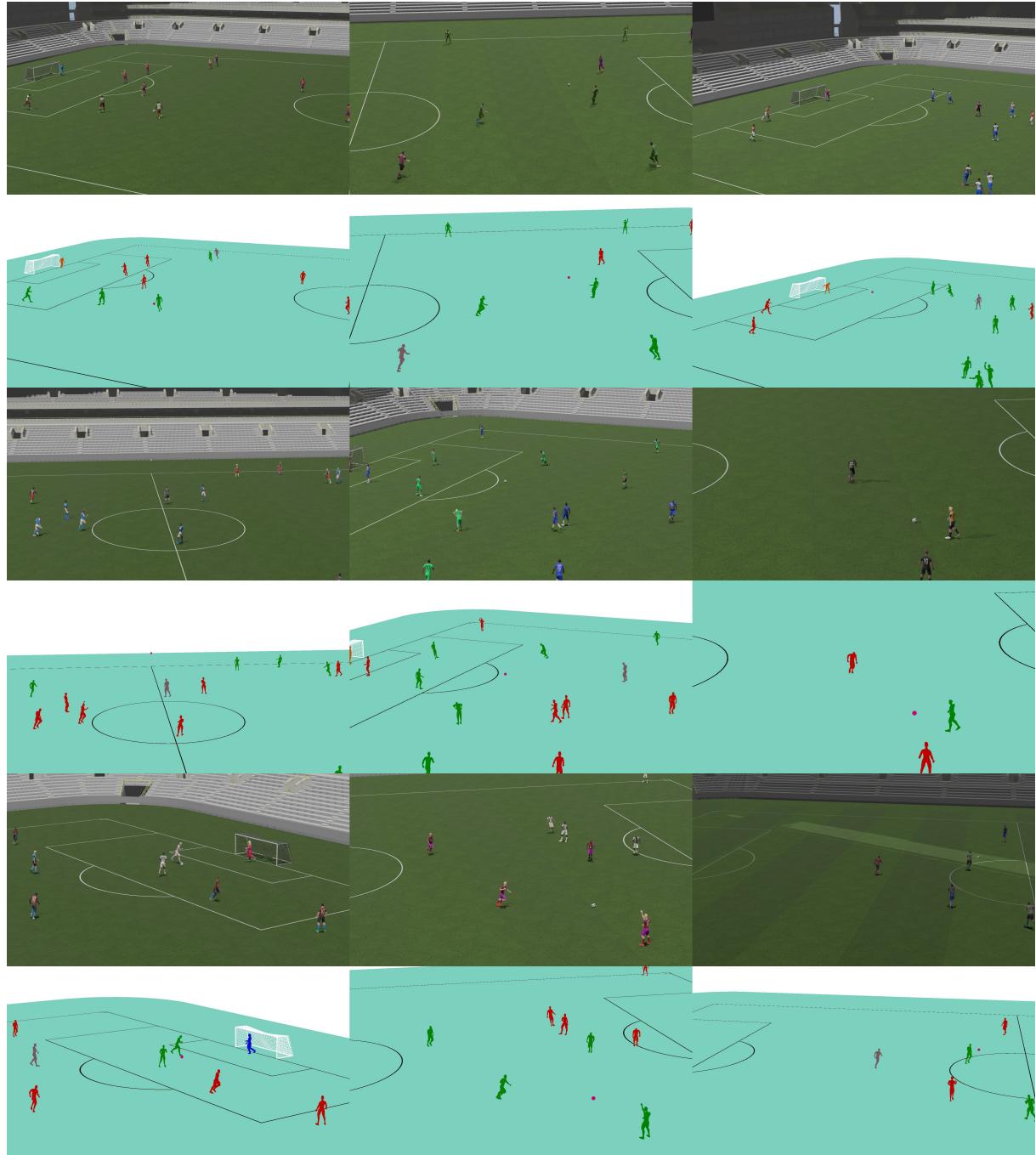


Figura 26: Resultados finales con sus imágenes verdaderas.

3.2.13. Limpiar la escena

Al final de cada renderización se ha debido de limpiar la escena. Efectivamente, si se generan 20 otros jugadores, acabarían un total de 40 jugadores en el terreno. Por eso, se ha programado una sección que permite “resetear” la escena. En un primer lugar se borran únicamente las texturas de los objetos generados previamente (no tiene sentido borrar la textura del campo de fútbol). También se borran los ficheros PNG que han servido a texturar esos objetos. En un segundo lugar, se resetean las posiciones de los porteros, de la pelota, de la cámara y de los dos soles. En un último lugar, se borran los jugadores y el árbitro. El hecho de borrar el árbitro y los jugadores se explicará en la próxima subsección. Una vez todo bien reseteado, se puede generar y renderizar una nueva escena.

3.2.14. Optimización

Antes de explicar por qué se ha optimizado el código, se necesita conocer los tiempos aproximativos de ejecución de cada parte (tiempos medidos con un portátil de la marca omen, con una GPU “GTX 1050 TI”):

- Generación de la escena (sin texturas) \approx 1.5 segundos
- Generación y aplicación de las texturas a los modelos \approx 17 segundos
- Limpieza de la escena \approx 0.1 segundos
- Tiempo de renderización de la escena con Eevee en 1920x1080 píxeles \approx 13 segundos
- Tiempo de renderización de la escena con Cycles \approx más de 2 minutos
- Tiempo de renderización de la escena con Eevee en 384x216 píxeles \approx 5 segundos
- Tiempo de renderización del *groundtruth* \approx 0.5 segundos

Antes de comparar los resultados, se debe de explicar la diferencia entre Eevee y Cycles. Eevee y Cycles son dos motores de renderizado en Blender. Eevee es un motor en tiempo real que produce resultados rápidos con un nivel de detalle básico, mientras que Cycles es un motor más avanzado, tarda más en renderizar, pero produce resultados más realistas con características como iluminación global, cáusticas y volumétricas. Eevee es ideal para la interactividad en tiempo real y la previsualización de animaciones, mientras que Cycles es mejor para renders finales de alta calidad. Como se puede ver en los resultados de tiempo de renderización, Eevee es mucho más rápido en este caso (más de 2 minutos contra 13-14 segundos). Además, con texturas low-poly y pocos detalles en la escena, los resultados generados por los dos motores son casi iguales.

Aunque se gane tiempo utilizando Eevee, una generación completa tarda unos 32 segundos. Se ha decidido añadir una opción a la generación, la posibilidad de guardar la textura de la generación anterior. Eso permite ganar 17 segundos a cada generación de imagen. En vez de cambiar de textura cada imagen, se puede hacerlo cada X imágenes. Si se decide cambiar de textura cada 20 imágenes, sobre una base de datos de 2000 imágenes, el tiempo de generación pasa de 17 horas y 47 minutos a 8 horas y 48 minutos.

Además, se puede ganar tiempo sobre la generación de la base de datos, reduciendo la calidad de las imágenes. Como se puede ver, se gana 12 segundos de renderización, pasando de una imagen 1920x1080 píxeles a 384x216. Por supuesto, si se reduce el tamaño de las imágenes, también disminuirá la calidad de la base de datos.

También, si se desea ganar un poco más de tiempo, se puede reducir el número de jugadores generados (por ejemplo, generar 20 % de la base de datos con 20 jugadores y 80 % con solamente 14). Cada jugador menos en el terreno es más o menos 1.2 segundos ganado por generación (con su textura).

Con todas esas optimizaciones se puede ganar mucho tiempo sobre toda la generación de la base de datos sin impactar su calidad.

3.2.15. Interfaz gráfica

Además de crear un código fácilmente modificable, se ha decidido implementar una interfaz gráfica. Esta interfaz está directamente integrada a Blender (aparece cuando se lanza el *add-on*). Esta interfaz permite cambiar una gran cantidad de variables, donde la mayoría ya se han visto anteriormente. Cada parámetro está organizado en una sección, lo cual permite una mejor organización. Por ejemplo, se puede directamente generar una o varias imágenes desde la interfaz. También se puede cambiar cuantos jugadores se crean por generación, se puede cambiar las probabilidades que afectan las texturas y las poses ... Cada parámetro tiene un título y una descripción que se puede apercibir si se deja el ratón encima. Esta interfaz gráfica se puede observar en la Figura 27.

3.3. ANOTACIÓN Y MODIFICACIÓN DE LAS IMÁGENES

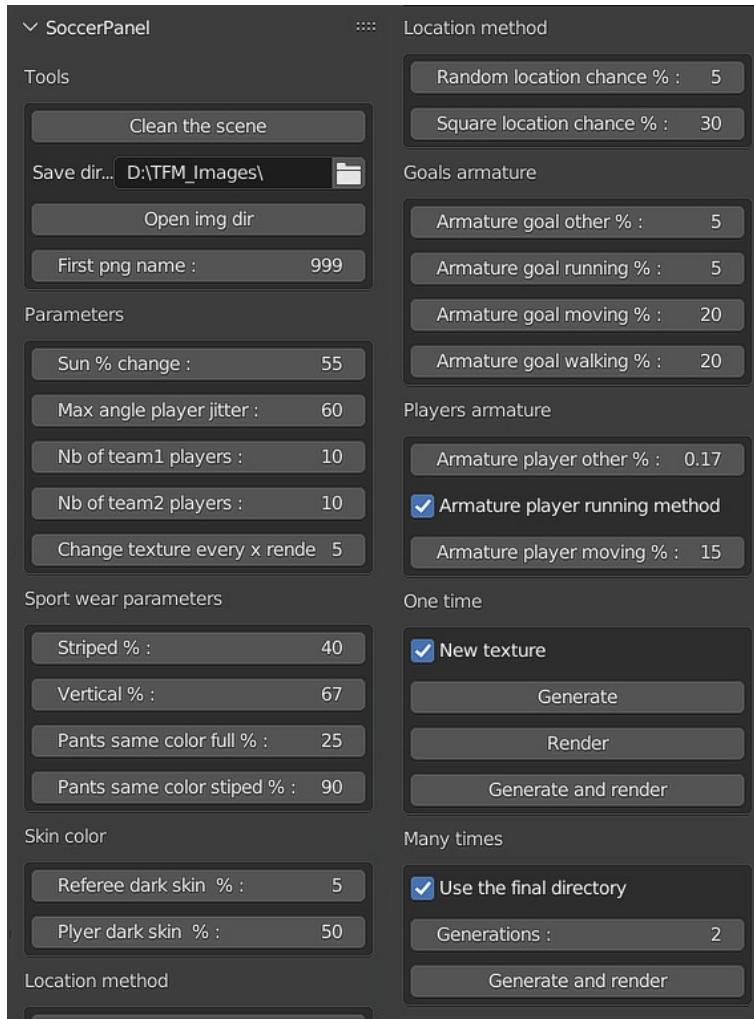


Figura 27: Interfaz gráfica de la generación de imagen.

3.3. Anotación y modificación de las imágenes

Para el modelo de detección de objetos, se ha decidido hacer aprendizaje por trasferencia con el modelo YoloV8. Se explicará en detalle esa decisión en el próximo capítulo. Para crear ese modelo con YoloV8 se necesitan varios datos. En esta sección se explicará cuáles son esos datos y como se ha logrado obtenerlos.

3.3.1. Modificación de la resolución de las imágenes

Primeramente, como todos los modelos de detección de objetos, YoloV8 necesita imágenes. Como se puede leer en la documentación de YoloV8¹⁶, el modelo se ha entrenado sobre imágenes de resolución 604x360, por eso se aconseja utilizar la misma resolución (para el aprendizaje por trasferencia). Además, se ha planteado la posibilidad de probar el modelo con imágenes de diferentes resoluciones.

Por eso, se ha creado un código en Python3 llamado “down_resolution.py” que permite cambiar la resolución de todas las imágenes de una base de datos. Este pequeño programa

¹⁶YoloV8 Documentación:<https://docs.ultralytics.com/>

puede reducir a cualquier resolución (proporcional) una imagen, utilizando principalmente el paquete Pillow¹⁷.

3.3.2. Anotación de las imágenes

YoloV8 no solo necesita imágenes, pero también necesita las anotaciones de esas imágenes (posición de los objetos y etiquetas de clase correspondiente). En la documentación de YoloV8 se exige un formato de anotación bien específico. Estas anotaciones tienen que almacenarse en un archivo texto (un fichero por imagen). Las especificaciones del archivo texto son:

- Una fila por objeto.
- Cada fila tiene el formato: “número de clase” “coordenada x del centro del cuadro delimitador” “coordenada y del centro del cuadro delimitador” “anchura” “altura”.
- Las coordenadas de los cuadros deben estar normalizados, de 0 - 1 (si las coordenadas están en píxeles).
- Los números de clase tienen índice cero (empiezan por 0).

Por ejemplo, en la Figura 28, se puede observar un ejemplo de anotación de una imagen representado 2 personas (clase 0) y una corbata (clase 27).

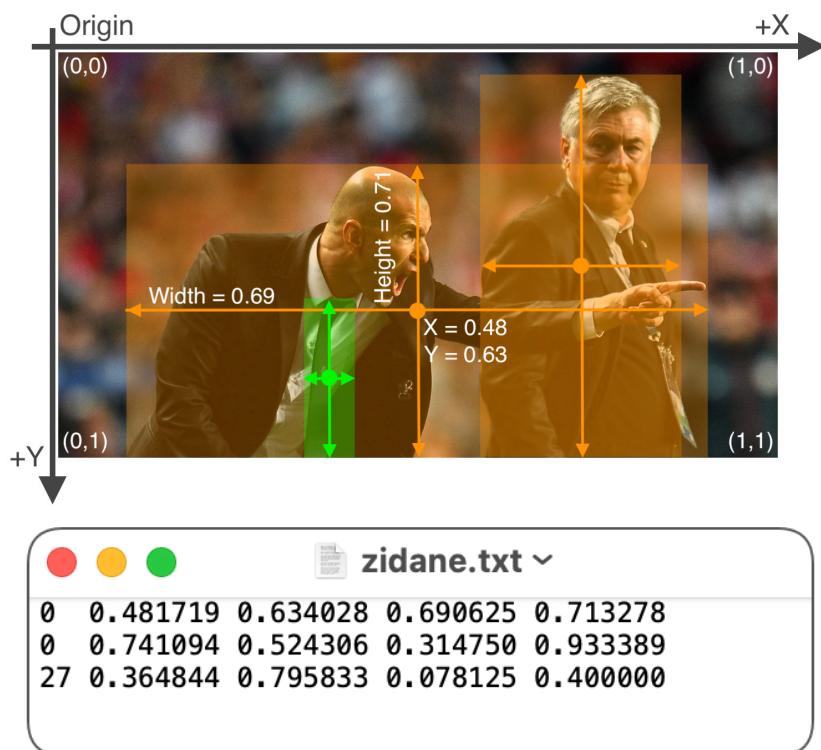


Figura 28: Ejemplo de anotación de una imagen para YoloV8.

¹⁷Pillow:<https://pillow.readthedocs.io/en/stable/>

3.3. ANOTACIÓN Y MODIFICACIÓN DE LAS IMÁGENES

Con el *add-on* de Blender que se ha desarrollado, solo se generan las imágenes con sus imágenes verdaderas (*groundtruth*). Entonces, se ha debido encontrar una manera de generar esas anotaciones a partir de las imágenes verdaderas. Por eso, se ha programado un código en Python3 llamado “annotation.py”. Ese código permite detectar el píxel más alto, bajo, más a la izquierda y más a la derecha de cada objeto. Efectivamente, como cada objeto tiene un color específico, solo hay que encontrar los píxeles correspondientes gracias al paquete “OpenCV”. Para no confundir los jugadores de un mismo equipo, cada jugador tiene su propio valor RGB. Por ejemplo, el jugador número uno del equipo exterior tiene un color de [1, 130, 0] sobre la imagen verdadera y el jugador 2 un color RGB de [2, 130, 0]. Además, el programa permite agrupar varios objetos en una sola clase. Eso permite agrupar todos los objetos “jugadores”, que serían 20 clases diferentes en una sola.

3.3.3. Verificación de las anotaciones

El último código complementario que se ha programado permite verificar visualmente el resultado de las anotaciones. Llamado “squares.py”, este código enseña las imágenes con los cuadros delimitadores a partir de las anotaciones. Se puede observar un resultado en la Figura 29.



Figura 29: Resultado de “squares.py”.

Capítulo 4

Experimentos y pruebas del modelo

Para encontrar un modelo con buenos resultados hay que hacer varios experimentos y pruebas. Durante este proyecto se han creado más de 20 modelos a partir de YoloV8 para encontrar el modelo final, variando sus parámetros y la base de datos. En esta sección se hablará de los modelos más importantes que han permitido encontrar el modelo final.

4.0.1. Parámetros de un entrenamiento

Para entrenar un nuevo modelo por transferencia (*transfer learning*) con YoloV8 se tiene que elegir con cuál modelo se va a empezar. Como se ha visto anteriormente, el objetivo del aprendizaje por transferencia es utilizar un modelo preentrenado como punto de partida para una nueva tarea. Así, se aprovecha los conocimientos adquiridos en un entrenamiento sobre un gran conjunto de datos y permite usar una base de datos mucho más pequeña para una nueva tarea. Por ejemplo, los modelos propuestos por YoloV8 se puede comparar en la Tabla 6.

Modelo	Resolución imágenes	Parámetros (en millones)
YOLOv8n	640x360	3.2
YOLOv8s	640x360	11.2
YOLOv8m	640x360	25.9
YOLOv8l	640x360	43.7
YOLOv8x	640x360	68.2

Tabla 6: Modelos propuestos por YoloV8.

Se ha decidido utilizar solamente los dos primeros modelos de Yolov8 (YOLOv8n y YOLOv8s) como base para obtener un modelo ligero y rápido de entrenar. Además de la elección del modelo de base, para el entrenamiento, también se debe elegir varios parámetros que influyen sobre el resultado del modelo como:

- El tamaño de las imágenes de la base de datos.
- El número de imágenes en la base de datos.
- El número de clases.

- El número de épocas (*epochs*): En el DL, una época se refiere a un ciclo del proceso de entrenamiento en el que el modelo recibe el conjunto de datos completo.
- El IOU mínimo: Mide el solapamiento entre los cuadros delimitadores predichos y los cuadros reales. Se calcula como la relación entre la intersección y la unión de las dos cajas (ver Figura 30).
- La confianza mínima: La confianza mide la certeza del modelo de que un objeto está presente en una imagen. Se representa mediante una puntuación de probabilidad asignada a cada objeto detectado, en la que las puntuaciones más altas indican una mayor confianza.

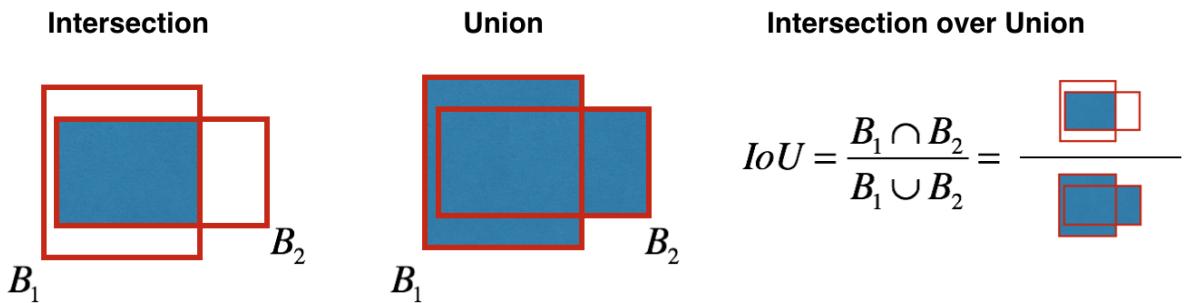
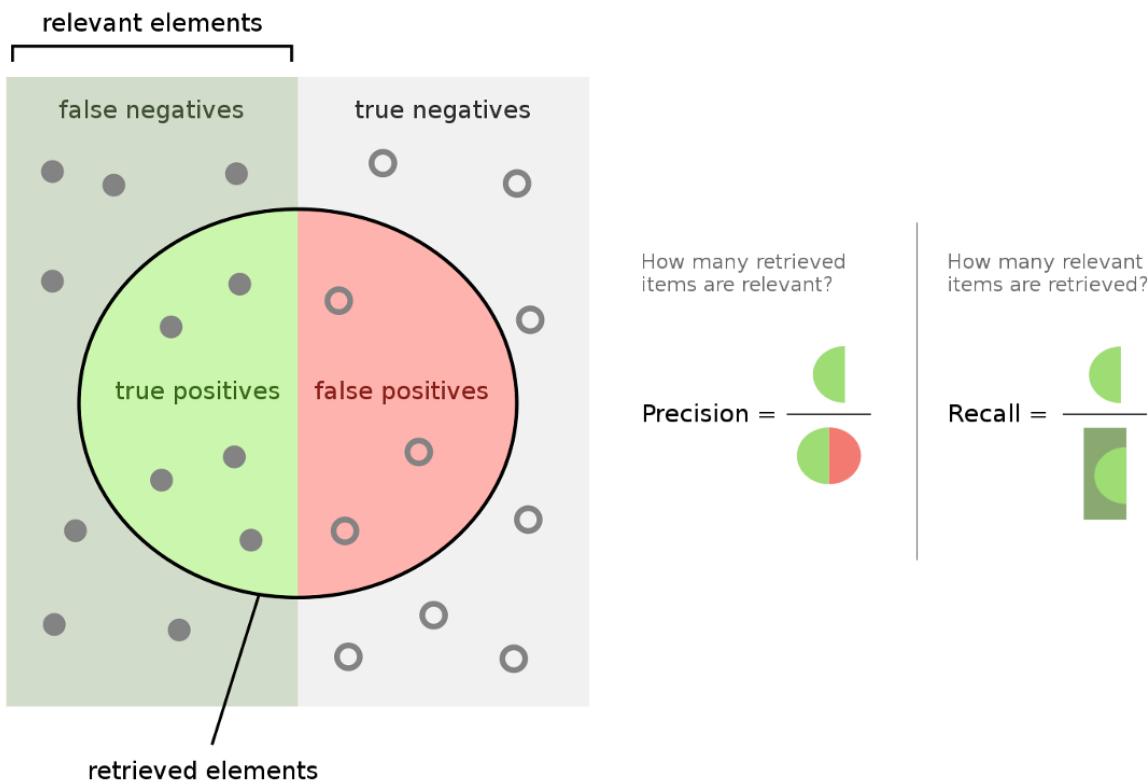


Figura 30: Explicación del IOU (*Intersection over union*).

4.0.2. Métricas de evaluación del modelo

Primeramente, para poder entender las métricas que se van a utilizar para evaluar los modelos, se debe de explicar lo que son los verdaderos positivos (TP), verdaderos negativos (TN), falsos positivos (FP) y falsos negativos (FN). En la detección de objetos, TP representa el número de casos positivos correctamente identificados. TN representa el número de casos negativos correctamente identificados (poco utilizado en detección de objetos). FP representa el número de instancias que se predijeron positivas, pero que en realidad fueron negativas. FN representa el número de instancias que se predijeron negativas, pero que en realidad eran positivas.

Gracias a esos valores (TP, TN, FP y FN) se pueden calcular el *recall* y la precisión. El *recall* mide el porcentaje de TP de todos los positivos reales, mientras que la precisión mide el porcentaje de TP de todos los positivos predichos (ver Figura 31).


 Figura 31: Explicación del *recall* y precisión.

Para poder evaluar cada modelo se ha utilizado unas métricas bien específicas. La primera ha sido la matriz de confusión, muestra el número de TP, TN, FP y FN en forma de tabla. Donde, la diagonal de la matriz muestra el número de predicciones correctas, mientras que los elementos no diagonales indican las clasificaciones erróneas. Proporciona una valiosa visión de la precisión de un modelo y ayuda a identificar qué clases se clasifican mal. La segunda métrica ha sido la precisión y la tercera el *recall*.

Finalmente, la última ha sido el F1 score. Mide el equilibrio entre el *recall* y la precisión. Es la media armónica de la precisión y la recuperación, y va de 0 a 1, donde una puntuación más alta indica un mejor rendimiento. Se calcula dividiendo el producto de la precisión y la recuperación por su suma (ver ecuación (3)).

$$F1 = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall}) \quad (3)$$

Los resultados de esas cuatro métricas se han obtenido haciendo tests sobre nuevas imágenes (que el modelo no ha visto nunca). Además, se han hecho, primeramente, las pruebas sobre imágenes sintéticas, y cuando se han obtenido buenos resultados, se han probado sobre imágenes reales.

4.0.3. Modelo con 24 clases

Los primeros modelos que se han probado tenían 24 clases (una clase para cada objeto, por ejemplo, el jugador exterior número 3 tenía la clase 16). Aunque se ha probado varios modelos con diferentes parámetros (cambiando el número de épocas, de imágenes ...), no

se ha conseguido obtener muy buenos resultados. Efectivamente, los modelos detectaban muy bien el árbitro, los porteros y la pelota, pero casi nunca a los jugadores. Por ejemplo, con el modelo llamado “Modelo_24clases” (ver Tabla 7 para ver sus parámetros), se ha obtenido un 89 % de TP para los árbitros y 75 % de TP para el portero exterior (Ver Figura 32). Esos dos resultados son muy buenos, pero cuando se observan los jugadores, casi todos tiene menos de 1 % de TP. Como se puede deducir, el modelo solamente fallaba sobre los jugadores. Se ha concluido que no era una buena idea de separar todos los jugadores en clases diferentes. Al tener 20 clases prácticamente iguales, cuando un jugador aparece en la imagen, el modelo no tiene bastante información para saber exactamente cuál es (una información que cambiaría eso, podría ser el número sobre la camiseta). Entonces, el modelo atribuye más o menos una confianza de 1/20 a cada clase de tipo “jugador”. Y como el modelo tiene que tener una confianza superior a 0.25 por objetos (ver Tabla 7), casi nunca el modelo detecta un jugador.

Modelo YoloV8	Imágenes	Resolución	IOU mín	confianza mín	épocas	clases
YOLOv8s	700	640x360	0.7	0.25	20	24

Tabla 7: Parámetros del modelo “Modelo_24clases”.

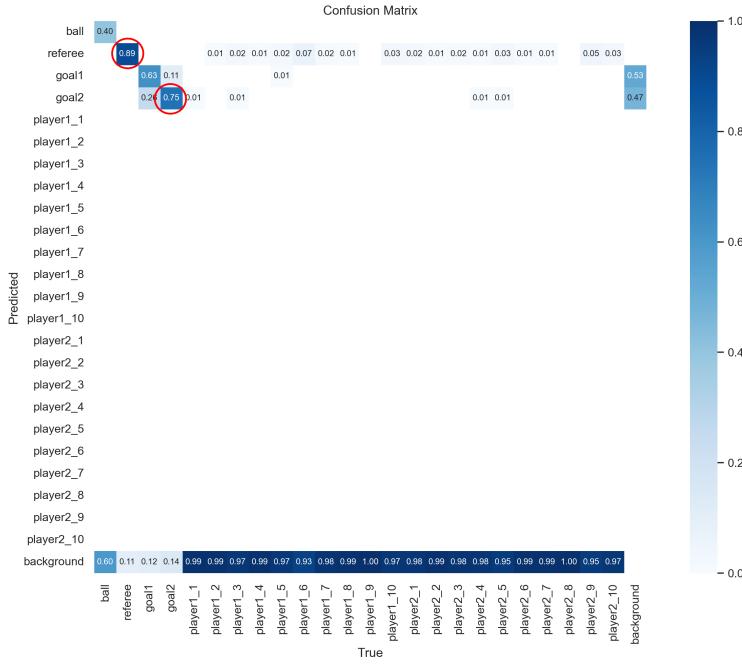


Figura 32: Matriz de confusión del modelo “Modelo_24clases”.

4.0.4. Modelo con 6 clases

Para intentar resolver el problema que se ha visto justo antes, se ha decidido utilizar solamente 6 clases. Los jugadores se han agrupado en dos clases: jugadores equipo exterior y jugadores equipo local. Los modelos resultaban tener el mismo defecto, no conseguían diferenciar con buenos resultados los jugadores locales y exteriores. Se ha deducido que el problema era el mismo, un modelo no puede diferenciar un jugador local o exterior si en la base de datos, las texturas son aleatorias. Como se puede ver en la matriz de confusión

del modelo “Modelo_6clases_20epochs” 33 (ver Tabla 8 para ver sus parámetros), los jugadores tienen un poco menos de 50 % de TP y un poco menos de 50 % de FP con el equipo contrario. Eso confirma que el modelo, al detectar un jugador, se equivoca más o menos una vez cada dos de equipo.



Figura 33: Matriz de confusión del modelo “Modelo_6clases_20epochs”.

Aunque los resultados no sean perfectos, se ha comparado diferentes modelos con diferentes épocas. Ha resultado que los modelos mejoran hasta aproximadamente la época 14-21. Más allá, los resultados se mejoran, pero muy lentamente (algunas clases pierden precisión y algunas ganan). Como se puede ver en la Figura 34 las matrices de confusiones de “Modelo_6clases_20epochs” y “Modelo_6clases_30epochs” son casi idénticas (ver Tabla 8 para los parámetros). YoloV8 se ha entrenado sobre millones de imágenes reales, más se entrena el modelo final con imágenes sintéticas y más se adapta a situaciones sintéticas. Entonces, se ha encontrado que más allá de 20 épocas los resultados en imágenes reales empeoran. A partir de este momento se ha decidido guardar la mejor época sobre un máximo de 20.

Nombre	YoloV8	Imágenes	Resolución	IOU mín	confianza mín	épocas	clases
1	YOLOv8s	700	640x360	0.7	0.25	20	6
2	YOLOv8s	700	640x360	0.7	0.25	30	6

Tabla 8: Parámetros del modelo “Modelo_6clases_20epochs” (1) y “Modelo_6clases_30epochs” (2).

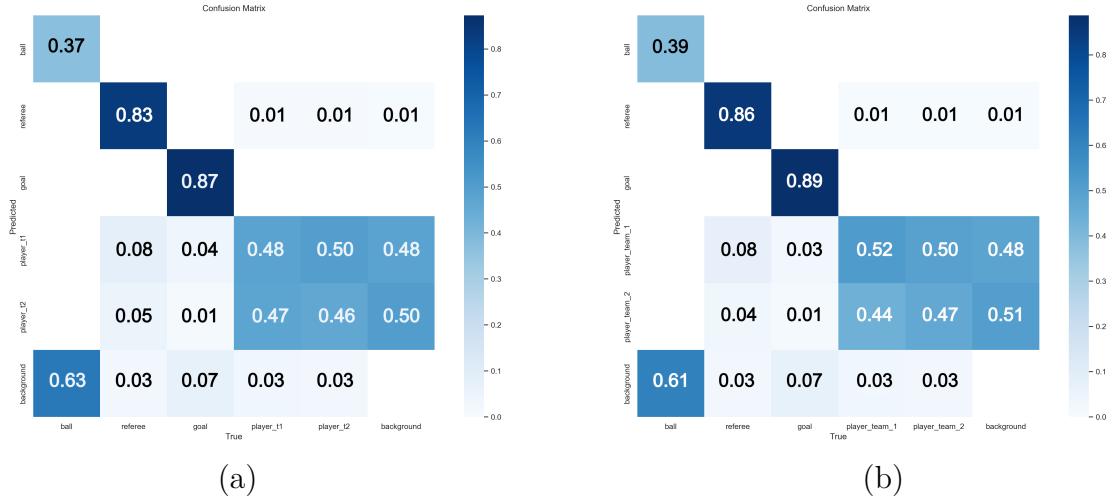


Figura 34: En (a) matriz de confusión de “Modelo_6clases_20epochs”. En (b) matriz de confusión de “Modelo_6clases_30epochs”.

4.0.5. Modelo con 4 clases

Se ha intentado agrupar todos los jugadores en una sola clase para eliminar el problema que se ha visto con los modelos a 24 o 6 clases. Siguiendo el mismo planteamiento, también se han agrupado los porteros para obtener mejores resultados. Utilizando los mismos parámetros que los modelos precedentes (pero cambiando el número de clases) se ha creado el modelo “Modelo_4clases”, y los mejores resultados se han obtenido con 19 épocas. Como se puede ver en la matriz de confusión de ese modelo (ver Figura 35), los resultados son muy buenos. Los jugadores tienen un TP de 0.96, los porteros de 0.86 y el árbitro de 0.83. Además, cuando los objetos se detectan con más de 0.4 de confianza se obtiene una precisión de aproximadamente 0.95 y un *recall* de aproximadamente 0.76 (ver Figura 36 y 37).

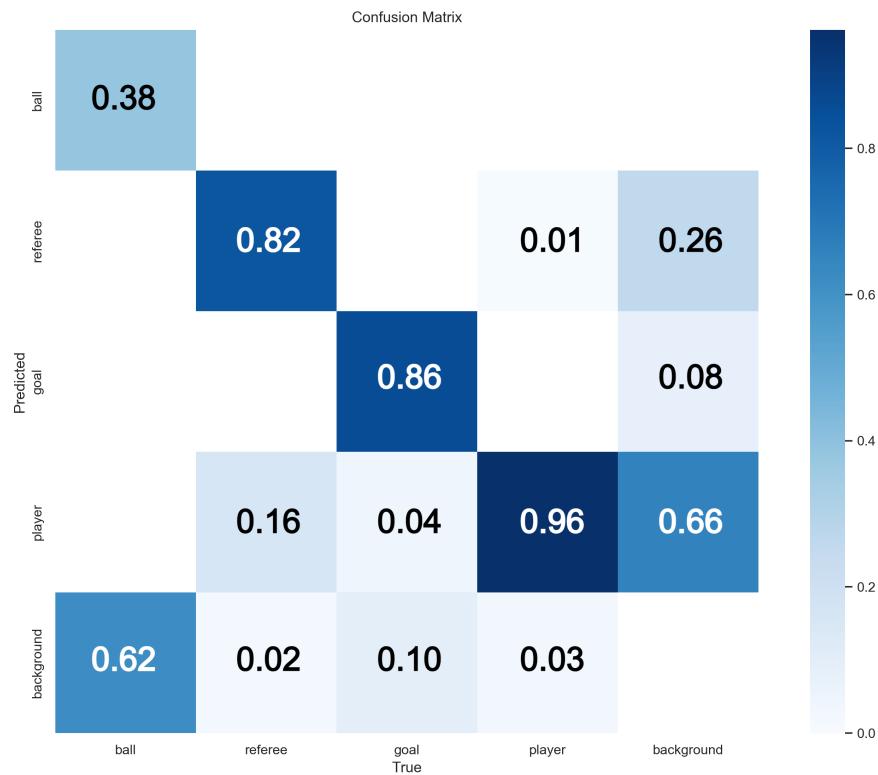


Figura 35: Matriz de confusión del modelo “Modelo_4clases”.

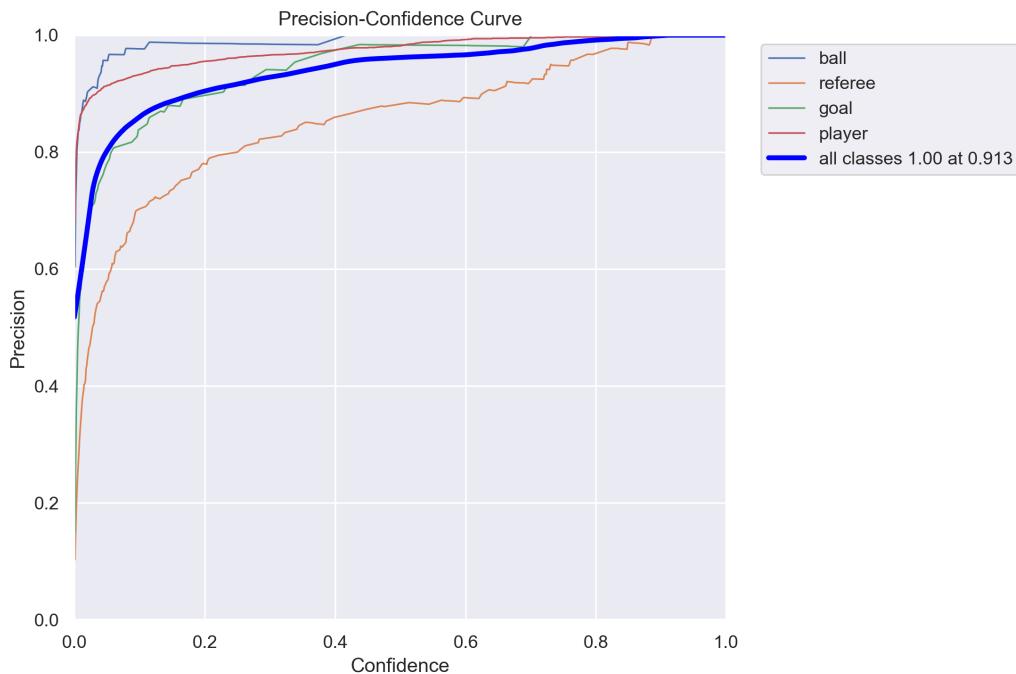


Figura 36: Curva de precisión de “Modelo_4clases”.

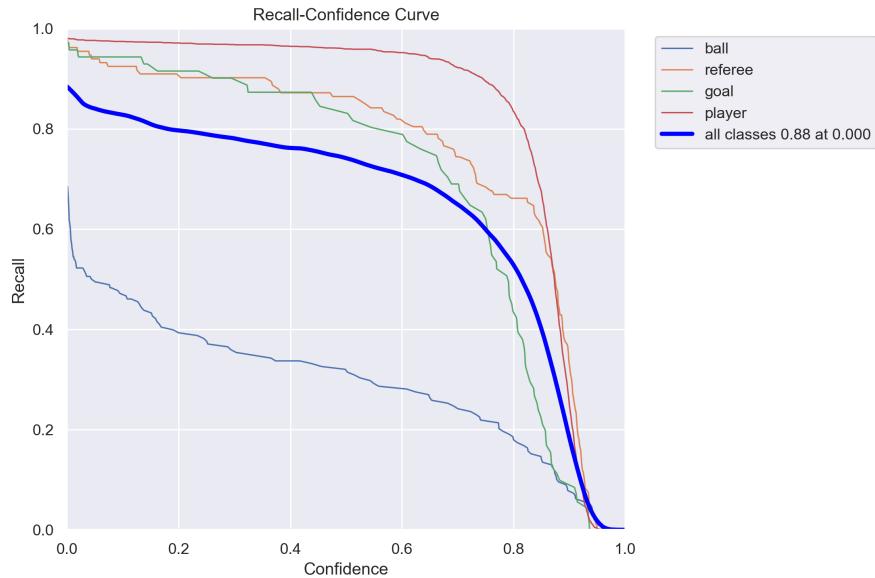


Figura 37: Curva de *recall* de “Modelo_4clases”.

Además de obtener muy buenos resultados en imágenes sintéticas, se pueden comprobar buenos resultados en imágenes reales (ver Figura 38). La métrica para medir el modelo sobre imágenes reales se explicará en la próxima subsección, donde se obtiene el mejor modelo.

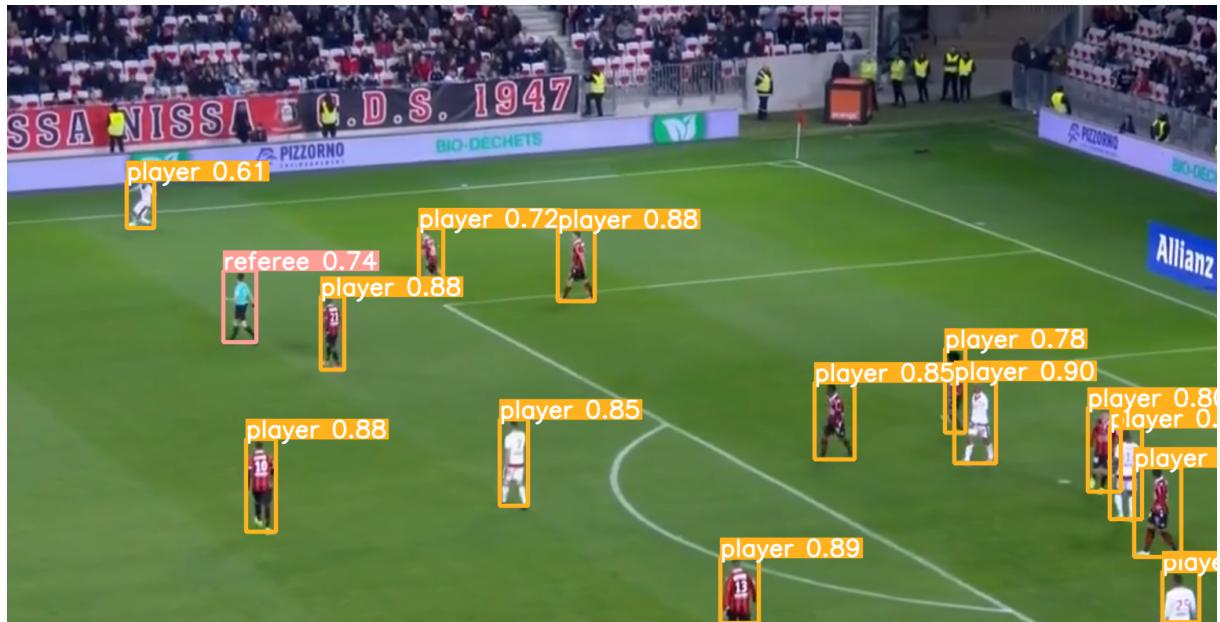


Figura 38: Ejemplo de detección del modelo “Modelo_4clases” sobre una imagen real.

4.0.6. Modelo personalizado

Con el modelo con 4 clases se ha obtenido muy buenos resultados, pero el objetivo era separar los jugadores y porteros en cuatro diferentes clases (domicilio y exterior). Como se ha visto antes, no era posible distinguir los jugadores con texturas aleatorias. Entonces,

para resolver ese problema, se ha generado una nueva base de datos con el objetivo que las texturas se parezcan a las de un solo partido de fútbol real. Se ha elegido un partido del Valencia C.F contra el Real Madrid, y como se puede ver en la Figura 39 las imágenes generadas se parecen bastante a la imagen real.

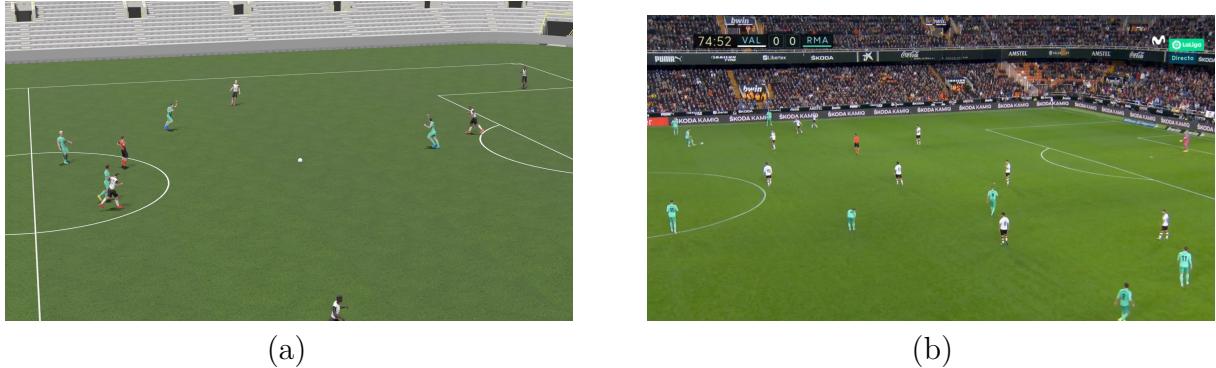


Figura 39: En (a) un render generado por el *add-on*. En (b) una imagen real del partido Valencia C.F / Real de Madrid.

Adaptando la base de datos a un solo partido, los resultados son mejores que con el modelo de 4 clases. Además, se pueden separar los jugadores y porteros en 4 clases diferentes, lo que resulta al final a un total de 6 clases. Entonces, se ha creado un modelo llamado “Val-Real-6clases”, donde sus parámetros se pueden observar en la Tabla 9.

YoloV8	Imágenes	Resolución	IOU mín	confianza mín	épocas	clases
YOLOv8s	700	640x360	0.7	0.25	19	6

Tabla 9: Parámetros del modelo “Val-Real-6clases”.

Sobre imágenes sintéticas, se ha obtenido con ese modelo resultados casi perfectos. Como se puede ver en su matriz de confusión (ver Figura 40), los jugadores a domicilio y exterior tienen un TP de 0.97, la clase portero a domicilio tiene un TP de 0.97 y el exterior de 1.00, el árbitro tiene 0.99 y la pelota 0.40. Como se puede ver, la única clase que no obtiene un buen resultado es la de la pelota. Con ningún modelo se ha conseguido más de 0.40 de TP para esa clase. Se ha supuesto que, al ser un objeto muy pequeño y únicamente blanco, la pelota se vuelve muy complicada a detectar, lo que podría ser un objetivo para trabajos futuros.

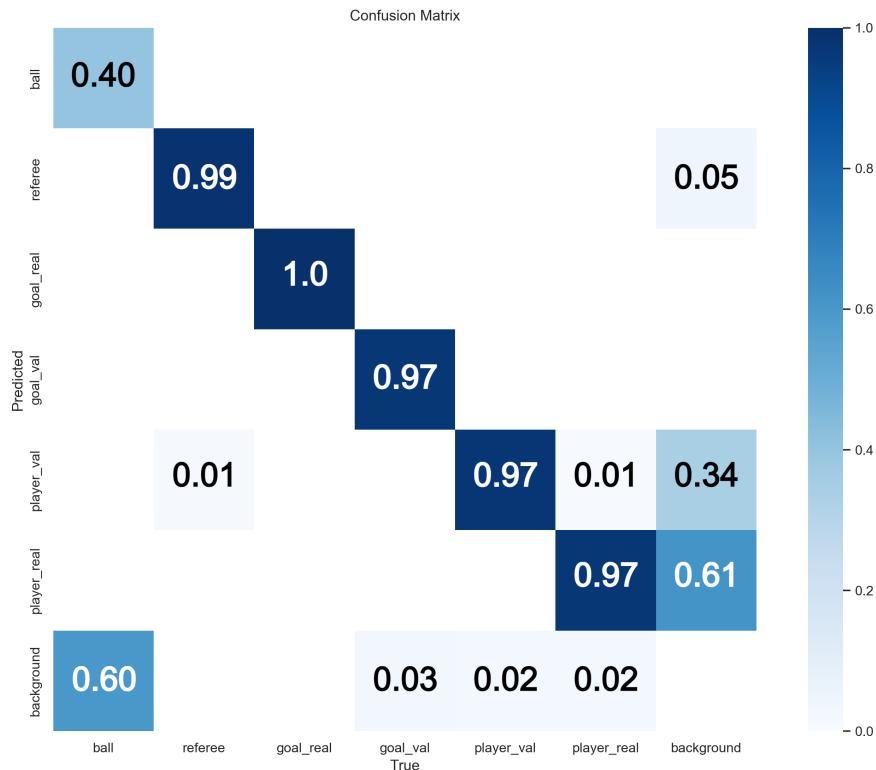


Figura 40: Matriz de confusión del modelo “Val-Real-6clases”.

También se ha obtenido una precisión casi perfecta y un *recall* casi perfecto antes de 0.6 de confianza (ver Figura 41 y 42). El hecho que el *recall* baje a partir de 0.6 de confianza no es un problema, esto significa que hay objetos que se detectan con menos de 0.6 de confianza, pero que siguen (según la curva de precisión) a más de 0.25 (por lo que siguen bien detectados).

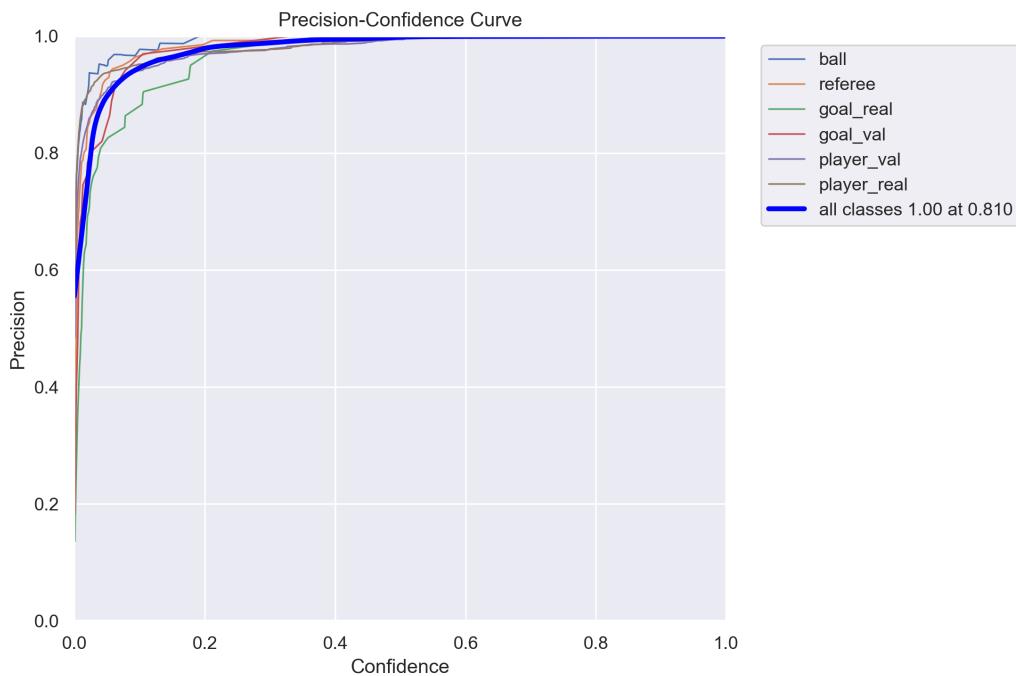


Figura 41: Curva de precisión de “Val-Real-6clases”.

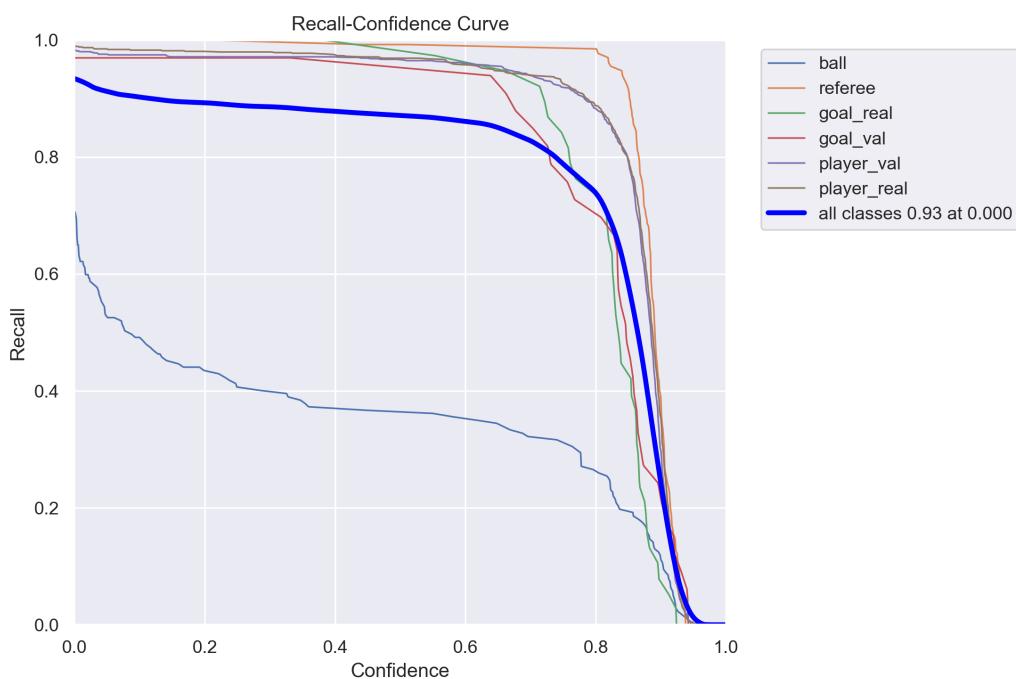


Figura 42: Curva de *recall* de “Val-Real-6clases”.

Además, también se ha obtenido una puntuación F1 buena. Su curva (ver Figura 43) compara la puntuación F1 con el umbral de confianza utilizado para clasificar los objetos, lo que ayuda a determinar el equilibrio óptimo entre precisión y recuperación en la detección de objetos. Aquí un buen umbral sería 0.75.

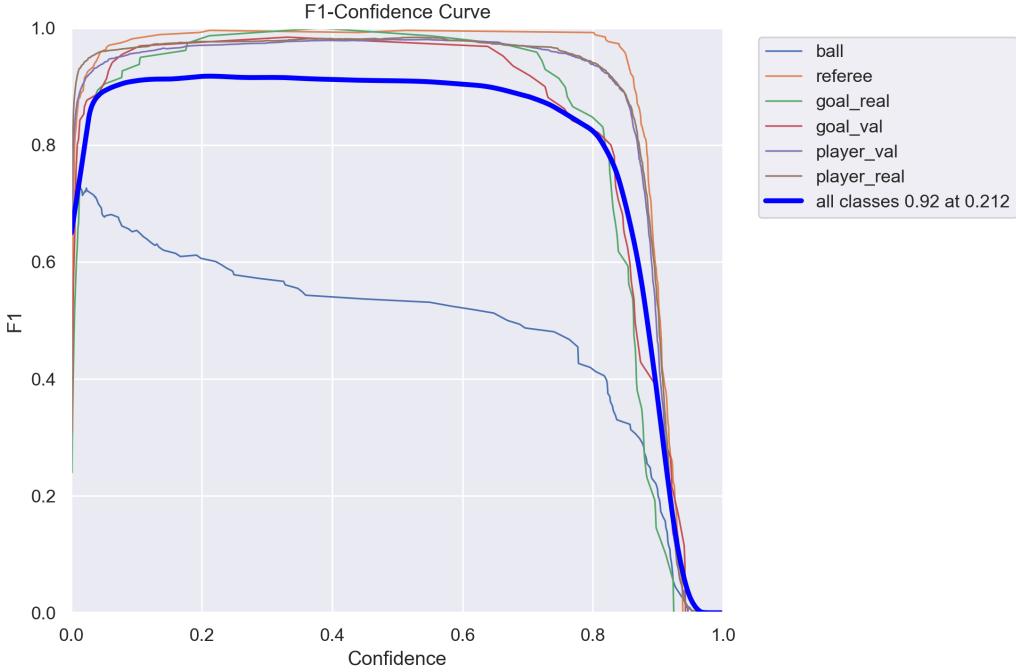


Figura 43: Gráfico que compara la puntuación F1 con el umbral de confianza del modelo “Val-Real-6clases”.

Los resultados sobre imágenes reales también son muy buenos. Algunos resultados pueden verse en la Figura 44 y 45. Como no existen imágenes verdaderas o anotaciones de los objetos de ese partido, se ha calculado manualmente el TP, FP, FN (el TN no tiene sentido en detección de objetos) a partir de 10 imágenes aleatorias. Se pueden ver los resultados obtenidos en la Tabla 10. El modelo también funciona sobre vídeos, porque los vídeos son solamente secuencias de imágenes.

Clase	TP	FP	FN
Jugadores Valencia	65/68	2/68	1/68
Jugadores Madrid	65/67	0/67	2/67
Árbitro	9/9	0/9	0/9
Portero Valencia	2/2	0/2	0/2
Portero Madrid	0/1	0/1	1/1
Pelota	3/10	0/10	7/10

Tabla 10: Resultados del modelo “Val-Real-6clases” sobre imágenes reales.

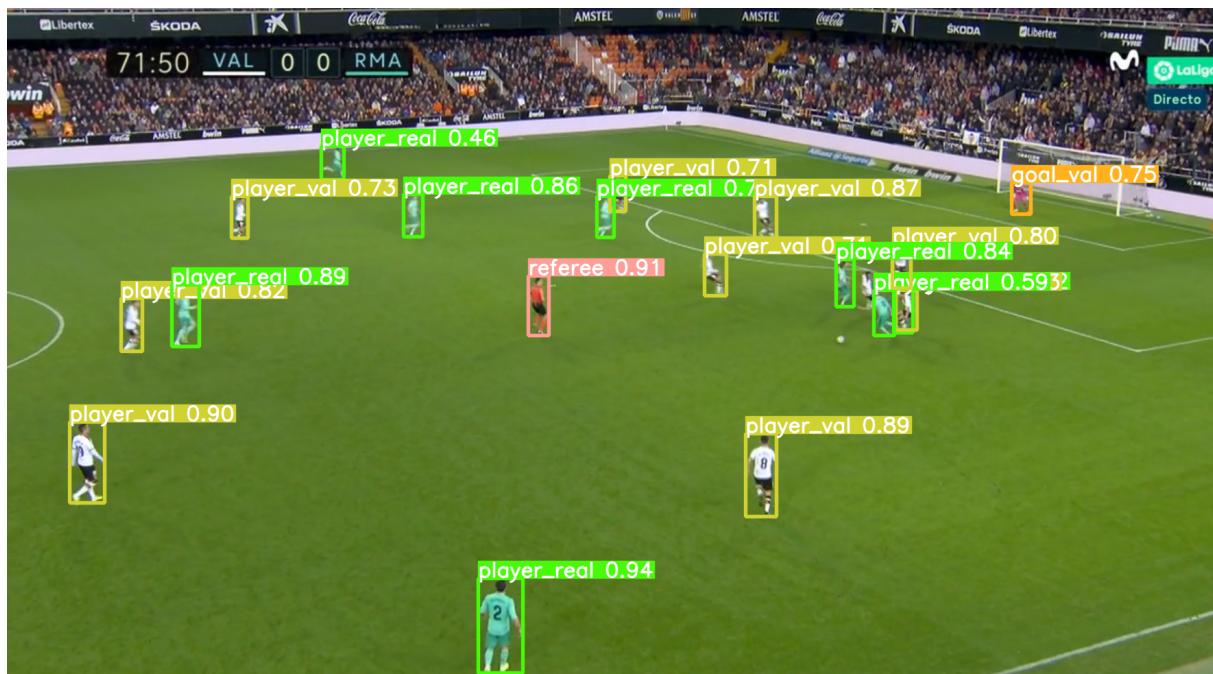


Figura 44: Resultado del modelo “Val-Real-6clases” sobre una imagen real.

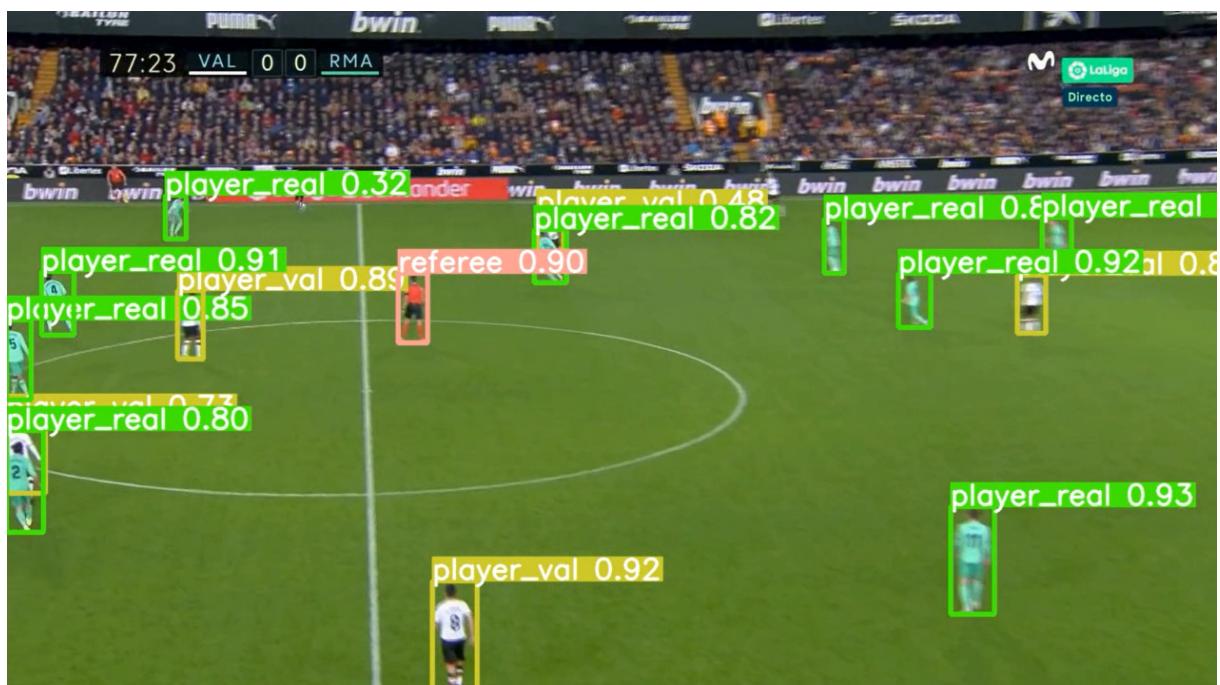


Figura 45: Resultado del modelo “Val-Real-6clases” sobre una imagen real (2).

Capítulo 5

Métricas del desarrollo

En este corto capítulo se detallará, las métricas del desarrollo del proyecto. Se verá, por ejemplo, cuanto tiempo se ha dedicado a cada parte o cantas líneas de códigos se han escrito durante ese trabajo fin de máster.

5.1. Tiempo dedicado

Como no se ha medido el tiempo exacto durante la elaboración de este proyecto, se ha hecho una estimación del tiempo dedicado a cada parte:

- Aprendizaje de Blender \approx 15 horas.
- Búsqueda de información \approx 20 horas.
- Software de generación de imágenes \approx 200 horas.
- Generación de imágenes (no presente, solo tiempo de espera) \approx 15 horas.
- Análisis de las imágenes (comparación con imágenes reales) \approx 3 horas.
- Códigos Python suplementarios \approx 3 horas.
- Elaboración del modelo de detección de objetos + activación del entrenamiento con GPU \approx 20 horas.
- Entrenamiento de los modelos (no presente, solo tiempo de espera) \approx 17 horas.
- Análisis y comparación de modelos \approx 5 horas.
- Redacción de la memoria (mi primer “gran” documento redactado en español) \approx 80 horas.

De manera general, se han dedicado aproximadamente 343 horas a este proyecto.

5.2. Métricas técnicas

Aproximadamente, el proyecto cuenta unas 2500 líneas de código, el *add-on* cuenta con 1639 líneas. Además del *add-on* de Blender, se han creado unas 233 líneas para el modelo y unas 637 líneas para pequeños programas de Python (como para reducir el tamaño de las imágenes).

Además de esas líneas de código, se ha estimado que se han generado unos 60.000 objetos durante este proyecto. El proyecto final de Blender cuenta (cuando una escena se ha generado) con 170 objetos, 610.496 vértices, 848.598 bordes, 313.613 caras y 616.504 triángulos. También se estima haber generado unas 2.500 imágenes con este programa de generación de imágenes.

Capítulo 6

Conclusiones

En este capítulo se presentarán las conclusiones de este trabajo fin de máster y las posibles continuaciones del mismo.

6.1. Objetivos cumplidos

El objetivo principal de este trabajo era obtener un modelo de detección de objetos en escenas de fútbol, entrenado solamente con imágenes sintéticas. Para cumplir ese objetivo se ha dividido el trabajo en dos partes: la generación de imágenes sintéticas y el modelo. Para la primera parte se ha creado un complemento al software Blender que permite generar imágenes sintéticas de fútbol. Además, el complemento permite, gracias a sus parámetros, generar imágenes personalizadas.

Para cumplir el segundo objetivo, se ha entrenado un modelo con aprendizaje por transferencia a partir de YoloV8. Las imágenes generadas por el complemento han demostrado ser útiles, ya que los resultados finales han sido muy buenos. Efectivamente, el modelo final ha conseguido, en imágenes reales, detectar correctamente 144 objetos sobre 155 (y 141 sobre 145 quitando la pelota). Este proyecto demuestra que, en algunos casos, los datos sintéticos pueden ser una buena alternativa a los datos reales, sobre todo cuando pueden ser difíciles de obtener.

6.2. Futuros trabajos

Este proyecto podría continuarse de varias maneras. Aunque los resultados sean muy buenos, se podría intentar mejorarlos, sobre todo cuando se intenta detectar objetos muy pequeños y con poca textura como una pelota de fútbol. También se podría hacer el mismo proyecto, pero cambiando el tipo de problema, como por ejemplo detección de objetos para vigilancia.

Sin embargo, quizás la mejor continuación de este proyecto sería construir un modelo de segmentación semántica con las imágenes generadas, en vez de usar YoloV8. La segmentación semántica consiste en clasificar cada píxel de la imagen. Todo el proyecto está más o menos listo para poder entrenar un modelo a esta tarea. Por ejemplo, las imágenes verdaderas (*groundtruth*) generadas por el complemento de Blender, han sido pensadas para

poder generar anotaciones permitiendo la segmentación. También se ha elegido trabajar con YoloV8 porque permite hacer aprendizaje por transferencia para segmentación. Haciendo pequeñas modificaciones al proyecto, se podría saber si se obtiene buenos resultados de segmentación con nuestras imágenes sintéticas. Otra posible continuación podría consistir en completar el sistema con algún método de *tracking*, para poder hacer seguimiento de los jugadores.

Apéndice A

Guía de instalación y utilización

Para la instalación y utilización del proyecto ir al Github¹ (<https://github.com/Luisrosario2604/Object-detection-soccer-synthetic-data>).

¹[Github:https://github.com/Luisrosario2604/Object-detection-soccer-synthetic-data](https://github.com/Luisrosario2604/Object-detection-soccer-synthetic-data)

Bibliografía

- [1] Abhinav Jain y col. “Overview and importance of data quality for machine learning tasks”. En: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, págs. 3561-3562 (vid. pág. 1).
- [2] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep learning*. MIT press, 2016 (vid. págs. 1, 3).
- [3] Alon Halevy, Peter Norvig y Fernando Pereira. “The unreasonable effectiveness of data”. En: *IEEE intelligent systems* 24.2 (2009), págs. 8-12 (vid. pág. 1).
- [4] CrowdFlower. *CrowdFlower DataScienceReport 2016*. Inf. téc. CrowdFlower, 2016 (vid. pág. 1).
- [5] Jia Deng y col. “Imagenet: A large-scale hierarchical image database”. En: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, págs. 248-255 (vid. págs. 1, 3).
- [6] Alex Krizhevsky, Ilya Sutskever y Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. En: *Communications of the ACM* 60.6 (2017), págs. 84-90 (vid. pág. 1).
- [7] towardsdatascience Tirthajyoti Sarkar. *Synthetic data generation — a must-have skill for new data scientists*. 2018. URL: <https://towardsdatascience.com/synthetic-data-generation-a-must-have-skill-for-new-data-scientists-915896c0c1ae> (vid. pág. 1).
- [8] Rohit Venugopal y col. “Privacy preserving Generative Adversarial Networks to model Electronic Health Records”. En: *Neural Networks* 153 (2022), págs. 339-348 (vid. pág. 1).
- [9] Mehrnaz Sabet, Praveen Palanisamy y Sakshi Mishra. “Scalable Modular Synthetic Data Generation for Advancing Aerial Autonomy”. En: *arXiv preprint arXiv:2211.05335* (2022) (vid. pág. 2).
- [10] Leinar Ramos y Jitendra Subramanyam. *Maverick Research: Forget About Your Real Data – Synthetic Data Is the Future of AI*. Inf. téc. 2021 (vid. pág. 2).
- [11] aimultiple Cem Dilmegani. *Synthetic Data to Improve Deep Learning Models in 2023*. 2022. URL: <https://research.aimultiple.com/synthetic-data-for-deep-learning/> (vid. pág. 2).
- [12] Samuel Hurault, Coloma Ballester y Gloria Haro. “Self-supervised small soccer player detection and tracking”. En: *Proceedings of the 3rd international workshop on multimedia content analysis in sports*. 2020, págs. 9-18 (vid. pág. 2).

- [13] Ying Yang y Danyang Li. “Robust player detection and tracking in broadcast soccer video based on enhanced particle filter”. En: *Journal of Visual Communication and Image Representation* 46 (2017), págs. 81-94 (vid. pág. 2).
- [14] Ian J. Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. Cambridge, MA, USA: MIT Press, 2016 (vid. pág. 3).
- [15] Aston Zhang y col. “Dive into deep learning”. En: *arXiv preprint arXiv:2106.11342* (2021) (vid. pág. 3).
- [16] David E Rumelhart, Geoffrey E Hinton y Ronald J Williams. “Learning representations by back-propagating errors”. En: *nature* 323.6088 (1986), págs. 533-536 (vid. pág. 3).
- [17] Ross Girshick y col. “Rich feature hierarchies for accurate object detection and semantic segmentation”. En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, págs. 580-587 (vid. pág. 3).
- [18] Joseph Redmon y col. “You only look once: Unified, real-time object detection”. En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 779-788 (vid. págs. 3, 4, 14).
- [19] Ross Girshick. “Fast r-cnn”. En: *Proceedings of the IEEE international conference on computer vision*. 2015, págs. 1440-1448 (vid. pág. 3).
- [20] Hei Law y Jia Deng. “CornerNet: Detecting objects as paired keypoints”. En: *Proceedings of the European conference on computer vision (ECCV)*. 2018, págs. 734-750 (vid. pág. 3).
- [21] Eduardo Arnold y col. “A survey on 3d object detection methods for autonomous driving applications”. En: *IEEE Transactions on Intelligent Transportation Systems* 20.10 (2019), págs. 3782-3795 (vid. pág. 3).
- [22] Mehran Yazdi y Thierry Bouwmans. “New trends on moving object detection in video images captured by a moving camera: A survey”. En: *Computer Science Review* 28 (2018), págs. 157-177 (vid. pág. 3).
- [23] James Philbin y col. “Object retrieval with large vocabularies and fast spatial matching”. En: *2007 IEEE conference on computer vision and pattern recognition*. IEEE. 2007, págs. 1-8 (vid. pág. 3).
- [24] Tsung-Yi Lin y col. “Microsoft coco: Common objects in context”. En: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part V 13*. Springer. 2014, págs. 740-755 (vid. pág. 3).
- [25] Fifa. *Ball Dataset*. <https://universe.roboflow.com/fifa/ball-z1138>. Open Source Dataset. visited on 2023-03-29. Oct. de 2022. URL: <https://universe.roboflow.com/fifa/ball-z1138> (vid. pág. 4).
- [26] Xavier Desurmont y col. “TRICTRAC Video Dataset: Public HDTV Synthetic Soccer Video Sequences With Ground Truth”. En: 2006 (vid. págs. 4, 5).
- [27] Karol Kurach y col. “Google research football: A novel reinforcement learning environment”. En: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, págs. 4501-4510 (vid. págs. 4, 5).
- [28] Jacek Komorowski y Grzegorz Kurzejamski. “Graph-Based Multi-Camera Soccer Player Tracker”. En: *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2022, págs. 1-8 (vid. págs. 4, 5).

- [29] Lia Morra, Francesco Manigrasso y Fabrizio Lamberti. “SoccER: Computer graphics meets sports analytics for soccer event recognition”. En: *SoftwareX* 12 (2020), pág. 100612. ISSN: 2352-7110. DOI: <https://doi.org/10.1016/j.softx.2020.100612>. URL: <https://www.sciencedirect.com/science/article/pii/S2352711020303253> (vid. págs. 4, 5).
- [30] Jianhui Chen y James J Little. “Sports camera calibration via synthetic data”. En: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 2019, págs. 0–0 (vid. pág. 5).
- [31] Jose Cerqueira Fernandes y Benjamin Kenwright. “Identifying and extracting football features from real-world media sources using only synthetic training data”. En: *arXiv preprint arXiv:2209.13254* (2022) (vid. pág. 5).
- [32] Divya Bhargavi, Erika Pelaez Coyotl y Sia Gholami. “Knock, knock. Who’s there?—Identifying football player jersey numbers with synthetic data”. En: *arXiv preprint arXiv:2203.00734* (2022) (vid. pág. 5).
- [33] Jonas Theiner y col. “Extraction of Positional Player Data From Broadcast Soccer Videos”. En: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Ene. de 2022, págs. 823-833 (vid. pág. 5).
- [34] Konstantinos Rematas y col. “Soccer on your tabletop”. En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, págs. 4738-4747 (vid. pág. 5).
- [35] Michal Sypetkowski, Grzegorz Kurzejamski y Grzegorz Sarwas. “Football Players Pose Estimation”. En: *International Conference on Image Processing and Communications Challenges*. 2018 (vid. pág. 5).
- [36] understand.ai Georgi Urumov. *Semantic Segmentation - The Misunderstood Data Annotation Type*. 2021. URL: <https://understand.ai/blog/annotation/machine-learning/autonomous-driving/2021/02/17/semantic-segmentation-misunderstood-data-annotation-type.html> (vid. pág. 9).
- [37] Victor W Lee y col. “Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU”. En: *Proceedings of the 37th annual international symposium on Computer architecture*. 2010, págs. 451-460 (vid. pág. 10).
- [38] All3DP. *Top 10: The Best 3D Modeling Software (Some Are Free)*. 2023. URL: <https://all3dp.com/1/best-free-3d-modeling-software-3d-cad-3d-design-software/> (vid. pág. 10).
- [39] techradar Mark Pickavance. *Best 3D modeling software and rendering software (March 2023)*. 2023. URL: <https://www.techradar.com/best/best-3d-modelling-software> (vid. pág. 10).
- [40] Roland Hess. *Blender foundations: The essential guide to learning blender 2.5*. Taylor & Francis, 2013 (vid. pág. 11).
- [41] Lance Flavell. *Beginning blender: open source 3d modeling, animation, and game design*. Apress, 2011 (vid. pág. 11).