TRABAJO FINAL DE MÁSTER

MÁSTER EN BIG DATA DEPORTIVO

COMPUTER VISION COMO HERRAMIENTA PARA LA DETERMINACIÓN DEL FUERA DE JUEGO

Cecilio José Martínez Pérez

DIRECTOR

David Fombella

Cracovia

23 de Junio de 2022

Índice

Agradecimientos, 3

0. Introducción, 4
1. Contexto y marco referencial, 5
1.1 Línea actuación FIFA, 5
1.2 Línea actuación Sony, 7
2. Metodología, 9
2.1 ¿Qué es Computer Vision?, 9
2.2 ¿Qué es OpenCV?, 9
2.3 ¿Qué es Mediapipe?, 10
2.4 ¿Qué es Estimación de Pose?, 11
2.5 Alternativas a Mediapipe, 12
2.6 Unity3d, 14
3. Caso práctico, 15
3.1P rocesado de imágenes, 16
3.2 Tracking de jugadores, 18
3.3 Renderizado3D, 20
3.4 Aspectos a mejorar, 22
4. Conclusiones, 24
5. Bibliografía, 25
6. ANEXO, 26

Agradecimientos

En primer lugar, dirigirme al profesorado, tutor del proyecto, ponentes y dirección académica por haber puesto su empeño a lo largo del transcurso de la VII Edición del Máster en Big Data Deportivo y por haber resuelto con diligencia las dudas y retos planteados por el alumnado.

En segundo lugar, agradecer a los alumnos Alberto Pérez Sancho, Eduardo Carbajal, Raúl Martin, compañeros de viaje desde el PreMaster por el apoyo prestado, buen humor y compartir sus puntos de vista.

En tercer lugar, para la realización de este TFM haber contado con el apoyo de Sebastian Jarzębak, árbitro internacional en Liga Ekstraklasa, por haberme asistido y dado su valoración y asesorarme en las cuestiones planteadas.

En último lugar, y no menos importante, a mi esposa, Stefan y Andrés, venido al mundo en noviembre durante el Máster, por el tiempo 'robado' para realización de este Máster, sin vuestro apoyo y paciencia no hubiera sido posible.

0. Introducción

El proyecto se enmarca en torno a la propuesta PFM 8 dedicada a Computer Vision en la recreación virtual del jugador para acciones de fuera de juego sujetas a revisión de VAR empleando técnicas de procesado de imágenes, inteligencia artificial y renderizado 3D.

Pretendemos ilustrar el estado del arte describiendo en capítulos posteriores las diferentes líneas de investigación de la FIFA en lo referente a mejoras de la técnica del VAR desde su puesta en marcha en 2018 y también una visión actual de lo que otras compañías pioneras en el sector como Sony está llevando a cabo para mejorar dicho proceso mediante su herramienta comercial Skeletrack.

Indudablemente no podremos competir en tiempo, esfuerzo y recursos humanos y materiales lo que grandes corporaciones están llevando a cabo en estos momentos, pero esto no es obstáculo para que a nivel de usuario surjan iniciativas que pretendan emular con medios más modestos un marco de trabajo similar, dada la difusión del conocimiento y la democratización del uso de herramientas Open Source.

Desde nuestro hogar, disponiendo de unos conocimientos en Python medio-avanzados, podemos explorar alguna de las potentes librerías como OpenCV e indagar en las infinitas posibilidades que ofrece actualmente Computer Vison aplicadas para nuestro caso práctico como el procesamiento de imagen y detección de pose del jugador en el momento de la jugada clave.

Gracias también a Unity3D, herramienta con gran uso en el diseño de videojuegos, pero que también su uso se puede hacer extrapolable a nuestra simulación, podemos crear una representación en 3 dimensiones de lo que hemos sido capaces de exportar en nuestro anterior código de Python.

Como broche final ilustraremos un caso práctico con todo lo mencionado en los puntos anteriores.

1. Contexto y marco referencial

1.1 Línea actuación FIFA

La aplicación de la tecnología a la regla del fuera de juego, que estipula que un jugador atacante no puede estar más allá del último defensor cuando recibe el balón, ha sido controvertida. Esto se debe a que todavía depende de que los árbitros asistentes humanos vean las imágenes de vídeo, con la ayuda de líneas de para ayudarles a tomar la decisión del fuera de juego.

Tras la implantación del sistema de árbitros asistentes de vídeo (VAR) después de su incorporación a las Reglas de Juego en marzo de 2018, la FIFA pretende seguir mejorando la tecnología del VAR en todos los niveles del juego. En particular, el desarrollo de una tecnología de fuera de juego semiautomatizada proporcionando al VAR información adicional y más precisa para ayudar al proceso de toma de decisiones del árbitro y para que el proceso de revisión sea lo más eficiente posible.

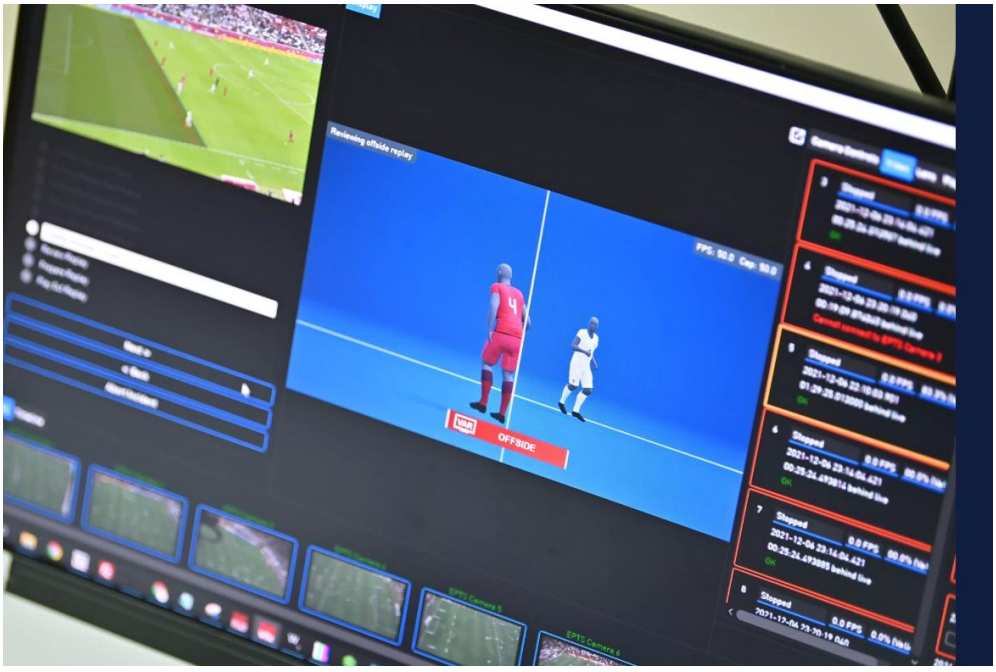


Figura1. Muestra de los avances en el VAR en la Copa Mundial de Clubes FIFA 2021[2]

Uno de los principales retos en el desarrollo de una tecnología avanzada de fuera de juego es la detección precisa y automatizada del punto de saque. El proveedor de la tecnología informó al grupo sobre posibles soluciones, como los datos de seguimiento de la tecnología de sensores o los datos de vídeo de los sistemas de cámaras

Figura2. Integración End-to-End Sistema EPTS Sevilla 2021

Además, el sistema tiene que identificar correctamente qué parte del cuerpo coloca a un jugador en fuera de juego o en juego. Las pruebas de precisión han demostrado que los operadores humanos tienden a elegir diferentes partes del cuerpo para las líneas de fuera de juego. También se ha avanzado en este ámbito, ya que el sistema automatizado presentado ha aprendido a modelar correctamente el esqueleto de un jugador. En el futuro, los algoritmos desarrollados del sistema deberían ser capaces de identificar automáticamente qué parte del cuerpo colocó al jugador en fuera de juego y a qué distancia[1].

El objetivo de los ensayos del Programa de Calidad de la FIFA es comprobar la interacción del equipo en su conjunto, así como el rendimiento del sistema en tres áreas principales: latencia, sincronía y calidad del vídeo.

Tras la certificación de los sistemas de VAR con el nuevo estándar mundial de calidad, el uso de un sistema certificado por la FIFA será obligatorio a partir de julio de 2022[2].·

1.2 Línea actuación Sony

Paralelamente en similar línea de investigación, la división Hawk-Eye que Sony adquirió y que es empleada en otros deportes populares como tenis y en futbol para la detección de la línea de gol, está desarrollando un software comercial denominado Skeletrack basado en similar concepto anterior aprovechando la gran experiencia que la compañía trae en lo referente a sensores y procesado de imágenes.

En una demostración de la nueva tecnología Skeletrack de Hawk-Eye en el CES 2022, Sony mostró cómo la "recreación virtual" de la acción en el campo podría utilizarse pronto para juzgar automáticamente las decisiones de fuera de juego con una precisión similar a la de su actual tecnología de línea de gol.

Skeletrack es potencialmente una gran mejora en el proceso actual de VAR porque combina la tecnología de seguimiento óptico con la tecnología de IA que puede analizar la captura de vídeo en tiempo real.

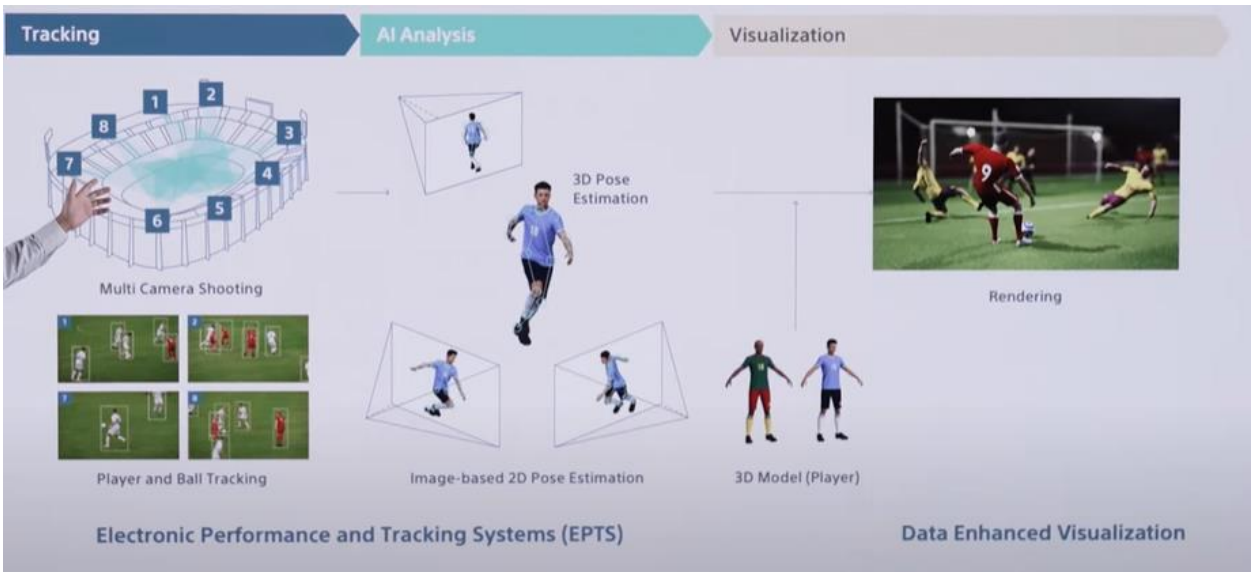


Figura3. Muestra de EPTS and Tecnología de Visualización de Datos.

Durante la demostración, Sony dijo que "Skeletrack analiza la posición del balón, así como las posiciones y posturas de los jugadores en tiempo real con la IA, y luego visualiza el juego a través de la recreación virtual". A continuación, mostró cómo podría aplicarse esto a una decisión de fuera de juego del VAR, con modelos 3D de los jugadores (y sus partes del cuerpo relevantes) que se analizan automáticamente en tiempo real para ver si se debe anular un fuera de juego.

Sony no ha revelado más detalles sobre cuándo podría llegar esta tecnología de ojo de halcón de última generación al VAR, limitándose a decir que "Skeletrack puede hacer avanzar la equidad y la transparencia de la toma de decisiones en el deporte". Pero si funciona tan bien como sugieren las demos, podríamos verlo en las ligas y torneos de fútbol en los próximos años.

Aunque todavía hay obstáculos para un sistema de fuera de juego 100% preciso, incluyendo la determinación del momento exacto en que se golpea el balón, el Skeletrack parece un avance prometedor respecto al proceso actual del VAR[3].

Figura4. Muestra Hawk-eye desarrollada por Sony

2. Metodología

2.1 ¿Qué es Computer Vision?

Computer Vision en adelante CV es un subcampo de machine learning que enseña a los ordenadores a ver y entender imágenes digitales. CV es una parte del dominio de la Inteligencia Artificial que puede ser usada en un abanico de grades posibilidades en el campo de conducción autónoma de vehículos, detección de objetos, robótica, seguimiento de objetos.

CV funciona en tres pasos básicos:

- Adquirir una imagen: las imágenes, incluso los conjuntos grandes, se pueden adquirir en tiempo real a través de videos, fotos o tecnología 3D para su análisis.
- Procesando la imagen: los modelos de aprendizaje profundo automatizan gran parte de este proceso, y se entrenan al ser alimentados primero con miles de imágenes etiquetadas o identificadas previamente.
- Entendiendo la imagen: el paso final es el paso interpretativo, donde se identifica o clasifica un objeto

2.2 ¿Qué es OpenCV?

OpenCV es una librería de código abierto principalmente usada para CV, procesamiento de imágenes y machine learning. Comenzó siendo un proyecto de investigación de Intel, siendo actualmente la biblioteca más popular para Computer Vision, contando con cientos de funciones para la captura, análisis y manipulación de las imágenes, que pueden venir de distintas fuentes como cámaras webcam, archivos de video e imagen.

Es una biblioteca multiplataforma que puede funcionar tanto en Mac,Windows o Linux.

Con el soporte de OpenCV podemos procesar imágenes y videos en distintas áreas de aplicación:

- Estimación de pose de cámara
- Reconocimiento facial
- Reconocimiento de gestos
- Segmentación
- Reconocimiento de objetos.
- Realidad aumentada

OpenCV está totalmente desarrollado en C++pero incluye conector con Python.

2.3 ¿Qué es Mediapipe?

Mediapipe es el marco multiplataforma de Google para crear distintas canalizaciones de procesamiento de datos.

Tareas que podemos realizar con la ayuda de MediaPipe:

- Detección de rostros y malla de rostros
- Pose y detección holística
- Detección y seguimiento de objetos
- Reconocimiento de mano
- Plantear estimación

A nivel de estructura básica, el modelo tiene una arquitectura de codificador-descodificador basada en MobileNetV2, las redes neuronales de Google para clasificación y detección visual. Mediante dichas redes, la proyección 2D de la imagen y un algoritmo de estimación 3D, el modelo es capaz de procesar una salida en 3D de dicho objeto[4].

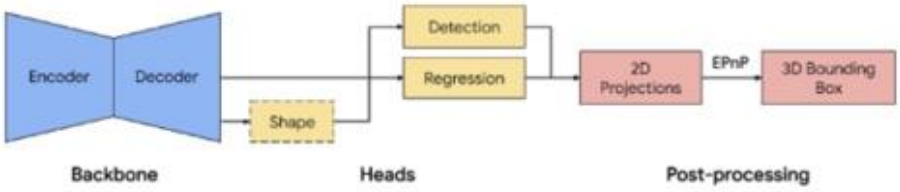


Figura3. Arquitectura de red y postprocesamiento para la detección de objetos 3D

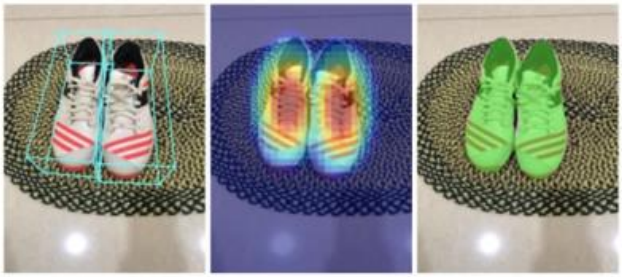


Figura5. Resultados de muestra de nuestra red-Izquierda Imagen 2D original con cuadros delimitadores estimados, Medio-Detección de objetos por distribución gaussiana, Derecha-Máscara de segmentación predicha

2.4 ¿Qué es Estimación de Pose?

Estimación de pose humana desde video pregrabado o en tiempo real desempeña un papel crucial en diversos campos, como el control de gestos de todo el cuerpo, la cuantificación del ejercicio físico y el reconocimiento del lenguaje de signos.

Media Pipe Pose es un marco de trabajo para el seguimiento de la postura corporal de alta fidelidad, que toma la información de los fotogramas de vídeo RGB e infiere 33 puntos de referencia en 3D en todo el cuerpo humano. Este método supera a otros y consigue muy buenos resultados en tiempo real.

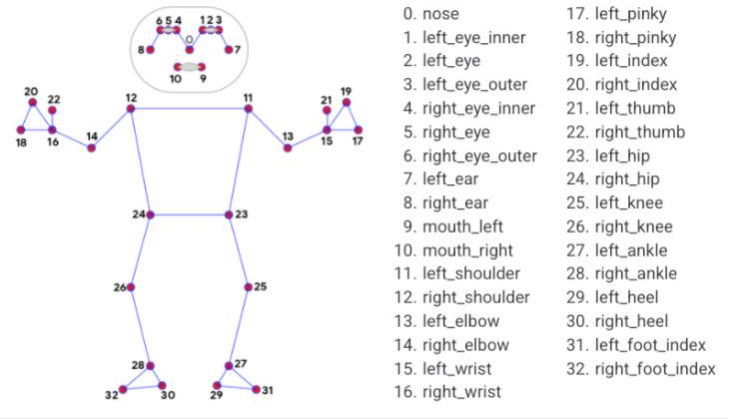


Figura6. Landmarks para Estimación de Pose[5]

2.5 Alternativas a Mediapipe

Los algoritmos de detección de pose se pueden dividir entre dos categorías:

-Top-down: Primeramente, de emplea un detector de personas, seguido de la estimación de las partes y luego calcula la pose para cada persona.

-Bottom-up: Se detectan todas las partes de las personas en la imagen, se asocian las partes pertenecientes a cada persona y se agrupan.

Destacamos la velocidad de procesamiento de MediaPipe(top-down) frente a Openpose (bottom-up), a la vez que se requiere de una menor carga computacional. También frente a disrupciones, es un modelo más robusto.

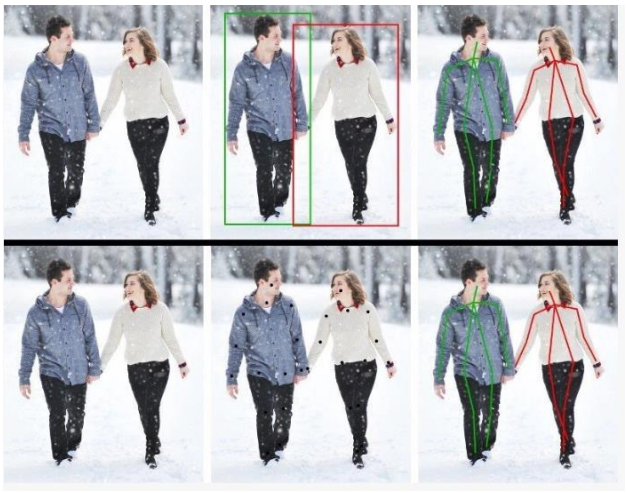


Figura7. Arriba: Aproximación top-down Abajo: Aproximación Botton-up

Otras alternativas que podemos encontrar al framework de MediaPipe serían Openpose(bottom up) , Alphapose (top down), OpenPifPaf(bottom-up) o Deepcut(bottom up).

Botton-up aproximación es más adecuada para imágenes concurridas, y con poses complejas y situaciones donde hay solapamiento, sin embargo, el coste es la precisión, como podemos comprobar, además del mayor coste computacional comentado.

Figura8. Comparativa algoritmos https://www.bilibili.com/video/BV1Kg41157wS/

2.6 Unity3d

Unity es un motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Microsoft Windows, Mac OS, Linux. La plataforma de desarrollo tiene soporte de compilación con diferentes tipos de plataformas

Crea imágenes ricas y ultrarrealistas con las herramientas asistidas por IA de Unity ArtEngine para automatizar[6].

Entre las ventajas de uso podemos enumerar su facilidad de usar, multiplataforma. Entre las desventajas el espacio ocupado por los proyectos y rendimiento computacional que se necesita para mover una simple implementación.

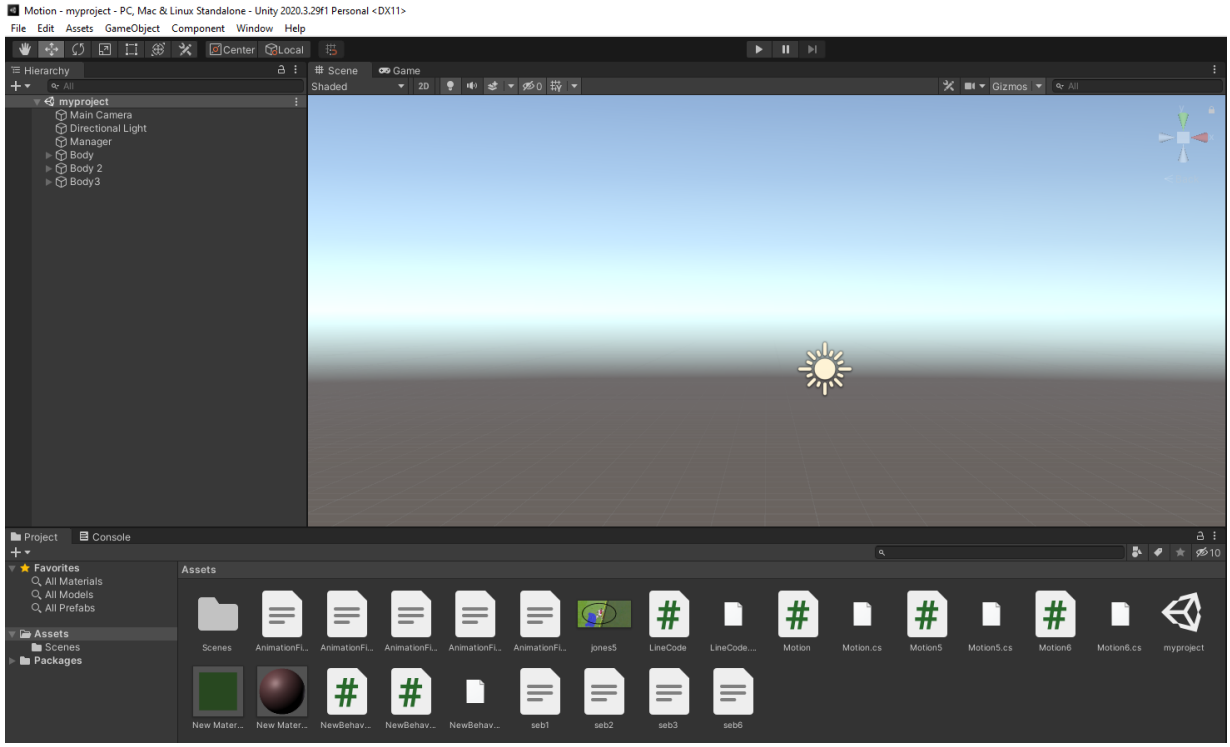


Figura 9. Unity interfaz

3. Caso práctico

3.1 Planteamiento del caso práctico

El planteamiento del caso práctico desde el proceso de captura de imagen hasta su visualización en 3D se puede dividir en 3 partes que detallaremos en los siguientes puntos:

La primera parte es la asociada a la captura de imágenes y tratamiento de la imagen previa a la estimación de la pose. Mediante diferentes scripts en Python seremos capaces de convertir una secuencia de video en imágenes individuales, seleccionar un área de interés dentro de nuestra imagen e incluso aplicar un difuminado a la imagen en cierta área de interés para desbloquear alguna situación en caso de que algoritmo de detección de jugador no sea capaz de lograrlo por superposición de jugadores.

Para la ejecución de los scripts en Python nos servimos de PyCharm, un entorno de desarrollo integrado, con amplias funciones en Python.

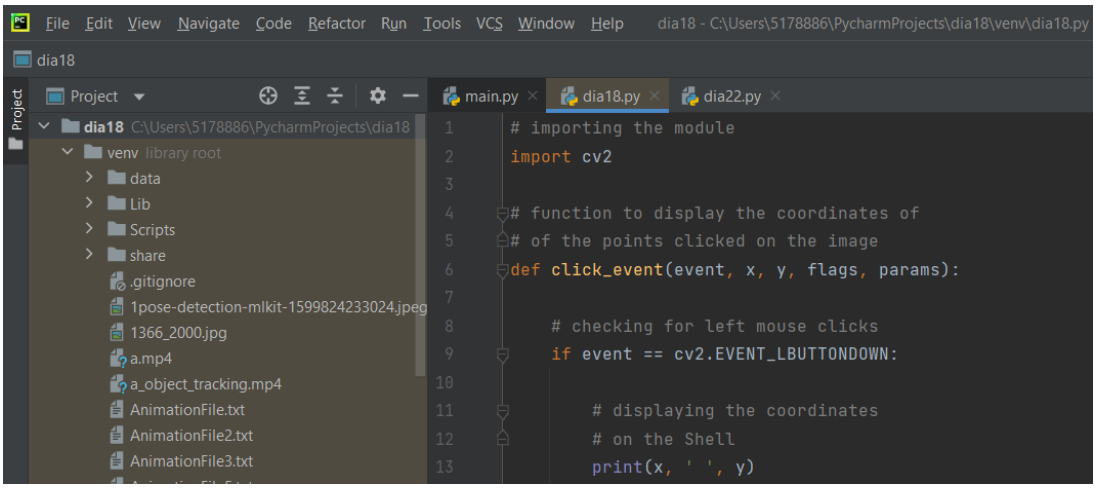
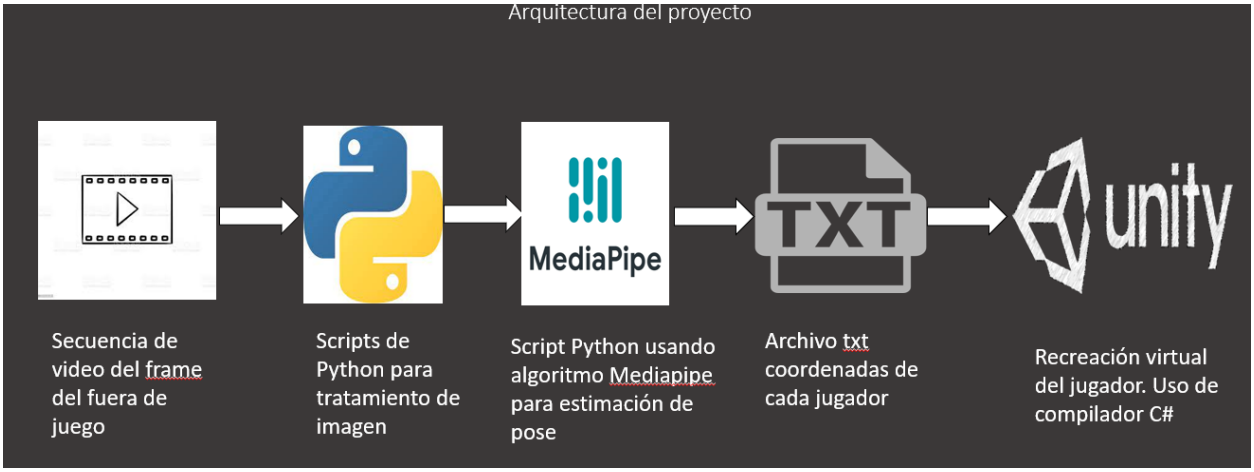


Figura 10. Interfaz PyCharm

La segunda parte utiliza el análisis de IA para convertir las secuencias de los jugadores en datos. Una vez seleccionado y adecuado el frame óptimo para revisión, a través de un script de Python seremos capaces de generar tantas salidas como jugadores en revisión del VAR sean necesarios.

La salida de este script nos proporcionará un fichero de texto con las coordenadas x,y,z de cada landmark.

La tercera es la creación de visualizaciones mediante esos datos. Con el software de Unity3d seremos capaces de representar un modelo en 3D recreando la escena del fuera de juego y añadir alguna figura geométrica si es necesario.



3.1 Procesado de imágenes

Para nuestro proceso tenemos que seleccionar el frame de la jugada del fuera de juego en el momento en que el balón es golpeado, presuponiendo que tendríamos acceso en tiempo real al contenido de esa cámara que capta esa secuencia de video, con las posibilidades que ofrece OpenCV podríamos ser capaces de procesar dicha imagen.

OpenCV cuenta con una potente función de edición de video. Empleando los siguientes métodos:

VideoCapture(File_path) : lectura de archivo video en formato mp4

imwrite(filename, img[, params]) : Guarda las imágenes en un directorio especifico

Con tal propósito ejecutaremos el script “extract images from video.py”disponible en anexo.

Por otra parte, OpenCV nos ayuda a controlar distintos tipos de evento al hacer click con el ratón y nos permite operar con ellos, esto nos es especialmente de utilidad a la hora de dibujar las líneas de campo ya que podemos obtener las coordenadas X e Y de nuestra imagen para luego

ser representada en Unity sin tener que utilizar otro software auxiliar de procesado de imagen. Ejecutaremos el script “extract pixel.py” disponible en el anexo.

También nos sirve de utilidad para conocer la región de interés del jugador en caso de hacer el difuminado de la imagen usando el código en Python en vez de usar un software paralelo de procesado de imagen, ya que una de las limitaciones con la que nos encontramos es que en caso de haber múltiples jugadores detecta solo uno de ellos.

OpenCV proporciona 4 técnicas de difuminado[8]. Emplearemos el método GaussianBlur para tal efecto.

Ejecutaremos el script “difuminado” disponible en el anexo.

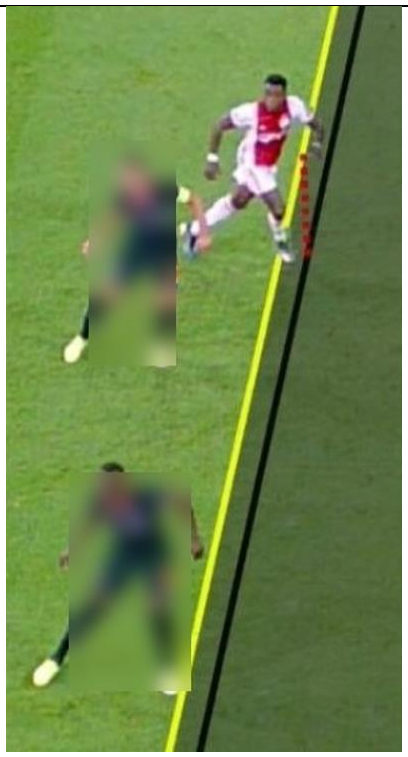


Figura 11.Salida script “Difuminado de imagen”

Figura 12.Salida script “Representación Píxel”

3.2 Tracking de jugadores

Para el seguimiento de jugadores nos valemos de código Python, importando las librerías más actuales disponibles anteriormente mencionadas necesarias tales cvzone proveniente de opencv y mediapipe.

Importamos el paquete PoseDetector que nos captura las 33 landmarks .

El fichero de salida almacena cada cada landmark sus coordenadas x,y,z por lo que tenemos un total de 99 parametros por secuencia.

Creamos un bucle que lea las imágenes y que despliegue una ventana para visualizar dicho contenido a través del método imshow.

Creamos nuestro objeto detector invocando a PoseDetector sin necesidad de ningún parámetro.

Usando este detector invocamos a findPose() este nos devolverá una imagen visualizando las landmarks.

Para registrar nuestras landmarks crearemos una lista sobre la zona de nuestro contorno de imagen bboxInfo

Ponemos condición en caso de detectar un contorno, crear un string para almacenar nuestros 33 landmarks.

Hacemos cierta modificación en el fichero de salida para obviar la referencia al identificador de la pose y que nos devuelva el trío de coordenadas *x,y,z*. Para la coordenada *y* debido al hecho que considera 0 el punto de la esquina superior izquierda pero en Unity lo considera desde abajo invertimos la coordenada, con lo que restando a la altura de nuestra imagen mediante el método shape[0] siendo 0 la referencia a la altura ya que shape nos devuelve la tupla en el orden de filas, columnas y canales (si la imagen es en color), conseguimos la referencia correcta para crear en Unity nuestra animación.

Después de almacenar la tupla, por cada coordenada, adjuntamos en una lista.

Creamos la lógica para guardar en fichero de salida al presionar una tecla.

Dicho código se encuentra en el script “tracking jugador” disponible en el anexo.

Figura13.Imagen original[9]

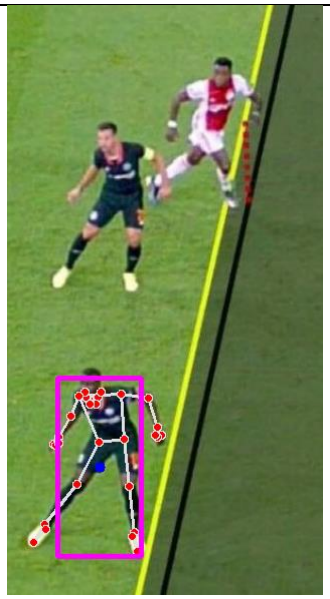


Figura14.PoseEstimation Jugador 1

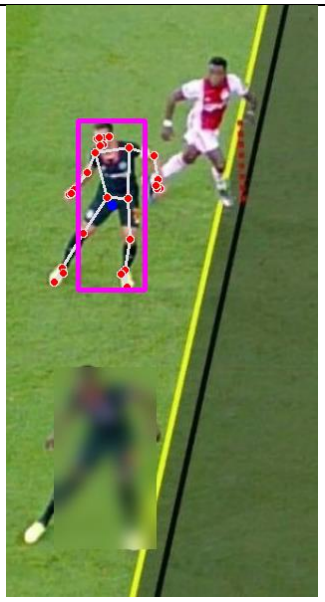


Figura15PoseEstimation Jugador 2

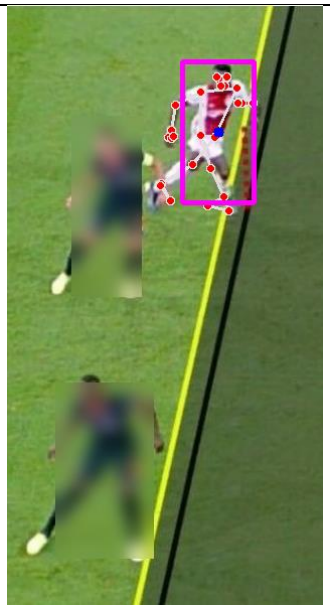


Figura16.PoseEstimation Jugador 3

3.3 Renderizado3D

Para el proceso de creación del jugador en la herramienta Unity, en primer lugar, creamos un objeto esfera, y luego creamos la clase Motion para cada uno de los jugadores, recibiendo como fichero de entrada la salida de nuestras coordenadas proporcionadas en Python desarrollando cierto código en C#, siendo C# una evolución que Microsoft realizó tomando como referencia C , C++ y Java.

Archivo disponible en el anexo bajo el nombre “Motion.cs”.

Representamos cada landmark como una esfera, por lo que tendremos 33 esferas por jugador.

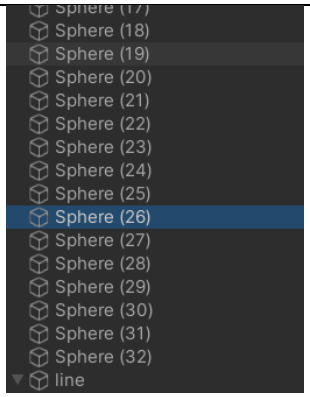


Figura17. Creación objeto esfera

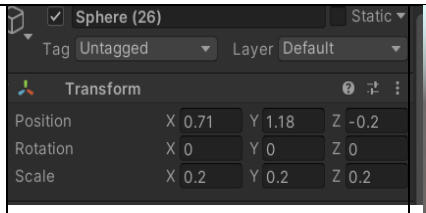


Figura18. Representación coordenada x,y,z

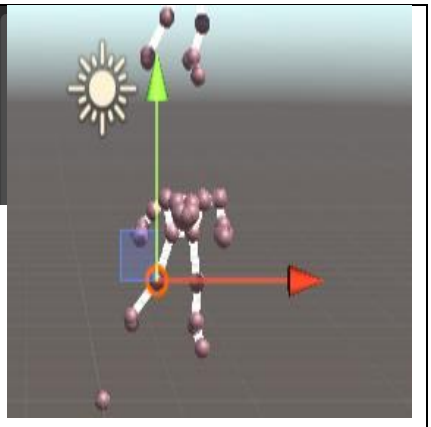


Figura19. Representación esfera sobre el cuerpo del jugador

Posteriormente para dar de mayor realismo a la figura del jugador, queda unir las esferas entre sí, siguiendo el diagrama de estimación de pose, es decir si por ejemplo tendremos que trazar una línea entre la rodilla y el tobillo derechos, usaremos las esferas 28 y 26, por lo que nuevamente crearemos un objeto línea para las uniones.

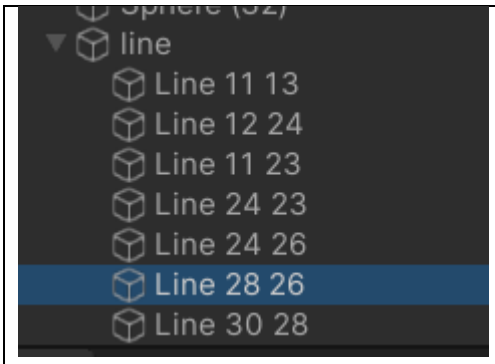


Figura20. Objeto línea

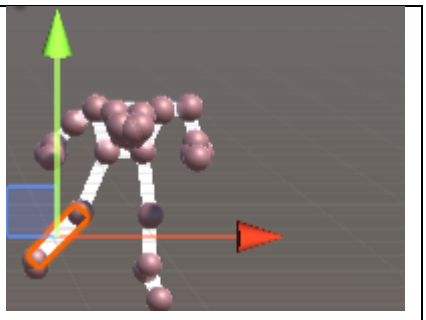


Figura21. Representación línea sobre el cuerpo del jugador

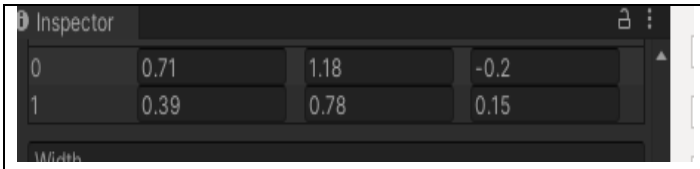


Figura22. Coordenadas de las esferas 26 y 28

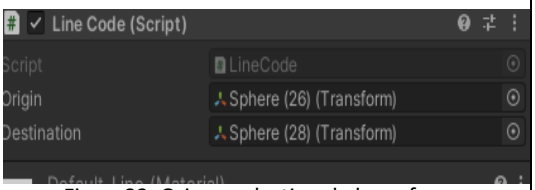


Figura23. Origen y destino de las esferas

Para salir de dudas y constatar el fuera de juego, trazamos la línea de fuera de juego y en la vertical representamos la posición más adelantada del jugador infractor que viene dada por las coordenadas de su hombro izquierdo.

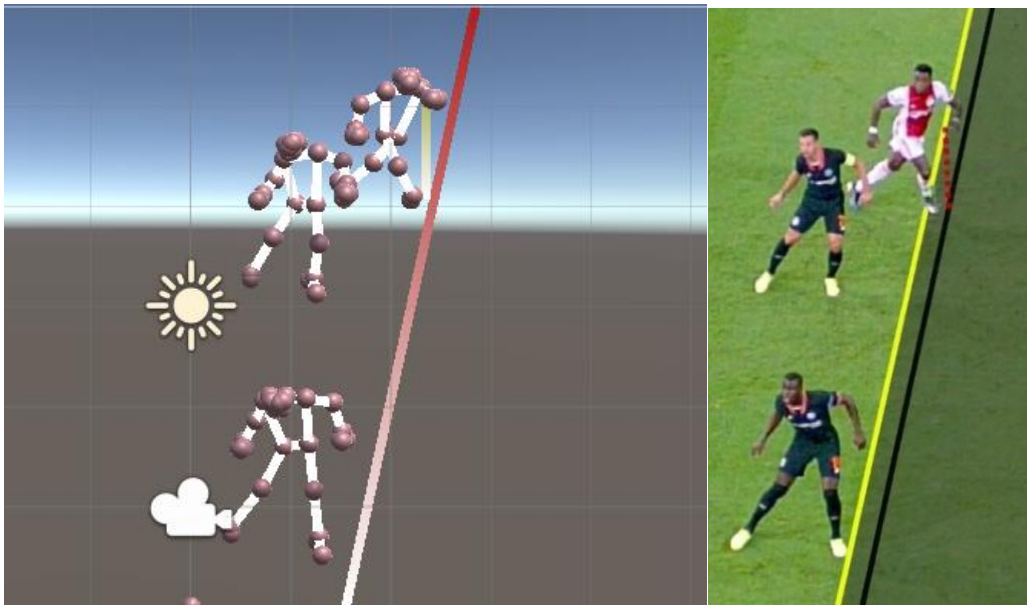


Figura24. Representación Unity3D de nuestros jugadores vs frame original

3.4 Aspectos a mejorar

Hay que considerar que potencialmente vamos a tener limitaciones en caso de superposición en el área del contorno de la imagen si disponemos de un único ángulo de filmación. Indudablemente el algoritmo de detección de contorno no podrá discernir cual parte del cuerpo se corresponde con cada persona debido al solapamiento. Es por ello por lo que por ejemplo la herramienta Skeletrack cuenta con distintas tomas de filmación a lo largo del estadio para posibles casos de superposición.

También otra limitación y mejora futura supondría ser capaz de procesar simultáneamente más de una detección de pose por imagen, actualmente solo somos capaces de lograr una pues es una limitación de Mediapipe, por lo que requiere de cierta previa manipulación como el difuminado del jugador más cercano de ahí que hayamos investigado en el script para selección del área de interés para acelerar el proceso.

Como mencionamos en el apartado 2.5 existen alternativas a Mediapipe pero en detrimento de nuestra precisión y coste computacional.

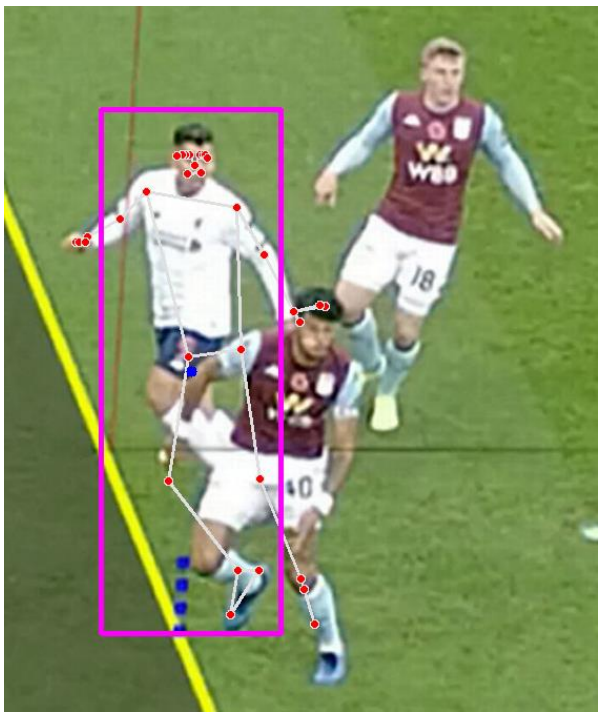


Figura25. Limitación algoritmo de seguimiento de jugador por solapamiento

Como posibles vías de investigación y futuras áreas de expansión del trabajo, sería optimizar el pipeline desde la toma de la imagen hasta la carga de la imagen en el cliente de tratamiento de la imagen, junto con una posible determinación automática del landmark en posible fuera de juego de los jugadores más adelantados, de manera que se puedan reducir los tiempos desde la sala VAR.

4. Conclusiones

El fin último de este Proyecto Final de Máster es la de dar una visión del estado del arte en que se encuentran los nuevos desarrollos del VAR en lo referente al fuero de juego. Como se mencionó al inicio, el fuera de juego es una de las jugadas que siempre más controversia ha causado dada la complejidad de determinar el momento exacto y los múltiples actores que intervienen.

La tecnología actual viene en nuestra ayuda para que la aplicación de la decisión final sea lo más justa posible en aras de la deportividad.

Aprovechando la oportunidad que se bridaba de realizar un PFM acerca de Computer Vision, previamente ya tenía inquietudes sobre dicho campo y ha venido como anillo al dedo durante la investigación para este proyecto, pues he podido profundizar sobre aspectos como detección de objetos, segmentación y clasificación siendo innumerables los casos prácticos dentro de la industria y más específicos para el deporte.

No hay ninguna duda de que es un campo floreciente que el futuro que nos depara es prometedor. La captación de la estimación de la pose aparte del VAR podría abrir nuevas puertas al estudio del rendimiento del jugador, analizando sus posturas, ya que son detalles que no se habían podido ver hasta ahora pues eran invisibles al ojo humano, que podrían ser aplicados a la prevención de lesiones, por ejemplo.

5. Bibliografía

1. https://www.fifa.com/football-development/news/fifa-organises-remote-demonstration-of-advanced-offside-technology FIFA 30 JUNIO 2020
2. https://www.fifa.com/en/news/fifa-certifies-var-systems-under-new-global-quality-standard FIFA 29 OCTUBRE 2021
3. https://www.techradar.com/news/sonys-skeletrack-tech-could-finally-solve-soccers-var-offside-controversies MARK WILSON 5 ENERO 2022
4. https://ai.googleblog.com/2020/03/real-time-3d-object-detection-on-mobile.html Adel Ahmadyan y Tingbo Hou, Software Engineers, Google Research 11 MARZO 2020
5. https://google.github.io/mediapipe/solutions/pose.html
6. https://unity.com/es/pages/more-than-an-engine
7. https://www.geeksforgeeks.org/extract-images-from-video-in-python/
8. https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html
9. https://twitter.com/dalejohnsonespn/status/1187401990642511873 @DaleJohnsonESPN
10. https://docs.unity3d.com/ScriptReference/Transform-localPosition.html

6. ANEXO

1 Script anexo “Extract images from video.py”

```
# Importing all necessary libraries
import cv2
import os
# Read the video from specified path
cam = cv2.VideoCapture("C:\\Users\\Admin\\PycharmProjects\\project_1\\openCV.mp4")

try:

        # creating a folder named data
        if not os.path.exists('data'):
                os.makedirs('data')

# if not created then raise error
except OSError:
        print ('Error: Creating directory of data')

# frame
currentframe = 0

while(True):

        # reading from frame
        ret,frame = cam.read()

        if ret:
                # if video is still left continue creating images
                name = './data/frame' + str(currentframe) + '.jpg'
                print ('Creating...' + name)

                # writing the extracted images
                cv2.imwrite(name, frame)

                # increasing counter so that it will
                # show how many frames are created
                currentframe += 1
        else:
                break

# Release all space and windows once done
cam.release()
cv2.destroyAllWindows()
```

2. Script anexo “Extract pixel.py”

```
# importing the module
import cv2

# function to display the coordinates of
# of the points clicked on the image
def click_event(event, x, y, flags, params):

	# checking for left mouse clicks
	if event == cv2.EVENT_LBUTTONDOWN:

		# displaying the coordinates
		# on the Shell
		print(x, ' ', y)

		# displaying the coordinates
		# on the image window
		font = cv2.FONT_HERSHEY_SIMPLEX
		cv2.putText(img, str(x) + ',' +
					str(y), (x,y), font,
					1, (255, 0, 0), 2)
		cv2.imshow('image', img)

	# checking for right mouse clicks
	if event==cv2.EVENT_RBUTTONDOWN:

		# displaying the coordinates
		# on the Shell
		print(x, ' ', y)

		# displaying the coordinates
		# on the image window
		font = cv2.FONT_HERSHEY_SIMPLEX
		b = img[y, x, 0]
		g = img[y, x, 1]
		r = img[y, x, 2]
		cv2.putText(img, str(b) + ',' +
					str(g) + ',' + str(r),
					(x,y), font, 1,
```

```
                                    (255, 255, 0), 2)
                cv2.imshow('image', img)

# driver function
if __name__=="__main__":

        # reading the image
        img = cv2.imread('lena.jpg', 1)

        # displaying the image
        cv2.imshow('image', img)

        # setting mouse handler for the image
        # and calling the click_event() function
        cv2.setMouseCallback('image', click_event)

        # wait for a key to be pressed to exit
        cv2.waitKey(0)

        # close the window
        cv2.destroyAllWindows()
```

3. Script anexo “Difuminado.py”

```
import cv2

# Read in image
image = cv2.imread('seb1.jpg')

# Create ROI coordinates
topLeft = (67, 109)
bottomRight = (137, 289)
x, y = topLeft[0], topLeft[1]
w, h = bottomRight[0] - topLeft[0], bottomRight[1] - topLeft[1]

topLeft1 = (51, 374)
bottomRight1 = (149, 549)
x1, y1 = topLeft1[0], topLeft1[1]
w1, h1 = bottomRight1[0] - topLeft1[0], bottomRight1[1] - topLeft1[1]

# Grab ROI with Numpy slicing and blur
ROI = image[y:y+h, x:x+w]
ROI2=image[y1:y1+h1, x1:x1+w1]

blur = cv2.GaussianBlur(ROI, (51,51), 0)
blur2 = cv2.GaussianBlur(ROI2, (51,51), 0)
# Insert ROI back into image
image[y:y+h, x:x+w] = blur
image[y1:y1+h1, x1:x1+w1] = blur2
#cv2.imshow('blur', blur)
cv2.imshow('image', image)
cv2.imwrite('C:/Users/5178886/PycharmProjects/dia18/venv/sebdif.jpg', image)
cv2.waitKey()
```

4 Script anexo “Tracking jugador.py”

```
import cv2
from cvzone.PoseModule import PoseDetector

cap = cv2.VideoCapture('off.jpg')

detector=PoseDetector()
posList =[]
while True:
    success,img=cap.read()
    img=detector.findPose(img)
    lmlist,bboxInfo= detector.findPosition(img)

    if bboxInfo:
        lmString=''
        for lm in lmlist:

            lmString+= f'{lm[1]},{img.shape[0]-lm[2]},{lm[3]},'
        posList.append(lmString)

    print(len(posList))
    cv2.imshow("Image",img)
    key=cv2.waitKey(10000)
    if key == ord('s'):
        with open ("AnimationFile5.txt",'w') as f:
            f.writelines(["%s\n" %item for item in posList])
```

5 Script anexo “Motion.cs”

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

public class Motion5 : MonoBehaviour
{
    public GameObject [] Body3;
       List<string> lines;
       int counter=0;
    // Start is called before the first frame update
    void Start()
    {
        lines = System.IO.File.ReadLines("Assets/seb6.txt").ToList();
    }

    // Update is called once per frame
    void Update()
    {
        print(lines[0]);
              string[] points = lines[counter].Split(',');

              for (int i=0;i<32;i++)
              {

              float x=float.Parse(points[0+(i*3)])/100;
              float y=float.Parse(points[1+(i*3)])/100;
              float z=float.Parse(points[2+(i*3)])/100;
              Body3[i].transform.localPosition= new Vector3(x,y,z);
              }


              counter +=1;
              if (counter==lines.Count){counter=0;}
              Thread.Sleep(30);
    }
}
```