Proyecto Fin de Grado
Grado en Ingeniería Robótica, Electrónica y Mecatrónica

Sistema para el Seguimiento de Atletas en Retransmisiones Deportivas

Autor: Juan Salinas Hernández
Tutor: Rubén Martín Clemente

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Sistema para el Seguimiento de Atletas en Retransmisiones Deportivas

Autor:
Juan Salinas Hernández

Tutor:
Rubén Martín Clemente
Profesor titular

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Grado: Sistema para el Seguimiento de Atletas en Retransmisiones Deportivas

Autor: Juan Salinas Hernández

Tutor: Rubén Martín Clemente

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mis padres

Agradecimientos

A mi familia, mi padre, mi madre y mi hermano, no sólo por su apoyo anímico y económico, sino por ser referentes constantes en mi vida. Les debo mis éxitos, presentes y futuros, serían imposibles sin ellos.

A mis amigos de siempre, que me acompañáis desde que tengo uso de razón y con los que sé que siempre puedo contar. Gracias por vuestro inconmensurable apoyo.

A mis compañeros de carrera, amigos todos, por su gran calidad humana y estar siempre dispuestos a ayudar al grupo. Los largos días de universidad y estudio no hubiesen sido igual sin vosotros.

A Alejandra, esa persona siempre dispuesta a escucharme y soportarme en esos momentos en los que nada parece salir bien, por ser capaz de ayudarme y sacarme una sonrisa siempre.

Finalmente, a esos profesores que han demostrado entrega y vocación por nuestro aprendizaje, y que han conseguido despertar cada día más nuestro interés por la ingeniería. Principalmente a mi tutor Rubén, que a pesar de los tiempos raros y complicados que hemos vivido recientemente, se ha sabido adapatar y persistir con su ayuda desde la distancia.

Juan Salinas Hernández

Sevilla, 2020

Resumen

En este proyecto se tratará de procesar y analizar retransmisiones deportivas con el fin de obtener de forma automatizada la mayor cantidad de información posible, como la detección y seguimiento de jugadores y pelota, detección del campo de juego y datos para estadísticas relevantes como distancias recorridas, trayectorias, etc. El objetivo será aplicar distintas técnicas de percepción y visión por computador para procesar la imagen, orientado a obtener determinado parámetro de forma automática. Además, se hará uso de filtros predictivos que ayudarán al seguimiento de los jugadores y a la detección en circunstancias más complejas.

Para todo ello, usaremos la herramienta de OpenCV en Python, una biblioteca libre orientada a la visión artificial.

Abstract

This project will try to process and analyze sports broadcasts in order to automatically obtain as much information as possible, such as the detection and monitoring of players and ball, detection of the playing field and data for relevant statistics such as distances traveled, trajectories, etc. The objective will be to apply different techniques of perception and computer vision to process the image, aimed at obtaining a parameter automatically. In addition, predictive filters will be used to help monitor players and switch in more complex circumstances.

For all this, we will use the OpenCV tool in Python, a free library oriented to computer vision.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xiv**

Índice de Tablas **xvi**

Índice de Figuras **xviii**

2 Introducción **1**
2.1 Visión por computador *1*
2.2 Objetivos *2*
2.3 Herramientas *3*
2.4 Estado del arte *3*

3 Detección de jugadores y balón **5**
3.1 Extracción del fondo *5*
3.1.1 Condición G>R>B 5
3.1.2 Gradiente de Sobel 5
3.1.3 Operaciones morfolóficas 7
3.1.4 Máscara del terreno de juego 8
3.2 Transformada de Hough *9*
2.3. Etiquetado *11*

4 Seguimiento **13**
4.1 Algoritmo de seguimiento *13*
4.2 Filtro de Kalman *15*
4.2.1 Teoría del filtro de Kalman 15
4.2.2 Implementación del filtro con OpenCV 19

5 Programa y funcionalidades **21**
5.1 Descargas de videos tests *21*
5.2 Programa principal *22*

6 Análisis y resultados **25**
6.1 Resultados obtenidos *25*
6.1.1 Aportación del filtro de Kalman 25
6.1.2 Oclusión momentánea 28
6.1.3 Iluminación no homogénea 30
6.1.4 Líneas del terreno de juego 31
6.1.5 Correspondencia de jugadores 33
6.1.6 Coste computacional 34
6.1.7 Resolución del video 35

6.2 Aplicación real *36*
6.2.1 Mapa de calor 36
6.2.2 Velocidad del jugador 37
6.2.3 Tiempo corriendo 38

7 Conclusiones, futuras investigaciones y otras aplicaciones **39**
7.1 Conclusiones *39*
7.1.1 Proceso de detección 39
7.1.2 Implementación del filtro de Kalman 39
7.2 Futuras investigaciones *39*
7.3 Otras aplicaciones *40*

8 Códigos **41**
8.1 Código programa principal *41*
8.2 Código programa de descarga de videos de prueba *48*

9 Refrerencias **49**

ÍNDICE DE TABLAS

Tabla 5.1 Frames y tiempos de procesamiento de cada video 35
Tabla 5.2 Tiempo corriendo de los jugadores 4 y 12 del video 1 38

ÍNDICE DE FIGURAS

Ilustración 1.1 Sistema de visión del ojo de halcón 4
Ilustración 2.1 Extracción del fondo con condición de color 5
Ilustración 2.2 Aplicación filtro de Sobel 7
Ilustración 2.3 Extracción con filtro de Sobel y condición de color 7
Ilustración 2.4 Ejemplo de dilatación, erosión, cerrado y apertura 8
Ilustración 2.5 Máscara del terreno de juego 9
Ilustración 2.6 Medidad tranformada de Hough [8] 9
Ilustración 2.7 Espacio de Hough [8] 10
Ilustración 2.8 Detección de rectas con transformada de Hough 10
Ilustración 2.9 Problema corte de jugasdores 11
Ilustración 2.10 Resolución corte de jugadores 11
Ilustración 2.10 Detección de jugadores 12
Ilustración 3.1 Medida de posición 15
Ilustración 3.2 Posición en el siguiente instante 16
Ilustración 3.3 Predicción con velocidad aleatoria 16
Ilustración 4.3 Implementación filtro de Kalman en Matlab 19
Ilustración 5.1 Detecciones de jugadores por frame 26
Ilustración 5.2 Falsos positivos en video 2 26
Ilustración 5.3 Faslo positivo del algoritmo de detección 27
Ilustración 5.4 Falso positivo por predicción del Filtro de Kalman 27
Ilustración 5.5 Secuencia predición de Kalman 27
Ilustración 5.6 Secuencia de oclusión del balón 28
Ilustración 5.7 Secuencia predicción de Kalman ante oclusión del balón 28
Ilustración 5.8 Imagen binarizada de juagadores en contacto 29
Ilustración 5.9 Resultado del filtro de Kalman en oclusión de jugadores 29
Ilustración 5.10 Detección ante 7 jugadores juntos 30
Ilustración 5.11 Detección de jugadores juntos en instantes posteriores 30
Ilustración 5.12 Ejemplo de iluminación no homogénea 31
Ilustración 5.13 Problemática de transformada de Hough ante jugadores cercanos 32
Ilustración 5.14 Tranformada de Hough sobre jugadores con línea discontinua 32
Ilustración 5.15 Varios jugadores sobre línea del terreno de juego 33
Ilustración 5.16 Fallo de correspondencia por predicciones imprecisas 33
Ilustración 5.17 Jugadores detectados al inicio y a lo largo del video 34
Ilustración 5.18 FPS de procesamiento por video 35

Ilustración 5.19 Mapa de calor del jugador 4 y 12 del video 1 37
Ilustración 5.20 Velocidad de los jugadores 4 y 12 del video 1 37
Ilustración 6.1 Procesado de jugada de rugby 40

INTRODUCCIÓN

En este capítulo se explicará de forma general las principales disciplinas científicas en las que se basarán el proyecto, poniéndolas en contexto y detallando su uso y alcance actual. Además, se definirá y acotará el objetivo concreto de este proyecto, exponiendo también los resultados que se esperan del mismo. Finalmente, se contextualizará con trabajos previos ya implementados en la industria y en el ámbito en el que se enfoca el proyecto.

1.1 Visión por computador

La visión por computador es una disciplina científica que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica para que puedan ser tratados por un ordenador [1]. Esta disciplina comenzó en la década de los 60, con el objetivo de emular la forma en la que los humanos reconocemos nuestro entorno para así dotar de inteligencia a una máquina. Uno de los primeros investigadores de los que se tiene constancia en estudiar esta disciplina es Larry Roberts, considerado uno de los padres de internet. El objetivo de su investigación fue el de extraer información 3D a partir de imágenes 2D, de forma que pudiera ver distintas perspectivas de un sólido. Esto lo consiguió captando diferentes imágenes de un bloque, y, procesándolas posteriormente para detector bordes según cambios bruscos en la intensidad de grises [2]. Desde entonces, esta rama ha seguido evolucionando, sobre todo en las últimas, debido a los grandes avances tanto en la captación de imágenes, consiguiendo cada vez mayor calidad y cantidad de información, con equipos más pequeños y ligeros, como en el postprocesado, contando con equipo de una gran capacidad de computación.

Como en gran cantidad de avances tecnológicos, en la visión por computador se ha tratado de aprender de la propia naturaleza, en cómo vemos los seres humanos, y posteriormente tratar de emularla con la tecnología que disponemos. Como sabemos, el ojo es el órgano que nos dota del sentido de la vista, sobre el cual incide la luz provocando una serie de impulsos nerviosos en la retina. Esto es gracias a nuestros foto-receptores, los conos y los bastones. Los conos son células sensibles a los colores, de forma que existen tres tipos de conos, cada uno sensibles a una longitud de onda distinta, las correspondientes a los colores rojo, verde y azul. Por otro lado, los bastones serán células sensibles a la intensidad de luz, es decir el brillo. El encargado de emular este funcionamiento en el ámbito de la visión artificial son las cámaras y sensores ópticos, de forma que disponemos de una serie de fotodiodos dispuestos en matriz, cada uno capaz de capar la intensidad para los colores rojo, verde o azul, y posteriormente almacenando una carga correspondiente a los fotones incidentes. Esta imagen es posteriormente digitalizada y procesada por un procesador.

Como vemos, la visión por computador tiene gran cantidad de aplicaciones en distintos ámbitos, como puede ser la edición y filtrado para el ámbito fotográfico, la capacidad de dotar de percepción a un robot para que pueda tomar decisiones en consecuencia en el ámbito de la robótica, manipulación de objetos en la industria, realizar un diagnóstico por medio de una imagen en el ámbito de la medicina, tomar y recomponer imágenes tomadas por un satélite en el ámbito de la astronomía, vigilancia automática en el ámbito de la seguridad o capacidad de detectar vehículos y leer matrículas para el control del tráfico. Su misión principal en todos ellos es la de hacer más comprensibles para el ser humano las imágenes captadas y, sobre todo, extraer la mayor cantidad de información de ellas de cara a poder automatizar al máximo nivel posible distintos procesos. En todos estos ámbitos, y otros más que no se han nombrado, la visión artificial está jugando un importante siendo una de las principales ramas a investigar para los futuros avances tecnológicos.

Además de estar presente en muchos ámbitos, la visión también tiene una gran cantidad de campos con los que está relacionado y en los que se basa. Como uno de los principales relacionados tenemos la inteligencia artificial, ya que, por un lado, la visión ayuda a la extracción de características con los que formar los datasets con los que entrenar los distintos algoritmos y redes neuronales, y por otro lado la visión se ayuda de la inteligencia artificial para la detección automática de ciertos elementos o patrones en una imagen, como por ejemplo el

reconocimiento facial. Otro campo en el cual también se basa de forma directa la visión artificial es el de procesamiento de señales ya que, gran parte de los métodos usados para procesar señales de una variable, se extienden de manera natural a señales de dos variables o multivariable que forman la visión. Por otro lado, tenemos la optoelectrónica, de la que se depende directamente ya que es la encargada de fabricar los distintos sensores con los que captamos las imágenes, de forma que ayuda con la investigación de distintos tipos de sensores que se ajusten a las características concretas que se deseen para determinada aplicación. Finalmente, también hay que destacar la aportación de la ciencia de computación, ya que el procesamiento de las imágenes conlleva un gran costo computacional, teniendo que operar sobre miles de píxeles por frame, y muchas veces, con la necesidad de procesar dichos frames a tiempo real, cosa que sería impensable sin los equipos actuales.

1.2 Objetivos

Tras exponer las bases de la visión por computador y los distintos ámbitos en los que puede actuar, pasamos a explicar la función que tendrá en este proyecto, y establecer la estructura que tendrá. Como hemos visto en el apartado anterior, la visión por computador tiene infinidad de aplicaciones, muchas de ellas orientadas a poder automatizar una tarea para la que antes se necesitaba de la dedicación de una o más personas. El caso ante el que nos enfrentamos tiene la misma intención. La función del programa que desarrollaremos será la detección y seguimiento automática de los jugadores a partir de las imágenes que ya se obtienen para las retransmisiones deportivas. Esta tecnología pretende ser útil para realizar las distintas estadísticas referentes al juego, como puede ser distancia recorrida por los jugadores, mapas de calor según la zona en la que se haya movido el jugador, porcentajes de posesión por equipo… Existen gran cantidad de estadísticas que pueden ser automatizadas tan solo con las imágenes captadas por las distintas cámaras, y obtenerlas a tiempo real, lo cual es muy interesante tanto por el lado del espectador, que puede conocer las estadísticas al instante según sucede el partido, como por parte del cuerpo técnico, ya que pueden contar con valiosa información para la toma de decisiones. Además, también ayudará al posterior análisis de los partidos, tanto del propio equipo para estudiar en que aspectos mejorar, como en el estudio del equipo contrario, para comprobar su estilo de juego y ver las posibles debilidades del equipo, muy útil en un mundo deportivo en el que los datos y su análisis juegan cada vez un papel más importante. También hay que destacar que el método propuesto es poco intrusivo con el juego, ya que es aplicable con la tecnología y cámaras que ya se usan en los eventos deportivos profesionales, y no tiene ninguna repercusión ni lastre en la actuación de los atletas, como si pudieran ser otros métodos aplicados como equipar al propio atleta de sensores como ya se hace para recopilar otra serie de parámetros, sobre todo en este caso usados para parametrizar el estado físico del deportista.

Para todo esto, tendremos los siguientes objetivos por pasos:

1. Detección de la pelota
2. Detección de los jugadores
3. Seguimiento de ambos con filtro del Kalman

Para todo esto aplicaremos una sucesión de técnicas existentes usadas en este ámbito, como puede ser la eliminación del fondo según unos márgenes en las intensidades de los colores RGB, el gradiente de Sobel para la detección de elementos en una imagen, la transformada de Hough para la detección de rectas, el filtro de Kalman, y otras técnicas morfológicas.

En consecuencia, el objetivo será aplicar todas estas técnicas de la forma correcta, definiendo los parámetros concretos que se ajusten a los mejores resultados posibles, con la intención de poder comprobar su bondad para la aplicación que se pretende que desempeñe. Para esto, estudiaremos cuánto tiempo es capaz nuestro programa de realizar el seguimiento de forma acertada, comprobando la cantidad de falsos positivos y falsos negativos que obtenemos, y mediremos cómo de efectiva es la predicción del filtro de Kalman según la posición real del jugador. También atenderemos al coste computacional que todo esto requiere, de forma que podamos valorar las capacidades del programar para actuar en tiempo real o no.

Una vez ajustados todos estos a los resultados, usaremos estos datos para calcular algunas estadísticas de forma automática, a modo de ejemplo del alcance y potencial que pueden tener estas técnicas aplicadas en casos reales.

1.3 Herramientas

Para facilitar esta tarea, usaremos la libraría OpenCV. Esta librería (Open Source Computer Vision), es una librería de Código abierto dedicada a la visión por computador y al aprendizaje automático, de forma que contiene más de 2500 funciones de algoritmos optimizados, entre los que se incluyen las técnicas más clásicas usadas en la industria. Estos algoritmos pueden usarse para detectar y reconocer caras, identificar objetos, clasificar acciones de personas en videos, seguir el movimiento de objetos, extraer modelos 3D de objetos, etc. Esta es una librería muy usada en empresas, grupos de investigación, por cuerpos gubernamentales y en toda la industria en general [3]. Además, esta librería estará implementada en diversos lenguajes como C++, Python, Java y MATLAB. Para este proyecto concreto usaremos Python.

El lenguaje escogido es Python, en primer lugar, debido a mi conocimiento y manejo previo del lenguaje, y, en segundo lugar, porque es un lenguaje muy usado para tareas de este tipo, donde OpenCV está implementado y en el que funciona con una eficiencia más que correcta.

Además de OpenCV, otras de las librerías que se usarán para leer las imágenes y videos será la creada por Adrian Rosebrock y publicada por medio de su blog de divulgación de visión por computador e inteligencia artificial, PyImageSearch. Esta será la libraría imutils.video, que usaremos para capturar cada uno de los frames de los videos que procesemos, en lugar del método implementado en OpenCV, ya que nos soluciona varios problemas, sobre todo en lo que respecta a velocidad y eficiencia. Esta función implementa sus procesos por medio de hilos, de forma que puede mejorar el procesamiento del pipeline hasta un 52% [6].

Finalmente se hará uso de otras librerías como argparse, con la que podremos introducir algunos argumentos por la línea de comandos, de forma que podamos modificar algunos parámetros como el video a leer o el modo del programa sin necesidad de realizar ningún cambio en la programación. Además, usaremos otras librerías de uso común como pueden ser numpy, time y math, para diversas operaciones necesarias a lo largo del programa.

1.4 Estado del arte

Desde inicios del siglo XXI son muchas las incorporaciones tecnológicas que se han realizado en el deporte, principalmente en tres líneas diferentes. La primera de ellas para obtener la mayor cantidad posible de información de cómo se sucede la competición, de forma que ayuden a analizar estrategias de juego, estado físico del atleta, etc. Por otro lado, se han realizado muchos avances para ofrecer una experiencia más completa para los telespectadores en las retransmisiones, de forma que no solo se ha avanzado en la calidad con la que se emiten los eventos, sino que también se ha avanzado en las distintas perspectivas que se captan, recomponiendo una imagen prácticamente en 3D con la que poder sumergirte en el juego y no perderte prácticamente nada de lo que ocurre sobre el terreno. Finalmente, una de las líneas en las que se están realizando nuevas incorporaciones y que está muy de moda en los últimos años, es la asistencia en el arbitraje por medio del video, con el objetivo a hacer la competición lo más justo posible, corrigiendo así el error humano que siempre conlleva la supervisión por una persona.

Con respecto a este último, uno de los deportes pioneros, por su aplicación desde hace década y media, y por su buena implementación, por acierto y por ser poco intrusivo en el ritmo de juego, es el tenis, con el sistema conocido como ojo de halcón. Este es un sistema de origen británico, que ha sido aplicado también en otros deportes con menos repercusión a nivel internacional como el billar inglés o el criquet. El objetivo de esta tecnología es esclarecer el bote de las bolas según su trayectoria y velocidad, para poder corroborar si ha entrado dentro o fuera. Este sistema se compone de un total de 10 cámaras que captan toda la pista, de forma que un programa informático es el encargado de procesar dichas imágenes para determinar donde ha botado la bola. El propio programa es capaz de a partir de los datos realizar una animación 3D con los datos recopilados, en tan solo unos segundos, de forma que se puede ver casi al instante tanto en el estadio como en la televisión. Además, tan solo cuenta con un error de unos 2-3 milímetros, por lo que este sistema es uno de los mejores ejemplos de aplicación de la visión artificial en el deporte, por los grandes resultados que arroja tanto en precisión como en tiempo [4].

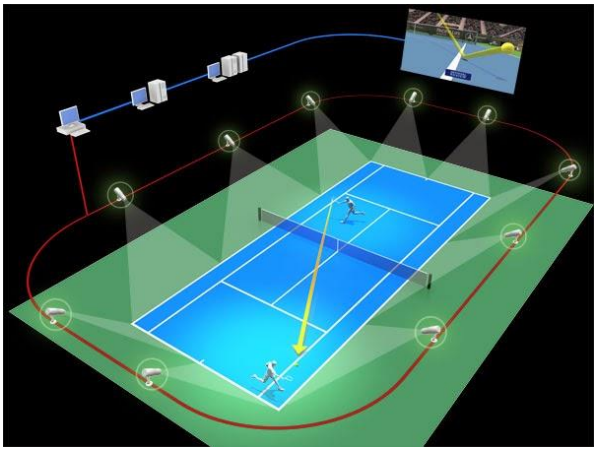


Ilustración 1.1 Sistema de visión del ojo de halcón

Este tipo de sistemas de video arbitraje ya se aplican también en gran cantidad de deportes olímpicos desde hace años con extraordinarios resultados. Uno de los últimos deportes con repercusión mundial en aplicarlos es el fútbol, donde las principales ligas y competiciones internacionales ya hacen uso de lo que denominan VAR (Video Assistan Referee). Este caso, a pesar de tener buenos resultados en algunos aspectos como determinar si el balón ha entrado en la portería o no, cosa que puede saber el árbitro de forma instantánea ya que llega la información a su reloj, todavía queda un largo camino en el que investigar para automatizar todavía más estas decisiones, ya que actualmente algunas de ellas pueden tardar hasta 2-3 minutos en decidirse, rompiendo el ritmo de juego.

En el aspecto en el que el fútbol si es un gran referente es en las retransmisiones deportivas, contando con nuevas incorporaciones como los sistemas de repetición 3D. Estos nuevos sistemas de repetición de jugadas en 3D utilizan hasta 42 cámaras de visión artificial ubicadas alrededor de un estadio para capturar imágenes que se reconstruyen en vistas 3D estereoscópicas de la acción y que luego se añaden a la transmisión en directo. Otros sistemas utilizan múltiples cámaras de visión artificial para generar vistas de 360 grados de la propia acción que se transmiten a los espectadores a través de cascos de realidad virtual para sumergirlos totalmente en la experiencia deportiva. La tecnología de las cámaras de visión artificial también se utiliza normalmente para realizar un seguimiento de los jugadores y del balón, mejorar el entrenamiento de los jugadores y reforzar la supervisión del estadio y los espectadores. El seguimiento del movimiento de los jugadores y sus parámetros, como las tácticas y la estrategia del partido, brindan tanto a los entrenadores como a los medios que retransmiten el partido una herramienta valiosa para el análisis general del juego. Los datos de las imágenes y los metadatos, como el tiempo en el campo y la distancia recorrida, se pueden usar para analizar rendimiento de los jugadores [5].

Hemos visto tan solo unos ejemplos de la infinidad de aplicaciones que tiene la visión artificial en el mundo del deporte, que como comprobamos, a pesar de tener ya múltiples aplicaciones en funcionamiento con espectaculares resultados, la industria tiene aún la necesidad de seguir avanzando en este sentido, por lo que se esperan grandes inversiones económicas que apoyen nuevas investigaciones al respecto.

DETECCIÓN DE JUGADORES Y BALÓN

En el presente capítulose abordará todo lo relativo a la detección de las pelota y los jugadores. Para ello, explicaremos las distintas técnicas que se han ido aplicando y los parámetros con los que se han ajustado dichas ténicas para obtener los mejores resultados.

2.1 Extracción del fondo

En primer lugar, procesaremos cada frame para determinar cuál es el fondo de nuestra imagen, que en el caso del fútbol será el terreno de juego y la grada, de forma que los separaremos de los jugadores y la pelota, que son los elementos principales que queremos procesar. Para ello, seguiremos el algoritmo propuesto en [7], en el que se propone extraer el campo por medio de dos métodos distintos, uno en el que discriminaremos según una condición de color de cada pixel RGB, y otra en el que haremos uso del gradiente de Sobel para la detección de bordes.

2.1.1 Condición G>R>B

En este método, binarizaremos la imagen según un criterio de color para extraer el terreno de juego. Como sabemos previamente que este terreno de juego será mayoritariamente verde, sabemos que cumplirá lo siguiente: Green>Red>Blue, según la información que tengamos del RGB de la imagen. De esta forma pondremos a 0 los píxeles de la imagen que cumplan dicha condición (visualmente como negro) y a 1 los que no la cumplan. Este método presenta un problema debido a que los grises también cumple la condición mencionada, por lo que el método no sólo discriminaría el terreno de juego que es lo que deseamos, sino que también discriminaría las líneas que limitan las distintas zonas del campo de juego y el balón.

A nivel de programación podremos hacerlo de forma sencilla gracias a la función *cv2.split()*, con la que separaremos los tres canales rgb en tres matrices distintas, por lo que podremos aplicar dicha condición tan solo con un par de comparaciones en estos canales. El resultado de aplicar esta condición es el siguiente:

Ilustración 2.1 Extracción del fondo con condición de color

2.1.2 Gradiente de Sobel

Para corregir el problema anterior en el que se discriminaba elementos importantes como el balón, implementaremos otro método que trabajará en conjunto con el anterior. Este método se basará en el gradiente de Sobel, que es un algoritmo muy usado en visión artificial ya que es muy eficiente para la detección de bordes

en una imagen. Este método se basa en la variación de la intensidad luminosa de un punto a otro, que será estudiado como un gradiente, que tendrá dos componentes, una para la dirección "x" y otra para la dirección "y". Este gradiente puede aproximarse como:

$$G = \begin{pmatrix} \frac{\Delta f}{\Delta x} \\ \frac{\Delta f}{\Delta y} \end{pmatrix};$$

Es decir, tendremos lo siguiente para cada punto (i, j) de la imagen f:

$$G_{i,j} = \begin{pmatrix} f_{i,j} - f_{i,j-1} \\ f_{i,j} - f_{i-1,j} \end{pmatrix};$$

De esta forma se obtendría el gradiente en cada punto, al que se le podría calcular la dirección y el módulo.

Sobel, basándose en este método propuso un operador de vecindad, ya que normalmente los operadores de gradiente tienden a aumentar el ruido de la imagen, por lo que Sobel añade un pequeño suavizado de la imagen en su operador, ayudando a eliminar parte del ruido y a minimizar la aparición de falsos bordes. Así Sobel propone las siguientes máscaras, con las que se realizará la convolución con la vecindad de cada uno de los pixeles:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix};$$

Como ya comentamos en la introducción, OpenCV tiene ya implementados todos los algoritmos populares en la industria para el tratado de imágenes, y esta no es una excepción, por lo que mediante la función *cv2.Sobel()* podremos calcular este gradiente fácilmente. Dado que calculamos este gradiente en un solo canal, pasaremos a esta función la imagen monocromática equivalente, convertida de la siguiente forma:

```
gray = cv.cvtColor(img, cv.Color_BGR2GRAY)
```

Además, se añade un filtrado previo para obtener mejores resultados, en este caso un filtro gaussiano:

```
gray = cv.GaussianBlur(gray, (5, 5), 0)
```

Así procedemos a calcular el gradiente de Sobel para cada dirección, que la marcamos en el argumento 3 y 4, junto con el argumento 5 donde indicamos el tamaño de la máscara:

```
dx = cv.Sobel(gray, cv.CV_32F, 1, 0, (3, 3))
dy = cv.Sobel(gray, cv.CV_32F, 0 1, (3, 3))
```

Finalmente, aplicaremos un umbral de forma que para valores de gradiente superiores este los tomaremos como un borde, mientras que desecharemos los inferiores. Este umbral se aplicará directamente sobre el módulo del gradiente en las dos direcciones.

```
mag, _ = cv.cartToPolar(dx, dy, angleInDegrees=True)
_, SobelG = cv.threshold(mag, umbral, 255, cv.THRESH_BINARY)
```

Así obtendríamos la siguiente imagen:

Ilustración 2.2 Aplicación filtro de Sobel

Que sumándola a la imagen obtenida en el proceso de discriminación según color se obtiene lo siguiente:

Ilustración 2.3 Extracción con filtro de Sobel y condición de color

2.1.3 Operaciones morfológicas

Una vez obtenidas las siluetas de los jugadores y del balón, uniendo los resultados de la discriminación por color y del gradiente de Sobel, tratamos de mejorar la información que obtenemos mediante operaciones morfológicas. Estas operaciones morfológicas serán la dilatación y la erosión, que combinadas forman las técnicas de apertura y cerrado. Con estas operaciones se pretenderá corregir las imperfecciones que podamos tener en las siluetas, como huecos internos o que estén separadas. Concretamente la técnica que usaremos será la de cerrado, para corregir el interior de las siluetas. Explicaremos en primer lugar como funcionan la dilatación y la erosión.

Para aplicar estas ténicas, lo primero que definiremos será una máscara. Esta máscara será una matriz que tendrá un tamaño predeterminado que definiremos según convenga, y aunque podrá tener distintas formas normalmente serán formas simétricas como una caja o un disco. Un ejemplo de mascara de disco de 5x5 es la siguiente:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Esta mascara será la que se multiplique por nuestra imagen binarizada, es decir compuesta solo de 0 y 1. Se multiplicará por cada píxel de la imagen y su vecindario, en este caso en una region de 5x5, y la forma en la que se ejecuta esta operación es donde reside la diferencia entre dilatación y erosión. Por un lado tenemos la erosión, que actúa como un filtro de máximo local, es decir, si al multiplicar el pixel y el vecindario por la máscara, resultará haber un pixel con resultado a 1, rellenamos el pixel que estamos procesando a 1. De esta forma, hacemos los objetos más visibles (expandirlos) y se consigue rellenar huecos. Por otro lado tendremos la erosión, que realiza exactamente lo contrario a la dilatación, es decir, funciona como un filtro de mínimo local, de forma que si al multiplicar la máscara por el pixel y su vecindario existe algún resultado a 0, fijamos el pixel procesado a 0. Con esta técnica se consigue eliminar pequeñas islas, quedando solo aquellos objetos más grandes.

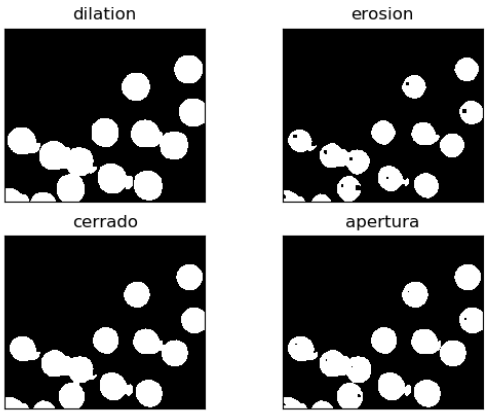


Ilustración 2.4 Ejemplo de dilatación, erosión, cerrado y apertura

Como hemos dicho, la técnica concreta que usaremos será la de cerrado. Esta técnica aplicará en primer lugar un dilatado, con un número de iteraciones concretas que definiremos nosotros, y posteriormente aplicará una erosión con la misma cantidad de iteraciones. Con esto, conseguiremos rellenar huecos dentro de los jugadores y la pelota, quedando como una silueta uniforme, sin afectar al tamaño, ya que erosionaremos todo lo dilatado. También se puede aplicar un cerrado, que será lo mismo pero en distinto orden, es decir, primero un erosionado y después un dilatado, de forma que eliminamos también los pequeños objetos no relevantes sin afectar demasiado a la forma del resto.

2.1.4 Máscara del terreno de juego

Como vemos en los resultados de los procesos anteriores, conseguimos eliminar de forma satisfactoria gran parte de la información del campo que no necesitamos para nuestra aplicación, pero aún tememos ciertos elementos que no deseamos en nuestra imagen, como puede ser la grada, los marcadores, etc. Dado que todos estos elementos indeseados se encuentran fuera del terreno de juego, que es nuestra área de interés, proponemos un procesamiento de la imagen en el que se obtenga de forma automática una máscara del terreno de juego en cada instante. Esta máscara se la aplicaremos a nuestra imagen procesada de forma que nos quedemos con aquellas zonas del interior de la máscara, eliminando así toda la zona de la grada. La idea original ha sido tomada del artículo *Player detection in field sports* [9], aunque se ha modificado la forma en la que se obtiene dicha máscara.

Para obtener esta máscara, haremos uso de la función *cv2.findContours()*, con la cual obtendremos todos los contornos de todos los elementos que tenemos en nuestra imagen, como es de esperar en una imagen de una retransmisión deportiva, el contorno más grande pertenecerá al terreno de juego. Así, haciendo uso de la función *max()*, obtendremos el contorno que tenga el área más grande, de forma que rellenando dicho contorno obtenemos la máscara que estábamos buscando.

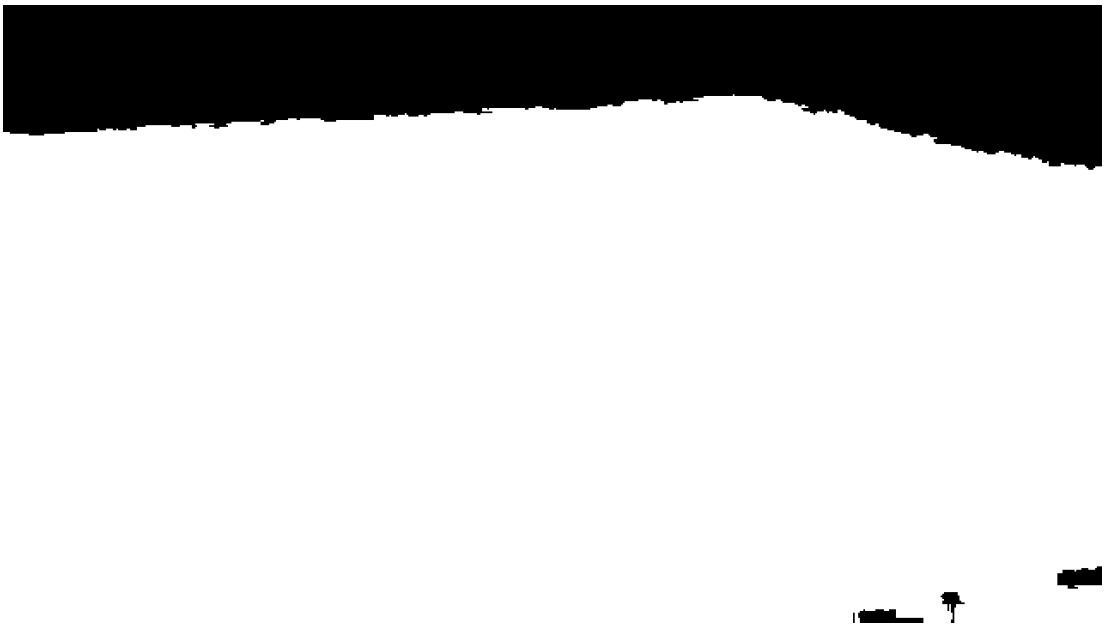


Ilustración 2.5 Máscara del terreno de juego

2.2 Transformada de Hough

La transformada de Hough es una técnica usada principalmente en visión por computador con la que podremos detectar de forma automática rectas, aunque también se puede adaptar para detectar otras formas como circunferencias o elipses. Este método nos será útil para nuestra aplicación en concreto para detectar las líneas del terreno de juego, pudiendo eliminarlas así de una forma sencilla. Dado que haremos la detección de los jugadores según las regiones de puntos conectados en una imagen binarizada, no nos interesan estas líneas ya que puede conectar a dos jugadores que estén sobre la misma línea del campo.

Este método se basará en un algoritmo que recorra las posibles rectas en la imagen, y según los votos de puntos coincidentes con dichas rectas se determina si es o no una recta destacable. Concretamente, por cada punto de una posible recta se trazan un número de rectas que pasen por este, recorriendo distintos ángulos. A estas rectas le calculamos una perpendicular que pase por el origen, determinando así el ángulo de la recta (θ) y la distancia al origen (ρ).

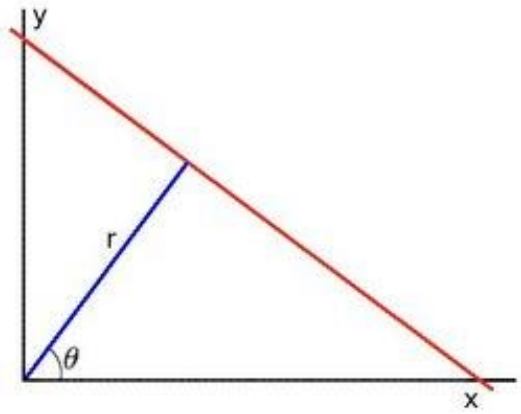


Ilustración 2.6 Medidad tranformada de Hough [8]

Una vez realizado esto por cada punto, obtendremos el grafo del espacio de Hough, en el que ρ será el eje de ordenadas y θ el de abscisas, y donde interpolaremos según los datos de cada recta de cada punto. De esta forma cada punto tendrá una curva, siendo el punto donde más intersecten dichas curvas donde obtengamos la recta con más votos. Dado que obtendremos muchas posibles rectas con diferente número de votos, se establece un umbral a partir del cual se determinará que la recta es destacable.

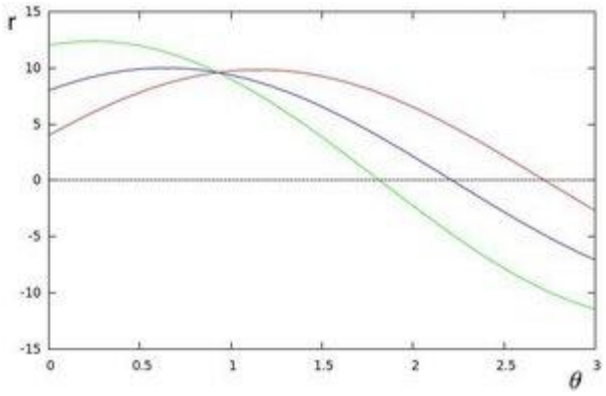


Ilustración 2.7 Espacio de Hough [8]

El algoritmo de la transformada de Hough ya viene implementado en OpenCV con la siguiente función:

```
Lines = cv2.HoughLinesP(img, rho, theta, threshold, lines = np.array([]), minLineLength, maxLineGap)
```

Como vemos, esta función tiene varios parámetros de entrada con los que podremos ajustar el algoritmo para obtener los mejores resultados para nuestra plicación en concreto. Estos parámetros son:

- Img: imagen de entrada a la que queremos aplicar el algoritmo.
- Rho: paso de distancia en pixeles para recorrer las posibles rectas.
- Theta: paso de ángulo en radianes para recorrer las posibles rectas.
- Threshold: número de intersecciones mínimas para detectar una línea.
- Lines: vector donde se almacenarán los parámetros ρ y θ de las líneas detectadas.
- minLineLegnht: mínimo número de puntos que pueden formar una línea. Líneas por debajo de este número serán descartadas.
- maxLineGap: hueco mínimo entre dos puntos para que estos se consideren de la misma línea.

Obtenemos los siguientes resultados para los parámetros: rho = 6, theta = pi/60, threshold = 800, minLineLength = 80, maxLineGap = 15.

Ilustración 2.8 Detección de rectas con transformada de Hough

Una vez realizado este cálculo, tan solo pondremos a 0 los pixeles por los que pasen estas rectas, de forma que queden borradas las líneas en nuestra imagen binarizada. Haciando esto, tendremos la problemática de que un

jugador quede partido en dos, como vemos en la siguiente imagen, lo que nos ocasionará que lo detecte como dos jugadores distintos o ni si quiera lo detecte.

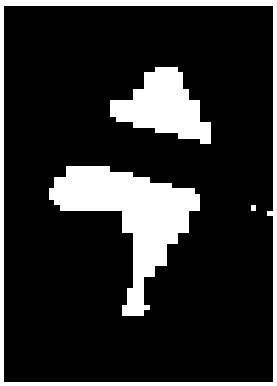


Ilustración 2.9 Problema corte de jugasdores

Dado que realmente no necesitamos borrar la línea entera, sino que lo que necesitamos es que esta línea no conecte los distintos elementos, se propone como solución hacer esta línea discontinua, de forma que corte esa "union" pero no corte la silueta del jugador en su totalidad. Puesto que las funciones de dibujado de OpenCV no permiten pintar una línea discontinua, realizamos una pequeña función para esta tarea. Como vemos en la siguiente imagen, ya no queda la silueta cortada.

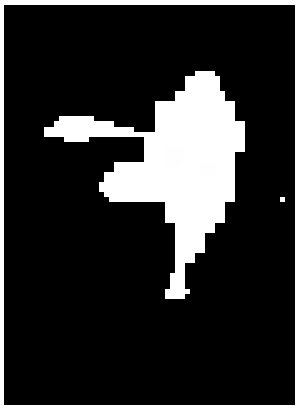


Ilustración 2.10 Resolución corte de jugadores

Esta función básicamente dibujará pequeñas rectas con la función *cv2.line* de OpenCV, calculando matemáticamente sus puntos de inicio y final para formar rectas equidistantes a lo largo de toda la línea detectada con la transformada de Hough.

2.3. Etiquetado

Finalmente, para detectar los jugadores después de todo el procesamiento, realizamos una función de etiquetado, a la que le pasamos como entrada la imagen binarizada procesada y la imagen original. Esta función se basará en otra función propia de OpenCV que detecta automáticamente las distintas regiones según los puntos blancos conectados entre sí. Esta función es la siguiente:

```
 Num_labels, labels_im, stats, centroids =
cv2.connectedComponentsWithStats(img)
```

Como vemos, a esta función le pasaremos la imagen binarizada, tras realizarle los procesos explicados en los apartados anteriores, y nos devolverá el número de etiquetas que ha detectado, el valor equivalente de cada región, los diferentes datos de la función y las coordenadas x e y del centro de la región.

Con los datos ofrecidos por esta función, discriminaremos si es un elemento de los que deseamos detectar según tres parámetros: área, la relación alto/ancho y la densidad. Estas características se obtienen directamente de la

salida de la función de connected components y con algunas operaciones matemáticas simples. Como sabemos, el objetivo es detectar los jugadores y el balón, de forma que con este método seremos capaces de realizar tal función de forma separada, es decir, según las características mencionadas, podremos diferenciar si un elemento es un jugador, si es el balón, o si es cualquier otro elemento que no se haya conseguido eliminar en el resto del proceso, pero al no cumplir con las características sería eliminado. Este procedimiento se asemeja al proceso de aprendizaje máquina, en el que en primer lugar se hace una extracción de características y posteriormente se clasifica según estas características y los experimentos anteriores, ajustando las características para evitar falsos positivos y falsos negativos.

En caso de cumplir las características mencionadas, obtenemos los parámetros para el bounding box de las propias características de connected components, que, junto con la característica del centroide, podemos representar, con las funciones de dibujado de OpenCV, un punto y un rectángulo alrededor del jugador en cuestión. Vemos a continuación la imagen que devolvería esta función:

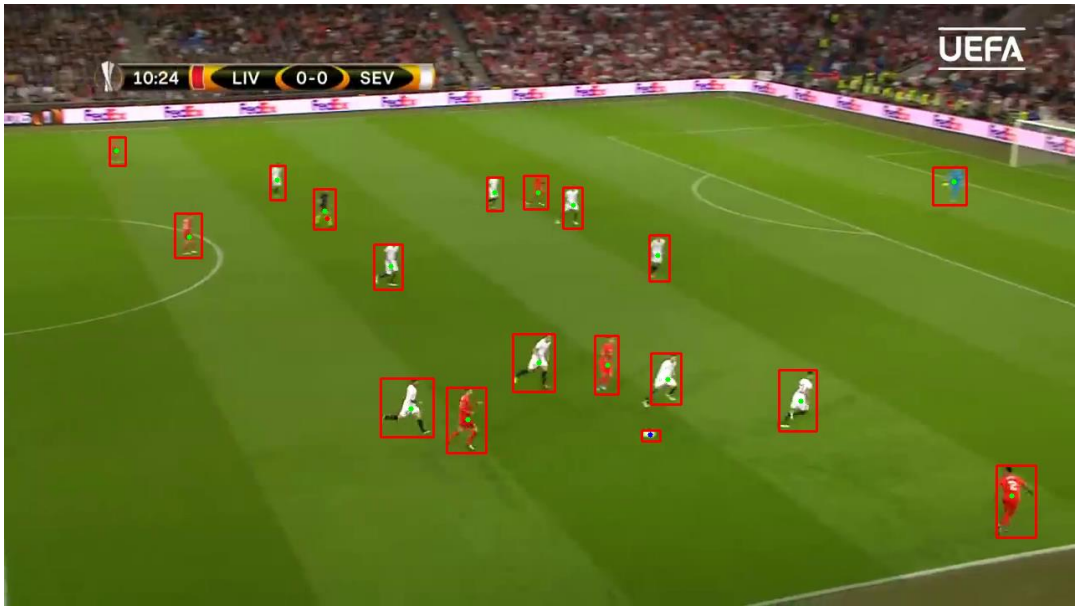


Ilustración 2.10 Detección de jugadores

Además, la función nos devolverá la información de la localización de los distintos jugadores, lo que nos será útil posteriormente para el seguimiento de los jugadores y el balón.

Como vemos, si ajustamos los distintos parámetros de cada procesado, podemos llegar a tener resultados bastante buenos: si atendemos a la cantidad de jugadores detectados, de todos los que aparecen en la imagen (verdaderos positivos), frente a la detección de otros elementos indeseados que no sean jugadores o pelota (falsos positivos). No obstante esta metodología tiene sus limitaciones, como veremos más adelante en el estudio de los resultados. Podemos adelantar que este algoritmo tendrá dificultades para detectar los elementos en determinadas ocasiones, como por ejemplo cuando exista cercanía u oclusión entre distintos elementos o haya mala calidad en la imagen. Por este motivo, planteamos el siguiente método de seguimiento, para que funcione en conjunto a este método y consiga solventar en la medida de lo posible estos problemas.

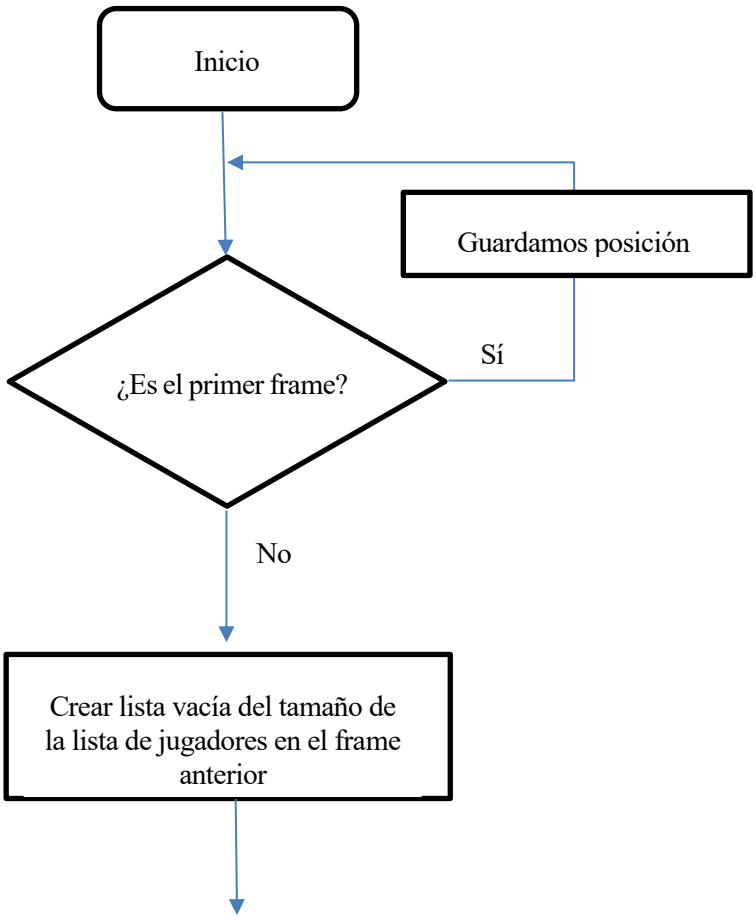
SEGUIMIENTO

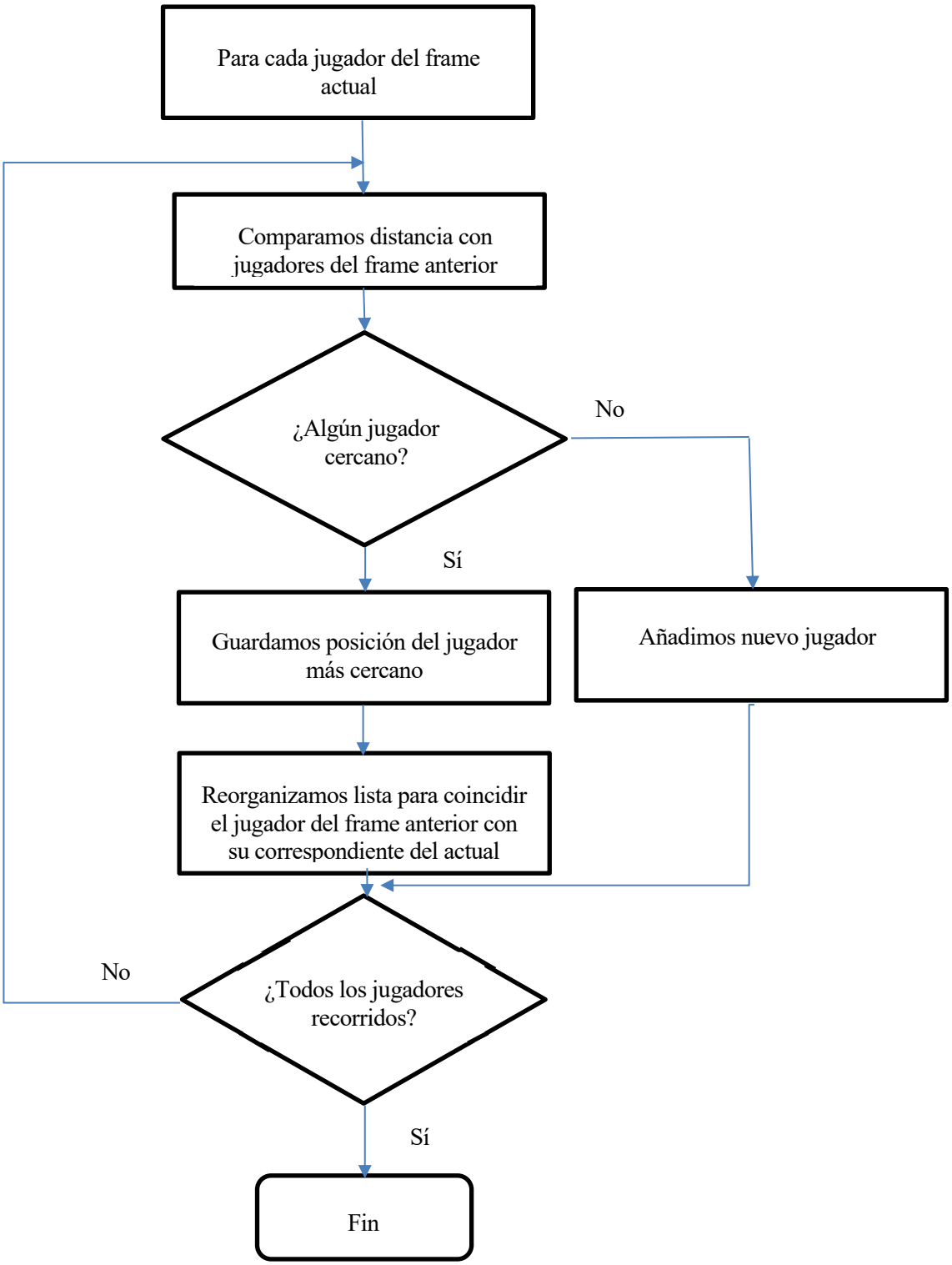
Este capítulo detallará como se ha abarcado el problema del seguimiento tanto de los atletas como del balón. Para ello se explicará en profundidad que algoritmo se ha desarrollado para la relación del jugador detectado en un frame con el mismo detectado en el siguiente frame, y además se explicará todo lo que se necesita saber de las técnicas predictivas implementadas.

3.1 Algoritmo de seguimiento

En primer lugar, será necesario desarrollar un algoritmo que nos permita asignar el elemento detectado en un frame con otro detectado en el frame anterior, de forma que determinemos que son el mismo y podamos realizarle un seguimiento a lo largo de todo el video. Además, esto será necesario para poder aplicar posteriormente el filtro de Kalman, ya que necesitaremos actualizar la posición de cada jugador en cada frame. En primer lugar, dado que sabemos que solo habrá un balón en el terreno de juego y que además gracias a la distinción según características realizadas en el apartado, podemos diferenciarlo del resto de elementos, no será necesario realizar ningún algoritmo para el balón, puesto que solo debería de haber detectado uno en toda la imagen.

En el caso de los jugadores sí será más complejo, puesto que tendremos una gran cantidad de ellos en la imagen, y además no será una cantidad fija, ya que por lo general estas grabaciones no abarcan todo el campo de juego, por lo que los jugadores pueden entrar o salir del plano. Para esta tarea, tomaremos como criterio principal la distancia. Dado que los elementos captados serán jugadores, por lo que sabemos que no alcanzarán una gran velocidad, junto que generalmente estas grabaciones se realizan a 24 FPS, se asume que la posición entre un jugador en el frame anterior y el actual será muy cercana. Para que sea más sencillo de entender, atenderemos en primer lugar al siguiente diagrama de flujos, y posteriormente detallaremos las distintas partes de este algoritmo.





Como vemos, este algoritmo se basará en una comparación entre los jugadores captados en este frame, con los captados en el frame anterior, ambos guardados en listas distintas. Para esta comparación, tendremos una distancia límite a partir de la cual consideraremos que es posible que dicho jugador se pueda corresponder con el anterior. De esta forma, si no hay ningún jugador cercano, consideraremos que es un nuevo jugador que no aparecía en la escena en frames anteriores, por lo que lo añadiremos a la lista.

Por otro lado, tendremos la posibilidad de que un jugador que anteriormente sí fue detectado, ahora no se detecte. Para este caso, como inicializamos una lista vacía del tamaño de los jugadores anteriores, si no se encuentra ningún jugador cercano para ese jugador, constará en dicha lista como *None.* Como el hecho de que no se haya detectado no implica necesariamente que no aparezca en la imagen, lo dejamos como *None.* De esta forma,

como ya explicaremos más en profundidad más adelante, podremos estimar, mediante la predicción del filtro de Kalman, donde se encontrarían los jugadores no detectados.

3.2 Filtro de Kalman

Para esta parte del seguimiento, nos basaremos en el artículo *Vision-based human tracking and activity recognition* [10], en el cual se propone estudiar el movimiento de una persona, apoyándose en algoritmos predictivos basados en el filtro de Kalman, con el objetivo de reconocer su actividad en función de su posición y velocidad. En este artículo, al igual que en el presente proyecto, habrá dos etapas, una en la que inicialmente se detecte el elemento deseado, y otra posterior en la que se realice el seguimiento. En este caso, como el estudio está orientado a la seguridad y vigilancia, propone realizar avisos si el peatón va muy rápido, si se acerca a una zona de seguridad o si merodea o está mucho tiempo en un mismo sitio. Para mi estudio concreto, se plantea la inclusión de un filtro de Kalman, basándonos en los estados de posición y velocidad, de cara a mejorar la eficiencia de nuestro algoritmo para reconocer tanto a los deportistas como al balón de juego.

3.2.1 Teoría del filtro de Kalman

El objetivo del filtro de Kalman será el de estimar ciertas variables de un sistema basándose en medidas con ruido, de forma que el algoritmo calculará las distintas probabilidades del estado del sistema. El filtro de Kalman consta principalmente de dos etapas: predicción y corrección.

- Predicción: Será la etapa en la que se estime el estado actual del sistema basándose en el estado anterior del sistema y las ecuaciones del modelo de dicho sistema.
- Corrección: En esta etapa se corregirá la estimación hecha en la etapa anterior haciendo uso de los datos de medición actuales.

Para mayor claridad del fundamento matemático de este algoritmo, lo explicaremos en un principio con un ejemplo sencillo con variables simples hasta llegar a las ecuaciones matriciales que forma este teorema.

Imaginémonos un móvil que se mueve en línea recta (una dimensión) a una velocidad concreta. Podemos conocer la localización de dicho móvil gracias a un dispositivo GPS que nos da la posición, pero como sabemos, los sensores tendrán una cierta incertidumbre, por lo que en este caso la posición será una variable aleatoria que seguirá una distribución normal, es decir tendrá una media y una varianza, y la probabilidad tendrá una distribución así:

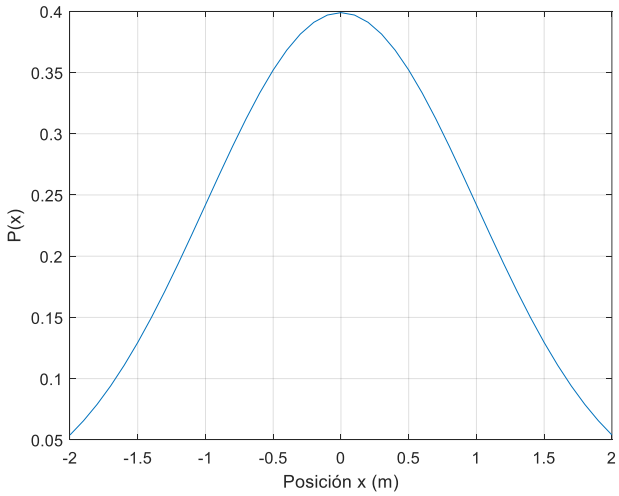


Ilustración 3.1 Medida de posición

Si sabemos que este móvil se mueve con una velocidad constante, pongamos 4 m/s, podremos estimar la posición de dicho móvil un incremento de tiempo después, según las siguientes ecuaciones:

$$p_k = p_{k-1} + V_{k-1}\Delta t$$

$$\mu_{p|k} = \mu_{p|k-1} + V_{k-1}\Delta t$$

$$\sigma^2_{p|k} = \sigma^2_{p|k-1}$$

Que representado gráficamente será así:

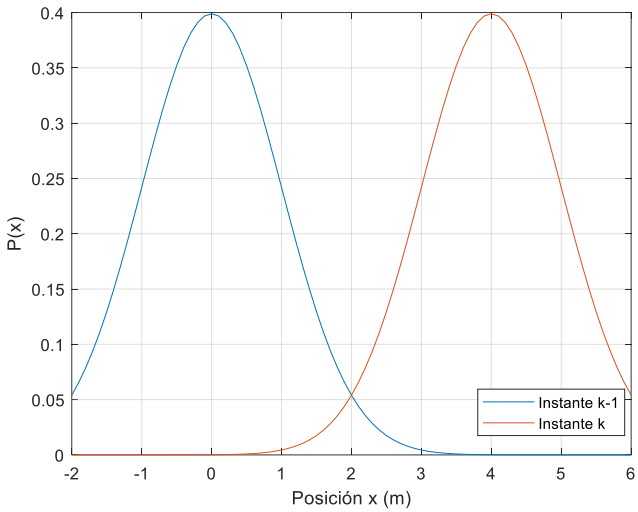


Ilustración 3.2 Posición en el siguiente instante

En este caso, tomamos la velocidad como una medida exacta en nuestro sistema, pero como sabemos la medida de velocidad también tendrá una incertidumbre, por lo que también será una variable aleatoria, quedando nuestras variables de la siguiente forma:

$$P \sim \mathrm{N}(\mu_p, \sigma_p^2)$$

$$V \sim \mathrm{N}(\mu_v, \sigma_v^2)$$

Sabiendo esto, y atendiendo a las propiedades de la esperanza y la media tenemos las siguientes ecuaciones:

$$E[P + V\Delta t] = E[P] + E[V]\Delta t$$

$$Var[P + V\Delta t] = Var[P] + \Delta t^2 Var[V]$$

De esta forma, podremos seguir realizando una predicción en los sucesivos instantes, pero debido a la velocidad aleatoria, aumentaremos la varianza en cada instante, por lo que si solo realizamos la predicción en cada instante obtendríamos lo siguiente:

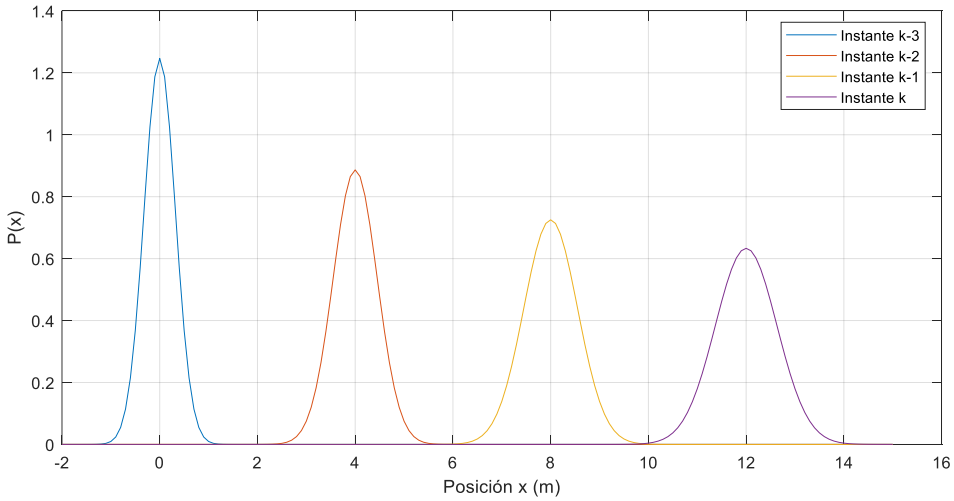


Ilustración 3.3 Predicción con velocidad aleatoria

Para la segunda etapa, la etapa de corrección, usaremos la nueva medida que nos ofrecen los sensores en ese instante. Esta medida también tendrá ruido, por lo que juntaremos el resultado de la predicción con el de la corrección, con el objetivo de calcular cual es la posición más probable en función de las dos etapas. Matemáticamente sabemos que la distribución conjunta de las dos variables aleatorias normales unidas será otra distribución normal. De esta forma para las siguientes variables aleatorias obtenidas en ambas etapas:

$$p \sim \mathrm{N}(\mu_p, \sigma_p^2)$$

$$z \sim \mathrm{N}(\mu_z, \sigma_z^2)$$

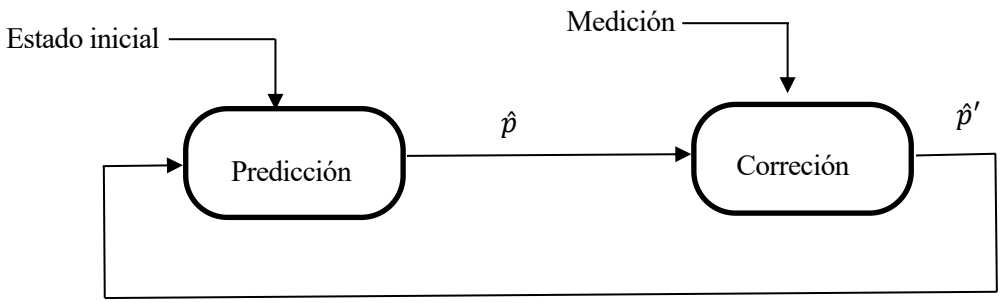
Obtenemos el siguiente resultado siendo p' la posición corregida:

$$k = \frac{\sigma_p^2}{\sigma_z^2 + \sigma_p^2}$$

$$\mu_p' = \mu_p - k(\mu_z - \mu_p)$$

$$\sigma_p' = \sigma_p - k\sigma_p$$

Si observamos el resultado que nos ofrece esta nueva varianza, podemos apreciar que reducimos la incertidumbre respecto a la fase de predicción, por lo que estamos mejorando nuestra medición. Por lo tanto, el diagrama general del algoritmo queda de la siguiente forma:



Ahora todo este proceso que hemos explicado con variable simples, procederemos a enunciarlo de forma análoga de forma matricial, que es como normalmente implementaremos el filtro de Kalman. Las ecuaciones para la predicción serán:

$$\hat{x}_k = x_{k-1} + A\hat{x}_k' + Bu + w_k$$

$$P_k = AP_{k-1}A^T + Q_k$$

Donde:

- X: Vector con las variables del sistema
- A: Matriz de predicción del siguiente estado
- B: Matriz de entradas externas al sistema
- U: Vector de entradas
- P_k: Matriz de covarianza de la variable de estado en el instante k
- Q_k: Ruido que añade el entorno al sistema, o como de fiable es nuestro modelo

Si seguimos con el ejemplo anterior, tendremos que nuestra matriz A será:

$$A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$

Y teniendo en cuenta que este sistema no tiene entrada, obtenemos el siguiente resultado, que es el mismo que el obtenido anteriormente:

$$\begin{bmatrix}\hat{p}_k\\ \hat{v}_k\end{bmatrix} = \begin{bmatrix}1 & \Delta t\\ 0 & 1\end{bmatrix}\begin{bmatrix}\hat{p}'_{k-1}\\ \hat{v}'_{k-1}\end{bmatrix}$$

$$p_k = p_{k-1} + V_{k-1}\Delta t$$

$$\hat{v}_k = \hat{v}'_{k-1}$$

De forma análoga, tendremos las siguientes ecuaciones matriciales para la fase de corrección:

$$\hat{x} = \hat{x}_k - K'(z_k - H\hat{x}_k)$$

$$P' = P - K'HP$$

$$K' = PH^T(HPH^T - R)^{-1}$$

Donde:

- z_k: Será la medida en ese instante
- R_k: Varianza del sensor de medida
- H: La matriz que asocial el estado del sistema con las medidas
- K': El resultado de la multiplicación de las variables normales de la predicción y la medición

Con estas cinco ecuaciones podremos calcular el filtro de Kalman en cada instante, tan solo necesitaremos definir correctamente las variables y realizar una estimación correcta de las varianzas de ruido de las variables. Para probar estas ecuaciones, y ver de una forma práctica el funcionamiento de este filtro, hemos desarrollado un pequeño programa en Matlab. En este programa simularemos las medidas de un vehículo que se desplaza en el plano x e y, donde provocaremos un cambio de dirección de la velocidad en el eje y para comprobar cómo se comportaría el filtro en esa situación. Las medidas serán tomadas por sensores con una determinada incertidumbre, por lo que para similar esto, generamos un ruido aleatorio en estas medidas. Para este caso práctico, el estado del sistema serán las variables de posición x e y, y las de velocidad en x e y, es decir tendrá un tamaño de 4. Por otro lado, tan solo se tomarán medidas de posición, por lo que nuestras medidas tendrán una dimensión de 2, es decir, las posiciones en los dos ejes. Teniendo esto en cuenta, declararemos las siguientes matrices, según lo visto en la teoría:

$$A = \begin{bmatrix}1 & 0 & T & 0\\ 0 & 1 & 0 & T\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$H = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\end{bmatrix}$$

$$R = \begin{bmatrix}r^2 & 0\\ 0 & r^2\end{bmatrix}$$

$$Q = \begin{bmatrix}q^2 & 0 & 0 & 0\\ 0 & q^2 & 0 & 0\\ 0 & 0 & q^2 & 0\\ 0 & 0 & 0 & q^4\end{bmatrix}$$

Donde *r* será el error aleatorio de la medida de la posición, y *q* será la confianza o el error de nuestro modelo. En este caso, *r* tendrá un valor de 5, ya que es el error introducido para la medida en la simulación, y *q*, tendrá un valor de 0.1, que en este caso ha sido ajustado experimentalmente según los resultados que se obtenían. Finalmente, T será el tiempo de muestro, para este ejemplo 1.

Por otro lado, simularemos también que durante unos instantes perderemos las medidas de nuestra posición (en un caso práctico podría ser perder la señal GPS al entrar en un túnel), de forma que probaremos el funcionamiento del filtro actualizando solo la fase de predicción. Los resultados obtenidos son los siguientes:

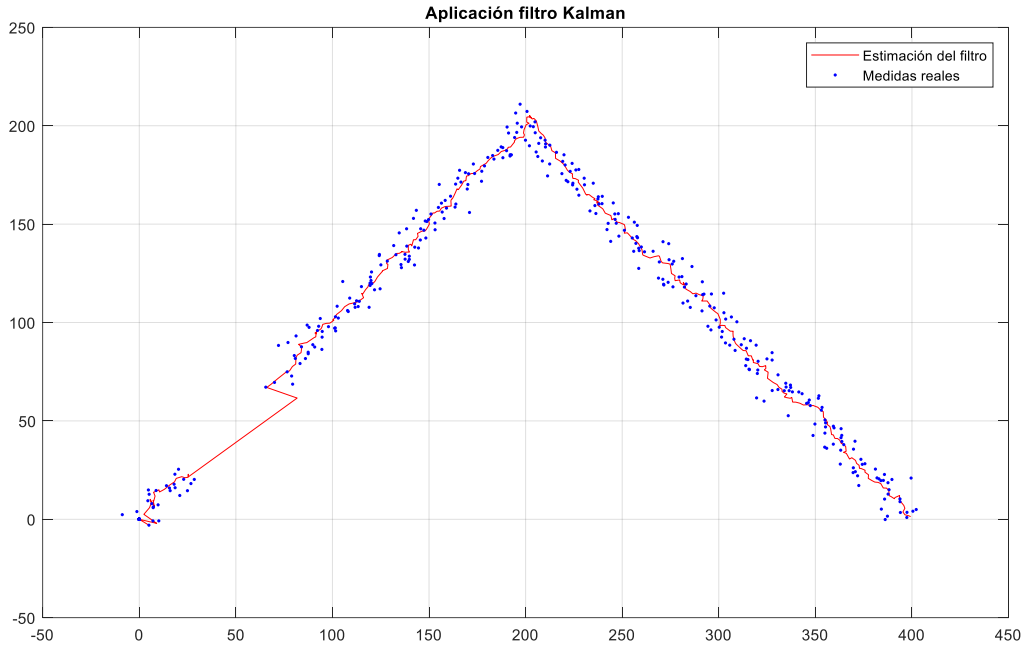


Ilustración 3.3 Implementación filtro de Kalman en Matlab

Como vemos, el filtro mejora las medidas tomadas por los sensores, ajustándose más a la posición real en caso de estar bien definidas las matrices que conforman el filtro. Si observamos los segundos del 25 al 75, es donde hemos simulado la pérdida de la medida de la posición. Como vemos, el filtro intentará predecir la posición según las medidas y predicciones anteriores, por lo que cabe esperar que aumentemos el error con la posición real, pero no demasiado según vemos en la práctica, y donde, además, conseguiremos recuperarnos rápidamente en cuanto llega la primera medida de posición.

3.2.2 Implementación del filtro con OpenCV

Al igual que sucedía ya con otras técnicas vistas anteriormente, el algoritmo del filtro de Kalman ya viene implementado en las funciones de OpenCV. En este caso, tendremos la clase de *KalmanFilter()*, de forma que podremos declarar un nuevo objeto por cada elemento a seguir. Al declarar un nuevo objeto de esta clase, deberemos indicar en los parámetros de entrada tanto la dimensión del estado como la dimensión de la medida. Posteriormente, la clase dispone de una serie de propiedades que deben ser declaradas por nosotros, ya que estos serán las variables a definir que hablábamos anteriormente en la teoría. La correspondencia entre estas será la siguiente:

- measurementMatrix: H
- measurementNoiseCov: R
- processNoiseCov: Q
- transistorMatrix: A

Estas matrices las declararemos igual que en el ejemplo práctico de Matlab realizado en el apartado anterior, es decir, tendremos como estados del sistema la posición y la velocidad en los ejes x e y, y como medidas la posición en los ejes x e y en cada frame. Lo que si variaremos respecto al ejemplo en Matlab serán las variables q y r. Si recordamos, la variable r era la que nos indicaba el ruido en la toma de medida de los sensores, dato que conocíamos exacto en nuestro ejemplo, pero que en este caso no podemos conocer con exactitud por lo que tendremos que estimarlo experimentalmente. Con respecto a la variable q, al igual que ya hicimos en el ejemplo, la estimaremos experimentalmente, dándole en esta ocasión menos confianza al modelo.

Para la actualización de cada una de las fases tendremos los métodos *predict()* y *correct()*, los cuales como su propio nombre indica serán los encargados de las fases de predicción y corrección. Para las actualizaciones, de

forma análoga a lo hecho en el ejemplo en Matlab, no realizaremos la fase de corrección en caso de no conseguir detectar al jugador por los métodos de detección ya explicados, de forma que estimaremos su posición tan solo con la predicción del filtro.

Más adelante, en el estudio de los resultados del proyecto, analizaremos el funcionamiento del filtro para la aplicación que nos ocupa, y mostraremos ejemplos y gráficas representativas que nos ayuden a dicha tarea.

PROGRAMA Y FUNCIONALIDADES

En este apartado, se explicará los distintos programas realizados para el proyecto, su estructura y sus distintas funcionalidades. El programa principal básicamente consistirá en una implementación en Python de las técnicas aplicadas anteriormente, y desarrollaremos un segundo programa que nos facilite la tarea de obtener los distintos videos para el testeo del programa principal. El objetivo que se ha perseguido en este sentido es el de poder realizar todas las pruebas y ajustes del programa desde la ventana de comandos.

4.1 Descargas de videos tests

El objetivo de este programa, como ya hemos introducido, será el de poder obtener nuestra base de testeo del programa principal de una forma sencilla desde la propia ventana de comandos. Como veremos más adelante, para este proyecto jugará un papel muy importante el testeo del programa con una gran cantidad de situaciones reales distintas. Esto es debido en primer lugar a que la visión artificial es muy dependiente de muchos factores externos, como puede ser la iluminación, que pueden hacer que nuestro programa funcione a la perfección en unas situaciones, y que en otras no sea capaz de detectar un solo jugador. Por otro lado, en el deporte que nos estamos centrando, el fútbol, existen también otra gran cantidad de factores variables que pueden alterar el funcionamiento de nuestra aplicación, puesto que el terreno de juego no tiene por qué ser exactamente igual, los distintos equipos llevarán equipaciones distintas, y, además, existen infinidad de situaciones de juego distintas que alterarán la detección y el seguimiento. Por este motivo, se ha desarrollado este programa, facilitando la tarea de obtención de videos con los que comprobar donde actúa bien el programa y donde no, pudiendo así mejorarlo según la experiencia.

Este programa se basará en la librería de código abierto *pytube* [12]. Esta librería dispone de una serie de funciones con las que podremos descargar videos de la plataforma de www.youtube.com de forma sencilla, ofreciendo además la posibilidad de descargarlo con la configuración de calidad que deseemos, siempre que el video subido a la plataforma lo permita. Además de esta librería, usaremos otra también usada en el programa principal, llamada *argparse* [13], con la que podremos programar distintos argumentos que puedan ser introducidos desde la ventana de comandos al realizar la llamada al programa. De esta forma podremos, sin abrir el programa ni modificarlo, indicar que video queremos descargar, con que nombre y donde queremos guardarlo, la resolución del video, y que parte concreta queremos descargar, ya que normalmente lo que buscaremos para este proyecto serán clips concretos, evitando así cambios de planos y poder estudiar el movimiento completo. Para esto último, se ha usado la librería de *moviepy* [14], que nos permite recortar los videos una vez ya descargados.

El programa se llamará yt_descarga.py, y podremos ver el código completo en el anexo del presente documento. Los argumentos del programa serán los siguientes:

- "-l" o "--link": link del video de yotube a descargar.
- "-n" o "--name": nombre del video destino, donde además podremos indicar la ruta concreta donde queremos que se guarde.
- "r" o "--resolution": resolución a la que queremos descargar el video.
- "-i" o "--inicio": segundo inicial del clip a guardar.
- "-f" o "--final": segundo final del clip a guardar.

De esta forma introduciendo en la ventana de comandos lo siguiente:

```
python yt_descarga.py -l https://www.youtube.com/watch?v=vR9ERQxq9BA
-n VideoEjemplo -r 720p -i 20 -f 30
```

Descargaremos el video del link introducido, con el nombre VideoEjemplo, a una resolución de 720p, del segundo 20 al 30. Cabe destacar que todos los argumentos serán necesarios introducirlos a excepción de la

resolución de descarga, que en caso de no indicarla se descargará a la resolución por defecto que será 720p. Por otro lado, aunque indiquemos la resolución concreta a la que queremos el video, muchas veces no estará disponible, por lo que al inicio del programa realizamos un filtrado de los distintos streams del video buscando dicha resolución, y en caso de no encontrarla, se descargará con el primer stream que se obtenga.

Posteriormente en el programa, una vez descargado el video, en formato mp4, se hace uso de la función *ffmpeg_extract_subclip(),* con la que cortaremos el video para los segundos que se han indicado. Esto nos creará otro archivo de video, también con formato mp4, que será el clip extraído del video completo. Finalmente, dado que tenemos los dos archivos, pero tan solo queremos el que contiene el fragmente deseado, eliminamos el video completo al finalizar el programa.

Con esto, ya podremos descargar todos los videos que deseemos para probar nuestro programa.

4.2 Programa principal

En este apartado explicaremos cual es la estructura del programa principal, de que modos dispondremos para las distintas funcionalidades y demás características importantes comprender sobre el programa.

El núcleo del programa estará formado por un bucle *While* que se ejecutará por cada frame del video que queramos analizar. Este archivo de video, lo leeremos como ya comentamos en la introducción con la función *FileVideoStream* de la librería *imutils,* ya que si hacemos uso de la función por defecto de OpenCV se obtienen peores resultados en cuanto a coste computacional al leer y analizar cada frame. En este bucle llamaremos a las distintas funciones, tanto de OpenCV como propias, en el orden correcto para el procesamiento de cada frame. Estas funciones principalmente serán las que realicen las distintas técnicas y algoritmos explicados en los apartados teóricos de detección y seguimiento. Estas funciones serán:

- condicionRGB: A esta función le pasaremos como entrada el frame a procesar, y nos devolverá la imagen binarizada con el fondo extraído y la máscara del terreno de juego. Esta función se basará en la función *cv.split()*, con la que podremos separar los canales r, g y b de nuestra imagen, para posteriormente compararlos según la condición G > R > B. Para la obtención de la máscara del terreno de juego usamos la función *cv.findcountours*(), con la que detectamos los contornos de las distintas figuras que aparecen en nuestra imagen binarizada, de forma que cogeremos el más grande de estos (asumiendo que se corresponderá con el terreno de juego que es el que ocupa mayor espacio en la imagen), y dibujaremos dicho contorno junto con su interior con la función *cv.drawContours(),* de forma que invirtiendo los colores con la función *cv.bitwise(),* obtendremos la máscara deseada.
- gradiente_sobel: A esta función le pasaremos como entrada el frame a procesar y un valor de umbral, y nos devolverá la imagen binarizada tras realizarle el gradiente de Sobel. En esa función, binarizaremos la imagen, calcularemos el gradiente de sobel para el eje x e y, las combinamos y discriminamos según el valor de umbral establecido, todo ello haciendo uso de las funciones de OpenCV ya explicadas en la teoría.
- TransformadaHough: Esta función calculará la transformada de hough de nuestra imagen a fin de detectar las líneas rectas del terreno de juego y eliminarlas. Esta devolverá la imagen con las líneas eliminadas, pasándole como entrada la imagen, y los valores de rho, theta, umbral, longitud mínima y hueco máximo para los parámetros de la transformada de Hough, ya explicados en apartados anteriores. En esta función simplemente ejecutaremos la función *cv.HoughLinesP(),* con sus parámetros, y la función propia *line_disc().*
- line_disc: Esta función nos dibujará una línea discontinua según indiquemos. Le pasaremos como entrada la imagen sobre la que queremos dibujar, el punto de inicio y fin de la recta, el color de la línea y el grosor, y nos devolverá la imagen con la recta discontinua dibujada. En esta función, calcularemos el paso que queremos para dividir nuestra recta continua en 16, valor tomado por defecto según resultados experimentales. Una vez hecho esto, dibujaremos cada línea discontinua según la división de forma intercalada, obteniendo así la recta discontinua sobre la imagen.
- etiquetado: Función con la que etiquetaremos cada uno de los elementos sobre la imagen procesada, detectando las distintas siluetas y eliminando según características. A esta función le pasaremos como

entrada la imagen procesada, y nos devolverá las posiciones x e y del balón y jugadores detectados, y imagen con los bounding box (rectángulos) sobre los elementos detectados (jugadores y balón en un principio). Para esto, aplicaremos la ya mencionada función *cv.connectedComponentsWithStats(),* que nos ofrecerá las distintas etiquetas de aquellos elementos con todos los puntos que la contienen conectados, juntos con una serie de características de cada uno de ellos. Posteriormente, recorremos todas las etiquetas detectadas, y determinaremos según las características ofrecidas por la función anterior si es un jugador, si es el balón, o en su defecto, si no es ninguno de los elementos que queremos detectar. Esto lo determinaremos con el área, la relación alto/ancho, y la densidad, que será área/(alto*ancho). Si es uno de los elementos deseados, dibujamos el bounding box con la función *cv.rectangle(),* y las cordenadas que obtenemos de las características del elemento. Dibujamos también un punto en el centroide con la función *cv.circle().* Guardaremos la posición en el caso del balón (solo hay uno), y las acumulamos en una lista en el caso de los jugadores.

- relacion_jugadores: A esta función pasaremos como entrada la posición de los jugadores detectados en el frame anterior con la de los detectados en el frame actual, devolviendo dicha lista ordenada según el número de jugador al que responda cada uno. La función realizará la tarea de relacionar un jugador detectado en un frame con el detectado en el frame siguiente. Esta función ayuda al seguimiento y a la actualización del filtro de kalman. En esta función básicamente aplicaremos el algoritmo explicado en el punto 3.1. Además, hace uso de otra función propia muy sencilla que calcula la distancia entre dos puntos, la cual también se hace uso varias veces durante el resto del programa.

Además del algoritmo de procesamiento de cada frame, tendremos en nuestro bucle *While* una estructura tipo *try except,* de forma que leeremos cada frame hasta que acabe el video, saltando así a la rutina de excepción donde realizaremos las distintas estadísticas y estudios de los datos recopilados del video.

Se ha programado también distintos modos de funcionamiento para el programa, que podremos determinarlos mediante los comandos que introducimos al iniciar el programa desde la ventana de comandos. Estos modos serán los siguientes:

- Modo 1: será el modo de depuración, en el que se ejecutará el video frame a frame, parando en cada uno de forma que se pueda analizar con facilidad como está actuando secuencialmente el programa.
- Modo 2: Será el modo continuo normal. Con este modo procesaremos y visualizaremos todos los frames de seguido, de forma que lo veremos al tiempo real al que se procese, que no será al tiempo real del video puesto que podrá ir más rápido o más lento según el coste computacional que suponga el procesamiento de cada frame.
- Modo 3: Será el modo de estadísticas, donde sacaremos todas las gráficas y estadísticas resultado de la ejecución del programa, tanto estadísticas del propio juego derivada del movimiento de los jugadores, como estadísticas que evalúen el funcionamiento del propio programa.

Con todo esto, la llamada al programa será:

```
  python main_v1.0.py -v "ruta del video a analizar" -m "modo de programa"
-o "visualización de salida" -j "Numero de jugador para estadísticas"
```

Correspondiéndose la enumeración del modo programa con la expuesta anteriormente, la ruta del video con la ruta entera de donde se encuentre el archivo o la ruta desde donde se encuentra el programa, la visualización de salida con el procesamiento que queremos visualizar, siendo 1 la imagen original con los elementos detectados marcados con bounding boxes, 2 la imagen binarizada con el proceso de discriminación por color, 3 la imagen binarizada con el gradiente de Sobel y 3 la imagen binarizada tras todo el proceso. Finalmente el número del jugador se corresponderá con uno de los jugadores detectados y enumerados por el programa de detección.

ANÁLISIS Y RESULTADOS

En el actual apartado expondremos los resultados obtenidos con el algoritmo y programa explicado en los apartados anteriores, exponiendo los casos en los que actúa con buenas prestaciones y aquellos en los que falla, realizando un análisis exhaustivo de lo obtenido, y mostrando además algunos ejemplos de aplicaciones reales para mostrar el alcance de las técnicas aplicadas. Todo esto lo haremos mostrando en primer lugar los resultados para diferentes clips de partidos de fútbol, de forma que tengamos una gran cantidad de situaciones diferentes, incluyendo situaciones donde se prevé un mal resultado, para posteriormente analizar en casos concretos como se comporta el programa, y mostrar gráficos y datos que sean representativos del funcionamiento del programa. Finalmente mostraremos las estadísticas que obtenemos con el programa, a modo de mostrar aplicaciones reales que puede tener este programa.

5.1 Resultados obtenidos

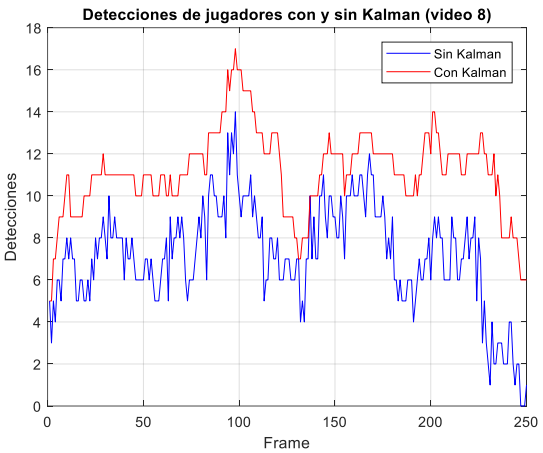
Como hemos dicho, hemos puesto a prueba nuestro algoritmo con diferentes videos de retransmisiones deportivas, con condiciones diferentes, de cara a evaluar con exactitud dónde funciona correctamente y dónde no. Para ver los distintos videos ya procesados de forma directa pueden dirigirse al siguiente link:

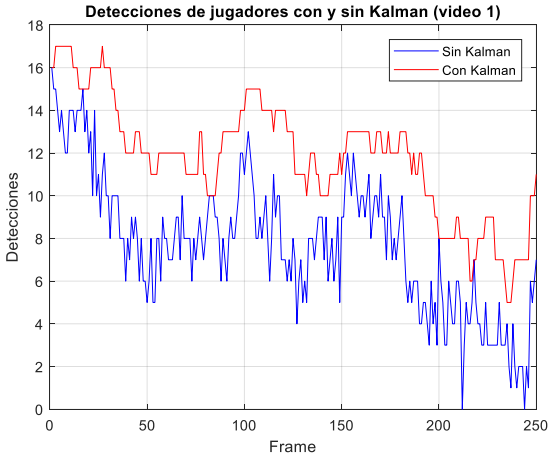
https://github.com/jsalinas98/tfg_resultados/tree/main/Videos

Si observamos los videos procesados en un primer vistazo, podemos apreciar como el algoritmo funciona con buenas presentaciones en la gran mayoría de circunstancias, consiguiendo detectar a la gran mayoría de jugadores durante la mayor parte del tiempo que dura la jugada, y sin tener excesivos falsos positivos. Para realizar un análisis más profundo, expondremos a continuación como actúan determinados elementos del algoritmo, así como qué resultados se obtienen en determinados momentos complicados.

5.1.1 Aportación del filtro de Kalman

Todo esto es principalmente gracias a la ayuda del filtro de Kalman junto con las técnicas de detección. Para poder apreciar esto, se ha dibujado con diferentes colores los bounding boxes que señalan el jugador detectado, según si se ha determinado por las técnicas de detección o si se ha determinado por la predicción del filtro de Kalman. Para poder evaluar esto de forma cuantitativa, vemos a continuación un gráfico de las detecciones con y sin filtro de Kalman para algunos de los videos procesados:





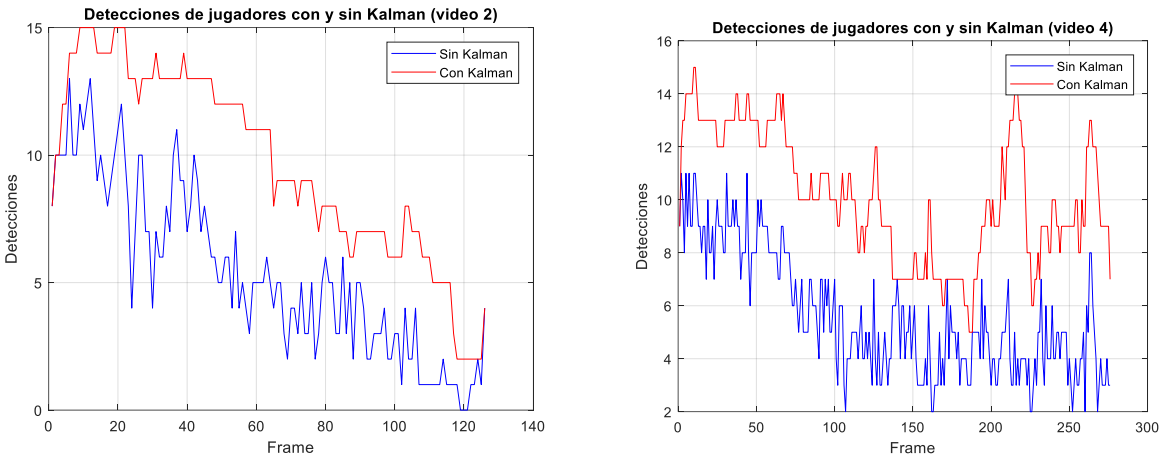


Ilustración 5.1 Detecciones de jugadores por frame

Vemos en los gráficos marcados con una linea azul los jugadores que detectaríamos si no aplicásemos el filtro de Kalman, y en rojo los que finalmente detectamos cuando aplicamos el filtro. Si nos fijamos en alguno de los videos, como el 2, parece encontrar más dificultades para encontrar los jugadores de forma continua por medio del algoritmo de detección, pero por medio de Kalman conseguimos mejorar esta detección hasta en más de un 50%. En otros videos vemos como también lo mejora, actuando en este caso en caso más puntuales, para los cuales se planteó el uso del filtro. Pero el número de detecciones no es suficientemente representativo para evaluar el funcionamiento de las predicciones del filtro, ya que podrían ser en su gran mayoría falsos positivos. Para evaluar esto, hemos cuantificado los falsos positivos, tanto solo del algoritmo de detección como de las predicciones de Kalman. El conteo ha sido realizado para el video 2, ya que según las gráficas anteriores creemos que puede ser el más crítico, y además no es posible obtener los datos de falsos positivos de forma autónoma, por lo que llevaría más tiempo realizarlo para cada video, pero asumimos un comportamiento parecido en el resto.

Ilustración 5.2 Falsos positivos en video 2

Como vemos en el gráfico, obtenemos un total de 86 falsos positivos, de los cuales 4 provienen del algoritmo de detección, y 82 provienen de las predicciones del filtro de Kalman. A pesar de que el número parece grande, tenemos que relativizar con la cantidad de frames del video en cuestión, ya que estos datos corresponden con los falsos positivos por frame. En este sentido, sabemos que tenemos un video de 130 frames, con una media de unos 8 jugadores detectados como verdaderos positivos. Desde esta perspectiva, no son demasiado malos los resultados, ya que la media de falsos positivos no llega ni a 1 por frame, teniendo en cuenta además la gran cantidad de jugadores que aparecen en pantalla.

A todo esto, habría que puntualizar una diferencia entre los falsos positivos según su origen. Por un lado, tenemos los falsos positivos del algoritmo de detección, los cuales tienen un claro origen, que es debido a que un elemento indeseado ha conseguido engañar al algoritmo haciéndolo coincidir con unas características similares a las de nuestros elementos deseados. Un ejemplo sería el siguiente:

Ilustración 5.3 Falso positivo del algoritmo de detección

Como vemos, para este ejemplo concreto, no es un tipo de falso positivo que suceda en muchas ocasiones, pero si es cierto que este tipo es el más crítico. Como hemos visto en la explicación del algoritmo de predicción, este realizará una predicción de donde se encuentra un elemento en base a una detección anterior del algoritmo de detección que posteriormente se ha dejado de detectar, realizándolo en los siguientes 10 frames si no se vuelve a detectar, después de los cuales se da por perdido el elemento y se deja de realizar predicciones. Atendiendo a este funcionamiento, podemos intuir que si obtenemos un falso positivo en la detección, tendrán como consecuencia 10 posteriores falsos positivos en las predicciones del filtro de Kalman. Por este motivo, decimos que este tipo de falso positivo es el más crítico, puesto que acarrea otra gran cantidad de falsos positivos en las predicciones, siendo este uno de los orígenes de falsos positivos en las predicciones.

Por otro lado, en las predicciones existe otro tipo de falso positivo, que serán aquellos de jugadores que previamente han sido detectados correctamente, pero que posteriormente se han dejado de detectar. Estos son los casos en los que entra en acción la predicción de Kalman, de forma que si esta predicción no es buena, genera otra sucesión de falsos positivos, como vemos a continuación:

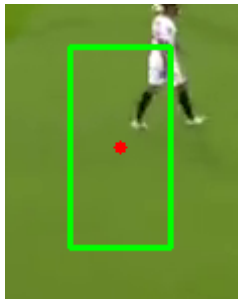


Ilustración 5.4 Falso positivo por predicción del Filtro de Kalman

Estos falsos positivos debidos a malas predicciones, son principalmente debidos a cambios de estados en los elementos difíciles de predecir con el filtro de Kalman. El más claro ejemplo de esto sería un cambio de ritmo en un jugador, es decir, tenemos un jugador corriendo a una cierta velocidad, el cual estamos detectando por medio del algoritmo, y cuando dejamos de detectarlo, este jugador frena en seco. En este caso, el filtro no es capaz de saber que el jugador ha frenado, puesto que no puede actualizar su estado con el estado real, lo que provoca que en la predicción indique como si el jugador hubiese seguido corriendo a la velocidad anterior. Un ejemplo de esto sería la siguiente secuencia:

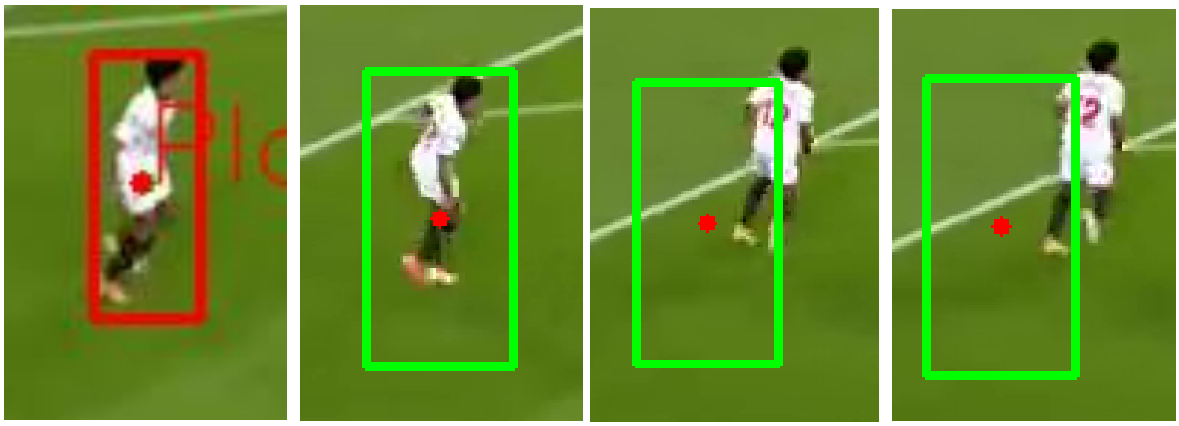


Ilustración 5.5 Secuencia predición de Kalman

Además, sabemos que las predicciones son menos fiables cuanto más tiempo llevamos sin actualizar con las posiciones reales, por lo que también acarreamos falsos positivos debidos a este motivo, y por el cual se estimó en el algoritmo que se desecharan a partir de 10 frames sin obtener medidas.

5.1.2 Oclusión momentánea

Otro de los problemas que nos encontramos en la visión en relación con el seguimiento de personas u objetos, es que durante la secuencia pueda ser parcial o totalmente ocluidos por otros elementos. Evidentemente, podemos prever que el algoritmo puramente de visión encargado de detectar a dicha persona u objeto no será capaz de detectarlo correctamente, puesto que ni si quiera lo ve. De hecho, el ser humano si tan solo viese ese frame, y estando completamente ocluido tampoco sería capaz de terminar que ahí hay una persona, pero si ve la secuencia entera, gracias a la información previa, si es capaz de saber que ahí hay una persona a pesar de no verla necesariamente. En este caso, al igual que en muchas otras aplicaciones tecnológicas, tomamos como ejemplo la forma en el que el ser humano o la naturaleza es capaz de hacer o saber ciertas cosas. Si nos fijamos, el filtro de Kalman emularía exactamente esto, la visión es capaz de detectar al jugador durante unos instantes determinados de tiempo, y posteriormente es ocluido, momento en el cual el filtro de Kalman, gracias a la información recopilada de los instantes anteriores es capaz de predecir donde está ese jugador.

En el entorno en el que nos encontramos, un partido de futbol, esta circunstancia que comentamos se dará con mucha frecuencia, ya que nos encontramos en un entorno con 22 jugadores, más los árbitros, en un espacio limitado. Los principales casos serán cuando un jugador tape a otro, o en el caso del balón, que lo tape al estar en contacto con el jugador, lo cual sucederá con mucha frecuencia. La siguiente secuencia es un claro ejemplo que se repite con frecuencia:

Ilustración 5.6 Secuencia de oclusión del balón

Aquí vemos como el jugador recibe un pase y controla el balón, momento en el cual el balón queda entre sus pies resultando parcialmente tapado para la cámara. En este caso, la visión no será capaz de distinguir el balón, porque además de estar parcialmente tapado, se encuentra junto con otro de los elementos a detectar, el jugador, de forma que resulta prácticamente imposible que el algoritmo actual sea capaz de diferenciar el balón de la silueta del jugador. Este sería el instante en el que Kalman realizaría su predicción. Vemos a continuación el resultado que nos ofrece nuestro programa para esta secuencia:

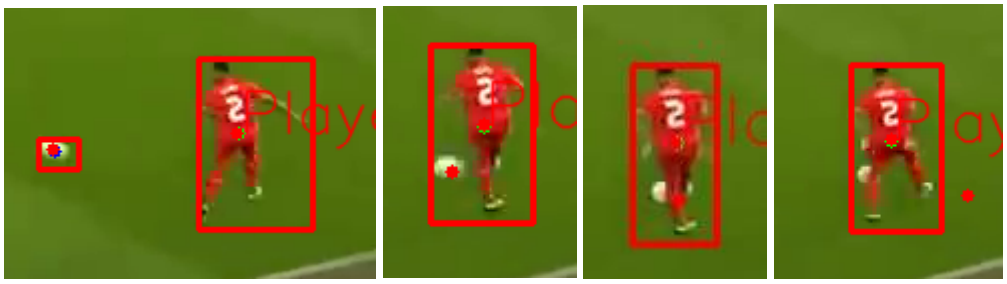


Ilustración 5.7 Secuencia predicción de Kalman ante oclusión del balón

En la secuencia vemos en un primer instante como el balón es detectado por nuestro algoritmo de detección, ya que lo vemos rodeado por el bounding box. En el siguiente instante, vemos como el algoritmo ya no es capaz de detectar el balón puesto que no aparece el bounding box dibujado, pero si vemos el punto que marcaría el centro del objeto. Este punto corresponde con la predicción que realiza el filtro de Kalman. Vemos como el punto realiza una predicción bastante fiable en la segunda secuencia, y aceptable en tercera. Pero si avanzamos

unos pocos instantes más, vemos la cuarta imagen de nuestra secuencia, en la cual la predicción ya tiene un error considerable. Esto se debe principalmente a la misma causa que exponíamos en el apartado 5.1.1 con respecto a cómo actúa el filtro de Kalman con los cambios de ritmo. Este caso es exactamente el mismo, tenemos el balón moviéndose a gran velocidad hasta que llega a los pies del jugador que lo controla, coincidiendo además con el momento en el cual perdemos la información de la detección. En este caso el filtro realiza sus predicciones como si el balón no se hubiese frenado, ocasionando una gran imprecisión cuando el jugador controla la pelota.

Como sabemos, la oclusión también puede ocurrir entre dos jugadores distintos, donde uno quede por encima de otro. Además, debido al funcionamiento del algoritmo, en este caso no es necesario ni si quiera que estén ocluidos en gran parte, sino con que tan solo estén en contacto ya provoca que ambas siluetas se identifiquen como solo una, alterando las características que tenemos en cuenta para determinar si es un jugador o no. Un ejemplo sería el de la siguiente imagen binarizada:

Ilustración 5.8 Imagen binarizada de juagadores en contacto

Los dos jugadores que vemos en la imagen se detectarán como un único elemento, de forma que el algoritmo detectará o un solo jugador, en caso de coincidir con las caracterísicas de un jugador, o ninguno en caso de no hacerlo. Vamos a continuación que resultado nos ofrece el programa tras aplicar el filtro de Kalman:

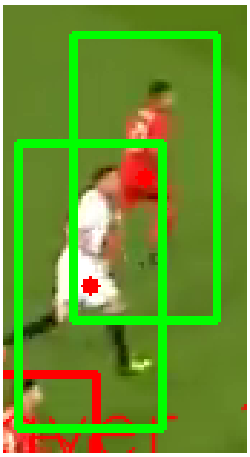


Ilustración 5.9 Resultado del filtro de Kalman en oclusión de jugadores

Vemos como en este caso el filtro realiza una predicción para ambos jugadores, puesto que lo vemos con el bounding box verde indicador de que es una predicción de Kalman. Además, el resultado que ofrece la predicción es bastante buena para ambos jugadores, por lo que podemos comprobar cómo de útil es la aplicación de este filtro para suplir este tipo de problemas en la visión.

Finalmente, expondremos el caso más crítico que puede suceder a este respecto. Este será el caso en el que tengamos una gran cantidad de jugadores juntos, y además desde el inicio de la secuencia, de forma que tenemos poca información previa de la posición y velocidad de cada jugador. Eso será recurrente en jugados del tipo falta o saque de esquina en los cuales los jugadores suelen partir todos de un espacio reducido. En la siguiente imagen, vemos una jugada de saque de esquina, donde tenemos hasta siete jugadores todos juntos, y donde el algoritmo de detección solo es capaz de detectar dos de ellos:

Ilustración 5.10 Detección ante 7 jugadores juntos

Como vemos es una situación complicada, y como tal tan solo se consigue detectar un porcentaje pequeño de todos los jugadores que aparecen. Pero este es el primer frame de la secuencia, por lo que todo el proceso y la información que dispone el programa se limita a este primer frame. Ahora mostramos dos resultados obtenidos en dos instantes posteriores a esta misma jugada:

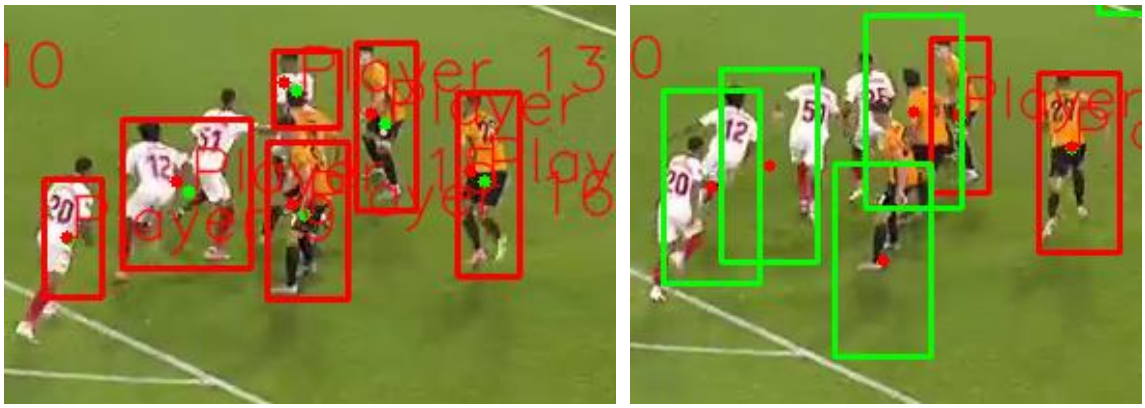


Ilustración 5.11 Detección de jugadores juntos en instantes posteriores

Vemos en la primera imagen, como en ese instante el algoritmo si ha sido capaz de detectar un mayor número de jugadores que en un inicio. En este caso en la imagen aparecen un total de ocho jugadores y se han detectado 6, dónde ha habido un par de fallos debido a un jugador en oclusión y otros dos que por cercanía los ha detectado como uno solo, pero teniendo en cuenta lo complicado de la situación podemos decir que es un buen resultado. Ahora pasamos a un instante poco posterior de la misma jugada. Observamos, en los rectángulos rojos, como otra vez el algoritmo de detección solo ha sido capaz de detectar dos jugadores, pero, sin embargo, gracias a la aplicación del filtro de Kalman, y apoyándonos en la información del instante anterior, conseguimos predecir con bastante acierto en que lugar se encuentran los otros cuatro jugadores detectados anteriormente.

Hemos podido comprobar como para este tipo de problemática, en el cual los jugadores están tapados o muy juntos, el filtro de Kalman, a pesar de tener algunos fallos reconocidos, ayuda bastante a mejorar los resultados en este tipo de circunstancias.

5.1.3 Iluminación no homogénea

Otro de los problemas que nos hemos encontrado en el funcionamiento del programa es debido a la iluminación. Es bien sabido que en el campo de la visión por computador, uno de los mayores problemas, que a su vez se convierte en uno de los principales puntos a perfeccionar para mejorar el funcionamiento de la aplicación, es la iluminación. La iluminación por un lado puede crear sombra o efectos indeseados, y nos otro modifica las tonalidades con las que percibimos los distintos colores del espacio grabado. Normalmente, cuando preparas un entorno para un sistema de visión, como se puede hacer por ejemplo en el mundo de la industria para la automatización de procesos industriales, la luz se convierte en el principal elemento para adecuar el entorno, implementando normalmente una iluminación difusa que ilumine la escena uniformemente.

Concretamente, para la aplicación de nuestro programa que es la detección y seguimiento en partidos de fútbol,

usando las imágenes de las retransmisiones por televisión, no tenemos la opción de adecuar el entorno y la iluminación según nuestras necesidades para tener mejores prestaciones, puesto la imagen se ha capturado con otro fin distinto a la visión por computador. Sin embargo, nos encontramos con un entorno bastante amigable, ya que los estadios para competir en competiciones profesionales necesitan por normativa unos determinados lúmenes, producido por una serie de focos distribuidos uniformemente por todo el estadio, lo que nos produce una iluminación bastante uniforme. Por lo general, en los resultados que hemos obtenido no ha habido ningún problema a este respecto en partidos que se disputan por la noche, ya que prácticamente la única iluminación es la de los focos del estadio. Cuando si se han tenido más dificultades ha sido en partidos disputados por el día, ya que la iluminación sola nos provoca una serie de sombras que dificultan la tarea de la visión. Vemos un ejemplo en las siguientes imágenes:

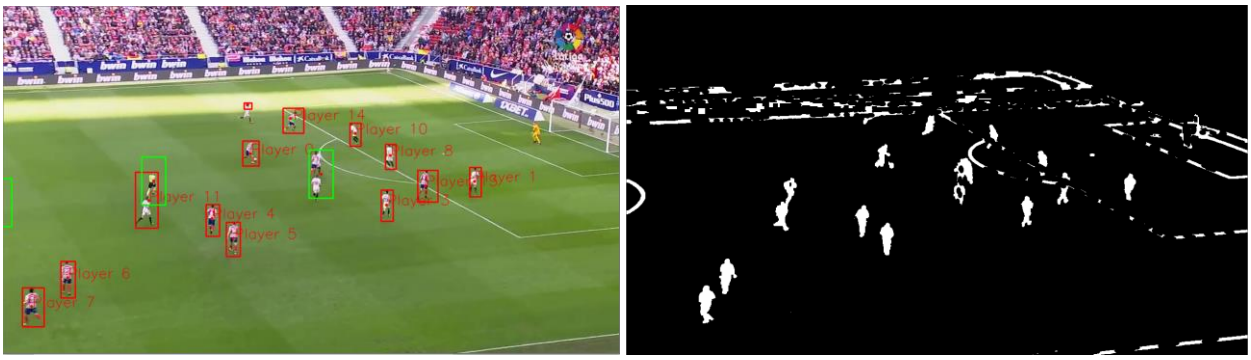


Ilustración 5.12 Ejemplo de iluminación no homogénea

Vemos dos imágenes correspondientes al procesamiento del video 2, a la izquierda la imagen original con los jugadores detectados y a la derecha la imagen binarizada de la que extraemos la información. Como podemos observar, estas imágenes proceden de un partido disputado de día, provocando sombras y una franja del campo donde incide el sol directamente. Si nos fijamos en la imagen de la derecha, esta diferencia en la iluminación nos provoca ruido e imprecisiones en la detección. Principalmente esto es debido al detector de bordes de Sobel, puesto que este cambio de iluminación es tomado por el gradiente como un borde. En este caso concreto no supone un gran problema ya que la mayoría de jugadores excepto uno se encuentran fuera de esta zona, pero para este jugador en concreto que se encuentra en dicha zona, el programa tiene grandes dificultades para llegar a detectarlo, por lo que esta iluminación no uniforme hace menos eficiente nuestro algoritmo.

5.1.4 Líneas del terreno de juego

Como ya se introdujo en la explicación de la transformada de Hough, la eliminación de las líneas de campo supone un problema. Por un lado, sabemos que es necesario eliminarlas puesto que, si no las eliminas y un jugador esta sobre ella el algoritmo no sería capaz de detectar a este jugador, pero por otro lado su eliminación conlleva otro tipo de problemas, ya que, si el jugador se encuentra sobre ella, eliminaremos parte el jugador junto con la línea, lo que ocasiona que el jugador quede partido en dos, de forma que se detectarán o dos jugadores distintos o ninguno.

Además del mencionado problema, la eliminación de las líneas por medio de la transformada de Hough conlleva otro problema visto en los resultados, y es que en ocasiones puede suceder que varios jugadores se alineen, de forma que la transformada determine la existencia de una línea recta compuesta por dichos jugadores.

Ilustración 5.13 Problemática de transformada de Hough ante jugadores cercanos

Para la solución de ambos problemas, se planteó borrar las líneas con la transformada de Hough, pero de forma discontinua, es decir en vez de dibujar la línea entera de forma que pueda separar a los jugadores, se dibujará "a trozos", de forma que lo más probable es que no acabe "partiendo" a los jugadores, sino que quedará siempre conectado por algún punto. Esto sí soluciona el mayor problema que tenían las líneas del campo, que era el conectar a todos los elementos sobre ella y marcarlos como uno solo, ya que identificamos las siluetas por medio de la función *connectedComponentes().* Teniendo las líneas discontinuas, si conseguimos separar los elementos sin interferir demasiado en el resto del funcionamiento. Vemos por ejemplo como actuaría en el caso anterior, ya en la imagen binarizada:

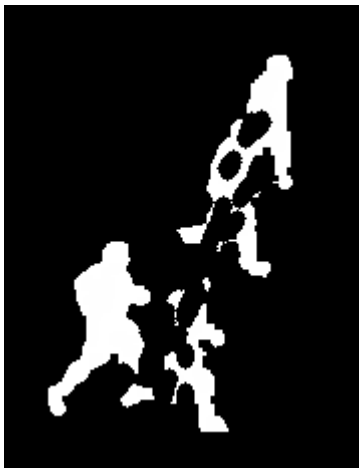


Ilustración 5.14 Tranformada de Hough sobre jugadores con línea discontinua

Como vemos si podremos seguir teniendo problemas aún con la línea discontinua, pero bastante menos que con la línea continua, de forma que al ocurrir con menos frecuencia si puede ser más fácilmente suplible con el filtro de Kalman, realizando predicciones en esos breves instantes que no detectemos los jugadores.

Vemos a continuación un ejemplo donde encontramos varios jugadores sobre una línea que conecta a todos ellos, y como el algoritmo es capaz de detectarlos a pesar de ello. Además, si observamos detenidamente el binarizado de dicha imagen podemos comprobar como efectivamente el hecho de que estas líneas sean discontinuas, ayuda a su detección.

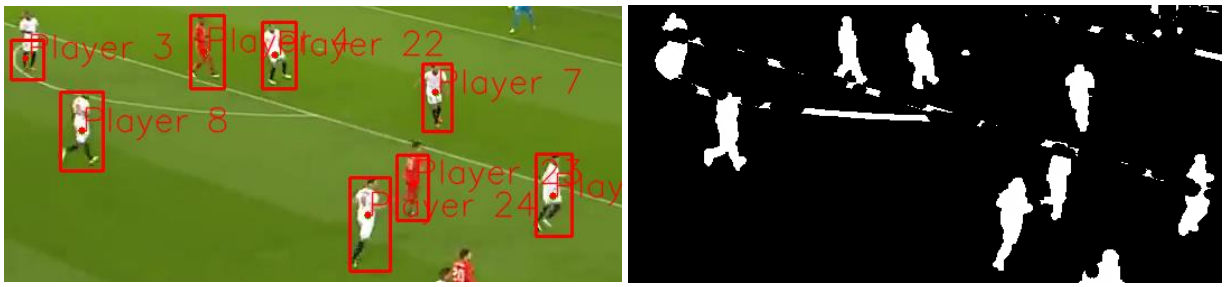


Ilustración 5.15 Varios jugadores sobre línea del terreno de juego

5.1.5 Correspondencia de jugadores

Otro de los problemas que nos encontramos en nuestra aplicación es la de la correspondencia de los jugadores de un instante a otro. Esta función es primordial para realizar un buen seguimiento, por que, a pesar de realizar una buena detección de cada jugador en cada frame, necesitamos complementar esta información con la asignación a un jugador ya detectado anteriormente, ya que, entre otras cosas, es necesario para la actualización del estado en el filtro de Kalman, herramienta fundamental en nuestro programa.

Como bien hemos explicado, la correspondencia se ha realizado según un criterio de cercanía, y se deja de realizar a partir de ciertos frames sin localizar al jugador. Este procedimiento tiene buenos resultados llevado a la práctica cuando el jugador es reconocido en cada frame directamente por el algoritmo de detección, que es el que ofrece posiciones más precisas. El mayor problema que nos encontramos es cuando se dejan de detectar dichos jugadores, y empieza a actuar el filtro de Kalman. Por lo general el filtro suele dar buenas predicciones, pero con cierta se dan dos casos que perjudican a los resultados, por un lado, tenemos cuando el jugador no es detectado durante varios frames seguidos, como sabemos en ese caso se da por perdido el jugador, de forma que, si posteriormente se vuelve a detectar, será tomado por el programa como un jugador nuevo. Al perder el jugador anterior, perdemos su seguimiento, y aunque lo hayamos vuelto a encontrar posteriormente, no tenemos forma a nivel de programa de relacionarlo con el seguimiento que ya teníamos del anterior, por lo que a nivel de estadísticas y demás es tomado como un jugador totalmente nuevo. Además, esta opción no puede ser eliminada por dos motivos. El primero es que a partir de un cierto número de predicciones seguidas sin actualizar la posición real del jugador el filtro de Kalman aumenta demasiado su incertidumbre, tanto que no podemos tomarlo como un dato fiable. Por otro lado, al ser retransmisiones deportivas es posible, y muy frecuente, además, que jugadores entren y salgan de plano, por lo que también se necesita un método en el que determinemos que un jugador ya no se encuentra en nuestro campo de visión, lo cual lo determinamos en nuestro programa cuando ya no somos capaces de detectar a dicho jugador.

Además, la segunda fuente de error más importante en este aspecto es cuando tenemos cambios de estados muy bruscos justo en el momento en el que dejamos de detectar el jugador, empleando solo la medición de la predicción del filtro de Kalman. En este caso, las medidas que se obtendrán estarán muy alejadas de la posición real del jugador, de forma que cuando volvamos a detectarlo ya no se relacionará con las medidas anteriores.

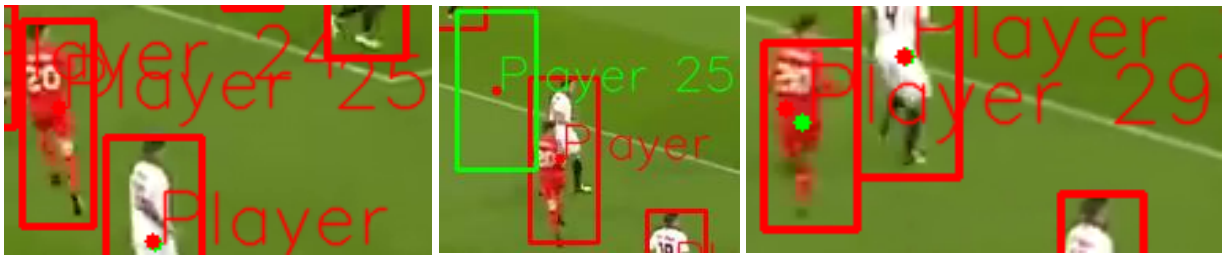


Ilustración 5.16 Fallo de correspondencia por predicciones imprecisas

Vemos en la secuencia de imágenes como un jugador, detectado como jugador 25, tras perderse en las detecciones y determinar su posición solo con predicciones, vemos como estas son imprecisas, de forma que al volver a detectarlo lo determinamos como un jugador nuevo, en este caso jugador 29.

Por otro lado, ya que la única condición es la proximidad, también podemos encontrarnos con fallos de correspondencia al cruzarse dos jugadores, aunque debido a los poco que varían las posiciones de un frame a otro, esto no sucede con mucha frecuencia.

Una vez explicado las causas que provocan este problema, vamos a tratar de cuantificarlo de cara a evaluar con qué frecuencia sucede. Para esto, expondremos gráficos de distintos videos con los datos de la cantidad de jugadores detectados en un inicio más los falsos positivos, la cantidad de jugadores totales que se han detectado a lo largo del video, y la cantidad de jugadores que han entrado en plano a lo largo del video, considerándolos como los únicos nuevos jugadores que pueden aparecer. De esta forma, para una detección y correspondencia perfecta, si tenemos siete jugadores en un inicio y entran dos nuevos en la imagen, al final del video tendríamos nueve jugadores totales detectados en toda la secuencia. Con estos datos, realmente no estamos cuantificando solo el buen funcionamiento de la correspondencia, sino que también es influido por falsos negativos que también serían considerados como nuevos jugadores, por lo que en estos datos veremos reflejados en realidad el resultado de la acumulación de los errores que hemos ido exponiendo en el análisis.

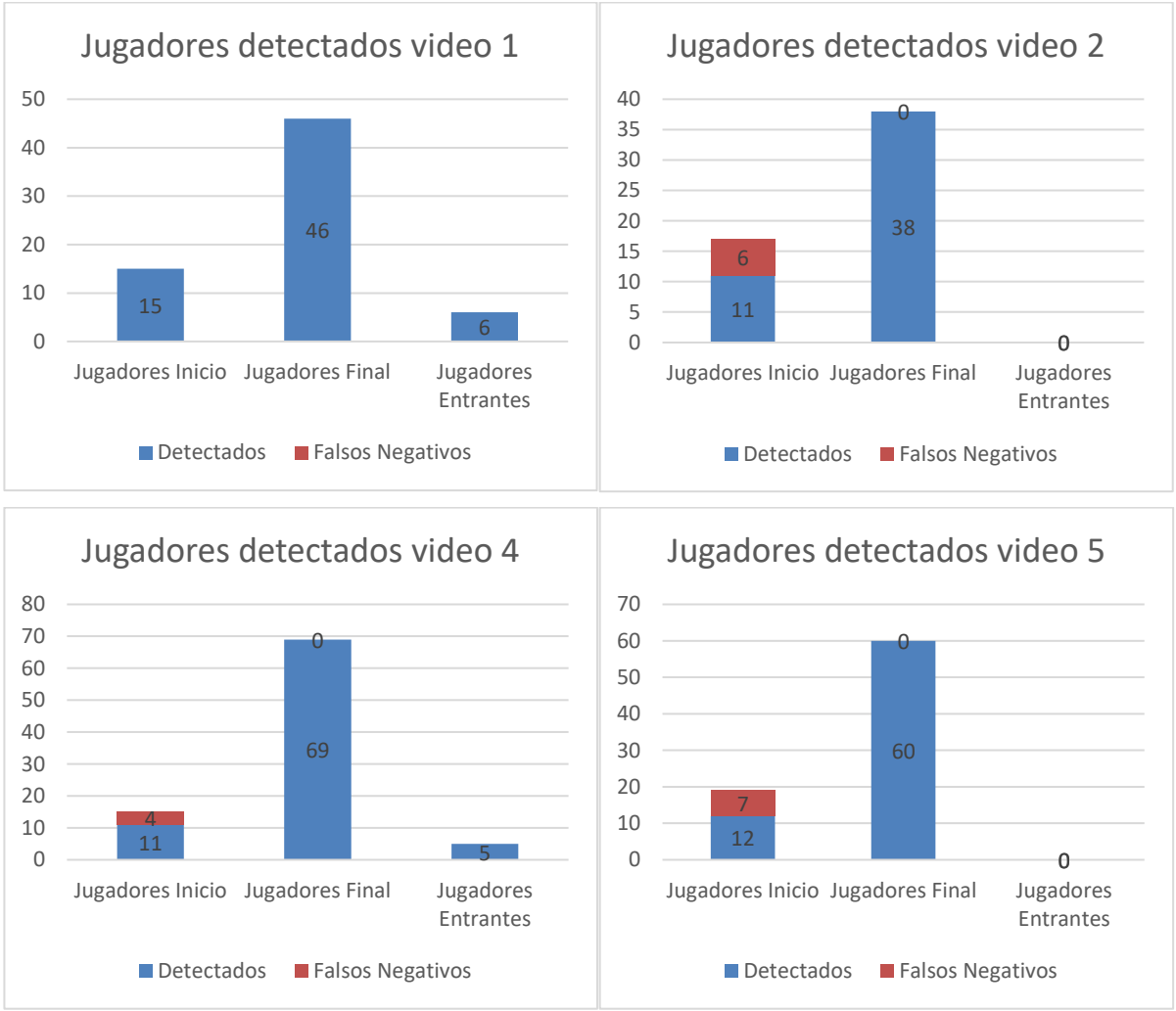


Ilustración 5.17 Jugadores detectados al inicio y a lo largo del video

A la vista de los gráficos comprobamos como el número de detectados a lo largo del video son entre 2 y 4 veces mayor de los que deberían ser en un caso ideal. Como decimos, en estos datos se reflejan tanto los fallos de correspondencia que se han comentado en este apartado, como los falsos positivos ocasionados por las distintas circunstancias ya mencionadas. Principalmente, si analizamos el comportamiento de nuestro programa de una forma más "manual", podemos ver como estos problemas, tanto de fallo de correspondencia como de falso positivo, son más frecuentes cuando la jugada se centra en el aérea de la portería. Si analizamos lo sucedido en esta zona, comprobamos como aglutina la gran mayoría de inconvenientes que se pueden tener para la detección de los jugadores y su seguimiento, ya que suelen haber gran cantidad de jugadores concentrados en una misma zona y hay numerosas líneas de terreno de juego que entorpecen la correcta detección. Es fácil comprobar cómo tan solo dos o tres jugadores en el área cercanos entre sí pueden dar numerosos fallos de correspondencia, aumentando significativamente las estadísticas anteriores. Sin embargo, si ponemos nuestra atención en este caso al resto de jugadores, que corren por la banda o se mueven por el centro del campo, lo usual es que el

algoritmo lo detecte correctamente a lo largo de toda la secuencia, consiguiendo determinar que es el mismo jugador desde que empieza la jugada hasta que acaba.

5.1.6 Coste computacional

Un importante aspecto a la hora de mostrar los resultados de un programa es cuantificar cuanto tarda nuestro ordenador en procesarlo. Concretamente, para nuestra aplicación es un punto muy a tener en cuenta, ya que este tipo de herramienta podría ser empleada para un análisis en tiempo real, de forma que necesitaríamos que el programa sea capaz de realizar sus cálculos a la misma velocidad a la que capturamos el juego. Para evaluar y mostrar cómo se comporta nuestro programa en este aspecto, realizaremos una sencilla cuenta con cada video que será la cantidad de frame que compone el video dividido por el tiempo en segundos que tarda en ejecutarse el programa para el video entero, obteniendo así los frames por Segundo que es capaz de procesar el programa. En este aspecto, también es muy importante tener en cuenta el equipo que se usa para ejecutar el programa, ya que dependiendo de la capacidad de cómputo de cada equipo el rendimiento del programa podrá ser mejor o peor. Las especificaciones del ordenador con las que se han realizado las pruebas son:

- SO Windows 10 Pro 64 bits
- Procesador i5 3.2GHz
- 8 GB RAM
- GTX 1060 3GB

Obtenemos los siguientes resultados:

	Video 1	Video 2	Video 3	Video 4	Video 5	Video 6	Video 7	Video 8
Frames	250	126	720	276	225	75	250	250
Tiempo (s)	14,76	8,21	12,6	17,51	14,52	5,36	16,94	15,36
FPS	16,93766938	15,3471376	57,1428571	15,7624215	15,4958678	13,9925373	14,7579693	16,2760417

Tabla 5.1 Frames y tiempos de procesamiento de cada video

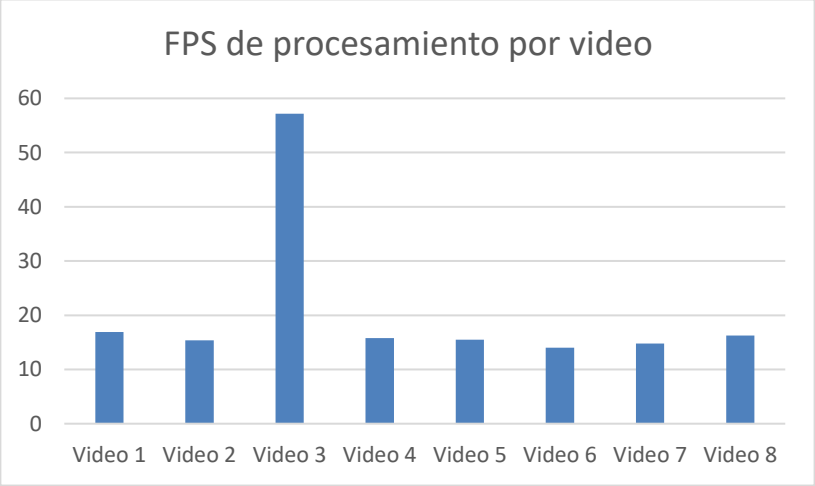


Ilustración 5.18 FPS de procesamiento por video

A la vista del gráfico, de puede destacar rápidamente el video 3 por su gran diferencia en velocidad con respecto al resto. Esto es debido a que este es el único video con una resolución menor a 720p y que además realiza muy pocas detecciones. Este video a menor resolución ha sido escogido por un motivo que expondremos en el siguiente apartado, pero de momento, no lo tendremos muy en cuenta a la hora del análisis del coste computacional. Si observamos el resto de videos, vemos como los FPS rondan entre los 14 y 17 según el video. Esta varianza es principalmente debido a la cantidad de jugadores que tenemos y detectamos en pantalla, ya que mientras más jugadores más veces tendremos que procesar algunos algoritmos pesados como el del filtro de Kalman, que correspondería a un filtro por jugador detectado, además del balón.

Si atendemos a los FPS de media que nos ofrecen los videos, excluyendo el video 3, comprobamos que se obtienen más o menos unos 15 FPS por video. Estas retransmisiones deportivas suelen tener algo más de FPS

capturados, normalmente 24 FPS. Por lo que podemos comprobar fácilmente como está cercano a una velocidad de procesamiento de tiempo real pero no llega. Pero esto sería en caso de procesar todos lo frames ofrecidos por la retransmisión, pero realmente, según lo visto en los resultados, no necesitamos tal cantidad de fotogramas para realizar una buena detección y seguimiento, ya que de un frame a otro la escena apenas cambia. De esta forma, se podría procesar la imagen cada 2 o 3 frames capturados, velocidad a la cual llega sin problemas, por lo que podemos determinar que podría tener una buena aplicación como procesamiento en tiempo real.

5.1.7 Resolución del video

Como ultimo problema a destacar en este análisis de los resultados, antes de proceder a mostrar los resultados de las posibles aplicaciones reales, es el de la resolución del video. Básicamente la resolución del video representa la cantidad de pixeles con los que se compone cada imagen, de forma que, a mayor resolución, mayor cantidad de pixels tenemos. En este aspecto el programa presenta un problema de compatibilidad, y es que, al cambiar la resolución, cambia muchos de los parámetros que usamos para las mediciones que determinan si el elemento es un jugador o no. El mayor ejemplo es la característica de área, que en este caso al tener menos cantidad de pixels en general, es de suponer que los propios jugadores estarán compuestos por menos pixels.

Por este motivo se ha incluido el análisis del video 3, un video de 480p, frente a los 720p del resto de videos. Si lo vemos, comprobamos fácilmente como el algoritmo no consigue detectar la mayoría de los jugadores, por el motivo que comentamos. Para este proyecto nos hemos centrado en videos 720p que son los más comunes en la actualidad, pero en caso de querer adaptar el programa a otros formatos, tan solo sería necesario modificar y ajustar algunos parámetros del programa.

5.2 Aplicación real

En este apartado mostraremos las utilidades que puede tener esta herramienta en una aplicación real. Como sabemos, en el ámbito deportivo las estadísticas juegan cada vez un papel más importante, tanto por el lado del espectador y la retransmisión, ya que ayudan a tener más información y saber con mayor precisión que está sucediendo, como por el lado de los deportistas y entrenadores, ya que estos datos ayudan a poder perfeccionar estrategias y entrenamientos. Para esta tarea, nuestro programa tiene mucho que aportar, ya que permite obtener estas estadísticas de forma automatizada, simplemente haciendo uso de las capturas de las retransmisiones. Para la recopilación de estadísticas y demás datos útiles, de cara a usarlo en un entorno real, nos hemos inspirado en el artículo [15], el cual trata de como automatizar estos datos precisamente con visión artificial para ofrecer las estadísticas en una aplicación web.

De esta manera, plantearemos una serie de estadísticas dentro de todas las posibles a modo de ejemplo. Estas estadísticas podremos visualizarlas si ejecutamos el programa en el modo 3 y agregamos el jugador al cual queremos realizar la recopilación de datos. Las estadísticas han sido centralizadas en este ejemplo para un único jugador por sencillez y ligereza del programa, pero el programa debería estar capacitado para realizarlas para todos los jugadores detectados.

5.2.1 Mapa de calor

La primera de las estadísticas planteadas es la de mapa de calor. Este mapa de calor usualmente marca sobre el terreno de juego por donde se ha ido moviendo dicho jugador, a lo largo de todos los datos guardados. Para esto, teniendo ya la posición del jugador en cada frame, obtenida con el programa, tan solo necesitamos realizar un registro de estas posiciones y posteriormente representarlas todas juntas sobre un rectángulo que represente el terreno de juego. Esta es una estadística muy útil puesto que representa todos los movimientos que ha realizado el jugador sobre el terreno de juego, indicándonos por qué zonas se mueve más. Mostramos a continuación algunos ejemplos de esta estadística:

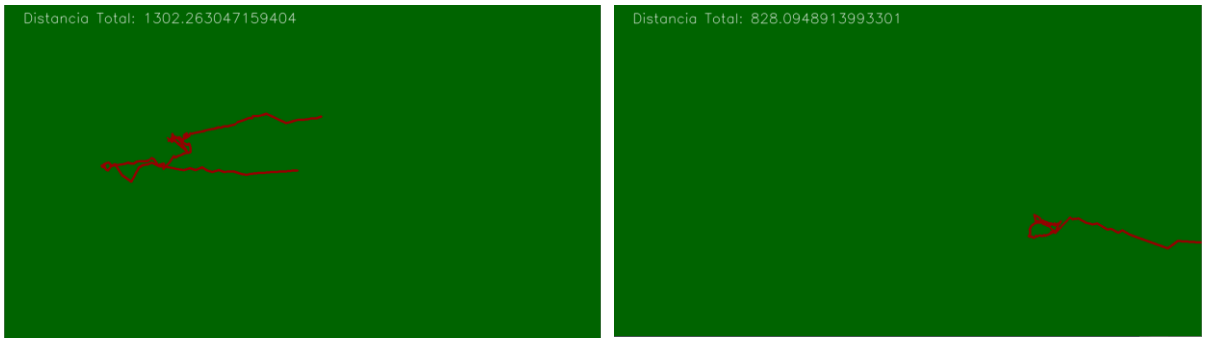


Ilustración 5.19 Mapa de calor del jugador 4 y 12 del video 1

Esta herramienta depende directamente de la detección, al igual que el resto de estadística que mostraremos en este apartado, de forma que, si tenemos una buena detección para los jugadores seleccionados, dispondremos de unas buenas características fiables y representativas de la realidad. En este caso tan solo destacar como el jugador 12 acaba saliéndose del plano durante el video, por lo que evidentemente dejamos de verlo representado en el mapa de calor. Además, complementamos el mapa de calor con la información de la distancia total recorrida por el jugador.

En cuanto al funcionamiento general si se ha destacar que este mapa de calor representa las posiciones de los jugadores según su posición captada con la cámara, que no se corresponde con su posición real en el terreno de juego. Para esto último, sería necesario detectar el terreno de juego al completo, de forma que detectando sus límites se pueda determinar la transformación matemática de las coordenadas en pixel de la cámara, a las coordenadas en metros del espacio real. Realizar esto tan solo a partir de la información captada por las cámaras de las retransmisiones sería más complicado, fuera del alcance del actual proyecto.

5.2.2 Velocidad del jugador

Otra de las estadísticas que podemos obtener de forma sencilla con este programa es la velocidad de los jugadores en cada instante. En este caso será fácil partiendo de los cálculos de distancia recorrida de la estadística anterior. Simplemente la distancia que se mueve un jugador en un frame será la velocidad de ese jugador en ese instante. De esta forma obtenemos la velocidad por frame, que en caso de quererlas por segundo tan solo sería necesario multiplicar el dato por los fps a los que graba la cámara. En este caso no se ha realizado, porque tampoco tendremos la distancia en metros por la problemática comentada en el apartado anterior, por lo que igualmente no podemos aportar la información en metros por segundo, que sería la unidad más adecuada. Vemos a continuación las velocidades de los mismos jugadores representados en el mapa de calor del apartado 5.2.1.

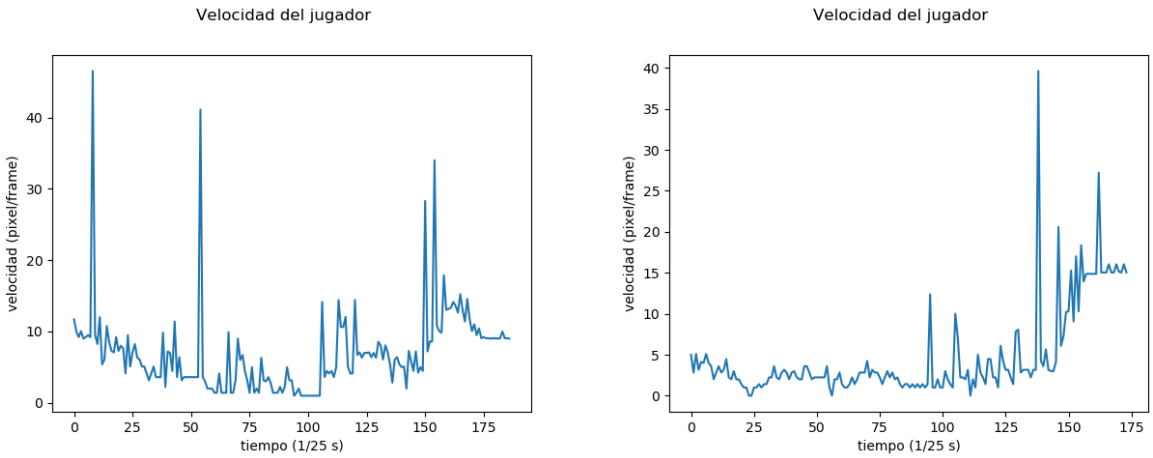


Ilustración 5.20 Velocidad de los jugadores 4 y 12 del video 1

El mayor problema de esta estadística, derivado del problema de no poder obtener las posiciones en las coordenadas reales del terreno de juego, es que un jugador en movimiento puede aparecer como si estuviese estático. Esto es debido a que la cámara está grabando con un plano dinámico, de forma que si mueve el plano a la misma velocidad y en el mismo sentido que el jugador, este aparecerá en las estadísticas como si estuviese

estático. Este problema se solventaría o con un plano fijo, o calculando las velocidades a partir de las posiciones reales en caso de disponer de ellas.

5.2.3 Tiempo corriendo

Finalmente ofreceremos una última estadística de ejemplo que será el tiempo que ha estado el jugador corriendo. Este dato será fácil de obtener a partir de la estadística de la velocidad del jugador, tan solo será necesario establecer un umbral de velocidad a partir del cual determinemos que el jugador está corriendo y no andando o parado. Este umbral lo hemos establecido de forma manual por un criterio experimental. Este dato lo imprimiremos por la ventana de comando al finalizar el programa. Los datos de tiempo corriendo para los mismos jugadores anteriores serán las siguientes:

Jugador corriendo	**Jugador 4**	**Jugador 12**
Tiempo (s)	7.48 s	6.96 s

Tabla 5.2 Tiempo corriendo de los jugadores 4 y 12 del video 1

Según la estadística ofrecida, ambos jugadores han estado la mayor parte de la jugada corriendo, ya que el total de tiempo de esta jugada es de 10 s. Si atendemos al video podemos comprobar como efectivamente esta estadística, a pesar del posble fallo comentado, no se aleja de la realidad en cuanto al tiempo que han estado corriendo.

CONCLUSIONES, FUTURAS INVESTIGACIONES Y OTRAS APLICACIONES

Para finalizar, expondremos en este apartado que conclusiones se han extraído de este Trabajo Fin de Grado y sus resultados, justificando si se ha llegado al alcance previsto en un principio para el proyecto. Además, comentaremos qué futuras líneas de investigación pueden partir de este proyecto y que otras aplicaciones puede llegar a tener.

6.1 Conclusiones

6.1.1 Proceso de detección

El presente proyecto tenía como primer objetivo aplicar una serie de técnicas de visión por computador para la detección de atletas a partir de las retransmisiones deportivas, con el fin de evaluar el funcionamiento de dichas técnicas. Estas se han implementado en un primer proceso que hemos definido a lo largo de este documento como proceso de detección. Básicamente este proceso estará formado por un algoritmo que ejecuta una serie de técnicas ya contrastadas en el sector de la visión por computador, y que se han implementado y ajustado para la aplicación concreta que se pretende llevar a cabo.

En este aspecto, obtenemos unos resultados acordes con los esperados, donde se ha conseguido ajustar estas técnicas para tener un buen porcentaje de acierto en dichas detecciones. A pesar de esto, este proceso si es cierto que tiene dificultades en algunas circunstancias concretas tal y como hemos comentado en el análisis de los resultados. Con esto nos hemos dado cuenta de cómo efectivamente estas técnicas, a pesar de ser muy útiles y eficaces si se implementan bien, tienen ciertas limitaciones. La mejor forma para suplir estas limitaciones será implementar otras técnicas nuevas que puedan ayudar a extraer mejor la información, o a realizarle un mejor tratamiento como se ha realizado por ejemplo en este proyecto con el filtro de Kalman. Una de las técnicas que podrían ayudar a mejorar de forma cuantitativa el programa sería aplicar algoritmos de machine learning, los cuales son muy utilizados en el sector y han supuesto un salto de calidad importante en cuanto al funcionamiento y alcance de la visión por computador.

6.1.2 Implementación del filtro de Kalman

Tras el proceso de detección, implementamos una de las técnicas más importantes para mejorar el funcionamiento del programa como es el filtro de Kalman. Este filtro es utilizado normalmente para optimizar las medidas de sensores con ruido, pudiendo además realizar predicciones en caso de perder la señal de estas mediciones como puede ser el caso del GPS.

Concretamente en este programa, su función será ayudar tanto al seguimiento de los atletas, como a la propia detección de estos. Para esto se hace uso de las predicciones que nos ofrece el filtro cuando no se consigue obtener la posición del jugador en algún instante. Esto, como bien comentamos en el apartado anterior, sucede con cierta frecuencia en el proceso de detección cuando se dan algunas circunstancias concretas. Llevado a la práctica nos damos cuenta como el uso de este método complementa perfectamente a las medidas que se obtienen del proceso de detección, consiguiendo solucionar gran parte de los problemas que nos encontramos en este proceso, y siendo la herramienta fundamental para realizar el seguimiento una vez detectados tanto los jugadores como el balón, de forma que sin él sería una tarea prácticamente imposible.

Además, gracias al seguimiento, podemos realizar numerosas estadísticas de juego, de las cuales también se han mostrado algunos ejemplos en este proyecto. Estas estadísticas parten de este seguimiento realizado con Kalman, de forma que, sabiendo la posición de un jugador a lo largo de una secuencia, podemos recopilar una serie de datos que son los que se pretenden automatizar con este programa. El resultado de las estadísticas ofrecidas es aceptable, dentro de las limitaciones ya nombradas, y no sería demasiado complicado a partir de este programa ofrecer otras estadísticas como posesión del balón, pases efectuados…

6.2 Futuras investigaciones

El actual proyecto, puede ser tomado como base para futuras líneas de investigación en torno al mismo campo de estudio en el que se centra el proyecto. Una de las principales líneas de investigación que a mi juicio sería interesante desarrollar, sería la implementación de técnicas de machine learning, como pueden ser las redes neuronales, que complementen las técnicas ya usadas. Si se entrenase un algoritmo de machine learning para detectar jugadores por ejemplo, se podrían eliminar gran cantidad de falsos positivos que se pueden tener con las técnicas actuales, ya que podríamos acceder al juicio del algoritmo de si es o no un jugador el elemento detectado. Además se podría entrenar el algoritmo como clasificador de jugadores, pudiendo determinar si el jugador detectado pertenece a un equipo o a otro, o si en su defecto pertenece al cuerpo arbitral. También destacar que en este caso, la implementación de machine learning junto con el actual programa, se prevé con un mejor rendimiento que en el caso de implementar tan solo machine learning para realizar toda la tarea, ya que si contamos con una detección previa, el algoritmo de machine learning puede actuar tan solo sobre las detecciones realizadas, no hacienda falta que recorra toda la imagen para detector un jugador.

Por otro lado otra investigación interesante, tal como hemos visto en el resultado de las aplicaciones reales, sería el de realizar un algoritmo que busque ciertos elementos característicos del campos de juego, a partir de los cuales se puedan usar de referencia para poder transformar las coordenadas de pixels a medidas reales del campo de juego. Con este algoritmo se conseguiría unas estadísticas más útiles y con más información de cara a un uso real.

6.3 Otras aplicaciones

Finalmente, comentaremos que el proyecto se ha centrado en un deporte concreto, el fútbol, dado que las técnicas de visión requieren apoyarse de algunas características destacables y comunes para los entornos que se pretende estudiar. Pero la estructura básica del algoritmo puede ser replicada para hacerla funcionar en otros deportes, con más o menos cambios según como de parecido sea el entorno de dicho deporte. Para exponer un ejemplo, hemos procesado una jugada de rugby con exactamente el mismo programa, sin realizar cambios, ya que tiene muchas similitudes con el fútbol en cuanto al entorno que es lo que nos interesa para la visión.

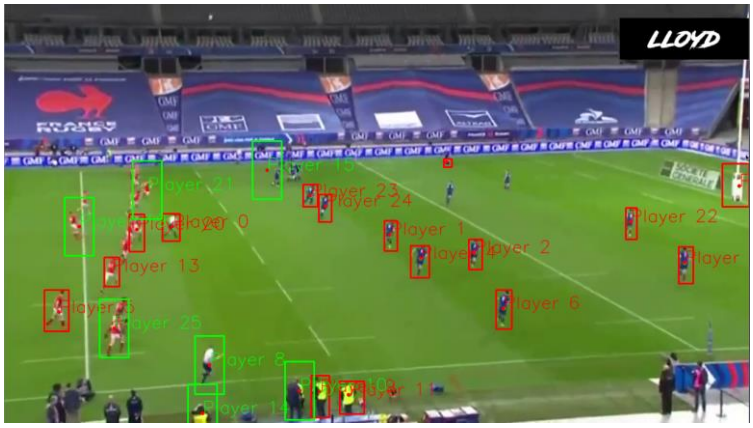


Ilustración 6.1 Procesado de jugada de rugby

Como vemos, no le haría falta realizarle muchos ajustes, quizás tan solo en la detección y seguimiento del balón de juego que es donde encontramos más diferencias. Esto mismo, se podría aplicar a otros deportes realizando los ajustes que fuesen necesarios. Por ejemplo, si quisiéramos implementar un algoritmo de detección y seguimiento en atletismo, podríamos hacerlo cambiando tan solo el algoritmo de supresión de fondo, ya que en este caso el fondo predominante no sería el verde del césped, sino el rojo de la pista de atletismo.

Además de a los deportes, este algoritmo también se puede extrapolar a otros campos como puede ser el de la vigilancia y seguridad, donde quiera detectar personas en un espacio concreto y realizarles un seguimiento en busca de comportamientos extraños o accedan a una zona restringida.

CÓDIGOS

7.1 Código programa principal

```
"""
Programa principal version 0.1
Detectar jugadores y pelota
"""
#importamos los paquetes necesarios
import argparse
import cv2 as cv
import numpy as np
import imutils
from imutils.video import FileVideoStream
import time
import math
from math import sqrt
import matplotlib.pyplot as plt


#construimos los argumentos necesarios para el programa
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", required=True,
	help="ruta del video a analizar")
ap.add_argument("-m", "--modo", default=1, type=int,
	help="modo del programa")
ap.add_argument("-o", "--output", default=1, type=int,
	help="elegir visualización")
ap.add_argument("-j", "--jugador", default=1, type=int,
	help="jugador a realizar estadisticas")
args = vars(ap.parse_args())

def distancia(p1, p2):
	"""
	Función sencilla para calcular la distancia entre puntos
	"""

	dist = sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)
	return dist

def line_disc(img,ini, fin, color, grosor):
	"""
	Funcion para dibujar una linea discontinua en una imagen
	"""

	paso_x = abs(ini[0]-fin[0])/16
	paso_y = (fin[1]-ini[1])/16

	for i in range(0,16,4):
		cv.line(img, (int(ini[0]+i*paso_x), int(ini[1]+i*paso_y)), (int(ini[0]+(i+1)*paso_x),
int(ini[1]+(i+1)*paso_y)), color, grosor)

	return img

def relaccion_jugadores(pos_ant, pos_act):
	"""
	Función para relacionar un jugador detectado en un frame con
	el detectado en el frame siguiente. Esta función ayuda al seguimiento
```

```
        y a la actualización del filtro de kalman.
        """

        i = 0

        pos_res = len(pos_ant)*[None]

        for c1 in pos_act:

                j = 0
                cambio = 0
                dist_min = 1000000

                for c2 in pos_ant:
                        dist_act = distancia(c1, c2)

                        if (dist_act < dist_min and dist_act<50 and pos_res[j]==None):
                                dist_min = dist_act
                                pos_j = j
                                cambio = 1

                        j = j + 1

                if (cambio == 1):
                        pos_res[pos_j] = c1
                else:
                        pos_res.append(c1)

        return pos_res

class FiltroKalman:
        """
        Clase de filtro de kalman para aplicar a cada uno de los elementos
        a realizar un seguimiento, además de ayudar a la detección en
        cirscunstancias especiales.
        """
        def __init__(self):
                q = 10
                r = 1
                self.kf = cv.KalmanFilter(4, 2)
                self.kf.measurementMatrix = np.array([[1, 0, 0, 0], [0, 1, 0, 0]], np.float32)
                self.kf.transitionMatrix = np.array([[1, 0, 1, 0], [0, 1, 0, 1], [0, 0, 1, 0], [0, 0, 0, 1]], np.float32)
                self.kf.measurementNoiseCov = np.array([[r**2, 0],[0, r**2]], np.float32)
                self.kf.processNoiseCov = np.array([[q**2,0,0,0],[0,q**2,0,0],[0,0,q**2,0],[0,0,0,q**2]],
np.float32)

        def Estimate(self, coordX, coordY):
                ''' Con esta función se estimará la posición del elemento en este instante'''
                measured = np.array([[np.float32(coordX)], [np.float32(coordY)]])
                predicted = self.kf.predict()
                if (coordX!=0 or coordY!=0):
                        predicted = self.kf.correct(measured)

                return predicted

# FUNCIONES

def condicionRGB(img):
        """
        Función para aplicar la condición G>R>B para extraer
        el campo de juego de nuestra imagen
```

```
        """

        b,g,r = cv.split(img)

        #Aplicamos condición g>r>b
        bgExtract1 = g>r
        bgExtract2 = r>b
        bgExtract1 = np.uint8(bgExtract1)
        bgExtract2 = np.uint8(bgExtract2)

        bgExtract = cv.bitwise_and(bgExtract1,bgExtract2)
        bgExtract = np.uint8(bgExtract)*255

        #Aquí extraemos y guardamos la máscara del campo
        bgMask = np.zeros((height,width), dtype=np.uint8)
        contours, hierarchy = cv.findContours(bgExtract, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
        c = max(contours, key = cv.contourArea)
        cv.drawContours(bgMask, [c], -1, 255, -1)

        bgExtract = cv.bitwise_not(bgExtract)

        return bgExtract, bgMask

def gradiente_sobel(img, umbral):
        """
        Función donde calcularemos el gradiente de sobel
        de nuestra imgaen y aplicaremos un umbral
        """
        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        gray = cv.GaussianBlur(gray,(5,5),0)
        dX = cv.Sobel(gray, cv.CV_32F, 1, 0, (3,3))
        dY = cv.Sobel(gray, cv.CV_32F, 0, 1, (3,3))
        mag, direction = cv.cartToPolar(dX, dY, angleInDegrees=True)
        _, SobelG = cv.threshold(mag,umbral,255,cv.THRESH_BINARY)
        SobelG = SobelG.astype(np.uint8)

        return SobelG

def etiquetado(img, imgOrig):
        """
        Proceso de etiquetado de cada una de las siluetas extraidas
        en nuestra imagen.
        Eliminamos además aquella mayores de una cierta área.
        """
        x_ball = 0
        y_ball = 0
        pos_jug = np.array([0,0])
        num_labels, labels_im, stats, centroids = cv.connectedComponentsWithStats(img)

        jugadores = []
        for label in range(num_labels):

                area = stats[label, cv.CC_STAT_AREA]
                alto = stats[label, cv.CC_STAT_HEIGHT]
                ancho = stats[label, cv.CC_STAT_WIDTH]

                densidad = area / (alto*ancho)

                #Caracteristicas de los jugadores
                if (area > 500) and (area < 2000):
```

```
                        if (alto/ancho > 1 and alto/ancho <3 ):
                                if (densidad>0.3 and densidad<0.9):
                                        x_bb = stats[label, cv.CC_STAT_LEFT]
                                        y_bb = stats[label, cv.CC_STAT_TOP]
                                        cv.rectangle(imgOrig, (x_bb, y_bb), (x_bb+ancho, y_bb+alto),
(0,0,255), 2)
                                        cv.circle(imgOrig, (int(centroids[label,0]), int(centroids[label, 1])), 3,
(0,255,0), -1)
                                        jugadores.append([int(centroids[label,0]), int(centroids[label, 1])])

                #Caracteristicas del balon
                if (area > 100) and (area < 300):

                        if (alto/ancho > 0.5 and alto/ancho <=1.5 ):
                                if (densidad>0.7 and densidad<0.9):
                                        x_bb = stats[label, cv.CC_STAT_LEFT]
                                        y_bb = stats[label, cv.CC_STAT_TOP]
                                        cv.rectangle(imgOrig, (x_bb, y_bb), (x_bb+ancho, y_bb+alto),
(0,0,255), 2)
                                        cv.circle(imgOrig, (int(centroids[label,0]), int(centroids[label, 1])), 3,
(255,0,0), -1)
                                        x_ball = int(centroids[label,0])
                                        y_ball = int(centroids[label,1])
                                        #print(area, densidad)


        return imgOrig, x_ball, y_ball, jugadores

def TransformadaHough(img, orig, rho, theta, threshold, minLenght, maxGap):
        """
        Función para calcular la tranformada de hough de nuestra imagen
        a fin de detectar la lineas rectas del terreno de juego y eliminarlas
        """
        lines = cv.HoughLinesP(img,
                                                rho,
                                                theta,
                                                threshold,
                                                lines = np.array([]),
                                                minLineLength = minLenght,
                                                maxLineGap = maxGap)

        for line in lines:
                for x1, y1, x2, y2 in line:
                        #cv.line(img, (x1, y1), (x2, y2), (0,0,0), 8)
                        img = line_disc(img, (x1,y1), (x2,y2), (0,0,0), 7)

        return img



#inicializamos el video a leer
print("[INFO] Leyendo clip...")
vs = FileVideoStream(args["video"]).start()
time.sleep(2.0)

jugadores_ant = 0
registro_posiciones=[]

kernal = np.ones((2,2), np.uint8)

KFJug = []
```

```
jug_perdido_cont = []
KFBall = FiltroKalman()
predictedCoords = np.zeros((2, 1), np.float32)

if (args["modo"]==1):
        ejecuta = 0
else:
        ejecuta = 1



#Bucle de lectura de cada frame
while True:
        try:
                frame = vs.read()
                height = frame.shape[0]
                width = frame.shape[1]

                #Condición g>r>b
                bgExtract, bgMask = condicionRGB(frame)

                #Aplicamos gradiente de sobel
                SobelG = gradiente_sobel(frame, 40)

                #Sumamos resultado de ambos procesosa
                imgFinal = bgExtract + SobelG

                #Aplicamos máscara de región de interés
                imgFinal = cv.bitwise_and(imgFinal,bgMask)

                #Destacamos siluetas con operaciones morfológicas
                imgFinal = cv.morphologyEx(imgFinal,cv.MORPH_CLOSE, kernal, iterations=2)

                #Buscamos rectas predominantes en la imagen
                imgFinal = TransformadaHough(imgFinal, frame, 6, np.pi / 60, 500, 80, 15)

                #Etiquetamos entra las siluetas detectadas, cual correspoende a jugadores y pelota
                frame, x_b, y_b, jugadores_act = etiquetado(imgFinal, frame)

                #Hacemos el seguimiento de los jugadores detectados
                if (jugadores_ant!=0):
                        #Relacionamos los jugadores detectados con los detectados anteriormente
                        pos_actualizada = relaccion_jugadores(jugadores_ant, jugadores_act)
                        print("## NUEVO FRAME ##")
                        print(pos_actualizada)

                        if (args["modo"]==3):
                                #Se guarda la información del jugador seleccionado
                                try:

        registro_posiciones.append([int(jugadores_ant[args["jugador"]][0]),int(jugadores_ant[args["jugador"]][1])]
)
                                except:
                                        print("No existe ese jugador\n")

                        if (args["modo"]==1):
                                #Ponemos texto identificador encima de cada jugador
                                num_jug = 0

                                for jugador in pos_actualizada:
                                        if jugador != None:
```

```
                                        frame = cv.putText(frame, 'Player ' + str(num_jug),
(jugador[0], jugador[1]), cv.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 1, cv.LINE_AA)
                                        num_jug = num_jug + 1

                        #Agregamos a la lista aquellos jugadores detectados nuevos
                        while (len(KFJug) != len(pos_actualizada)):
                                KFJug.append(FiltroKalman())
                                jug_perdido_cont.append(0)

                        jugadores_ant = pos_actualizada

                else:

                        jugadores_ant = jugadores_act

                #Estimación con filtro de kalman
                for j in range(len(KFJug)):
                        #Comprobamos que el jugador no lleve más de 10 frames sin detectarse
                        if jug_perdido_cont[j]<11:
                                #Si se ha actualizado la posición, actualizamos Kalman con las nuevas medidas
                                if (pos_actualizada[j] != None):
                                        JugCoords = KFJug[j].Estimate(pos_actualizada[j][0],
pos_actualizada[j][1])
                                        jug_perdido_cont[j] = 0
                                #Si no se ha actualizado la posición, realizamos una predicción con Kalman
                                else:
                                        JugCoords = KFJug[j].Estimate(0, 0)
                                        jugadores_ant[j] = [JugCoords[0], JugCoords[1]]
                                        jug_perdido_cont[j] = jug_perdido_cont[j]+1

                                        #Ponemos texto identificado para los jugadores detectados por
predicción
                                        if (args["modo"]==1):
                                                frame = cv.putText(frame, 'Player ' + str(j), (int(JugCoords[0]),
int(JugCoords[1])), cv.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 1, cv.LINE_AA)
                                        cv.rectangle(frame, (int(JugCoords[0])-25, int(JugCoords[1])-50),
(int(JugCoords[0])+25, int(JugCoords[1])+50), (0,255,0), 2)

                                cv.circle(frame, (int(JugCoords[0]), int(JugCoords[1])), 3, (0,0,255), -1)
                        #Damos por perdido el jugador si no aparece durante 10 frames consecutivos
                        else:
                                jugadores_ant[j] = [0,0]

                #Estimación de la pelota con filtro de Kalman
                predictedCoords = KFBall.Estimate(x_b, y_b)
                print(predictedCoords)
                cv.circle(frame, (int(predictedCoords[0]), int(predictedCoords[1])), 3, (0,0,255), -1)
                '''
                print("##Reales##")
                print(x_b, y_b)
                print("##Predichas##")
                print(int(predictedCoords[0]), int(predictedCoords[1]))
                '''

                #Abrimos ventana con la visualización de las distintas imágenes procesadas
                #cv.imshow('Sobel', SobelG)
                #cv.imshow('bgExtract', bgExtract)
                if (args["output"]==1):
                        cv.imshow('Original', frame)
                if (args["output"]==2):
                        cv.imshow('Discriminacion color', bgExtract)
```

```
                if (args["output"]==3):
                        cv.imshow('Sobel', SobelG)
                if (args["output"]==4):
                        cv.imshow('Procesado Final', imgFinal)


                #Comprobamos si qe ha pulsado la tecla para salir
                key = cv.waitKey(ejecuta) & 0xFF
                if key == ord("q"):
                        break
                #Si pulsamos la letra "s" guardaremos el actual frame como una imagen png
                elif key == ord("s"):
                        cv.imwrite("Imagenes/last_save.png", frame)

        except:
                """
                Esta excepción saltará cuando el video llegue a su fin.
                En esta rutina de excepción se finalizarán determinados procesos y se sacarán las estadísticas de
                la jugada
                """
                cv.destroyAllWindows()
                vs.stop()

                if (args["modo"]==3):
                        mapa_calor = np.zeros((height,width,3), np.uint8)
                        mapa_calor[:] = (0,100,0)
                        distancia_recorrida = 0
                        tiempo_corriendo = 0
                        velocidades = []
                        for j in range(1,len(registro_posiciones)):
                                if (registro_posiciones[j][0]!=0 or registro_posiciones[j][1]!=0):
                                        cv.line(mapa_calor,
(registro_posiciones[j][0],registro_posiciones[j][1]), (registro_posiciones[j-1][0],registro_posiciones[j-1][1]),
(0,0,150), 3)
                                        distancia_actual =
distancia((registro_posiciones[j][0],registro_posiciones[j][1]), (registro_posiciones[j-1][0],registro_posiciones[j-
1][1]))
                                        distancia_recorrida = distancia_recorrida + distancia_actual
                                        velocidades.append(distancia_actual)
                                        if distancia_recorrida > 3:
                                                tiempo_corriendo = tiempo_corriendo + 1
                                print(distancia_actual)

                        print("ESTADÍSTICAS RECOPILADAS DEL VIDEO")
                        print("DISTANCIA RECORRIA POR EL JUGADOR:")
                        print(distancia_recorrida)
                        print("VELOCIDAD MÁXIMA DEL JUGADOR:")
                        print(max(velocidades))
                        print("TIEMPO QUE HA ESTADO EL JUGADOR CORRIENDO:")
                        print(tiempo_corriendo/25)

                        plt.plot(range(len(velocidades)),velocidades)
                        plt.xlabel('tiempo (1/25 s)')
                        plt.ylabel('velocidad (pixel/frame)')
                        plt.suptitle('Velocidad del jugador')
                        plt.show()

                        cv.putText(mapa_calor, 'Distancia Total: ' + str(distancia_recorrida), (40,40),
cv.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 1, cv.LINE_AA)
                        cv.imshow('Estadistica 1', mapa_calor)
                        cv.waitKey(0)
```

```
            break
```

7.2 Código programa de descarga de videos de prueba

```
"""
Pequeño script para guardar los clips de los momentos concretos de videos de youtube
de forma sencilla y rápida.
"""

#importar librerias necesarias
import argparse
import pytube
from moviepy.video.io.ffmpeg_tools import ffmpeg_extract_subclip
from os import remove

#construir los argumentos necesarios para el programa
ap = argparse.ArgumentParser()
ap.add_argument("-l", "--link", required=True,
        help="link del video")
ap.add_argument("-n", "--name", required=True,
        help="nombre de destino del video")
ap.add_argument("-r", "--resolution", type=str, default="720p",
        help="resolucion de descarga")
ap.add_argument("-i", "--inicio", required=True, type=int,
        help="segundo inicial del video")
ap.add_argument("-f", "--final", required=True, type=int,
        help="segundo final del video")
args = vars(ap.parse_args())

#proceso de descargar del video
print("[INFO] Descargando video de YouTube...")
yt = pytube.YouTube(args["link"])
try:
        yt.streams.get_by_resolution(args["resolution"]).download()  #Lo guarda en el mismo directorio que el
programa, con yt.title tengo el titulo
#Lanzamos una excepcion puesto que con algunos videos falla obtener el video filtrando la resolucion
except:
        print("[INFO] Descargando del primer stream")
        stream = yt.streams.first()
        stream.download()

#proceso para extraer el clip concreto del video
print("[INFO] Extrayendo clip indicado...")
try:
        ffmpeg_extract_subclip("{}.mp4".format(yt.title), args["inicio"], args["final"],
targetname="{}.mp4".format(args["name"]))
except:
        ffmpeg_extract_subclip("descarga.mp4", args["inicio"], args["final"],
targetname="{}.mp4".format(args["name"]))
#Para listar todos los stream del video
#lstst=yt.streams.all()
#for st in lstst:
#    print(st)

#Finalmente eliminamos el video completo de la carpeta
remove("{}.mp4".format(yt.title))
```

REFRERENCIAS

[1] Visión artificial. (2020, 20 octubre). En Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial#:%7E:text=La%20visi%C3%B3n%20artificial%2C%20tambi%C3%A9n%20conocida,simb%C3%B3lica%20para%20que%20puedan%20ser

[2] Roberts, Lawrence. (1963). Machine Perception of Three-Dimensional Solids.

[3] https://opencv.org/about/

[4] A. (2019, 17 septiembre). ¿Cómo funciona el ojo de halcón en tenis? Ertheo, education and sports. https://www.ertheo.com/blog/ojo-halcon-tenis-2/

[5] I. (2020, 1 abril). Del «replay» al control de «hooligans»: la visión artificial en el deporte. [R]evolución artificial. https://blog.infaimon.com/del-replay-al-control-de-hooligans-la-vision-artificial-en-el-deporte/

[6] Rosebrock, A. (2020, 18 abril). Faster video file FPS with cv2.VideoCapture and OpenCV. PyImageSearch. https://www.pyimagesearch.com/2017/02/06/faster-video-file-fps-with-cv2-videocapture-and-opencv/

[7] M. M. Naushad Ali, M. Abdullah-Al-Wadud and Seok-Lyong Lee. An Efficient Algorithm for Detection of Soccer Ball and Players. Department of Industrial and Management Engineering Hankuk University of Foreign Studies, REPUBLIC of KOREA

[9] Cem Direkoglu. Melike Sah. Noel E. O'Connor. (2017, 5 December). Player detection in field sports. Springer Nature.

[10] Robert Bodor, Bennett Jackson, Nikolaos Papanikolopoulos. Vision-Based Human Tracking and Activity Recognition. Dept. of Computer Science and Engineering, University of Minnesota.

[11] G. (2020a, enero 30). Filtro de Kalman: Deducción y... Lab. Gluón. https://www.laboratoriogluon.com/filtro-de-kalman-deduccion-ejemplos/

[12] pytube — pytube 9.6.4 documentation. (s. f.). pytube. Recuperado 28 de octubre de 2020, de https://python-pytube.readthedocs.io/en/latest/

[13] https://docs.python.org/3/library/argparse.html

[14] moviepy. (2020, 7 mayo). PyPI. https://pypi.org/project/moviepy/

[15] J.M.F. Rodrigues, P.J.S. Cardoso, T. Vilas, S. Bruno, P. Rodrigues, A. Belguinha, C. Gomes. A computer visión based web application for tracking soccer players. University of the Algarve.