
# Detector de movimiento y seguimiento de jugadores en el campo con Opencv.

*Football players tracking with Opencv.*

Diego Alexander Chanto García

La Laguna, 8 de septiembre de 2015

D. **Jesús Miguel Torres Jorge**, con N.I.F. 43.826.207-Y profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática de la Universidad de La Laguna, como tutor

D. **Jose Demetrio Piñeiro Vera**, con N.I.F. 43.774.048-B profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

*"Detector de movimiento y seguimiento de jugadores en el campo con Opencv."*

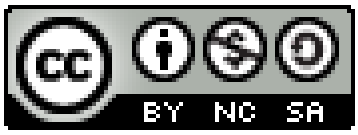
ha sido realizada bajo su dirección por D. **Diego Alexander Chanto García**, con N.I.F. 79.072.856-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de septiembre de 2015.

# Agradecimientos

Quería agradecer a mi tutor Jesús Miguel Torres Jorge por su apoyo durante la elaboración de este proyecto.
También dar las gracias a Alberto Hamilton Castro por su labor en la coordinación de la asignatura "Trabajo Fin de Grado".
A Demetrio Piñeiro por su colaboración en los días previos a la entrega de este proyecto.
Por último a mi novia y a mi familia por su apoyo durante el desarrollo de este proyecto y de mi progreso académico en general.

# Licencia



## Resumen

*El objetivo de este trabajo ha sido la creación de un sistema capaz de realizar un seguimiento de los jugadores, en este caso de fútbol, por el terreno de juego y poder comprobar cuales son las zonas del campo más visitadas por cada jugador para posteriormente realizar un estudio con estos datos. Ya que este sistema podría ser utilizado por los distintos equipos que deseen realizar los estudios con sus jugadores, las fases de preparación y documentación son críticas en el desarrollo. Será necesario realizar pruebas con vídeos de partidos reales para evitar posibles fallos.*



**Palabras clave:** Detección de personas, seguimiento de personas, Opencv, HOG, SURF, SIFT.

**Abstract**

*The main goal of this project has been the development of a football player tracking system that let us know which parts of a football field have been visited the most by any player in the field. With the collected data every team could study how a player moves in the field and using this data change or correct the tendency of movement of any player. It will be necessary testing this system with real videos of a football match with the intention of avoiding problems in the future and make sure that our system works properly.*

**Keywords:** People detection, people tracking, Opencv, HOG, SURF, SIFT.

# Índice General

# Índice de figuras

# Índice de tablas

# Capítulo 1. Introducción

## 1.1 ¿Cómo nace?

Este sistema de seguimiento de jugadores de fútbol en el terreno de juego nace por la necesidad de saber y poder estudiar cuál es la tendencia a moverse por el campo de cada jugador durante un partido de fútbol. Con los datos obtenidos se podría corregir los movimientos de cada jugador para obtener un mayor rendimiento.

## 1.2 Planteamiento del problema

Después de lo introducido en el apartado anterior, se plantea la necesidad de definir muy bien las herramientas que se van a utilizar para el desarrollo de este sistema. Aunque existen proyectos similares sobre seguimiento de personas, no he encontrado ningún sistema capaz de realizar un seguimiento de los jugadores y marcar las zonas visitadas por este para su posterior estudio.

## 1.3 Requisitos

El principal objetivo es que este sistema sea lo más fiable posible para obtener unos resultados precisos.

Será necesario dejar todo bien documentado desde la primera fase del proyecto, para poder realizar cambios futuros sobre el código en caso de querer mejorar el sistema.

Una vez que el sistema este funcionando podrá ser utilizado por cualquier equipo que desee realizar un estudio sobre sus jugadores.

En un futuro a este sistema se le podrá ir añadiendo nuevas características en función de las necesidades de los equipos.

Por tanto, como conclusión inicial, se plantea la creación de un sistema que puede ser utilizado por cualquier persona, en el que sólo será necesario poseer conocimientos sobre el deporte al que se aplica, en este caso el fútbol.

# Capítulo 2.
# Toma de contacto

En el capítulo anterior se ha introducido el planteamiento del problema y las necesidades y requisitos del proyecto. En este capítulo, se abordarán las herramientas utilizadas para el desarrollo del proyecto.

## 2.1 Estado del arte

Comencemos definiendo los conceptos sobre los que se basará este proyecto.

- Detección de movimiento: Consiste en la utilización de una serie de sensores que responden a un movimiento físico y que se encuentran generalmente en sistemas de seguridad o en circuitos cerrados de televisión por ejemplo. Existen diferentes aplicaciones para un sensor de movimiento: seguridad, entretenimiento, iluminación, etcétera.
- Detección de personas: Consiste en llevar a cabo la utilización de una serie de algoritmos que mediante la introducción de una serie de parámetros permitan distinguir en una imagen entre una persona y un objeto cualquiera.

Centraremos nuestra investigación pues, en estos dos conceptos. Ya que para poder llevar a cabo el desarrollo del proyecto debemos comprenderlos.

Todo sistema de detección de movimiento necesitará de los siguientes elementos:

- Un sensor.
- Algoritmos para detectar movimiento.
- Movimiento.

En cuanto al sensor, éste será en nuestro caso la cámara de nuestro portátil pero que podrán utilizarse otro tipo de cámaras para detectar movimiento en tiempo real o incluso utilizar vídeos para realizar la detección del movimiento.

Los algoritmos utilizados para la detección de movimiento han sido "*BackgroundSubstractorMOG*", "*BackgroundSubstractorMOG2*", "*SURF*", "*SIFT*" que serán explicados en profundidad en los próximos capítulos.

Por último no podemos olvidarnos del movimiento ya que sin la existencia del movimiento en nuestras imágenes estas técnicas no tendrían ninguna utilidad. En nuestro caso el movimiento que realmente nos interesa es el de las personas para poder realizar el seguimiento sobre ellas.

## 2.2 Reuniones preliminares

Una vez que el proyecto se encontró completamente definido, comenzaron las reuniones con el tutor. En ellas se planteó tanto la forma de trabajo como las herramientas necesarias para que el proyecto saliese adelante.

En estos primeros encuentros se puso de manifiesto una serie de técnicas utilizadas en la planificación del desarrollo software. Son el "*Brainstorming*" y el "Juicio de Expertos". Se empiezan a presentar una serie de propuestas y a descartar otras para llevar a cabo el proyecto, e incluso se llegan a descartar aquellas herramientas que no son capaces de satisfacer nuestras necesidades.

## 2.3 PCL

La primera de las propuestas es la de utilizar una herramienta de software libre para el procesado de imágenes 2D/3D.

- PCL: Se corresponde con las siglas en inglés "*Point Clouds Library*" (en español, Librería de Nubes de Puntos). PCL es un proyecto Open Source para el procesamiento de imágenes 2D/3D y nubes de puntos. Posee numerosos algoritmos del "Estado del Arte" que pueden ser utilizados por ejemplo para conseguir la unión de los puntos en una nube de puntos 3D. PCL está desarrollada bajo los términos de la licencia BSD, es decir, es gratuito tanto para uso un uso comercial como para un uso más centrado en la investigación.

- Ventajas:
  - ❖ Permite realizar estimaciones entre imágenes 2D/2D, imágenes 2D/3D e imágenes 3D/3D.
  - ❖ Posee una gran modularidad.

## 2.4 OPENCV

En este apartado encontramos la segunda propuesta para el desarrollo de nuestro proyecto.

- OPENCV: Se trata de una biblioteca de visión artificial originalmente desarrollada por intel. Desde su primera aparición en el año 1999, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Su uso extendido se debe ya que al igual que PCL está desarrollada bajo la licencia BSD.
  - Ventajas:
    - ❖ Proporciona un entorno de desarrollo fácil de utilizar y altamente eficiente.
    - ❖ Aprovecha la capacidad que proveen los procesadores multi núcleo.
    - ❖ Es multiplataforma con versiones para GNU/LINUX, Mac OS X y Windows.

Debido a que contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras o visión robótica, ésta ha sido la herramienta por la cual nos hemos decantado para llevar a cabo el desarrollo de este proyecto.

# Capítulo 3.
# Desarrollo de la aplicación

En este capítulo, se entrará en detalle acerca del proceso que se siguió para realizar este sistema.

## 3.1 Herramienta y técnicas utilizadas

Como ya se ha dicho en el apartado anterior la herramienta por la cual nos hemos decantado para la realización de este proyecto ha sido OPENCV.

Entre las técnicas utilizadas para la detección de movimiento encontramos:

- **Background subtraction (BS):** En español, substracción de fondo, se trata de una técnica que sirve para crear una máscara que contiene una imagen binaria con los pixeles que pertenecen a un objeto que se encuentra en movimiento en nuestra escena por medio del uso de una serie de cámaras estáticas.

  Como su propio nombre indica, BS calcula una máscara del primer plano mediante la substracción del fondo entre el *frame* actual y un modelo de fondo que contiene la parte estática de la escena, o en general, todo aquello que pueda ser considerado como fondo, tal y como se puede observar en la siguiente imagen.

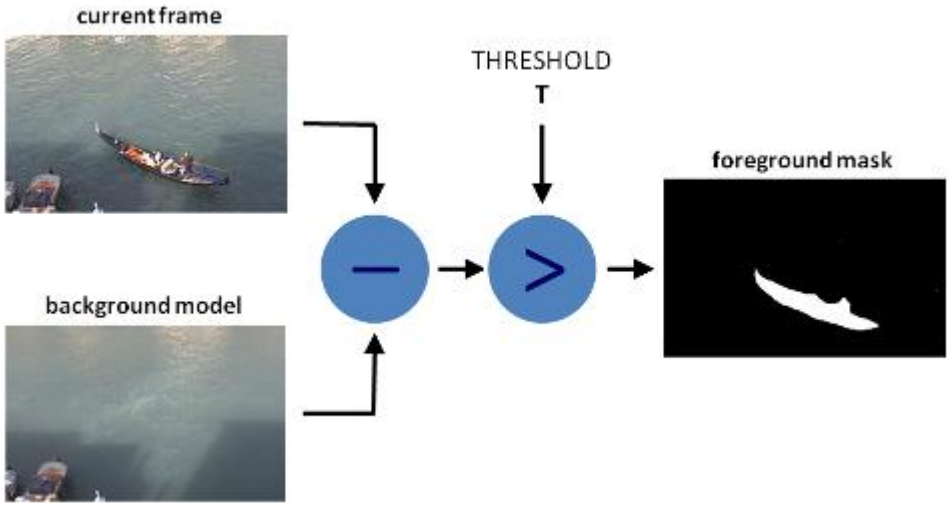



Figura 3.1: Funcionamiento de Background subtraction.

Este modelo consta de dos pasos principales:

- Inicialización: Aquí un modelo inicial del fondo de la imagen es gestionado.
- Actualización del fondo de la imagen: En este paso este modelo es actualizado con la finalidad de adaptarlo a los posibles cambios que se puedan producir en la escena.

- **Feature Detection and Description:** Incluye una serie de métodos que tienen como objetivo la extracción de información de las imágenes y la toma de decisiones en cada punto de las mismas si se encuentran una serie de características de un tipo dado en ese momento. Esta serie de características suelen representarse en forma de puntos aislados, curvas continuas o regiones conectadas.

  No existe una definición universal de qué define una característica(*feature*), esa definición exacta se define en función del problema o del tipo de aplicación. Dado que, una característica es una "parte de interés" en una imagen.

  "*Feature detection*" realiza operaciones de procesamiento de imágenes en bajo nivel. Normalmente es una de las primeras operaciones que se le aplica a una imagen y examina cada pixel para comprobar si encuentra alguna característica relevante en uno o varios pixeles determinados.

  Por lo tanto, la propiedad deseable para un detector de características es la repetitividad para poder encontrar las mismas características entre dos imágenes distintas en la misma escena.

  Entre las características que se buscan en una imagen encontramos:

  - Bordes: En la práctica se define a los bordes como un conjunto de puntos en una imagen que poseen un fuerte gradiente de magnitud. Lo que nos permite poder detectar cual sería la forma del objeto en la imagen.
  - Esquinas/Puntos de interés: Los términos esquinas y puntos de interés se utilizan como sinónimos y se refieren a las características puntuales en una imagen, que tienen una estructura de dos dimensiones.

- Blobs/Regiones de interés: En español manchas, y proporcionan una descripción complementaria de las estructuras de la imagen en términos de regiones, a diferencia de las esquinas que son más de punto similares.

| Feature detector | Edges | Corner | Blob |
|---|---|---|---|
| CANNY | X | | |
| SOBEL | X | | |
| KAYYALI | X | | |
| HARRIS & STEPHENS | X | X | |
| SUSAN | X | X | |
| SHI & TOMASI | | X | |
| LEVEL CURVE CURVATURE | | X | |
| FAST | | X | X |
| LAPLACIAN OF GAUSSIANS | | X | X |
| DETERMINANT OF HESSIAN | | X | X |
| MSER | | | X |
| PCBR | | | X |
| GRAY-LEVEL BLOBS | | | X |

Tabla 3.1: Clasificación de los detectores de características.

De entre todos los detectores de características que se encuentran implementados actualmente, en los que me he centrado para realizar las pruebas han sido:

- *SURF(Speed Up Robust Features)*
- *SIFT(Scale-Invariant Feature Transform)*
- *ORB(Oriented FAST and Rotated BRIEF)*

- **Matching:** El *matching* permite saber cuáles son los puntos que coinciden entre una imagen y la siguiente y de esta forma poder dibujarlos. Para realizar esta tarea encontramos:

- BFMatcher: Permite encontrar aquellos puntos iguales en una imagen por fuerza bruta. Lo que hace es tomar un punto dentro de la primera imagen y compararlo con todos los puntos de la otra imagen.
- FLANN(Fast Library for Approximate Nearest Neighbors): Se trata de una biblioteca que permite realizar búsquedas rápidas de los vecinos más cercanos. Contiene una colección de algoritmos que permiten decidir cuál es el mejor vecino más cercano y un sistema para elegir automáticamente el mejor algoritmo y los parámetros más óptimos en función del conjunto de datos.

  En los estudios realizados se ha encontrado que FLANN es más rápido que las otras librerías desarrolladas hasta la fecha.

- Meanshift.

## 3.2 Desarrollo

Para llevar a cabo el desarrollo de este proyecto han sido necesarios una serie de pasos:

- Cargar un vídeo o capturar las imágenes por medio de un sensor (cámara)
- Realizar la supresión de fondo
- Utilizar los detectores de características y realizar el *matching*
- *Blob detection*
- Uso de *HOGDescriptor* para realizar el tracking

Cada uno de estos puntos será desarrollado en los siguientes apartados.

## 3.3 Cargar vídeo

Para llevar a cabo esta tarea se ha utilizado la clase `VideoCapture`, que permite realizar capturas desde archivos de vídeo o desde una cámara. Esta clase provee una API en C++ para poder realizar estas capturas.

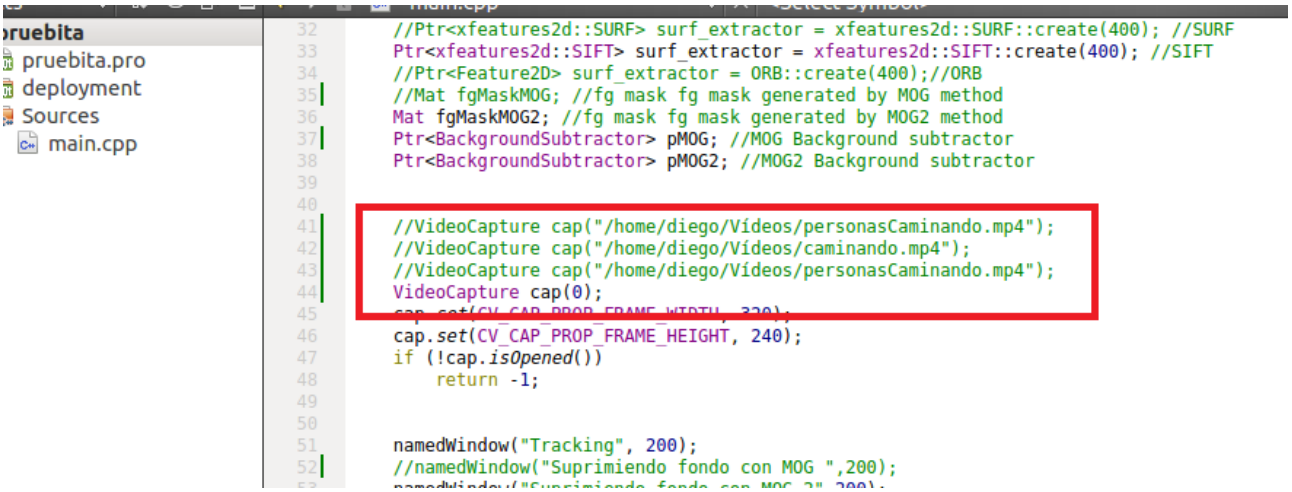



Figura 3.2: VideoCapture

En la imagen anterior observamos las dos formas distintas de capturar el vídeo. Por una parte se le puede pasar la ruta del vídeo y por otra se puede indicar el índice de la cámara de la cual deseamos realizar la captura, en este caso el índice es el 0.

## 3.4 Supresión de fondo

Una vez que hemos cargado nuestro vídeo procederemos a ir guardando tanto el *frame* actual como el *frame* anterior para realizar las operaciones pertinentes, en el caso de la supresión de fondo simplemente nos hará falta ir analizando *frame* a *frame* el vídeo.

Como hemos indicado en el apartado anterior, para la supresión del fondo utilizamos "*BackgroundSubtractor*", después de haber probado con "`BackgroundSubtractorMOG()`" y "`BackgroundSubtractorMOG2()`" me he decantado por la utilización de la segunda técnica ya que devolvió unos resultados más fiables a la hora de mostrar aquellos elementos que se encontraban en movimiento.

La primera opción devolvió imágenes en las que casi no se apreciaba cuales eran aquellos objetos que se encontraban en movimiento. Aunque la segunda opción si mostraba los movimientos, hizo falta realizar algunos ajustes ya que había ciertos objetos que no se encontraban en movimiento y sin embargo los mostraba.

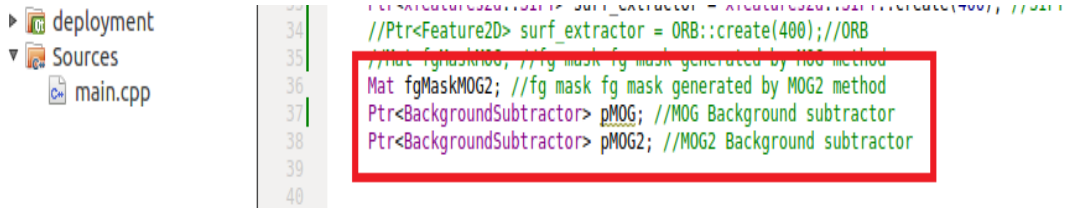



Figura 3.3: Declaración de la máscara y la variable del *BSMOG2*

Con la variable pMOG2 podremos empezar a realizar la substracción de fondo y posteriormente mostrar los resultados.

```
namedWindow("keypoints",200);

//pMOG =  createBackgroundSubtractorMOG(); //MOG approach
pMOG2 = createBackgroundSubtractorMOG2(); //MOG2 approach

while (true)
{
    cap >> frame;
    if (!frame.data)
```

Figura 3.4: Enfoque *MOG* y *MOG2*.

En la siguiente imagen se puede observar como en la imagen a pesar de no haber movimiento en algunos de los objetos, estos aparecen en la imagen de supresión de fondo.


Figura 3.5: Supresión de fondo con MOG2

Se puede observar en la imagen de la derecha (marcado con rojo) como aparece el ventilador y sin embargo éste no se está moviendo.

Para solucionar este problema se ha utilizado una función morfológica que entre las acciones que puede realizar permite erosionar o dilatar la imagen, por ejemplo.

La función utilizada en nuestro caso ha sido `Erode()`, la idea básica de la erosión podríamos compararla con la erosión del suelo, por ejemplo, ya que erosiona los límites del objeto en primer plano intentando mantener siempre la imagen del primer plano (aquella que está en movimiento) en blanco, mientras que el fondo se mantiene de color negro.

Con esta función, lo que pasa es que todos los pixeles cerca de los límites de la imagen en movimiento son descartados, reduciendo el tamaño o el grosor del objeto que se encuentra en primer plano (disminuye el blanco en la imagen). Por tanto, es útil para eliminar el ruido o separar dos objetos en movimiento que se encuentran conectados, etcétera.

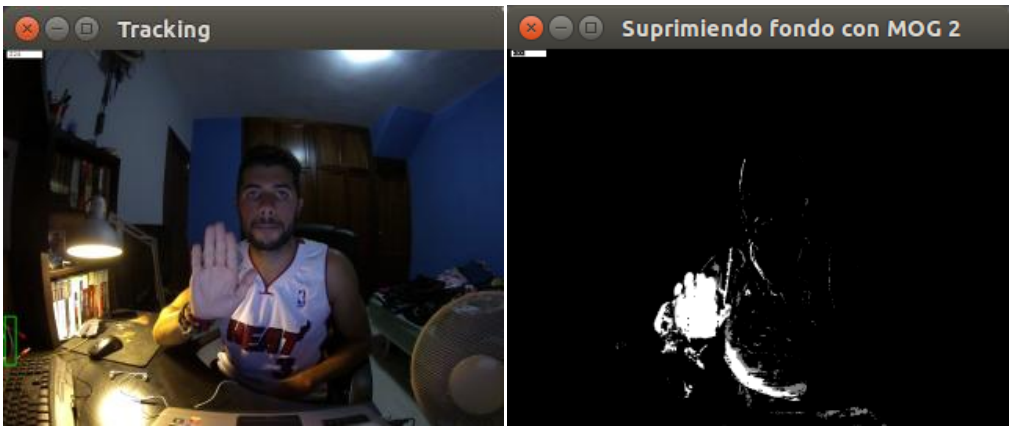



Figura 3.6: Supresión de fondo con *MOG2* aplicando `erode()`

Como se puede observar en la imagen anterior una vez que hemos utilizado `erode()` la reducción de ruido es considerable. Sobretodo podemos observar como el ventilador desaparece y el ruido presente en la zona de la cara disminuye.

Otra forma de ver los resultados para la supresión de fondo sería aplicando la función `findContours()` cuyo objetivo es el de encontrar los contornos en una imagen, es decir, como resultado solo se mostrarían los contornos del objeto en movimiento.

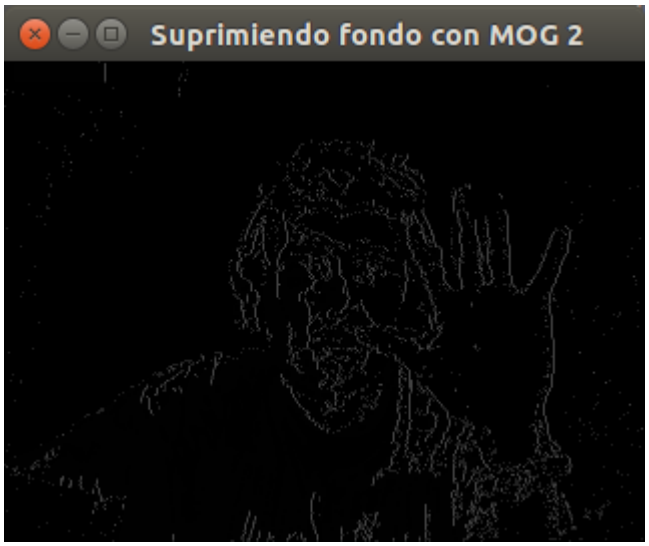



Figura 3.7: Supresión de fondo y contornos

## 3.5 Extracción de blobs

Para empezar explicaremos el significado de lo que es un "*Blob*". Un blob es un grupo de pixeles en una imagen que tiene una propiedad común, por ejemplo valores dentro de una escala de grises. El objetivo de la técnica de extracción de blobs es la de identificar y marcar estas regiones que poseen dicha propiedad.

En este punto ya tenemos una serie de pixeles catalogados como de primer plano y que suelen agruparse en regiones que se corresponden con los objetos que se encuentran en movimiento en el *frame* original. La extracción de blobs nos permitirá, por tanto, identificar y marcar estas regiones que poseen dicha propiedad con un círculo o un cuadrado en el *frame* original.

### 3.5.1 ¿Cómo funciona la extracción de blobs?

Esta técnica está basada en un algoritmo que se encuentra controlado por una serie de parámetros. Dichos parámetros son los siguientes:

- Color: Dependiendo de la selección que hagamos filtrará por colores oscuros (0) o por colores claros (255).
- Tamaño: Se filtrarán los blobs en función del tamaño, debemos proporcionar el valor para el mínimo y el máximo valor del área en el que filtraremos, por ejemplo si ponemos el valor de minArea = 100 se filtrará todos los blobs que tengan menos de 100 pixeles.
- Forma: Ahora encontramos que el filtrado por medio de la forma contiene tres parámetros distintos:
    - Circularidad: Medirá qué tan cerca de un circulo se encuentra el blob.
    - Convexidad: Buscará aquellas formas que sean lo más convexas posible.
    - "Inercia del radio": Lo que viene a medir es cuán elongada la forma del objeto es. Por ejemplo para encontrar algún objeto con forma de elipse.

### 3.5.2 ¿Cómo configurar los parámetros?

Estos parámetros se pueden configurar de una manera muy sencilla, simplemente indicamos cuales son aquellos que queremos indicar y sus valores. En la siguiente figura podemos observar como configurar estos parámetros.

Una vez que ya tenemos los parámetro configurados podemos hacer el seguimiento de los jugadores, en este caso se ha realizado un filtrado por color detectando así a cada uno de los jugadores que se encuentran en el terreno de juego y al árbitro.


Figura 3.8: *Blobs*.

como se observa en la imagen los jugadores tienen un punto rojo sobre cada uno de ellos y que les sigue se muevan a donde se muevan. También se puede observar como aparecen blobs en la grada. Para solucionar este problema habría que encontrar homografías de tal manera que se toman las cuatro esquinas del campo y sólo buscar dentro de ese perímetro.

```
}

Mat H = findHomography( obj, scene, CV_RANSAC );

//-- Get the corners from the image_1 ( the object to be "detected" )
std::vector<Point2f> obj_corners(4);
obj_corners[0] = cvPoint(0,0); obj_corners[1] = cvPoint( img_object.cols, 0 );
obj_corners[2] = cvPoint( img_object.cols, img_object.rows ); obj_corners[3] = cvPoint( 0, img_object.rows );
std::vector<Point2f> scene_corners(4);

perspectiveTransform( obj_corners, scene_corners, H);

//-- Draw lines between the corners (the mapped object in the scene - image_2 )
line( img_matches, scene_corners[0] + Point2f( img_object.cols, 0), scene_corners[1] + Point2f( img_object.cols, 0), Sc
line( img_matches, scene_corners[1] + Point2f( img_object.cols, 0), scene_corners[2] + Point2f( img_object.cols, 0), Sc
line( img_matches, scene_corners[2] + Point2f( img_object.cols, 0), scene_corners[3] + Point2f( img_object.cols, 0), Sc
line( img_matches, scene_corners[3] + Point2f( img_object.cols, 0), scene_corners[0] + Point2f( img_object.cols, 0), Sc
```

Figura 3.9: Búsqueda de homografías.

## 3.6 Features Detection and Description

Los detectores de características que se han utilizado en este proyecto son los siguientes:

- *SIFT*

- *SURF*
- *ORB*

### 3.6.1 SIFT(Scale-Invariant Feature Transform)

Se trata de un algoritmo que permite obtener una representación visual de una imagen y extraer información detallada y específica de esta imagen. En cuanto a su sucesor (*SURF*) es un poco más lento en términos de procesamiento y *matching* de imágenes. A pesar de ser más lento que *SURF* permite que se reduzca en mayor medida el radio de búsqueda de los puntos característicos entre las imágenes.

Figura 3.10: SIFT BFMatcher

### 3.6.2 SURF(Speed Up Robust Features)

Se trata de un algoritmo que permite obtener una representación visual de una imagen y extraer información detallada y específica de esta imagen. Esta información es tratada para realizar una serie de operaciones como podría ser, por ejemplo, reconocimiento de objetos o personas.

*SURF* es un detector de características de alto rendimiento. Este algoritmo está basado en su predecesor *SIFT*, aunque utiliza un esquema diferente lo que permite obtener mejores resultados como podría ser una mayor rapidez en el *matching* y en el procesamiento.

Con *SURF* se intenta calcular todos los vecinos, una vez calculados se estima cuál es la orientación dominante calculando la suma de todos los resultados dentro de una ventana deslizante que abarca un ángulo de pi/3.

*SURF* utiliza "respuesta de ondas" para buscar tanto vertical como horizontalmente a los vecinos dentro de una región determinada con un tamaño de 20sX20s, donde se toman todos los puntos característicos donde "s" es el tamaño. Se realiza una división de 4x4 subregiones y se toman las respuestas de onda tanto horizontal como verticalmente y se forma un vector como el siguiente:

$$v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|).$$

Figura 3.11: Vector de "Respuestas de onda".

En la siguiente imagen podemos hacernos una idea de cómo es la ventana deslizante que se comentó antes.

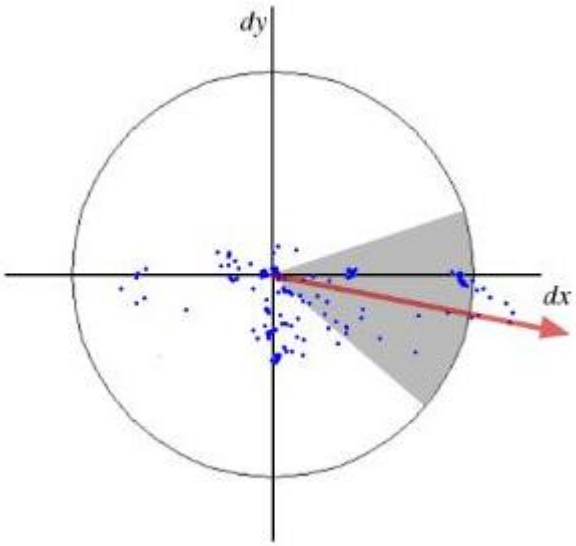



Figura 3.12: Ventana deslizante

Se han realizado pruebas con este algoritmo y se han obtenido varios resultados. Utilizando *SURF* y `BFMatcher`*(Brute Force Matcher)* se obtienen demasiados puntos característicos (*keypoints*) entre la primera imagen y la segunda.

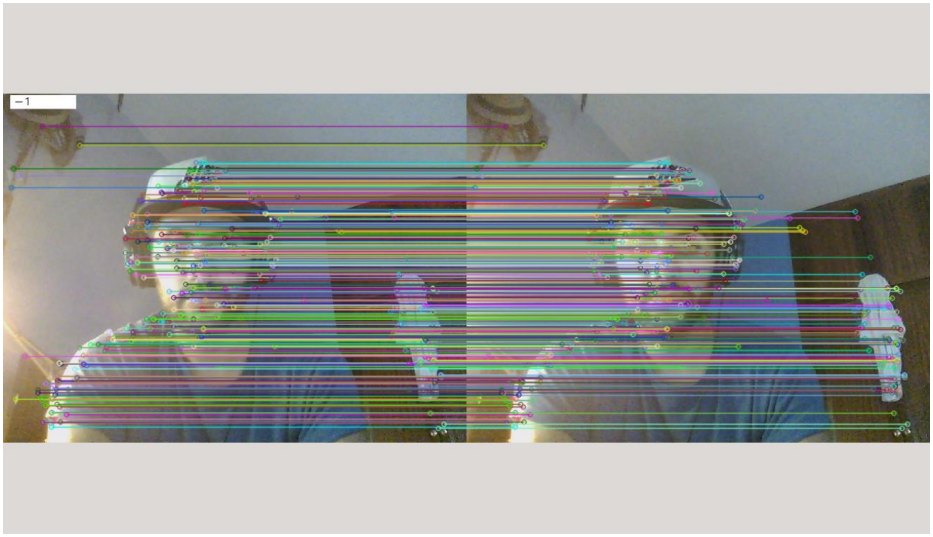


Figura 3.13: SURF y BFMatcher.

Para reducir la cantidad de puntos característicos que encuentra SURF, se ha calculado la mínima y la máxima distancia entre las que se encontrarán estos puntos dibujando solamente aquellas coincidencias que cataloguemos como buenas.

Aunque se debe prestar especial atención, ya que en caso de realizar un mal cálculo de los puntos característicos podemos obtener resultados como el que se muestra en la siguiente imagen, en los que se hace un *matching* de puntos que no tienen nada en común.

Figura 3.14: Mala aplicación SURF

### 3.6.3 ORB (Oriented FAST and Rotated BRIEF)

Una vez que se han realizado las pruebas con SIFT y SURF, se observa que mediante el cálculo de la mínima y la máxima distancia entre los puntos característicos los resultados que se obtienen son notablemente mejor que los dos algoritmos utilizados anteriormente cuando aplicamos las mismas técnicas de *matching*, tanto `BFMatcher` como `FLANNMatcher`. En la imagen que aparece a continuación se observa como la cantidad de puntos característicos se reduce notablemente con respecto a los algoritmos anteriores.

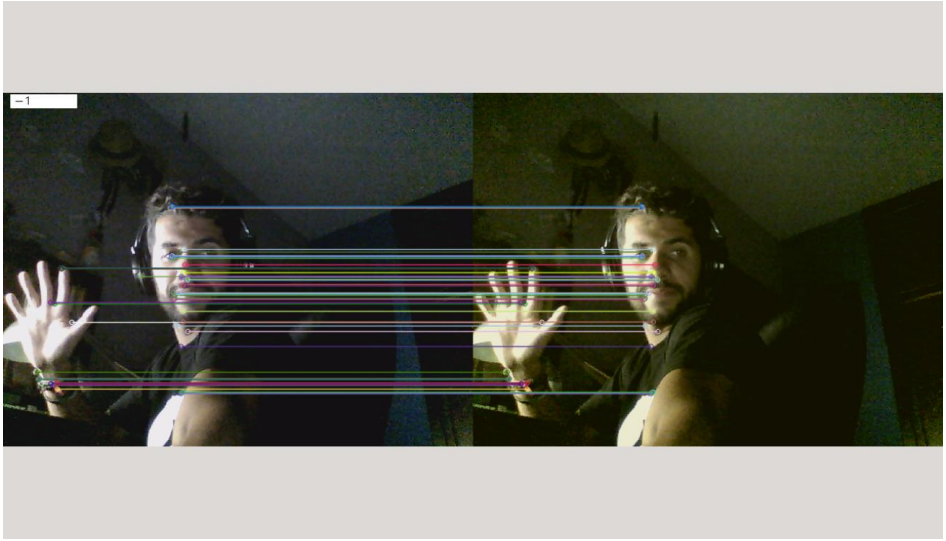


Figura 3.15: Aplicación de *ORB*.

Por tanto, como *ORB* nos permite realizar una mejor búsqueda los puntos característicos entre dos imágenes, sobre todo entre los objetos que se encuentran en movimiento, éste ha sido el algoritmo utilizado como detector de características.

## 3.7 Histograms of Oriented Gradients

Uno de los detectores de personas más populares y exitosos actualmente sería el *HOG* (*Histograms of Oriented Gradients*). Se trata de una especie detector de características. La finalidad de este algoritmo es el de realizar la detección de personas mediante el uso de una ventana deslizante que se mueve a través de la imagen, previamente para realizar dicha detección se ha "entrenado" a este algoritmo para que sepa diferenciar entre una persona y un objeto.

La ventana deslizante que utiliza HOG tiene un tamaño de 64 pixeles de ancho y 128 pixeles de alto.

En la siguiente imagen podemos observar el funcionamiento de este algoritmo.


Figura 3.16: Funcionamiento HOG.

## 3.8 Meanshift

En último lugar hemos un algoritmo conocido como *meanshift* que consiste en considerar una serie de puntos, se crea una ventana (rectángulo) y se debe mover dicha ventana hasta la zona que contenga una mayor densidad de pixeles. Una vez que nos encontramos en esa zona procedemos a calcular el centroide de la ventana que hemos pintado en el *frame* actual. Si el objeto se mueve procederemos a calcular nuevamente el centroide en la posición en la que se encuentre actualmente el objeto y así sucesivamente una vez que ya tengamos centroides calculados procederemos a unirlos mediante una línea por ejemplo, tomando como referencia inicial el primer centroide de que aparece y como referencia final el último que aparezca, de tal manera que se pueda dibuje la trayectoria que sigue el objeto en movimiento. En la siguiente imagen se puede observar el resultado.

# Capítulo 4. Presupuesto

Este proyecto no ha tenido ningún gasto en cuanto a materiales físicos(*hardware*) se refiere. Por otro lado, no han surgido gastos de licencias o *sotfware*.

Aun así mediremos el coste total del proyecto en función de las horas empleadas en el desarrollo.

El desarrollo del sistema ha sido realizado en su totalidad por el alumno de manera autónoma, con lo cual no haría falta añadir gastos en desplazamiento por ejemplo.

## 4.1 Tabla

| Concepto | Precio |
|---|---|
| Horas de trabajo | 260x10€= 2500€ |
| TOTAL | 2500€ |

Tabla 4.1. Tabla de presupuestos.

# Capítulo 5.
# Conclusiones y líneas futuras

A grandes rasgos se ha realizado un proyecto que intenta utilizar técnicas ya existentes para aportar una solución a un problema que actualmente no la tiene. Uno de los puntos más interesantes de este proyecto podría ser su aplicación en el mundo real.

Como anécdota, este proyecto surgió debido a que a pesar de existir muchos programas deportivos en la televisión, en especial de fútbol, en los que se estudia minuto a minuto cada una de las acciones del partido, mostrando también las estadísticas de cada uno de los jugadores, entre las que podemos encontrar la distancia recorrida durante el tiempo que ha estado sobre el terreno de juego, los pases que ha realizado con éxito, entre otras. Sin embargo la ausencia de un gráfico por ejemplo que muestre cuales son las zonas más visitadas por un jugador durante el partido, impulsó el desarrollo de este proyecto.

Por otra parte, este proyecto me ha servido para utilizar y aprender un tipo de técnicas que durante el tiempo que estuve en la carrera no tuve la oportunidad de utilizar en profundidad.

Aunque al principio del proyecto se presentaron algunas dificultades, en relación a la instalación de ciertas librerías e incompatibilidades con el sistema operativo con el que empecé a trabajar, una vez que fueron resueltas el proyecto comenzó a tomar forma.

Las pruebas que se han realizado hasta la fecha han proporcionado buenos resultados aunque todavía es necesario utilizar estas técnicas con imágenes reales de un partido de fútbol ya que no se ha encontrado un vídeo con las condiciones necesarias para aplicarlas.

## 5.1 Trabajos futuros

A pesar de que el sistema está casi terminado, se pueden plantear una serie de mejoras para completar su funcionalidad en el futuro.

### 5.1.1 Medición distancia recorrida y pases completados

Se podría añadir un sistema que mida la distancia que ha recorrido cada uno de los jugadores y los pases que han realizado durante todo el partido teniendo aquellos completados con éxito y aquellos fallados.

### 5.1.2 Detección del balón.

Como ya se ha realizado la detección de los jugadores, otra mejor podría ser la de realizar la detección y seguimiento del balón, de tal forma que lo marcaríamos de un color cuando se encuentre dentro del campo y otros colores dependiendo de la situación que se presente, si pasa la línea de fondo, si traspasa la línea de la portería (gol), o si sale por la línea de banda evitando así que se produzcan jugadas polémicas en los partidos.

## 5.2 Mantenimiento

Este sistema no necesita de un mantenimiento continuo. Simplemente se podrían aplicar acciones de mantenimiento en caso de que se desarrollen nuevas técnicas de detección de movimiento por ejemplo que ofrezcan unos mejores resultados de los que se han obtenido hasta la fecha.

# Capítulo 6.
# Summary and Conclusions

If we take a look of this project, we can see how some techniques that already exist to solve a problem that does not have yet a solution. One of the most interesting things of this project is its application in the real life.

As anecdote, this project was born because in many sport programs on tv show people a lot of statistics as number of successful passes, the distance that a player has done in the time he has been on the field. However I missed a system able to follow players on the field and draw their trajectory just to show which are the most visited parts of the field.

This project made me know and uses some techniques that I had never used.

At the first of this project some problems with libraries and the operative system that I was using appeared. Once this problems were solved I could start the development of it.

Some tests have been made and we have get very good results but it is necessary to use a video of a soccer game in good conditions to apply the techniques used.

## 6.1 Future work

Despite this system is almost done, we can make some changes in the future to improve its functionality in the future.

### 6.1.1 Distance measure and successful passes

A system that measures the distance that a player has made during the game or a system that counts the successful or missed passes made can be included.

### 6.1.2 Ball detection

We have already made a system that tracks players during a soccer match, so we could include a ball detection system that tracks the ball and let us know when the ball go in the goal or when it goes out of the field by using colors depending on the situation.

## 6.2 Maintenance

This system does not require an everyday maintenance. We can apply maintenance actions just in case some new techniques are developed and show better results than the techniques use so far.

# Capítulo 7. Bibliografía


[1] http://opencv.org/

[2] http://docs.opencv.org/modules/nonfree/doc/feature_detection.html?highlight=surf

[3] http://docs.opencv.org/modules/features2d/doc/feature_detection_and_description.html?highlight=orb

[4] http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html?highlight=flannbasedmatcher#flannbasedmatcher

[5] http://docs.opencv.org/modules/core/doc/drawing_functions.html#line

[6] http://docs.opencv.org/3.0-beta/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html

[7] http://www.cs.ubc.ca/research/flann/

[8] http://docs.opencv.org/modules/gpu/doc/object_detection.html

[9] http://docs.opencv.org/3.0-beta/modules/video/doc/motion_analysis_and_object_tracking.html#meanshift

[10] http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html?highlight=flannbasedmatcher#bfmatcher-bfmatcher