Universidad de Murcia

Grado en Ingeniería Informática

Curso 2017/2018

Seguimiento visual automatizado de eventos en deportes de equipo

Trabajo Fin de Grado

Autor: Antonio Saavedra Sánchez antonio.saavedra@um.es
Director: Pedro E. Lopez de Teruel Alcolea pedroe@um.es

Índice

Índice de figuras

Declaración de autenticidad

Resumen

Extended Abstract

1. Introducción

1.1. ¿Qué es la visión artificial?

Los seres humanos somos capaces de percibir con facilidad el entorno tridimensional que nos rodea. Durante décadas, el funcionamiento de nuestra visión ha sido fruto de extensivo estudio por parte de psicólogos de la percepción, pero aún no se conoce el funcionamiento de manera exacta [1].

Paralelamente, los investigadores en visión artificial han desarrollado técnicas matemáticas para, a partir de imágenes, reconstruir la apariencia tridimensional de objetos. Actualmente somos capaces de, entre otras cosas, generar un modelo 3D parcial de un entorno a partir de miles de fotografías solapadas, seguir el movimiento de una persona sobre un fondo complejo y, con moderado éxito, reconocer y nombrar a las personas de una fotografía a partir de características como el pelo, la cara o la ropa. Entre otras aplicaciones más avanzadas se encuentran detectar y clasificar objetos o elaborar descripciones verbales a partir de imágenes o vídeos.

Sin embargo, a pesar de estos avances, hacer que un ordenador sea capaz de interpretar una imagen al mismo nivel que lo haría un niño sigue siendo un sueño sin cumplir. La dificultad del problema que nos ocupa suele ser subestimada, ya que se suele tender a pensar que las partes difíciles de la inteligencia artificial son las cognitivas y no las relativas a la percepción.

Pero, ¿qué es lo que hace tan difícil la visión por computador? En parte es que, dado nuestro desconocimiento del funcionamiento de nuestra visión, es un *problema inverso*, dada una cantidad insuficiente de información (una imagen o un video), tratamos de llegar a las incógnitas que nos permiten especificar una solución completa. Es decir, a partir de una imagen, tratamos de reconstruir sus propiedades, como las formas, iluminación y color.

Ahora bien, no todo son malas noticias: la visión por computador tiene numerosos casos de uso hoy en día. Entre sus aplicaciones encontramos cosas tan mundanas como (Szeliski):

- **Reconocimiento de caracteres óptico:** Se ha llegado a reconocer caracteres numéricos con una tasa de error del 0.23 % [2].
- **Inspección mecanizada:** inspección de partes de automóviles o aeronaves a fin de, por ejemplo, encontrar defectos en el metal (Figura 1a).
- **Construcción de modelos 3D** construcción automatizada de modelos 3D a partir de fotografías aéreas como las que usa Google Maps (Figura 1b).
- **Seguridad en automóviles:** reconocimiento de obstáculos tales como peatones u otros coches, en condiciones en las que el radar o el lidar no funcionan bien, en tecnologías como Mobileye.
- **Reconocimiento de huellas dactilares y biométricas:** Usado ampliamente hoy en día en teléfonos móviles.

(a) (b)

Figura 1. Algunas aplicaciones de la visión artificial: (a) sistema de digitalizado por luz blanca WLS400A. (b) reconstrucción 3D en Google Maps de la Puerta del Sol.

1.2. Breve historia de la visión artificial

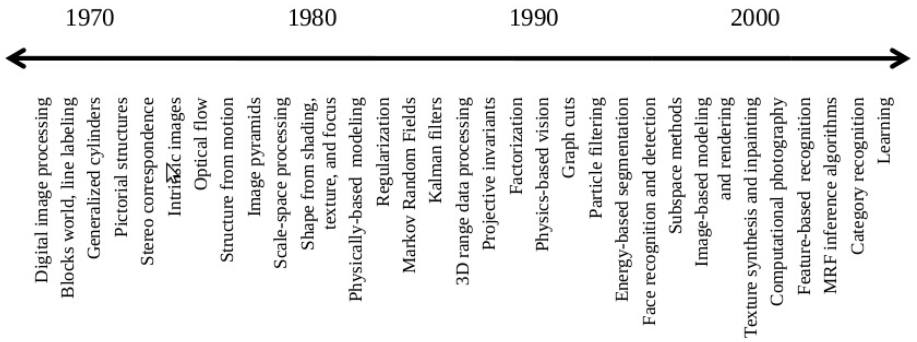


Figura 2. Timeline de las aportaciones más importantes a la visión artificial [1]

La investigación en visión artificial comienza en la **década de los 70**. En esta época, los pioneros en el campo de la IA y la robótica creían que resolver el problema de la entrada visual sería un trabajo fácil hacia la resolución de problemas de mayor envergadura. De hecho, tanto es así, que en 1966 Marvin Minsky le asignó a su alumno en el MIT Gerald Jay Sussman un proyecto de verano que consistía en enlazar una cámara a un ordenador y que este describiera lo que veía.

Lo que distinguió la visión por computador del campo del procesado de imágenes digitales fue la intención de reconstruir la estructura tridimensional del mundo a partir de las imágenes. Los primeros intentos en este sentido estaban relacionados con la extracción de ejes y posterior inferencia de la estructura 3D del objeto.

En la **década de 1980** los esfuerzos se centraron en técnicas matemáticas mas sofisticadas para el análisis. Empiezan a usarse ampliamente técnicas como las pirámides de imágenes, y continuó la investigación en una mejor detección de bordes y contornos. Comienzan a adoptarse técnicas de detección de formas a partir de sombreado, textura, o enfoque. Posteriormente, algunos investigadores vieron que estas técnicas podían ser generalizadas usando el mismo modelo matemático si se planteaban como problemas de optimización. También aparecieron

las primeras redes neuronales aplicadas al campo de la visión por computador

En la **década de 1990** continuó la investigación en todo lo anteriormente mencionado, pero algunos de los temas vieron un aumento en su actividad. Hubo un esfuerzo por solucionar el problema de la estructura a partir del movimiento. El trabajo comenzado en la década anterior consistente en usar medidas detalladas de color e intensidad en combinación con modelos físicos precisos acabó por crear su propio subcampo llamado visión basada en la física.

Los algoritmos de tracking mejoraron, incluidos los de tracking por contorno usando contornos activos, así como los basados en intensidad. Normalmente se aplicaron al tracking de caras o de cuerpos completos.

La segmentación de imagen fue un tema de investigación activo, y produjo técnicas como *mean shift*. Comienzan a aparecer técnicas de aprendizaje basadas en la estadística, de las que surgen los primeros algoritmos de reconocimiento de caras.

El avance más notable en esta década fue la interacción con la computación gráfica. La idea de manipular imágenes del mundo real para crear animaciones tomó forma mediante técnicas de *morphing* y se aplicó después a técnicas de interpolación y creación de imágenes panorámicas. A la vez, empezaron a introducirse técnicas consistentes en crear modelos 3D a partir de colecciones de imágenes.

Durante la **década de los 2000** continuó la interacción entre el campo de la visión y el de los gráficos. Una tendencia de esta década fue la proliferación de técnicas basadas en *features* para el reconocimiento de objetos. Estas técnicas también dominan otras tareas del reconocimiento como el reconocimiento de la escena o de localización.

Otra tendencia, que sigue presente en la actualidad, es la aplicación de técnicas de *machine learning* a problemas de visión artificial, como es el uso de redes neuronales convolucionales. El crecimiento de esta técnica coincide con la gran disponibilidad en Internet de enormes cantidades de datos etiquetados, así como la multiplicación de la capacidad de cómputo, lo que facilitó las tareas de aprendizaje [1].

Al igual que en el campo del *machine learning*, las redes neuronales han cobrado una importancia vital, y se han logrado verdaderamente grandes avances haciendo uso de ellas. Un ejemplo del estado actual del campo es el trabajo de He et al. [3] que muestra el avance en segmentación de imagen que se ha logrado usando ANN.

También hay avances en la conducción autónoma usando visión por computador, de hecho NVIDIA tiene actualmente en funcionamiento una plataforma para coches autónomos llamada NVIDIA DRIVE. Dicha plataforma, nuevamente, hace uso de técnicas de deep learning para detectar obstáculos, señales, etc.

También existe Kinect, que es un dispositivo cuyo propósito es el de estimar la posición y los gestos del usuario mediante cámaras de profundidad. Kinect surgió en 2010 como un controlador de videojuegos, pero se puede usar para otros propósitos.

Figura 3. Imagen promocional de Kinect

Figura 4. Imagen de la instalación de SportVU

2. Estado del arte

2.1. Visión artificial en los deportes

En cuanto a las aplicaciones actuales de la visión por computador en los deportes, cabe comenzar mencionando la que es quizá la más célebre: el ojo de halcón. Este sistema se utiliza en deportes como el tenis, cricket, rugby, volleyball y, recientemente, en fútbol. Tiene origen en 2001, cuando se empezó a utilizar en las retransmisiones televisivas de cricket, y se usaba para hacer seguir las trayectorias de las bolas.

En el caso del tenis, que es el uso mas presente en el imaginario popular, el sistema se implantó por primera vez en un torneo de primer nivel en 2006 en la copa Hopman, y el primer grand-slam en emplearlo fue el Open de Australia de 2007. En estos primeros compases de su utilización, hubo ciertas controversias debido a algunos errores del sistema.

En el fútbol, tras algunas controversias como la mundial de 2010, cuando un *gol fantasma* de Inglaterra no visto por el árbitro no permitió el empate ante Alemania, se implementó en el mundial de clubes de 2012 la llamada *goal-line technology*.

El funcionamiento del sistema es similar en todos los deportes que lo utilizan: consiste en una serie de cámaras (10, en tenis) que toman imágenes de distintos ángulos del area de juego, y usando principios de triangulación, se calcula la localización actual de la pelota y se 'predice' su posición siguiente [4].

Existen otras soluciones que permiten recabar estadísticas del partido, como es el caso de SportVU. El sistema, en el caso del fútbol, consiste en la colocación de 3 (figura 4) o 6

Figura 5. Imagen promocional del sistema Intel 360 Replay

cámaras (en algunos sistemas más), y recaba datos tales como distancia viajada, velocidad media, velocidad máxima, mapas de calor, posesión, etc. Sin embargo, está disponible para más deportes, como baloncesto, fútbol americano, béisbol, hockey de hielo y rugby.

Además podemos encontrar otros sistemas que se utilizan para análisis deportivo y/o ayuda al espectador de TV. Entre estos se encuentra el desarrollado por Intel: Intel 360 Replay. Este sistema ofrece repeticiones en 3D mediante el uso de 38 cámaras de resolución 5K. Se lleva usando durante unos años en la NBA y la MLB y recientemente en fútbol, siendo el clásico de la temporada pasada uno de los primeros grandes partidos en usarlo.

En deportes en los que el fondo es muy uniforme, como pueden ser los que emplean pistas de césped, se puede emplear para implementar un efecto de croma en la pista. Esto permite hacer dibujos en el campo de juego a fin de mostrar jugadas.

Uno de las áreas de investigación actuales es la de automatizar el control de cámaras durante las retransmisiones deportivas. Esta tarea es complicada ya que es un doble trabajo, requiere un tracking de la acción (que puede ser muy rápida por momentos) y la búsqueda de un buen plano que ayude al espectador a entender qué está pasando en el partido. Esta tecnología está aun muy lejos de ser una realidad en un futuro cercano. No obstante, un acercamiento es el de que un realizador para controle una cámara principal que es capaz de determinar la posición a la que apunta, y una serie de cámaras 'esclavas' apunten a dicha posición, de manera que se pueda ver la acción desde distintos puntos de vista. [5]

3. Análisis de objetivos y metodología

3.1. Objetivos del proyecto

En este trabajo se tratará de, usando imágenes de una cámara en vista cenital, hacer un sistema capaz de detectar y seguir (mediante *tracking*) tanto los jugadores como balón de un partido de volleyball.

Una vez hecho esto, las posiciones de los jugadores y del balón deberían quedar registrados en un archivo CSV, divididas por jugadas y sets. Para hacer esta división, lo ideal es que funcione de forma automática, aunque lo podría hacer un operario mediante pulsaciones de

teclas.

El proyecto parte suponiendo que la cámara mencionada ya ha sido instalada. Por tanto, los detalles de la instalación de la parte hardware y la extracción de imágenes del campo de juego quedan fuera del alcance del trabajo.

3.2. Herramientas utilizadas y metodología

Durante la realización de este trabajo se han utilizado una serie de herramientas que pasaré a listar y detallar a continuación.

Python

Python es un lenguaje de programación interpretado, multiparadigma y de tipado dinámico. Nace a finales de los 80, pero alcanzó una mayor popularidad a partir de mediados de los 2000, poco después de la versión 2.0 del lenguaje. En la actualidad, Python es uno de los lenguajes más utilizados en materia de procesado científico y *machine learning*.

La totalidad del desarrollo del software de este trabajo se realizará en este lenguaje. Se ha tomado esta decisión debido a la gran sencillez que aporta, en conjunto con el hecho de que es multiplataforma. Otra gran ventaja del lenguaje es la disponibilidad de librerías como OpenCV, que será la piedra angular del proyecto.

OpenCV

OpenCV surge en 1999, originalmente desarrollada por Intel, como librería de C++ de visión artificial y *machine learning*. En la actualidad cuenta con versiones para Python, Java, MATLAB entre otros lenguajes. Se encuentra disponible para Linux, Mac y Windows. OpenCV cuenta con implementaciones de más de 2500 algoritmos de visión artificial, que cubren un gran abanico de casos de uso, desde reconocimiento de objetos hasta control de tráfico o seguridad.

Como se ha comentado antes, OpenCV es una parte fundamental de este proyecto, ya que todo lo referente a la visión artificial va a hacer uso de esta librería.

Git

Git es una herramienta de control de versiones originalmente desarrollada por Linus Torvalds para mantener el kernel de Linux. Su desarrollo comienza en abril de 2005 y su primera versión estable se lanzó en diciembre de ese mismo año. En la actualidad se ha convertido en uno de los sistemas de control de versiones más importantes.

El uso de Git (o cualquier VCS) es, en mi opinión, indispensable por pequeña que sea la envergadura del proyecto a desarrollar, debido a la posibilidad de tener siempre disponibles las versiones anteriores del código. Esto hace extremadamente sencillo volver a una versión anterior

en caso de necesidad, y proporciona una buena manera de hacer backups incrementales del código. También es muy útil de cara al trabajo en equipo, en nuestro caso, la supervisión de los avances del trabajo por parte del director.

PyQt

PyQt es una adaptación multiplataforma de la librería gráfica Qt. Esta librería está desarrollada por la compañía inglesa Riverbank Computing. PyQt cuenta con toda la funcionalidad de programación de aplicaciones gráficas disponible en la librería de la que proviene.

PyQt ha facilitado en gran medida la programación de una interfaz gráfica, lo que es interesante de cara a facilitar el uso de la aplicación al usuario medio

4. Diseño y resolución

En este apartado vamos a pasar a explicar los detalles del desarrollo del proyecto. Como se ha dicho, el proyecto consiste en realizar un sistema capaz de seguir los movimientos del balón y las jugadoras de un partido de volleyball mediante imágenes proporcionadas por una cámara en vista cenital.

El seguimiento se puede realizar por *tracking*, sustracción de fondo, o bien un sistema que aúne ambas cosas para una mayor robustez.

Una vez resuelta la parte del seguimiento, el siguiente objetivo sería registrar las posiciones detectadas a un fichero de salida en formato CSV, mediante el cual se puedan hacer análisis de distintas métricas del juego.

El desarrollo se va a centrar, como he ido aclarando, en una aplicación en Python utilizando OpenCV y PyQt para la parte de interfaz gráfica. De cara a la construcción del sistema, la estrategia a seguir es desarrollar separadamente sus partes y posteriormente implementarlas para que funcionen de manera conjunta.

A continuación detallaré estas partes y cómo se ha resuelto su implementación. Las partes principales del sistema son: sustracción de fondo, seguimiento (*tracking*), interfaz gráfica y métodos de consistencia temporal

4.1. Sustracción de fondo

Para empezar a detectar formas en la imagen, primero nuestro sistema tiene que ser capaz de hacer una estimación del fondo. La técnica mediante la cual se hace esto se denomina sustracción de fondo. Esta es, por tanto, una parte primordial de nuestro sistema que debe funcionar de una manera suficientemente robusta. Este debe ser capaz, además, de distinguir a las jugadoras del balón, y de etiquetar cada una de las formas de una manera consistente entre los frames del vídeo.

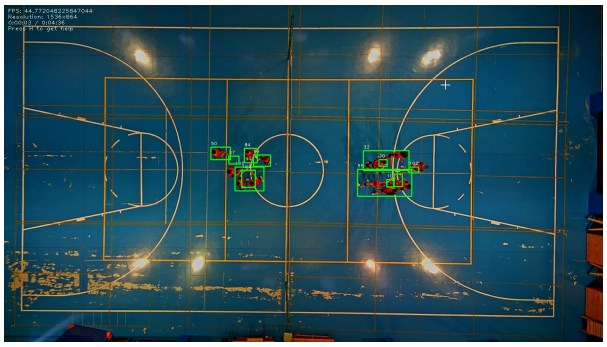


Figura 6. Imagen de salida sin la operación de supresión de máximos

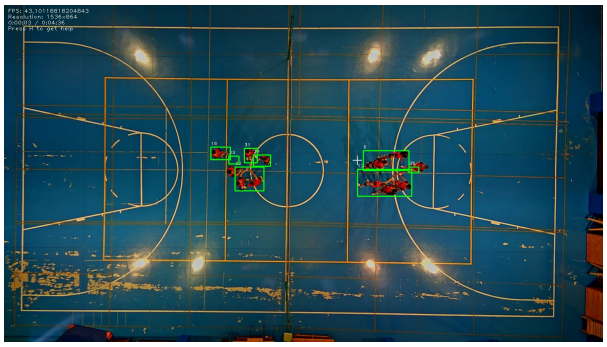


Figura 7. Imagen de salida tras la operación de supresión de máximos

Para ello, tras detectar las formas que forman parte del primer plano de la imagen se realizan una serie de operaciones. La primera de ellas es la **supresión de no máximos**. En ciertos momentos del vídeo, puede que no se detecte una forma de manera completa, sino que se detecta el contorno y ciertas partes separadas. Cuando esto ocurre, la imagen de salida tiene la forma que vemos en la figura 6 donde pueden verse varios cuadrados redundantes que podrían englobarse en uno más grande. Tras aplicar la supresión de no máximos, la imagen queda como en la figura 7, donde puede verse que los cuadrados contenidos dentro de otros han sido suprimidos.

La segunda operación es el **test de circularidad**, el cual nos sirve para saber cuál de las formas detectadas en la imagen es el balón. Para ello, calculamos el coeficiente de circularidad $C = \frac{4*\pi*área}{perímetro^2}$. Cuanto mayor sea C, mayor probabilidad de que la forma sea la de un círculo, es decir, la del balón. En un frame cualquiera, la forma cuyo coeficiente C tenga un valor mayor de 0.6 y sea el mayor de todos los contornos detectados, se marcará en rojo en la imagen de salida, mientras que el resto se marcarán en verde.

La última de las operaciones que se realizan es el etiquetado de los contornos detectados. Esto es importante ya que de un frame a otro, no tenemos una manera rápida de saber qué contorno del frame actual corresponde con qué contorno del anterior. Para solucionar esta problemática, en el primer frame se da una etiqueta a todos los contornos que se detecta. A partir de aquí, en los siguientes frames se irá etiquetando en base a un emparejamiento mutuo entre los contornos, calculando una matriz de distancias de todos los contornos entre sí. Una vez calculada esta matriz, si para un contorno c, su más cercano es otro contorno k y si para k el mas cercano es c, esto quiere decir que son el mismo contorno. En caso de quedar contornos

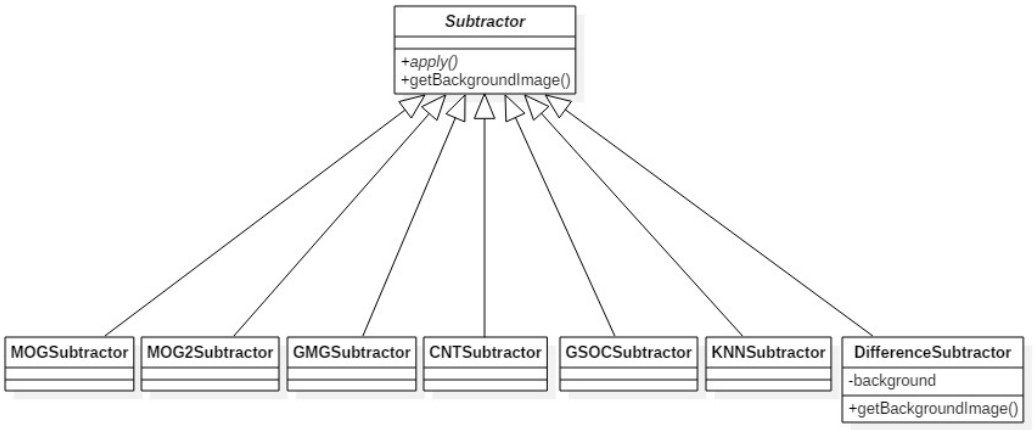


Figura 8. Estructura de clases de los sustractores de fondo

sin pareja después de esto, simplemente se les da una etiqueta nueva

Una vez establecidas estas operaciones secundarias, tenemos que solucionar el problema principal, la sustracción de fondo. Teniendo en cuenta que las imágenes proceden de una cámara fija y cenital, y que por tanto el fondo siempre va a ser exactamente el mismo, una primera aproximación ingenua a este problema podría ser tan simple como realizar una diferencia entre los frames del video.

Por desgracia, las condiciones de iluminación no siempre van a ser totalmente iguales, lo cual va a dificultar en gran medida la detección de objetos usando este método. Además, no siempre vamos a tener disponible una imagen limpia del campo en un vídeo para usar como modelo de fondo. Por ello es complicado realizar un modelo del campo para las diferentes condiciones de iluminación a lo largo de un día mediante resta de frames.

Ademas de la sensibilidad a cambios en la iluminación, este método adolece de otra desventaja: no es capaz de detectar sombras. Las imágenes que usamos son las de un campo de volleyball dentro de un pabellón, y los focos de dentro de este hacen que sea bastante común tener sombras alargadas y notables en las jugadoras.

Una vez hemos podido ver que esta manera de proceder no es todo lo viable que cabría desear, tendremos que empezar a plantearnos otras opciones a nuestra disposición. OpenCV proporciona distintos algoritmos de sustracción de fondo, los cuales podemos aprovechar y comparar para ver cuál es más apropiado en nuestro propósito.

Para facilitar el uso de estos algoritmos, se ha hecho uso de la programación orientada a objetos que nos ofrece Python. Se ha creado una clase abstracta llamada Subtractor de la cual heredan todas las clases que hagan uso de los algoritmos de OpenCV así como el de diferencia de frames. Dicha clase proporciona una interfaz común a todas las clases, que hará mas sencillo el uso de ellas. La estructura resultante puede verse en la figura 8.

El funcionamiento interno de las clases resultantes, como se ha dicho, corresponde a las propias implementaciones de OpenCV. Pasaré a explicar los fundamentos teóricos de cada uno de estos algoritmos.

MOG

MOG, mezcla de gaussianas (*Mixture of Gaussians*), fue descrito en [6] partiendo de otro algoritmo ya existente de Grimson *et al* [7, 8, 9]. Ambas versiones crean un modelo del fondo en base a una mezcla de k gaussianas con $3 \leq k \leq 5$, cada una de las cuales representa un color. La probabilidad de que un pixel tenga el valor x_N en tiempo N es de

$$p(x_N) = \sum_{j=1}^{k} w_j \eta(x_N; \theta_j)$$

donde w_k es el peso de la componente guasiana k-ésima. $\theta(x; \theta_k)$ es la ecuación que define la distribución de la componente, representada por

$$\theta(x; \theta_k) = \theta(x; \mu_k, \Sigma_k) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_k|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)}$$

donde μ_k es la media y $\Sigma = \sigma_k^2 I$ es la covarianza de la componente. Las distribuciones se ordenan en base a su fitness, definido por $\frac{w_k}{\sigma_k}$ y las B primeras son el modelo de fondo con $B = \arg\min(\sum_{j=1}^{k} w_j < T)$ donde T es el límite de probabilidad mínima a partir de cual un pixel se considera fondo. Un pixel se considera parte de los objetos de la escena si está 2.5 desviaciones por encima de una de las B distribuciones.

La diferencia entre la implementación original de Grimson y la de este algoritmo es las funciones de actualización de cada frame. En el caso del original las funciones de actualización son:

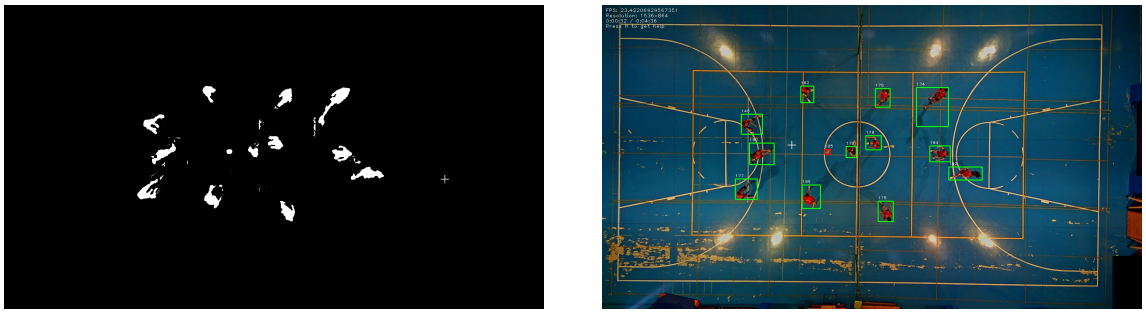
$$\hat{w}_k^{N+1} = (1-\alpha)\hat{w}_k^N + \alpha\hat{p}(\omega_k|x_{N+1})$$
$$\hat{\mu}_k^{N+1} = (1-\alpha)\hat{\mu}_k^N + \rho x_{N+1}$$
$$\hat{\Sigma}_k^{N+1} = (1-\alpha)\hat{\Sigma}_k^N + \rho(x_{N+1} - \hat{\mu}_k^{N+1})(x_{N+1} - \hat{\mu}_k^{N+1})^T$$
$$\rho = \alpha\eta(x_{N+1}; \hat{\mu}_k^N, \hat{\Sigma}_k^N)$$
$$\hat{p}(\omega_k|x_{N+1}) = 1 \text{ si } \omega_k \text{ es la primera componente gaussiana; 0 si no}$$

Una novedad de MOG respecto a su predecesor es la capacidad para discernir las sombras de los objetos que las proyectan. Para ello se emplea una diferencia de la componente cromática y del brillo y si pasa de ciertos límites, se marca como sombra. Más concretamente,

El funcionamiento de MOG en un frame puede verse en la figura 9, como puede verse, el algoritmo detecta los cuerpos con cierto nivel de fidelidad, aunque en algunos momentos puede ser especialmente sensible a una jugadora estática. Por otro lado, la detección de las sombras no es en absoluto perfecta, y no en pocas ocasiones se marcan partes de la sombra como componentes de la imagen. El rendimiento es correcto, no se aprecian ralentizaciones graves.

MOG2

MOG2 es un algoritmo propuesto en los trabajos [10] y [11] de Z. Zivkovic.



(a) (b)

Figura 9. Salida del algoritmo MOG (a) Imagen binarizada del algoritmo (b) Imagen del vídeo tras aplicar la máscara

GMG

GMG surge en un trabajo de Godbehere *et al.* [12]. En este trabajo aclaran los autores que se inspiraron en los algoritmos ya existentes en OpenCV. El método tiene una peculiaridad respecto al resto, y es que requiere de un número T de frames determinado para inicializar el modelo de fondo. Esta inicialización puede ser interesante en caso de contar con vídeos en los que se vea el fondo al principio sin ningún tipo de oclusión, en nuestro caso, que se vea el campo sin ningún jugador. Desgraciadamente, esto no siempre es posible, por lo que la efectividad inicial puede verse limitada debido a esta condición.

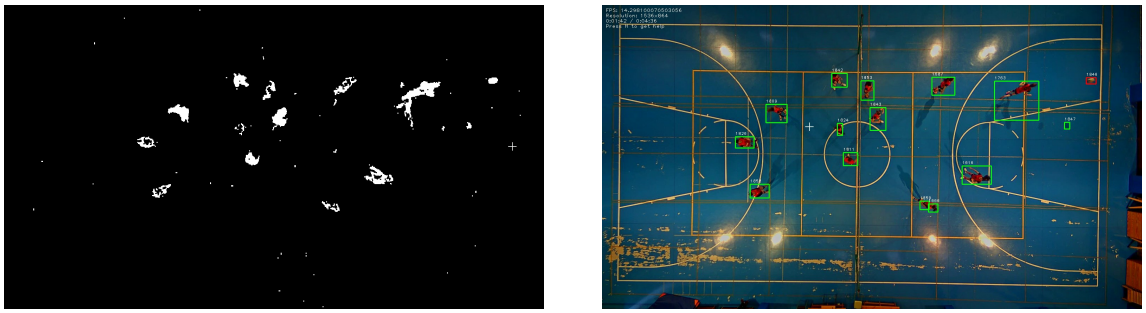
Durante la inicialización, se guarda un histograma $\hat{H}_{ij}(k)$ en espacio RGB por cada pixel previamente cuantizado. Los primeros T frames del vídeo se utilizan, como datos de entrenamiento para estimar la función de probabilidad de cada pixel, es decir, el modelo de fondo. El proceso de inicialización comienza en cada frame generando un vector $f_{ij}(k) = L(\hat{I}_{ij}(k)) \in \mathcal{F}$ mediante una función L a partir de la entrada. Una vez generados todos los vectores, se calcula el histograma $\hat{H}_{ij}(T) = \frac{1}{F_{tot}}\Sigma_{x=1}^{T} f_{ij}(x)$. F_{tot} es el número total de distintos factores observados, que siempre debe ser menor o igual que F_{max}, una constante del sistema. En caso de que sea mayor, se eliminan las observaciones más antiguas hasta que se cumpla que $F_{tot} \leq F_{max}$.

A partir de aquí se utiliza inferencia bayesiana para calcular la probabilidad de que un píxel dado sea fondo a partir del factor observado. Sea $p(F|f)$ la probabilidad de que un pixel con factor $f_{ij}(k)$ sea parte del primer plano y $p(B|f)$ la probabilidad de que este mismo pixel sea fondo:

$$p(B|f) = \frac{p(f|B)p(B)}{p(f|B)p(B) + p(f|F)p(F)}$$

Siendo $p(f|B) = f_{ij}(k)^T \hat{H}_{ij}(k)$, $p(f|F) = 1 - p(f|B)$, $p(F)$ es un parámetro constante que afecta a la sensibilidad del algoritmo y $p(B) = 1 - p(F)$. Una vez calculado $p(B|f)$ para cada pixel, obtenemos una imagen $P(k)$ a la que se le aplican una serie de transformaciones morfológicas de manera que se eliminen las posibles pequeñas formas anómalas que puedan detectarse y las componentes de la imagen aparezcan conectadas. Por último se aplica un umbral a la imagen para generar a partir de ella una imagen en binario que servirá como máscara y será la salida del sustractor.

Por último se debe actualizar el histograma que se usa como modelo de fondo de manera



(a) (b)

Figura 10. Salida del algoritmo GMG (a) Imagen binarizada del algoritmo (b) Imagen del vídeo tras aplicar la máscara

que se adapte a posibles cambios en la escena. Si un pixel esta marcado como frente de la escena, el histograma para él no se actualiza. Si no, si el factor $f_{ij}(k)$ tiene peso 0 en el histograma y se ha excedido la constante F_max, se debe eliminar algún factor del histograma, el de menor peso. A continuación se actualiza el histograma con el nuevo factor a añadir: $H_{ij}(k+1) = (1-\alpha)H_{ij}(k) + \alpha f_{ij}(k)$ siendo α el parámetro de adaptación (*learning rate*) del algoritmo, y cuanto más grande sea, antes se eliminan las observaciones anteriores.

En cuanto al desempeño del algoritmo, en la figura 10 podemos ver que es extremadamente sensible al ruido de la cámara, cosa que hace que la detección de objetos no sea estable. Esta inestabilidad hace que constantemente se pierdan de un frame a otro algunos de los contornos, y haya que asignarles etiquetas nuevas, lo que explica que en la imagen haya etiquetas tan altas. Se aprecian algunos fallos en la detección de sombras, por ejemplo los de la jugadora más arriba a la derecha y el balón en la imagen.

El rendimiento no es especialmente bueno, y se puede apreciar que cuando el ruido de imagen es muy grande y se detectan muchos contornos falsos, el funcionamiento es extremadamente lento. Además, durante los frames de inicialización, también se notan algunos bajones de FPS. Con todo, estos problemas no son especialmente serios y un reescalado de la imagen bastaría para paliarlos.

CNT

Este método fue desarrollado como open source en [13]. El algoritmo fue planteado por el desarrollador para que fuera eficiente en máquinas de bajo rendimiento, como raspberryPi. Se basa en "contar" (que es de donde viene su nombre) cuántos frames se mantiene estable un pixel, y a mayor estabilidad, mayor probabilidad de que este sea fondo.

El método consta de 2 parámetros principales: minPixelStability y maxPixelStability. La utilidad de estos parámetros es, respectivamente, indicar el número de frames que un pixel debe mantenerse en el mismo color para ser considerado estable, y el máximo crédito que se le permite a un pixel en el histograma.

En mi caso, usando este método, la reproducción del vídeo llega sobradamente a los 30 fps, con lo que puede ser útil si existe la necesidad de que la aplicación funcione en tiempo real,

además de contar con un modo para paralelizar la ejecución de manera que sea incluso más rápida. El algoritmo es relativamente sencillo en su funcionamiento: en cada frame se calcula la diferencia entre el color actual y el del frame anterior, si no se diferencian por encima de cierto límite, se incrementa la estabilidad del pixel en cuestión. Cuando la estabilidad de un pixel llega a ser igual o mayor que la que se le pasa al algoritmo por parámetro se empieza a considerar parte del fondo.

Existe la opción de añadir el uso de un histograma, y aquí es donde cobra importancia el segundo parámetro. Además de llevar un conteo de la estabilidad del propio pixel, se mantiene un histograma con las estabilidades que ha tomado este, y si un pixel supera el valor máximo se deja de sumar. De esta manera el algoritmo puede reponerse rápidamente a que un objeto que ha pasado largo tiempo en escena se mueva.

GSOC

GSOC fue implementado durante un *Google Summer of Code*, no procede de ningún artículo, sino que parte de un algoritmo existente (LSBP). El funcionamiento de este algoritmo no tiene documentación más allá de aclarar que se basa en ¿¿saliencia de vídeo??

En general, el algoritmo funciona de manera bastante robusta en lo que respecta a detectar formas en la escena, e incluso es capaz de detectar sin grandes problemas las sombras. Sin embargo, no logra detectar el balón de manera consistente, por lo que sería necesario usar algún tipo de algoritmo de tracking en conjunción a este.

Por otro lado, a pesar de su robustez, el algoritmo es el más lento de los que se han probado, lo que lo hace bastante desaconsejable para aplicaciones en tiempo real.

KNN

[11]

4.2. Seguimiento

4.3. Interfaz gráfica

5. Conclusiones y vías futuras

Referencias

[1] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010. ISBN 1848829345. Disponible online.

[2] D. Cireşan, U. Meier, and J. Schmidhuber. Multi-column Deep Neural Networks for Image Classification. *ArXiv e-prints*, February 2012.

[3] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. *ArXiv e-prints*, March 2017.

[4] Wikipedia contributors. Hawk-eye — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Hawk-Eye, 2018. [Online; accessed 14-April-2018].

[5] Thomas Moeslund, Graham Thomas, and Adrian Hilton. *Computer Vision in Sports*. Springer, 2014. ISBN 978-3-319-09395-6.

[6] P. KaewTraKulPong and R. Bowden. *An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection*, pages 135–144. Springer US, Boston, MA, 2002. ISBN 978-1-4615-0913-4. doi: 10.1007/978-1-4615-0913-4_11. URL https://doi.org/10.1007/978-1-4615-0913-4_11.

[7] W. E. L. Grimson, C. Stauffer, R. Romano, and L. Lee. Using adaptive tracking to classify and monitor activities in a site. In *Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.98CB36231)*, pages 22–29, Jun 1998. doi: 10.1109/CVPR.1998.698583.

[8] C. Stauffer and W. E. L. Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, volume 2, page 252 Vol. 2, 1999. doi: 10.1109/CVPR.1999.784637.

[9] C. Stauffer and W. E. L. Grimson. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):747–757, Aug 2000. ISSN 0162-8828. doi: 10.1109/34.868677.

[10] Z. Zivkovic. Improved adaptive gaussian mixture model for background subtraction. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 28–31 Vol.2, Aug 2004. doi: 10.1109/ICPR.2004.1333992.

[11] Zoran Zivkovic and Ferdinand van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recognition Letters*, 27(7):773 – 780, 2006. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2005.11.005. URL http://www.sciencedirect.com/science/article/pii/S0167865505003521.

[12] A. B. Godbehere, A. Matsukawa, and K. Goldberg. Visual tracking of human visitors under variable-lighting conditions for a responsive audio art installation. In *2012 American Control Conference (ACC)*, pages 4305–4312, June 2012. doi: 10.1109/ACC.2012.6315174.

[13] Sagi Zeevi. The background subtractor cnt project. https://github.com/sagi-z/BackgroundSubtractorCNT, 2017.