# 2.- INTRODUCCIÓN TEÓRICA:

## 2.1.- Estructura:

En este capítulo se pretende plasmar toda la teoría relacionada con los algoritmos que vamos a utilizar en la implementación de nuestro sistema de seguimiento y detección de caídas.

En un principio desarrollaremos el **estado del arte** actual relativo a los trackers o algoritmos de seguimiento, diferenciando entre los métodos de segmentación y reconocimiento y los métodos para el análisis del movimiento. A continuación, una vez expuestos todos los algoritmos realizaremos una comparación entre ellos, resaltando sus ventajas e inconvenientes, para finalmente seleccionar aquellos que vamos a emplear en nuestro sistema.

Por último, realizaremos una descripción más exhaustiva de aquellos algoritmos seleccionados, indicando cómo funcionan y sus fundamentos matemáticos.

# 2.2.- Estado del arte:

## 2.2.1.- Introducción

Durante estos últimos años, la visión por computador ha sufrido grandes avances y dentro de esta disciplina, el análisis del movimiento ha sido objeto de importantes estudios. La razón principal por la que esto ha ocurrido es porque los primeros sistemas de visión trabajaban con escenas estáticas y con entornos conocidos y estructurados, dando lugar a técnicas muy interesantes pero de poca aplicabilidad dentro del mundo real. Sin embargo, en la mayoría de los sistemas de visión actuales existen objetos en movimiento, por lo que será necesario que estas aplicaciones presenten la **capacidad de analizar el movimiento** utilizando técnicas y herramientas específicas.

En general, los cambios que se producen en las imágenes capturadas se deben al movimiento de los objetos de la escena, a cambios en su estructura, tamaño o forma, movimientos de la cámara que realiza la grabación o cambios en la iluminación. Como ya hemos comentado a lo largo de la memoria, en este tipo de sistemas es necesaria una velocidad de computación alta, por lo que el análisis del movimiento se centrará sólo en los **movimientos relativos entre la cámara y el objeto**, tratando al resto como posibles fuentes de error que habrá que controlar y minimizar.

El principio básico sobre el que se sustenta la teoría del análisis del movimiento es que los objetos suelen presentar un movimiento suave a lo largo del tiempo debido a la inercia. Sobre este hecho se fundamenta toda la teoría del tracking, ya que la única forma que existe para que un ordenador pueda seguir a un objeto es **anticipándose** a su movimiento, es decir, predecir cuál va a ser la próxima posición del mismo.

Lógicamente, para conseguir esta caracterización del movimiento son necesarias una serie de consideraciones. En primer lugar, hay que determinar de forma precisa cuál es el objeto o partes del objeto que hay que seguir, ya que dentro de una imagen existe mucha información que puede provocar que los algoritmos se pierdan y no sean capaces de realizar el seguimiento. Por otro lado, también precisamos caracterizar la evolución del movimiento de dichas partes, ya que en caso contrario la predicción de su posición no sería posible. Esta caracterización del movimiento se conseguirá extrayendo un **modelo dinámico** que describirá su evolución a lo largo del tiempo.

### 2.2.2.- Método de trabajo

De acuerdo con todo lo dicho hasta el momento, los algoritmos de seguimiento por visión realizan en general las siguientes tareas:

- Selección de los objetos a seguir: Dentro de una secuencia de imágenes capturadas en un entorno no estructurado, existen muchos elementos que pueden distraer al sistema provocando la pérdida del objeto. En general, en los sistemas de seguimiento por visión no se sigue al objeto completo, sino sólo algunas partes del mismo que presentan determinadas características como su textura, color o forma que las hacen distintas al resto de elementos presentes en la imagen y por lo tanto hacen que su identificación sea más sencilla, fiable y robusta frente a situaciones indeseadas.

- Estimación del movimiento: Una vez seleccionados los elementos que vamos a seguir, tendremos que encontrarlos dentro de la siguiente imagen. Para ello será necesario determinar un recuadro de búsqueda no muy grande y una técnica adecuada para que el tiempo de computación no sea excesivo y así podamos realizar el seguimiento en tiempo real y de forma efectiva.

- Determinación de la posición y orientación: Utilizando los datos obtenidos con la localización de los elementos en la imagen, será necesario obtener los valores de la posición y orientación de los mismos, ya que nos servirán para refinar el modelo dinámico. La determinación de estos dos parámetros (posición y orientación) constituyen el resultado final del proceso de seguimiento.

- Filtrado de los parámetros: Los parámetros obtenidos en el paso anterior no son exactos debido al ruido en las medidas, por lo que realizaremos un filtrado de las mismas en base al modelo dinámico establecido.

- Predicción del movimiento: Como ya se ha explicado, la idea básica del seguimiento por visión es la de anticiparse y predecir el movimiento del objeto. En este paso la aplicación buscará las posibles nuevas localizaciones de los elementos, consiguiendo así reducir la ventana de búsqueda en la siguiente iteración. En general, es bastante probable que entre imagen e imagen los elementos no hayan variado demasiado su posición (suposición de movimiento suave y continuado) por lo que con la ayuda del modelo dinámico será posible realizar una predicción bastante aproximada.

Todos estos pasos (excepto la determinación de los elementos a seguir) se repetirán continuamente para cada una de las imágenes capturadas en la grabación, quedando finalmente un esquema como el que se muestra a continuación:

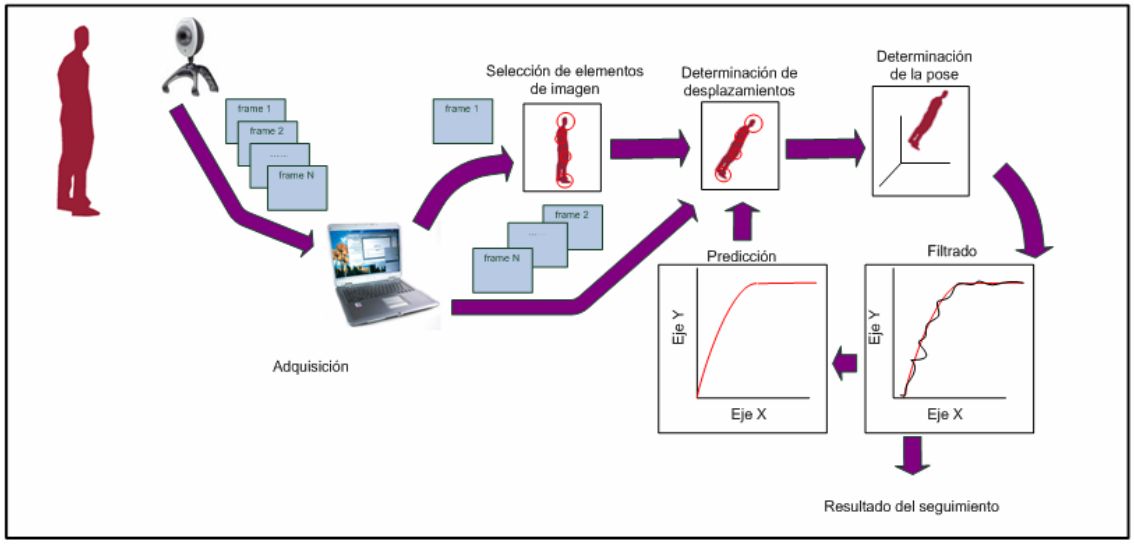


Figura 2.1.- Esquema general de un sistema de seguimiento

El desarrollo de estas tareas implicará la utilización de distintas técnicas adecuadas a los requerimientos generales de la aplicación para la que se vaya a destinar el sistema (seguimiento de tráfico, eye-trackers, detección de caídas, etc) y a los requerimientos particulares de cada una de estas tareas.

Es importante señalar que las soluciones adoptadas para cada etapa dependerán directamente de las técnicas empleadas en las etapas anteriores, de forma que si la característica determinante para elegir los elementos a seguir ha sido el histograma, la técnica para la búsqueda y determinación de la posición (siguientes etapas) tendrán que basarse también en el histograma.

### 2.2.3.- Algoritmos:

En el apartado anterior hemos detallado la metodología de trabajo que hemos de seguir para poder realizar el seguimiento de objetos en movimiento. Otra posible clasificación de las fases de este proceso se muestra en este otro esquema:

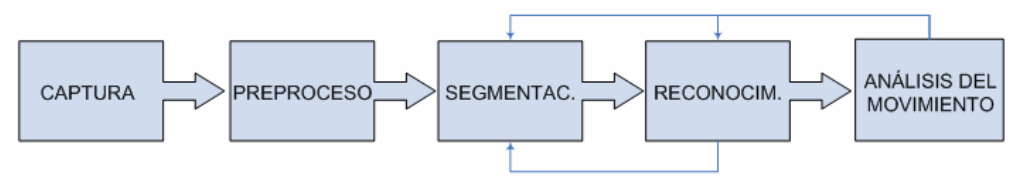


Figura 2.2.- Fases de un sistema de seguimiento por visión

Como podemos observar, la primera fase consistente en la captura y preprocesado de cada imagen es común prácticamente para todos los sistemas de visión por computador. Sin embargo, las fases dos y tres divergen en función del sistema que queramos diseñar y en función de la aplicación para la que dicho sistema esté pensado.

Existen en la actualidad gran cantidad de técnicas distintas para las fases dos y tres del esquema anterior, siendo posibles múltiples combinaciones de las mismas para la implementación de nuestro detector de caídas. Comenzaremos en primer lugar con las técnicas centradas en la segmentación y reconocimiento del elemento de interés y posteriormente nos centraremos en las técnicas de análisis del movimiento. Nuestra intención es desarrollar de forma breve cada una de ellas, para así poder compararlas y justificar la decisión de utilizar unos métodos frente a otros.

### 2.2.3.1.- MÉTODOS DE SEGMENTACIÓN Y RECONOCIMIENTO

Antes de comenzar a estudiar cada una de estos métodos sería interesante hacer una clasificación general de los mismos en base a la forma en la que realizan su cometido. Esta clasificación se divide en los métodos tradicionales basados en **detección de características** (*featured-based)* que utilizan datos como el color o las formas geométricas y los **métodos basados en reconocimiento de patrones** (*image-based*) que son más actuales y que obtienen los resultados mediante un aprendizaje previo y la utilización de un modelo o patrón como referencia.

- Detección de características: Este tipo de métodos emplean características de bajo nivel como los niveles de gris, el color, los bordes o la existencia o no de movimiento para reconocer y encontrar los elementos dentro de las imágenes.

- Reconocimiento de patrones: También se conocen como detectores basados en aprendizaje. Este tipo de métodos no emplean características de tan bajo nivel como el anterior, sino que se

aprovechan del conocimiento a priori de la forma del elemento a seguir para establecer un patrón o modelo de referencia que luego buscar en las imágenes analizadas. Estos detectores se basan en redes neuronales artificiales que distinguen entre las entradas que sí coinciden con los elementos buscados y entradas que no coinciden.

En general podemos decir que los métodos basados en características son apropiados para procesado de imágenes en tiempo real cuando el color y el movimiento son características propias del elemento buscado, mientras que para imágenes estáticas los elementos basados en reconocimiento de patrones son mejores pues disminuye la carga computacional y la eficacia y exactitud a la hora de encontrar el objeto es mucho mayor. De todas formas, esta afirmación será cierta o no en función de la aplicación concreta para la que estemos trabajando.

Una vez enunciados los grandes grupos en los que podemos englobar los distintos algoritmos existentes en la actualidad vamos a desarrollar algunos de los más conocidos de forma breve.

### A.- SEGMENTACIÓN MEDIANTE PATRONES:

La técnica de segmentación mediante patrones, tal y como su nombre indica, se engloba dentro de los métodos de reconocimiento de patrones. Ésta es una de las técnicas de procesado digital de imágenes más antiguas y utilizadas para la identificación o localización de objetos en imágenes en aplicaciones tan diversas como la robótica, imágenes médicas, astronomía, etc.

Su funcionamiento básico es muy sencillo: empleando como muestra una imagen generalmente de dimensiones reducidas que representa el objeto que se quiere encontrar (o alguna parte del mismo), realizamos una búsqueda de la misma sobre un área determinada de la imagen capturada por la cámara, también llamada ventana de búsqueda. Esa imagen muestra recibe el nombre de **imagen patrón**.

Figura 2.3.- Segmentación mediante patrones

Para realizar la búsqueda, el patrón se desplaza sobre la ventana de búsqueda, aplicando una función de coincidencia que tomará un valor dependiendo del grado de semejanza entre patrón e imagen. Este grado de semejanza se puede establecer de muchas maneras, desde métodos sencillos aplicando una simple diferenciación píxel a píxel, hasta métodos donde los píxeles son agrupados en estructuras más complejas (como líneas, contornos, bloques, polígonos, etc.) para luego ser buscadas en la imagen. Al final de este proceso, el punto de la imagen que devuelva el máximo valor según el criterio de semejanza establecido será el punto donde estará localizado el objeto que estamos buscando. Es importante resaltar que actualmente es la correlación uno de los criterios de semejanza más extendidos.

Si hacemos este proceso para varias imágenes pertenecientes a una secuencia de video, obtendremos que aquellas posiciones donde el criterio de semejanza devuelva su máximo valor nos proporcionarán la trayectoria seguida por el objeto. Es por esta razón que este método se emplea en muchas aplicaciones de seguimiento por visión.

Es bastante habitual que el objeto que tratamos de encontrar no sea exactamente igual que la imagen patrón que se tiene, debido a posibles transformaciones a lo largo del tiempo como traslaciones, giros, acercamientos o alejamientos, cambios de color o intensidad e incluso deformaciones del mismo. Por esa razón, una solución común consiste en considerar que el objeto puede sufrir transformaciones afines, como se muestra en el artículo *"Good features to track"* de J.Shi y C. Tomasi de 1994.

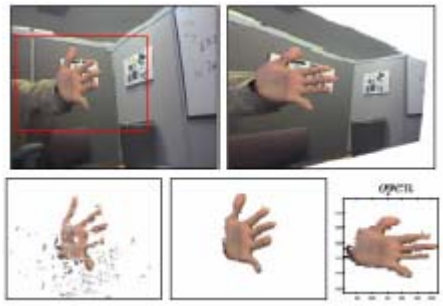

Figura 2.4.- Localización y transformación de la imagen de una mano

El principal problema que se plantea en este método es la gran carga computacional que supone el tener que evaluar el criterio de semejanza o correlación para cada posible localización del patrón. Si además consideramos también la posibilidad de contemplar transformaciones geométricas en el patrón la dificultad es aún mayor, pues se requiere un mayor número de comparaciones además de tener que realizar el proceso de transformación o *warping* de la imagen patrón.

## B.- SEGMENTACIÓN POR RECONOCIMIENTO DE FORMAS:

Este método podríamos considerarlo una variante del método de segmentación mediante patrones y por lo tanto lo englobaremos también dentro del reconocimiento basado en patrones. Como ya hemos comentado antes, en este tipo de métodos se recurre a un **conjunto de muestras** del objeto a buscar que se utilizarán luego para su identificación y localización dentro de otras imágenes.

Este método se basa en algoritmos sencillos que buscan formas concretas, como por ejemplo, una forma circular oscura rodeada de un área blanca como caracterización de la pupila del ojo. Podría parecer en primera instancia igual que el método de segmentación por patrones, sin embargo éste método es más robusto puesto que se centran más en el contraste entre regiones que en los valores absolutos de los píxeles que las forman, siendo efectivos en escenarios donde hay cambios de luminosidad o donde hay variaciones en la forma buscada.

Uno de los principales inconvenientes que presentan los dos métodos (por patrones y por reconocimiento de formas) es que el reconocedor debe ser previamente entrenado con las muestras del objeto de interés. Lógicamente, para obtener un reconocedor lo suficientemente efectivo es necesario una base de datos amplia con muestras de “objeto” y de “no-objeto”. Se planteará por lo

tanto el compromiso entre la efectividad del reconocedor y el tiempo de cómputo necesario para el reconocimiento del objeto.

### C.- SEGMENTACIÓN MEDIANTE "BLOBS":

Este es el primer método que explicaremos de los basados en características. En el contexto del procesamiento digital de imágenes, denominaremos *blob* a una agrupación de píxeles que verifican una determinada propiedad (como por ejemplo el color, la intensidad, píxeles en movimiento, texturas, etc.) y que están conectados entre sí, es decir, que son píxeles vecinos.

El *blob* o agrupación de píxeles viene caracterizado por una serie de parámetros como son su área, su contorno o su centroide entre otros, de modo que cuando se realice el seguimiento del movimiento de dicho *blob*, bastará con determinar la nueva posición del centroide del mismo en la nueva imagen para determinar su nueva posición y por lo tanto conocer su trayectoria.

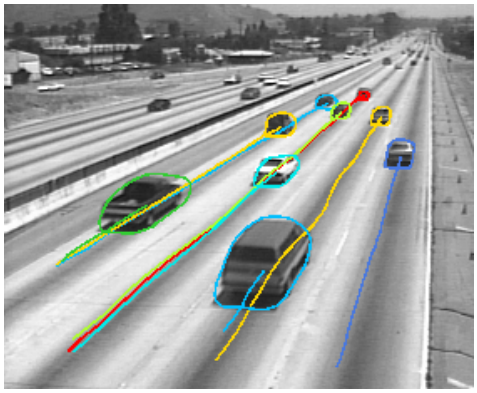

Figura 2.5.- Segmentación mediante blobs

Este método es muy interesante y se usa mucho en la actualidad debido a que es muy simple y a que existen librerías completas de funciones para la creación, modificación, manejo y análisis de los *blobs*.

Uno de los principales problemas que se plantea en los algoritmos de segmentación de la imagen mediante *blobs* es la aparición de falsos positivos (agrupación de píxeles que se consideran parte del elemento de interés y que en realidad son objetos del fondo o distractores) y la aparición de falsos negativos (agrupación de píxeles que se considera que forman parte del fondo y que en realidad son uno de los elementos buscados). Para poder solventar este problema se hace necesario por tanto un postprocesado de la imagen en base a una serie de criterios como son el área de los *blobs* (si sabemos cuál es el área

aproximada que debe ocupar el elemento de interés podemos eliminar aquellos *blobs* que posean un área fuera del rango de valores), su perímetro, su textura, su color, etc. Lógicamente, el criterio utilizado para el postprocesado de la imagen y la eliminación de estos falsos positivos o negativos debe ser distinto al criterio utilizado para realizar la segmentación original. Esto nos obliga a tener conocimiento de más de una característica básica del elemento de interés, lo que no siempre es posible.

Por otro lado, es importante resaltar también otra serie de problemas que se nos pueden plantear con este tipo de segmentación, como por ejemplo la obtención de múltiples *blobs* asociados a un único objeto (teniendo pues que reagrupar dichos *blobs*), obtención de un único *blob* asociado a varios elementos de interés (teniendo que separar los *blobs* en el postprocesado), etc. Como vemos, la segmentación inicial obtenida con este método no siempre es satisfactoria y el postprocesado se vuelve bastante laborioso, de forma que la sencillez de cómputo que se nos garantizaba inicialmente con este método se ve mermada por la serie de operaciones que hay que realizar después para garantizar una segmentación adecuada para la aplicación.

## D.- SEGMENTACIÓN POR *"BACKGROUND SUBTRACTION"*

El método de segmentación mediante la substracción de los objetos de interés (*foreground*) del fondo (*background*) se puede considerar uno de los métodos más empleados en las aplicaciones de seguimiento debido a que los objetos de interés en este tipo de algoritmos son aquellos elementos que se encuentran en movimiento.

Con el uso de un buen algoritmo que separe el fondo de la imagen de los objetos de interés, el resto de operaciones que se realicen después durante el análisis del movimiento (cálculo de la posición, trayectorias, velocidad, etc.) podrán centrarse exclusivamente en el elemento que estamos siguiendo, ignorando los píxeles pertenecientes al fondo que son distractores y generan muchas ambigüedades. Además, la ventana de trabajo se ve reducida, aumentando por lo tanto la velocidad de procesamiento y la eficiencia de la aplicación.

Existen en la actualidad muchos métodos para realizar la separación fondo-objeto, por lo que a continuación comentaremos algunos de ellos de forma breve.

## D.1.- CHROMA-KEYING

La técnica más conocida por ser la más sencilla y la más utilizada en efectos especiales es lo que se conoce como "*bluescreening*" o "*chroma-keying*". Este método consiste en la grabación de los objetos de interés sobre un fondo azul o verde, para posteriormente eliminar dicho fondo aplicando un sencillo filtro basado en el color. Evidentemente, los objetos de interés no deben presentar zonas con ese color pues automáticamente serían eliminadas.

Figura 2.6.- Sustracción de fondo mediante chroma-keying

Este método, a pesar de estar muy extendido para aplicaciones televisivas, de cine, fotografía o experimentos controlados en el laboratorio, no suele servir para aplicaciones de procesado visual reales, ya que los objetos se encuentran en fondos no controlados.

## D.2.- DIFERENCIACIÓN ENTRE IMÁGENES

Otro método para la sustracción del fondo es lo que se conoce como "*frame differencing*" o diferenciación entre imágenes. Este algoritmo consiste en restar dos imágenes consecutivas, de forma que si la diferencia de intensidad entre los píxeles situados en la posición (x,y) es menor que un determinado umbral se considerará que dicho píxel no ha variado y que por lo tanto no pertenece al objeto en movimiento. Lógicamente este método es muy pobre pues si el objeto posee regiones con colores parecidos al fondo estas regiones se considerarán fondo también y por el contrario, si en la imagen existen sombras o cambios de luminosidad éstos pasarán a formar parte de lo que la aplicación consideraría objeto.

D.3.- MODELADO ESTADÍSTICO DEL FONDO

Si nos centramos en métodos más potentes que los anteriores hablaremos de los algoritmos basados en modelos estadísticos, los cuales modelan el comportamiento de cada uno de los píxeles pertenecientes al fondo.

El modelo estadístico más simple que podemos aplicar es aquel que asume que la intensidad de cada uno de los píxeles pertenecientes al fondo varía de forma independiente de acuerdo a una distribución normal. Para calcular los parámetros (media y varianza) de dicha distribución normal asociada a cada píxel es necesario acumular datos de bastantes imágenes donde no aparezca el objeto en movimiento y sólo haya fondo, lo cual no siempre es posible. Por último, una vez caracterizado el fondo del sistema diremos que un píxel pertenece al objeto en movimiento si su valor no se comporta como la distribución normal calculada para su posición (considerando un cierto margen de error).

La técnica anterior, a pesar de ser más sofisticada es muy poco robusta, pues es sensible a cambios de luminosidad, pequeños movimientos de la cámara y no se adapta a los posibles cambios que se puedan producir en el fondo, como la introducción de nuevos elementos, por ejemplo un bolso. Para mejorar esta técnica se introducen lo que se conocen como **algoritmos adaptativos para la sustracción del fondo**, los cuales son capaces de funcionar correctamente en entornos desestructurados y solventan los problemas que acabamos de mencionar.

Existen en la actualidad muchísimas variantes de estos algoritmos adaptativos que no desarrollaremos en este apartado, pero cuyo funcionamiento general consiste en una actualización del modelo estadístico asociado utilizando para ello un filtro adaptativo sencillo. Gracias a esta actualización del modelo la introducción de distractores en el sistema no nos afectan pues pasado cierto tiempo, si permanecen estáticos, éstos pasaránn a formar parte del fondo.

## E.- SEGMENTACIÓN MEDIANTE CONTORNOS:

Nos encontramos ante otro método basado en características. La segmentación mediante contornos es una técnica de reciente aparición que es mucho más robusta que el resto de técnicas que hemos visto hasta el momento para aquellas aplicaciones que trabajen en entornos desestructurados. La segmentación de contornos no es una única técnica, si no un conjunto de técnicas que se denominan **contornos activos.**

Su funcionamiento básico consiste en el modelado de las fronteras entre el objeto de interés, el fondo y el resto de objetos de la imagen. Lógicamente esta tarea no es trivial, y en ocasiones requerirá la ayuda de información adicional sobre la forma del objeto de interés para poder extraer su contorno. Gracias a esto y como ya hemos comentado, esta técnica es mucho más robusta frente a la presencia de ruido o de elementos distractores en la imagen, permitiendo así la segmentación de imágenes más complejas como por ejemplo las imágenes médicas.

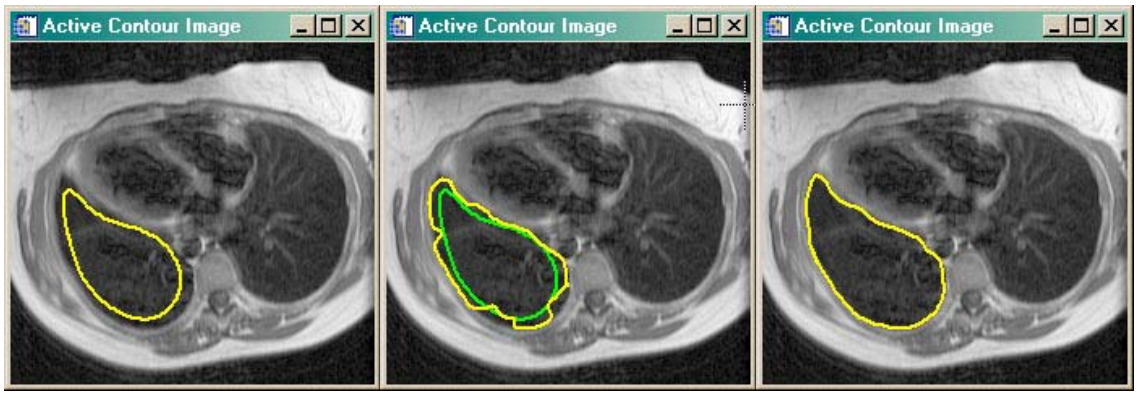


Figura 2.7.- Ejemplo de contornos activos en imágenes médicas

Como ya hemos mencionado, estas técnicas precisan un modelo a priori del objeto de interés lo cual nos restringe el abanico de aplicaciones para las que podemos utilizarlo. Sin embargo, este hecho nos proporciona también ventajas, pues en general los resultados obtenidos no precisan un procesado posterior y además son directamente interpretables al basarse en un modelo ya conocido.

Como comentábamos antes para el método de segmentación mediante *blobs*, uno de los principales problemas al que nos enfrentamos es la aparición de falsos positivos y negativos. Sin embargo, con los contornos activos, al emplear un modelo ya conocido del elemento de interés, la presencia de estos errores disminuye considerablemente.

Los contornos activos pueden ser clasificados en *snakes* (serpientes), patrones deformables y contornos dinámicos. A continuación desarrollaremos algunas características de cada grupo.

- Serpientes (snakes): Hasta el momento, con el resto de mecanismos de segmentación analizábamos los datos de la imagen y a partir de la información obtenida se generaban los *blobs*, o los contornos, etc. Con los *snakes* utilizaremos la información a priori de modo que no esperaremos a que propiedades deseables de los contornos como la continuidad o la suavidad provengan de los datos de la imagen, sino

que estas propiedades serán asumidas desde el principio. La forma de conseguir esto es imponer un modelo elástico de curva continua y flexible que posteriormente se ajustará a los datos de la imagen.

- Patrones deformables: Como ya hemos comentado, en los contornos activos lo que utilizaremos es un modelo elástico de curva continua. En los *snakes* el modelado a priori es muy sencillo, pues sólo asume propiedades básicas de continuidad y suavidad; sin embargo, para los patrones deformables el modelado a priori se puede hacer más complejo trabajando con varias curvas flexibles y con un conjunto de parámetros que controlen el tamaño de cada curva, los ángulos con que se unen al resto, etc. Este modelo es mucho más potente que el anterior para buscar estructuras conocidas.

- Contornos dinámicos: Este tipo de contornos activos son los que se utilizan cuando necesitamos localizar objetos en movimiento. Al problema de segmentar la información de la imagen se le añade el hecho de tener que seguir los elementos segmentados. Para solucionar este problema será necesario añadir inercia, factor de amortiguamiento y fuerzas de restauración al *snake* estático y además, el modelo a priori tendrá que ser un modelo dinámico.

Por último hemos de decir que un contorno activo no está pensado en principio para resolver el problema de búsqueda automática de contornos en la imagen, sino para refinar la solución obtenida por otros métodos menos exactos. Es decir, partiendo de un contorno cercano a la solución (un contorno establecido manualmente o utilizando métodos clásicos de segmentación), utilizando los contornos activos dicha solución evolucionará hasta el contorno buscado. Esto lo consiguen gracias a un **proceso de minimización de energía**.

## F.- MÉTODOS BASADOS EN EL ESPACIO DE ESTADOS

Estos métodos han adquirido bastante fama durante estos últimos años debido al enfoque probabilístico que le dan al problema del seguimiento dentro de los algoritmos basados en características. Si se considera el conjunto de objetos a seguir como un sistema dinámico cuyo estado varía en el tiempo, es necesario utilizar un modelo que describa esa evolución de forma conveniente. Esos modelos vienen descritos en términos del **espacio de estados**.

Según el tipo de sistema al que nos enfrentemos en nuestra aplicación encontraremos distintos tipos de algoritmos:

- Sistemas lineales:
  - LMS (Least Mean Square)
  - KF (Kalman Filter)
- Sistemas con ligeras no linealidades:
  - EKF (Extended Kalman Filter)
- Sistemas no lineales:
  - PF (Particle Filter)
  - Condensation (**Con**ditional **Dens**ity Propag**ation**)

A continuación desarrollaremos el filtro de Kalman para sistemas lineales y el filtro de partículas para sistemas no lineales por ser los algoritmos más extendidos.

F.1.- FILTRO DE KALMAN

El filtro de Kalman, a pesar de ser propuesto en el año 1960, es uno de los métodos más populares en la actualidad y que aún hoy sigue siendo objeto de investigación y de su utilización en diversas aplicaciones.

Este algoritmo lo que pretende es estimar el estado de un proceso **minimizando el error cuadrático medio**. Este método es muy potente debido a que soporta estimaciones de estados pasados, presentes y futuros y debido a que se pueden realizar estimaciones aunque no se tenga un conocimiento exhaustivo del funcionamiento del sistema modelado.

Este algoritmo es un algoritmo recursivo, lo que significa que sólo son necesarios el estado estimado de la iteración anterior y la medida actual para estimar el estado actual del sistema en cuestión, no como otros sistemas donde es necesario un historial completo de estados antiguos para poder realizar la estimación. Otra característica importante es el hecho de que es un filtro que trabaja en el dominio del tiempo (al contrario que la mayoría de los filtros que trabajan en el dominio de la frecuencia y por lo tanto luego tienen que transformar sus resultados al dominio del tiempo para poder ser interpretados y aplicados).

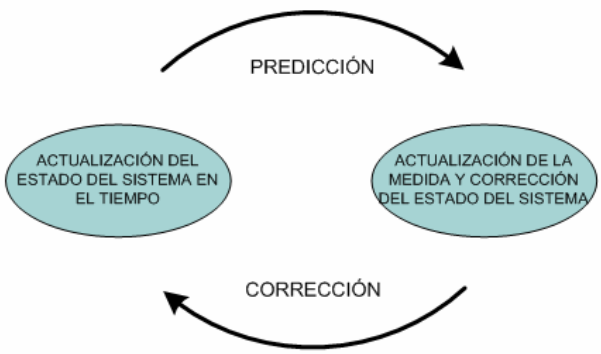


Figura 2.8.- Fases de funcionamiento del filtro de Kalman

Como se puede apreciar en la figura anterior, el filtro de Kalman tiene dos fases distintas: **Predicción y actualización**. En la fase de predicción utiliza la estimación propia de la iteración anterior para producir una nueva estimación para el estado actual y el fase de actualización utiliza la información medida en el instante actual para refinar la anterior predicción del estado y así conseguir una estimación mucho mejor.

El principal problema que plantea este método es que sólo es efectivo para sistemas caracterizados por una distribución normal de media cero y desviación determinada y para sistemas que se ven afectados por ruidos blancos gaussianos. Para el caso de sistemas no lineales aparecen otros métodos como el filtro de partículas (Particle filtering) y el algoritmo CONDENSATION que veremos a continuación.

## F.2.- FILTRO DE PARTÍCULAS. ALGORITMO CONDESATION

Para el caso de sistemas dinámicos no lineales, no gaussianos e incluso multimodales empleamos estos dos algoritmos.

El método conocido como filtro de partículas fue propuesto en el año 1993 como algoritmo complementario al filtro de Kalman y con la intención de solventar las limitaciones de éste. La idea principal del filtro de partículas consiste en la representación de la función de densidad de probabilidad que caracteriza al sistema mediante un **conjunto de muestras discretas con pesos asociados**, y estimar dichas muestras y sus pesos para el instante siguiente.

Al igual que el filtro de Kalman, el filtro de partículas es un algoritmo recursivo donde podemos diferenciar las siguientes cuatro etapas:

- *Inicialización*: En esta etapa generaremos de forma aleatoria valores para las variables que definen el estado de cada partícula. En general cada una de estas partículas comienza con el mismo peso $1/N$, donde N es el número total de partículas. En caso de que conozcamos la distribución a priori podremos aplicar un peso concreto a cada partícula.
- *Actualización*: Aquí asignaremos un valor (peso) a cada una de las partículas que nos indicará la calidad de la predicción que se realizó para cada una de ellas.
- *Estimación:* Estimamos el estado del sistema en ese instante. Como cada partícula tiene un estado diferente, se considera que el sistema tiene el estado de la partícula con más peso o se hace una media ponderada con los estados de todas las partículas.
- *Predicción:* Con el nuevo conjunto de partículas obtenidas, se seleccionan aquellas de mayor calidad (con mayor peso) y sobre ellas se aplica el modelo de movimiento, y así se predice el estado del sistema en el instante siguiente.

Ya dijimos que este método era un algoritmo iterativo, por lo que después de esta última etapa de predicción se obtiene un nuevo conjunto de partículas al que le volveremos a aplicar la etapa de actualización y así sucesivamente. A continuación mostramos una figura con la evolución de las partículas durante una iteración:

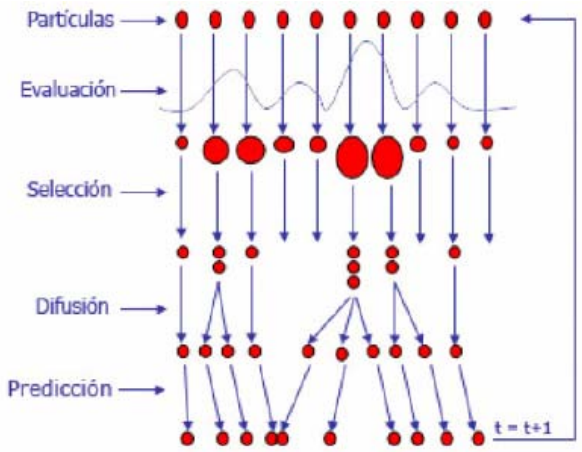


Figura 2.9.- Fases del algoritmo basado en el filtro de partículas

Como podemos observar, después de la etapa de actualización cada una de las partículas tiene un peso asociado; aquellas partículas con poco peso serán eliminadas y las partículas con mucho peso se subdividirán en otras partículas.

Al final siempre debe quedar el mismo número de partículas, sobre las que de nuevo se volverá a aplicar el algoritmo.

En cuanto al algoritmo CONDENSATION (Conditional Density Propagation), es una variante del algoritmo de filtro de partículas que permite realizar seguimiento de contornos sobre secuencias de imágenes y que hoy día conforma la base de la mayoría de los algoritmos de seguimiento basados en la "propagación de la densidad condicional", es decir, en la propagación de la función densidad de probabilidad.

## G.- ALGORITMO CAMSHIFT

Es otro de los métodos basados en características. Este algoritmo fue desarrollado por Gary R. Bradski de Intel en el año 1998 con la intención de crear un sistema para el **seguimiento de caras basado en el tono de la piel** que fuera más potente que los existentes hasta el momento (como los basados en contornos activos, eigenfaces, etc.) que eran demasiado costosos computacionalmente hablando.

El algoritmo CAMSHIFT proviene de "***C****ontinuously* ***A****daptative* ***M****ean* ***Shift***" y es una versión modificada del algoritmo Mean Shift, basado en gradientes de densidad que lo que pretende es encontrar el máximo en las funciones de densidad de probabilidad. En nuestro caso, la función de densidad de probabilidad es la distribución del color en una secuencia de video y para poder representar el color como una distribución emplearemos el **histograma**.

Los principales requisitos buscados por Bradski en su investigación fueron:

1. Conseguir que su sistema fuese capaz de **funcionar en tiempo real**, es decir que fuese capaz de procesar al menos 30 imágenes por segundo.
2. Que consumiese los mínimos recursos posibles del sistema, de tal forma que pudiese funcionar como parte integrante de un interfaz de usuario con muchas otras más aplicaciones funcionando a la vez.
3. Que fuese capaz de funcionar con cámaras sencillas y baratas y que no necesitase lentes calibradas o alta resolución.

Para conseguir estos objetivos, el algoritmo CAMSHIFT se centra en realizar un **seguimiento basado en el color**, y en concreto en la tonalidad, pues está optimizado para trabajar con objetos donde la tonalidad (*hue*) sea única y por lo tanto el histograma no sea multidimensional.

En general, los métodos que se basan en el color suelen ser muy lentos y complejos debido a la utilización de correlaciones basadas en el color, utilización de blobs, suavizado y predicción mediante filtro de Kalman, consideraciones de contorno, etc. para poder solventar problemas como el ruido, distractores en la escena, oclusiones del objeto de interés, cambios de luminosidad o los efectos de la perspectiva. Sin embargo, el algoritmo CAMSHIFT es capaz de mitigar todos estos problemas sin hacer uso de todas esas operaciones de postprocesado.

Su funcionamiento básico es muy sencillo: empleando como referencia el histograma del color del elemento de interés, buscaremos la mayor agrupación de píxeles con ese tono dentro de lo que llamamos la ventana de búsqueda (que es el área de la imagen donde el algoritmo ha establecido que es más probable que encontremos al objeto). En base a esto, esa agrupación de píxeles se corresponderá con el objeto que estamos siguiendo. Una vez encontrado, utilizaremos el tamaño y la posición actual del objeto para establecer el tamaño y la posición de la ventana de búsqueda de la siguiente iteración; de ese modo no será necesario procesar la imagen completa en cada ejecución del bucle, haciendo que este algoritmo sea bastante rápido.

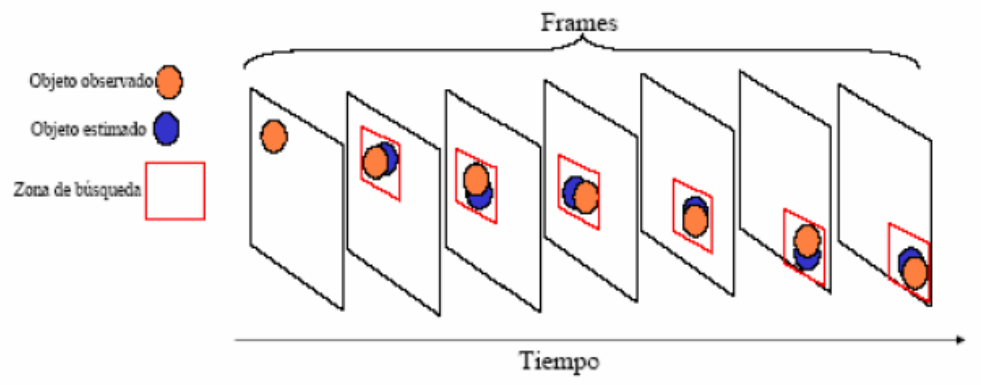


Figura 2.10.- Seguimiento mediante el algoritmo Camshift

En cada iteración obtendremos la posición (X,Y) del elemento buscado, su área (que será proporcional a Z, la distancia que haya entre el objeto y la cámara) y su ángulo de inclinación.

### 2.2.3.2.- MÉTODOS DE ANÁLISIS DE MOVIMIENTO

Una vez vistas las técnicas para la segmentación y reconocimiento vamos a centrarnos en los métodos de análisis del movimiento. Estos métodos se pueden clasificar en tres grandes grupos: Métodos continuos, métodos discretos y diferenciación entre imágenes.

### A.- DIFERENCIACIÓN ENTRE IMÁGENES

En un principio, los trabajos sobre análisis del movimiento comparaban cada una de las imágenes de la secuencia píxel a píxel, para ver si cada píxel de la imagen había sufrido un cambio o no. El resultado de esta operación es lo que se denomina **imagen binaria de diferencias**, la cual vale '0' en los píxeles que no han sufrido cambio y '1' en los píxeles que sí lo han sufrido. Matemáticamente hablando, dentro de una secuencia de imágenes, la función $I(x,y,t)$ representa el valor de la intensidad del píxel posicionado en las coordenadas $(x,y)$ de la imagen capturada en el instante $t$, en base a esto y tras comparar dos imágenes capturadas en los instantes $t_1$ y $t_2$, la imagen binaria de diferencias vendrá dada por $ID_{1,2}(x,y)$ cuya fórmula es la siguiente:

$$ID_{1,2}(x,y) = \begin{cases} 1 \text{ si } I(x,y,t_2) - I(x,y,t_1) \geq \tau \\ 0 \text{ en caso contrario} \end{cases}$$

Si tenemos en cuenta que un objeto en movimiento tiene en la imagen un área de varios píxeles, podemos aplicar criterios de vecindad a la imagen binaria de diferencias y así obtener más información sobre las áreas en movimiento de la imagen.

Como hemos visto, $ID_{1,2}(x,y)$ nos da información sobre los cambios producidos entre dos imágenes; si lo que queremos es analizar una secuencia completa de imágenes utilizaremos lo que se conoce como **imágenes de diferencias acumulativas**, de forma que podremos detectar más fácilmente objetos con un movimiento más lento.

Estas técnicas son interesantes por su simplicidad, ya que consumen muy pocos recursos en los cálculos; sin embargo son muy poco precisas y poco robustas, pues se ven afectadas gravemente por los cambios de iluminación, y es por eso que en la actualidad casi no se utilizan.

### B.- MÉTODOS CONTINUOS O DIFERENCIALES

Como ya hemos indicado anteriormente, la función $I(x,y,t)$ es la que determina la intensidad de cada píxel dentro de una secuencia de imágenes. Si somos capaces de determinar los gradientes o variaciones espaciales y temporales de esta función seremos capaces de calcular lo que se conoce como **flujo óptico** (optical flow), que nos dará información sobre cómo es el movimiento de los píxeles dentro de la imagen y a lo largo del tiempo. (En apartados posteriores de la memoria desarrollaremos más a fondo los principios del flujo óptico y la teoría subyacente).

### C.- MÉTODOS DISCRETOS

Hasta el momento hemos considerado siempre que el movimiento de los objetos es suave y que su desplazamiento es pequeño, sin embargo, para aquellos casos en los que el desplazamiento observado entre los píxeles de dos imágenes consecutivas es muy grande utilizaremos los *métodos discretos*. En este caso lo que se buscan no son píxeles o agrupaciones de éstos, sino características específicas de la imagen, como bordes, esquinas, texturas, etc. El problema que se nos plantea ahora es el establecimiento de correspondencia entre regiones de dos imágenes, buscando su similitud en un proceso conocido como *matching*.

## 2.2.3.3.- COMPARACIÓN DE ALGORITMOS Y SELECCIÓN

Una vez detallados todos los algoritmos que existen para el seguimiento de objetos vamos a justificar la elección de unos métodos frente a otros en base a la aplicación que queremos desarrollar, un detector de caídas.

En primer lugar, en cuanto a los métodos de segmentación y reconocimiento, vamos a desechar los dos métodos "*image-based*" (el algoritmo basado en patrones y el basado en reconocimiento de formas) ya que como ya explicamos anteriormente, para su correcto funcionamiento es necesario un entrenamiento previo del reconocedor y para ello es necesario poseer una base de datos adecuada con imágenes del elemento que deseamos encontrar. En nuestro caso, el elemento que vamos a seguir es una persona, la cual puede presentar infinitas poses (agachado, levantado, sentado, tumbado, de lado, de frente, de espaldas, cerca de la cámara presentando un tamaño, lejos de ella con un tamaño menor, etc.) Es por ello, que una base de datos adecuada para el

reconocimiento de siluetas humanas es muy compleja y por lo tanto no sería factible.

En un principio también se planteó la posibilidad de emplear como patrón la cara de una persona y no la silueta completa del individuo, sin embargo de la literatura existente sobre estos métodos se puede extraer que para que el reconocedor funcione bien la base de datos tiene que ser bastante extensa y variada lo cual resultaba muy complejo y además no nos presentaba garantías suficientes de obtener un resultado satisfactorio en cuanto a la velocidad del reconocedor, no pudiendo así detectar caídas. Por lo tanto descartamos los métodos basados en patrones y nos quedamos con los basados en características, que son menos exactos pero mucho más eficientes computacionalmente hablando.

En cuanto a los métodos basados en características, antes de seleccionar un método en concreto debemos tener claros cuáles son los puntos fuertes que debe presentar el algoritmo para que funcione nuestra aplicación:

- Debe ser un algoritmo sencillo, para que la carga computacional no sea muy alta y nuestra aplicación pueda funcionar en tiempo real.
- Debe funcionar correctamente con personas, que son los elementos que vamos a seguir en nuestro sistema. Si además el algoritmo está optimizado para trabajar con alguna característica propia de alguna parte del cuerpo, como por ejemplo la cara, será un serio candidato para ser elegido.
- Debemos tener en cuenta que la aplicación es un detector de caídas, y que las caídas son movimientos muy veloces, por lo que el algoritmo no sólo debe garantizar su funcionamiento en tiempo real, sino ser todo lo veloz posible para no perder al individuo durante una caída.
- El algoritmo debe ser bastante robusto, de forma que siga funcionando correctamente ante cambios de luminosidad, oclusiones del elemento a seguir, etc.

Teniendo en cuenta estas consideraciones y las ventajas e inconvenientes que hemos resaltado en la explicación de cada uno de los métodos decidiremos qué algoritmo utilizar para la implementación de la aplicación.

El método de segmentación mediante *blobs* hemos decidido no utilizarlo debido a que, a pesar de todas las librerías y funciones existentes al respecto que nos facilitarían el trabajo, presentan el grave inconveniente de tener que aplicar un postprocesado a la segmentación para la minimización de los falsos positivos y negativos, haciendo que la velocidad de proceso disminuya y por lo tanto la

aplicación sea difícilmente capaz de detectar movimientos tan rápidos como caídas.

El método de contornos activos es un método muy potente y que consigue resultados extraordinarios en la determinación de contornos, sin embargo, uno de los inconvenientes que presenta es que para conseguir esos resultados normalmente se suele aplicar primero un método clásico de segmentación para posteriormente refinarlo mediante los contornos activos. Además, la mayoría de aplicaciones que lo utilizan son aplicaciones con imágenes estáticas, pues la utilización de contornos dinámicos es bastante compleja. En la detección de caídas la velocidad es mucho más importante que la exactitud de la segmentación, por lo que hemos decidido no emplear este algoritmo.

En cuanto a la segmentación por *background subtraction* hemos de decir que es la más extendida para aplicaciones en las que el movimiento es la característica predominante de la escena, como es nuestro caso. Como ya hemos visto, existen numerosos métodos para este tipo de segmentación de los cuáles sólo los algoritmos adaptativos cumplen las necesidades de nuestra aplicación. Como ventajas a destacar, su sencillez, su especialización en la detección de movimiento y su robustez frente a cambios de luminosidad o a elementos distractores en la imagen. Por estas razones se plantea como uno de los algoritmos candidatos para nuestra aplicación.

En cuanto al algoritmo CAMSHIFT, no es un algoritmo tan extendido como la separación fondo-objeto en aplicaciones de seguimiento sin embargo también está pensado para aplicaciones donde el movimiento es un factor primordial. Sus principales ventajas son el hecho de ser mucho más rápido que cualquiera de los algoritmos anteriores pues está optimizado para emplear los mínimos ciclos de CPU posibles; por otro lado, nos aporta directamente la posición del centro, el área y la orientación del objeto que estamos siguiendo, siendo esto una información muy importante para nuestra aplicación. Como principal inconveniente nos encontramos con que esta velocidad se debe a que el algoritmo se optimizó para trabajar con objetos donde la tonalidad (*hue*) no sea heterogénea, por lo que nos veríamos limitados a trabajar sólo con los píxeles de la piel de las personas que sí cumplen esta característica. Por las ventajas expuestas anteriormente, sobre todo por su velocidad, lo consideraremos como otro algoritmo posible para la aplicación.

Por último nos queda por analizar los algoritmos basados en los espacios de estados, como el filtro de Kalman, los filtros de partículas o el algoritmo CONDENSATION. Esta serie de métodos, al igual que los dos anteriores, están principalmente pensados para aplicaciones de seguimiento de objetos, por lo que dentro de este ámbito son muy potentes y obtienen unos resultados muy

satisfactorios en general. Como principal ventaja es que son mucho más robustos que los anteriores y parecen funcionar mejor en entornos desestructurados, como inconveniente es que son veloces, pero no tanto como el algoritmo CAMSHIFT que acabamos de mencionar. Por su potencia y por su versatilidad (pues no tienen requerimientos tan específicos como el algoritmo CAMSHIFT) tampoco los desecharemos.

Como vemos, de todos los algoritmos existentes nos hemos quedado al final con tres: *background subtraction,* CAMSHIFT y espacio de estados. De todos ellos utilizaremos el primero debido a que es el método más extendido en la actualidad dentro de la implementación de aplicaciones de seguimiento de objetos, dejando los otros dos métodos pendientes en caso de que el primero no nos ofrezca los resultados deseados.

- ✘ Algoritmo basado en patrones
- ✘ Algoritmo de reconocimiento de formas
- ✘ Algoritmo mediante blobs
- ✘ Algoritmo mediante contornos activos
- ☑ Algoritmo de sustracción de fondo
- ✘ Algoritmo mediante contornos activos
- ▶ Algoritmo CAMSHIFT
- ▶ Algoritmos de Filtro de Kalman, filtro de partículas y CONDENSATION.

Todo lo que hemos visto es referente a los algoritmos de segmentación y reconocimiento. Nos centramos ahora en los algoritmos de análisis de movimiento para determinar velocidades, direcciones, sentido y flujo.

En relación a los métodos de diferenciación entre imágenes podemos decir que son demasiado sencillos y muy poco precisos, pues sólo aportan información de existencia de movimiento en determinadas posiciones, pero no aportan ningún dato más referente a velocidades, vectores que nos indiquen la nueva posición del píxel que se ha movido, etc. Datos que en definitiva necesitamos para un correcto análisis de la escena. Por estas razones no son suficientes para nuestra aplicación.

Referente a los métodos diferenciales, son los más interesantes no sólo porque son los más extendidos en aplicaciones basadas en el seguimiento de objetos, sino porque el movimiento característico en nuestra aplicación es un movimiento continuo y suave (no hay discontinuidades en la trayectoria de los píxeles) y por lo tanto la utilización de gradientes para la determinación de información como el sentido del movimiento, velocidad, etc. es el más adecuado. Por esa razón, utilizaremos para el detector de caídas los métodos

diferenciales y desecharemos los métodos discretos (característicos de movimientos discontinuos y amplia separación entre píxeles de imágenes consecutivas).

- ✘ Métodos de diferenciación entre imágenes
- ☑ Métodos diferenciales. Algoritmos basados en el flujo óptico.
- ✘ Métodos discretos.

Una vez desarrollados los algoritmos existentes y determinados cuáles van a ser los que vamos a emplear, pasaremos a realizar una explicación más exhaustiva de los mismos en el siguiente apartado.

## 2.3.- Técnicas empleadas:

Como ya hemos dicho, en este apartado vamos a completar la breve introducción que hicimos de cada una de las técnicas que vamos a emplear en nuestra aplicación para la detección de caídas incluyendo los fundamentos matemáticos en los que éstas se basan.

### 2.3.1.- Segmentación con "Background Subtraction".

De todos los algoritmos existentes para la segmentación fondo-objeto, los algoritmos basados en filtros adaptativos son los más adecuados para nuestra aplicación, gracias a su robustez en entornos desestructurados y a su adaptabilidad ante posibles modificaciones en la escena.

Existen actualmente numerosas variantes de este tipo de sistemas, por lo que la elección de unos algoritmos frente a otros suele complicarse. En nuestro caso, teniendo en cuenta que la librería que vamos a utilizar para la implementación del sistema es OpenCV[1] (Open Source Computer Vision Library) de Intel®, de todos los algoritmos existentes nos decantaremos por aquellos que tengan funciones relacionadas implementadas en la librería, ya que dichas funciones están optimizadas en la mayoría de los casos, obteniendo así un funcionamiento más eficiente que el obtenido si programáramos todo el algoritmo desde cero.

[1] En el siguiente capítulo haremos una breve introducción sobre dicha librería

Dentro de la librería de OpenCV (versión Beta 4.a, en la carpeta ../OpenCV/cvaux/src) encontramos tres ficheros: cvbgfg_acmmm2003.cpp, cvbgfg_common.cpp y cvbgfg_gaussmix.cpp donde podemos encontrar todas las funciones necesarias. En concreto tenemos dos métodos **cvGaussBGModel** y **cvFGDStatModel**.

El artículo sobre el que se basa el primer método es "*An Improved Adaptative Background Mixture Model for Real-time Tracking and Shadow Detection*" de P. KaewTraKulPong y R. Bowden en Proc. 2nd European Workshop on Advanced Video-Based Surveillance Systems, mientras que el segundo método se basa en el artículo "*Foreground Object Detection from Videos Containing Complex Background*" de Liyuan Li, Weimin Huang, Irene Y. H. Gu y Qi Tian en ACM-MM 2003. De estos dos artículos, decidimos implementar el primero pues en nuestra aplicación de detección de caídas el procesamiento en tiempo real es un requisito indispensable y además, como nuestro sistema va a funcionar en interiores no tendremos un fondo tan complejo como si trabajáramos en exteriores.

Este primer artículo es una mejora del algoritmo propuesto por Grimson et al. consistente en un modelo probabilístico del fondo donde el histórico de valores de intensidad de cada uno de los píxeles de la imagen viene caracterizado por una mezcla de gaussianas(normalmente de tres a cinco) ponderadas con un peso característico y que sirven para determinar la probabilidad de que el color de dicho píxel permanezca igual entre esa imagen y la siguiente.

#### 2.3.1.1- Modelo GMM por Grimson et al.

Este algoritmo consiste en el modelado de cada uno de los píxeles de la imagen mediante una mezcla de *K* distribuciones Gaussianas, de ahí el nombre de GMM (Gaussian Mixture Model). De esa forma, la probabilidad de que un determinado píxel valga $x_N$ en el instante *N* viene dada por:

$$p(x_N)=\sum_{j=1}^{K} w_j \cdot \eta(x_N,\theta_j) \qquad [2.1]$$

Donde $w_k$ es el peso asociado a la k-ésima componente Gaussiana del modelo, y $\eta(x,\theta_k)$ es la distribución Normal de la k-ésima componente, que se puede expresar como:

$$\eta(x,\theta_k)=\eta(x,\mu_k,\sum_k)=\frac{1}{(2\pi)^{D/2}\cdot\left|\sum_k\right|^{1/2}}\cdot e^{-\frac{1}{2}(x-\mu_k)^T\sum_k^{-1}(x-\mu_k)} \qquad [2.2]$$

Siendo $\mu_k$ la media de la distribución y $\sum_k=\sigma_k^2 I$ la covarianza asociada a la k-ésima componente.

Cada una de estas Gaussianas se emplean para representar diferentes colores; en general *K* será un número pequeño entre 3 y 5. Por otro lado, en la expresión [2.1] hemos introducido el concepto de peso asociado a cada distribución $w_k$, el cual representará la proporción de tiempo que ese color ha permanecido en la escena.

Antes de continuar con el desarrollo del algoritmo GMM propuesto por Grimson et al. introduciremos un nuevo concepto al que ellos denominan *fitness*. En general, en una imagen en la que existen elementos en movimiento los objetos de un único color que permanecen inmóviles tienden a formar agrupaciones compactas en el espacio del color, mientras que los objetos en movimiento tienden a formar agrupaciones más expandidas debido a los reflejos de estos objetos móviles sobre las distintas superficies que hay en la imagen. Para medir esto introduciremos un nuevo concepto al que Grimson et. al denominan *fitness*, cuyo valor viene dado para cada distribución como $w_k/\sigma_k$.

Teniendo en cuenta esto último, las *K* distribuciones Gaussianas se ordenarán en función a su valor de *fitness* y de todas esas distribuciones nos quedaremos sólo con las *B* primeras para modelar el *background* o fondo de la escena. Gracias a esto, los colores con los que modelamos el fondo se corresponderán con los *B* colores que han permanecido estáticos durante más tiempo en la imagen. El valor del parámetro *B* vendrá dado por la siguiente expresión:

$$B=\arg_b\min\left(\sum_{j=1}^{b}w_j>T\right) \qquad [2.3]$$

Donde *T* es el umbral que determina si una distribución continúa o no considerándose en el modelo.

Por último, la segmentación se realiza considerando píxeles **pertenecientes al objeto** a todos aquellos píxeles cuyo valor se encuentre **más alejado de 2.5 desviaciones estándar** de cualquiera de las *B* distribuciones que modelan el fondo.

Ya hemos comentado que una de las principales características de este algoritmo GMM es su capacidad de adaptación ante posibles cambios en la escena. Para conseguir esto y que además el algoritmo siga funcionando en tiempo real se aplica un esquema de actualización de las Gaussianas y sus pesos. Esta actualización recibe el nombre de **actualización selectiva** y consiste en los siguientes pasos:

1. Para cada píxel obtenemos un valor de color o intensidad que compararemos con el modelo GMM existente
2. Si encontramos una distribución Gaussiana asociada a dicho color actualizaremos sus parámetros asociados siguiendo las siguientes ecuaciones.

$$\hat{w}_k^{N+1} = (1-\alpha)\hat{w}_k^N + \alpha\hat{p}(\omega_k \mid x_{N+1}) \quad [2.4]$$

$$\hat{\mu}_k^{N+1} = (1-\alpha)\hat{\mu}_k^N + \rho x_{N+1} \quad [2.5]$$

$$\hat{\Sigma}_k^{N+1} = (1-\alpha)\hat{\Sigma}_k^N + \rho(x_{N+1} - \hat{\mu}_k^{N+1})(x_{N+1} - \hat{\mu}_k^{N+1})^T \quad [2.6]$$

$$\rho = \alpha\eta(x_{N+1};\ \hat{\mu}_k^N, \hat{\Sigma}_k^N) \quad [2.7]$$

$$\hat{p}(\omega_k \mid x_{N+1}) = \begin{cases} 1 & ;si\ \omega_k\ es\ la\ primera\ componente\ Gaussiana\ encontrada \\ 0 & ;en\ cualquier\ otro\ caso \end{cases} \quad [2.8]$$

Donde $\omega_k$ es la k-ésima componente Gaussiana y $1/\alpha$ es la constante de tiempo que determina el cambio.

3. Si no se encuentra ninguna distribución que concuerde con el valor del píxel, la componente Gaussiana menos probable de todas (la que tenga un menor peso) será reemplazada por una nueva distribución cuya media será el valor de dicho píxel, la varianza será un valor muy alto y el peso asociado será pequeño.
4. Por último, en base a los artículos de Grimson et al. XXX, sólo los parámetros $\alpha$ y $T$ necesitan ser inicializados para el funcionamiento del sistema.

Este método propuesto por Grimson et al. es muy potente pues funciona sin problemas ante cambios de luminosidad, se adapta a la eliminación o introducción de nuevos elementos en la escena y es capaz de ignorar pequeños movimientos producidos por la cámara o por los árboles o vegetación existentes en la escena; sin embargo, al principio el aprendizaje y actualización de las

Gaussianas es muy lento, especialmente en ambientes con mucho movimiento y tampoco es capaz de distinguir los objetos de interés de las sombras que estos proyectan.

El método propuesto por P. KaewTraKulPong y R. Bowden trata de solventar esos problemas modificando las ecuaciones empleadas para la actualización del modelo (actualización de las distribuciones), la inicialización del mismo y la introducción de un detector de sombras, obteniendo un algoritmo más potente y más eficiente en la segmentación.

#### 2.3.1.2- Mejora del método. El algoritmo EM

Una forma típica de optimizar los modelos basados en la composición de gaussianas (GMM: Gaussian Mixture Model) como el propuesto por Grimson et al. consiste en emplear el algoritmo EM (Expectation Maximisation). Este algoritmo es un método iterativo que garantiza la convergencia hacia un máximo local en un espacio de búsqueda. Como este algoritmo hay que aplicarlo para cada uno de los píxeles que forman el fondo de la imagen, tendremos que utilizar un método EM *online*, es decir, aplicado en tiempo real.

Para evitar tardar tanto en obtener un modelo GMM donde los colores del fondo sean las distribuciones predominantes, P. KaewTraKulPong y R. Bowden proponen en su artículo inicializar el sistema empleando las ecuaciones del algoritmo EM *online* para pasar después a emplear las ecuaciones de actualización del método de "las L ventanas más recientes" (*L-recent windows update equations*). Pasaremos de un método a otro después de que L imágenes hayan sido procesadas.

En general, el método EM *online* proporciona una buena estimación del modelo GMM antes de que las L muestras hayan sido procesadas, por lo que tendremos garantía de que la inicialización es correcta. Gracias a esta estimación inicial del modelo al algoritmo de segmentación convergerá mucho más rápido a un modelo GMM estable del fondo consiguiendo que su funcionamiento posterior sea mucho mejor. Por otro lado, las ecuaciones de actualización del método de las *L* ventanas dan mayor prioridad a aquellos datos más recientes de modo que el sistema de segmentación pueda adaptarse más rápidamente a posibles cambios en el entorno. A continuación mostramos las ecuaciones para el algoritmo EM y para el método de las *L* ventanas.

Ecuaciones de actualización para el algoritmo EM:

$$\hat{w}_k^{N+1} = \hat{w}_k^N + \frac{1}{N+1}\left(\hat{p}(\omega_k \mid x_{N+1}) - \hat{w}_k^N\right) \quad [2.9]$$

$$\hat{\mu}_k^{N+1} = \hat{\mu}_k^N + \frac{\hat{p}(\omega_k \mid x_{N+1})}{\sum_{i=1}^{N+1} \hat{p}(\omega_k \mid x_i)}(x_{N+1} - \hat{\mu}_k^N) \quad [2.10]$$

$$\hat{\Sigma}_k^{N+1} = \hat{\Sigma}_k^N + \frac{\hat{p}(\omega_k \mid x_{N+1})}{\sum_{i=1}^{N+1} \hat{p}(\omega_k \mid x_i)}\left((x_{N+1} - \hat{\mu}_k^N)(x_{N+1} - \hat{\mu}_k^N)^T - \hat{\Sigma}_k^N\right) \quad [2.11]$$

Ecuaciones de actualización para el algoritmo de las *L* ventanas:

$$\hat{w}_k^{N+1} = \hat{w}_k^N + \frac{1}{L}\left(\hat{p}(\omega_k \mid x_{N+1}) - \hat{w}_k^N\right) \quad [2.12]$$

$$\hat{\mu}_k^{N+1} = \hat{\mu}_k^N + \frac{1}{L}\left(\frac{\hat{p}(\omega_k \mid x_{N+1})x_{N+1}}{\hat{w}_k^{N+1}} - \hat{\mu}_k^N\right) \quad [2.13]$$

$$\hat{\Sigma}_k^{N+1} = \hat{\Sigma}_k^N + \frac{1}{L}\left(\frac{\hat{p}(\omega_k \mid x_{N+1})(x_{N+1} - \hat{\mu}_k^N)(x_{N+1} - \hat{\mu}_k^N)^T}{\hat{w}_k^{N+1}} - \hat{\Sigma}_k^N\right) \quad [2.14]$$

Como ya hemos dicho, en el artículo de P. KaewTraKulPong y R. Bowden también se propone un método para la eliminación de sombras que sin embargo no está implementado en las funciones de la librería OpenCV por lo que no lo desarrollaremos aquí.

#### 2.3.1.3- Resultados

Empleando las funciones de librería basadas en la teoría anterior se implementó un programa sencillo para comprobar que el comportamiento inicial del sistema era satisfactorio utilizando este algoritmo. Sin embargo, el resultado que obtuvimos no fue el esperado, pues bien es cierto que el algoritmo realiza una segmentación fondo-objeto impecable (a falta de solucionar el aspecto de las sombras) pero la velocidad de procesamiento del mismo no es suficiente para nuestra aplicación, ya que nosotros precisamos una velocidad aproximada de 30 frames por segundo y este método no es capaz de alcanzarla.

Otro problema que pudimos apreciar es que el algoritmo está pensado para adaptarse a posibles cambios en la escena, de forma que todos aquellos elementos que permanezcan inmóviles durante un determinado tiempo terminarán siendo considerados como parte del fondo. Para un entorno en el que los objetos de interés estén continuamente moviéndose este hecho es muy interesante, sin embargo, en nuestra aplicación la persona que esté siendo monitorizada puede permanecer quieta durante un tiempo indeterminado por lo que terminaría perdiéndose en la imagen.

Debido a todo esto, nos vimos obligados a cambiar de algoritmo. En primer instancia consideramos la posibilidad de utilizar otro método adaptativo basado en la sustracción fondo-objeto que tardara más tiempo en actualizarse en caso de inmovilidad del sujeto, sin embargo, vimos que esa opción no era adecuada puesto que el sistema no sería capaz entonces de solventar cualquier otra variación en la escena, como la introducción de nuevos objetos, considerando a éstos como elementos de interés cuando en realidad no lo son. Además, tampoco tendríamos garantía de obtener un procesamiento de las imágenes lo suficientemente veloz, ya que para cada imagen tenemos que procesar cada uno de los píxeles lo que supone una carga computacional demasiado alta. Por estas razones decidimos no utilizar los algoritmos basados en la separación fondo-objeto en base al movimiento.

Es importante señalar que todo el código implementado hasta el momento, las funciones de librería y los resultados obtenidos se pueden consultar en el **anexo B**, pues a pesar de no ser un algoritmo adecuado para nuestra aplicación puede ser interesante para otro tipo de aplicaciones, como por ejemplo detectores de movimiento en sistemas de seguridad, pudiendo reutilizarse todo el trabajo y el código.

De los otros dos algoritmos que habíamos dejado como posibles suplentes, CAMSHIFT y los basados en el espacio de estados, existen funciones relacionadas en la librería OpenCV por lo que podemos implementar nuestra aplicación utilizando cualquiera de ellos. Debido a que nuestro principal requisito es la velocidad de procesamiento, terminamos decantándonos por el algoritmo CAMSHIFT por ser el más rápido de los dos.

- ✘ Algoritmo basado en patrones
- ✘ Algoritmo de reconocimiento de formas
- ✘ Algoritmo mediante blobs
- ✘ Algoritmo mediante contornos activos
- ✘ Algoritmo de sustracción de fondo
- ✘ Algoritmo mediante contornos activos
- ☑ Algoritmo CAMSHIFT

- Algoritmos de Filtro de Kalman, filtro de partículas y CONDENSATION.

## 2.3.2.- Algoritmo CAMSHIFT.

El algoritmo CAMSHIFT que vamos a utilizar para la implementación de nuestra aplicación es el que viene desarrollado en el artículo "*Computer Vision Face Tracking for Use in a Perceptual User Interface*", de Gary R. Bradski (Microcomputer Research Lab, Santa Clara, CA, Intel Corporation).

Este apartado pretende completar las nociones básicas introducidas sobre el algoritmo en apartados anteriores para comprender su funcionamiento y conocer la teoría subyacente.

### 2.3.2.1- Introducción.

El algoritmo CAMSHIFT se implementó en primera instancia con intención de realizar un seguimiento eficiente de la cabeza y la cara, para ser integrado luego como parte de una interfaz de usuario más completa. Este método es una adaptación del algoritmo **Mean Shift**, capaz de encontrar la moda (mean) de una determinada distribución mediante iteraciones consecutivas en la dirección del máximo incremento de la densidad de probabilidad.

El nombre de CAMSHIFT proviene de *Continuosly Adaptative Mean Shift*, y la principal diferencia con el algoritmo Mean Shift consiste en que CAMSHIFT utiliza funciones de distribución adaptativas, es decir, que las funciones de densidad de probabilidad no son estáticas y fijas como en el algoritmo Mean Shift, sino que se tienen que recalcular para cada una de las imágenes de la secuencia. En el algoritmo Mean Shift el valor de la función de distribución sólo se recalcula en caso de que el elemento que estamos siguiendo modifique sensiblemente su tamaño, forma o color.

Ya hemos visto que ambos algoritmos emplean funciones de densidad de probabilidad. Estas funciones reciben el nombre de *backprojections*, y consisten en una imagen en escala de gris donde cada píxel representa la probabilidad de pertenecer al objeto que estamos siguiendo. Para calcular la probabilidad de cada píxel emplearemos el histograma del objeto a seguir, compararemos el valor del píxel en la imagen original con el valor de dicho histograma y asignaremos un valor proporcional a la similitud de ambos dentro de la imagen

*backprojection*. En el siguiente apartado explicaremos más a fondo este concepto.

#### 2.3.2.2- Histograma Backprojection

Como ya hemos comentado, las funciones de densidad de probabilidad con las que trabajan tanto el algoritmo CAMSHIFT como el Mean Shift se determinan utilizando un método que asocia el valor de cada píxel con la probabilidad de que dicho píxel pertenezca al objeto que estamos siguiendo. Un método muy común para realizar esto es el llamado "Histograma *Back-Projection*". Antes de continuar con la explicación de este nuevo concepto definiremos el concepto de histograma.

Definiremos **histograma de una imagen en escala de gris** como una función discreta en cuyo eje X se representan los colores de la gama de grises y en el eje Y se representa el porcentaje de píxeles que hay en la imagen con ese determinado color. De ese modo, podemos ver el histograma como una función de distribución de probabilidad que nos indica la probabilidad de que dentro de una imagen un píxel cualquiera sea de un determinado color. Para el caso de imágenes en color, en vez de trabajar con un único histograma se trabaja con un histograma multidimensional, es decir, tres histogramas correspondientes a cada uno de los planos en que se representa la imagen; por ejemplo, para el caso de utilizar una representación RGB, tendremos un histograma para el plano R, otro histograma para el plano G y otro para el plano B.

Una vez que ya tenemos claro el concepto de histograma, pasaremos al método Histograma Back-Projection. Empleando como base el histograma del objeto que estamos siguiendo, asociaremos a cada píxel de una imagen cualquiera el valor correspondiente del histograma asociado a su color, de tal forma que el resultado final será una imagen que se corresponderá con una función de distribución que nos indicará la probabilidad de que cada píxel pertenezca al objeto de interés. A continuación presentamos una figura explicativa del proceso.

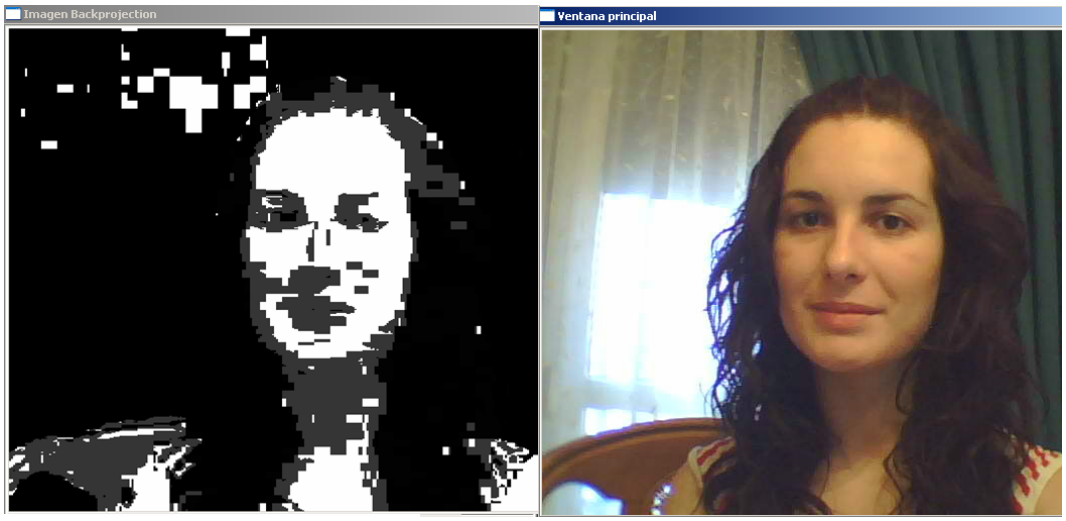


Figura 2.11.- Imagen Backprojection del tono de la piel

Cuando trabajamos con imágenes en color, en lugar de un único histograma tenemos 3 histogramas distintos. Utilizando el método Histograma Back-Projection en imágenes en color (como es nuestro caso) de cada una de las imágenes de la secuencia de vídeo obtenemos tres nuevas imágenes *backprojection*, por lo que la carga computacional se vería triplicada. Para evitar esto, el algoritmo CAMSHIFT no trabaja con histogramas multidimensionales, sino sólo con el histograma correspondiente al plano HUE (tono), empleando la representación HSV para las imágenes en color. Gracias a esto, de cada imagen original sólo obtendremos una imagen *backprojection* por lo que el algoritmo tendrá una carga computacional mucho más reducida. En el siguiente apartado justificaremos el por qué de la elección del plano H.

#### 2.3.2.3- Representación HSV de una imagen y aproximaciones en el modelo.

En este apartado vamos a detallar en primer lugar en qué consiste la representación HSV de una imagen. A continuación justificaremos la razón por la cual el modelo se ha basado en el plano H (hue) exclusivamente y mencionaremos también las aproximaciones que hemos realizado para obtener mejores resultados en el algoritmo.

El espacio HSV diferencia tres planos, el plano H correspondiente al tono o color, el plano S correspondiente a la saturación (cómo de concentrado está el color) y el plano V correspondiente al brillo.

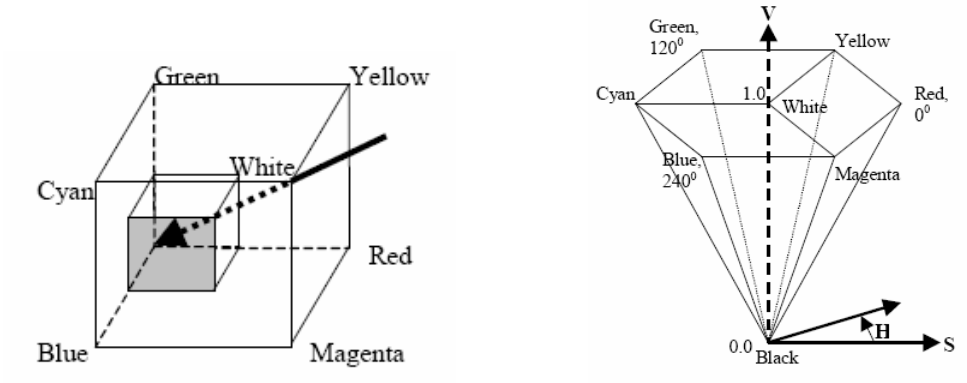


Figura 2.12.- Representación RGB (izquierda) y HSV (derecha) del color

Atendiendo a la figura anterior, la representación HSV (**H**ue – **S**aturation – **V**alue) se corresponde a la proyección del cubo RGB sobre su diagonal principal, que va del color blanco al color negro (ver imagen de la izquierda); esta proyección da como resultado el cono hexagonal de la imagen de la derecha. Si analizamos dicho cono, vemos que al movernos por el eje V obtenemos nuevo conos más pequeños correspondientes a subcubos menores en el espacio RGB asociados con colores cada vez más oscuros.

Un algoritmo pensado para el seguimiento de caras, como es el caso del algoritmo CAMSHIFT, empleará sólo el plano H, ya que dicho plano es suficiente para caracterizar el color de la piel. Es bastante común pensar que distintas razas requerirán histogramas del plano H distintos, sin embargo eso no es cierto, ya que excepto para los albinos, todos los humanos son del mismo tono; las personas con un color de piel más oscura simplemente tienen una mayor saturación del color piel, y ya hemos visto que el valor de la saturación pertenece a otro plano distinto dentro del espacio HSV.

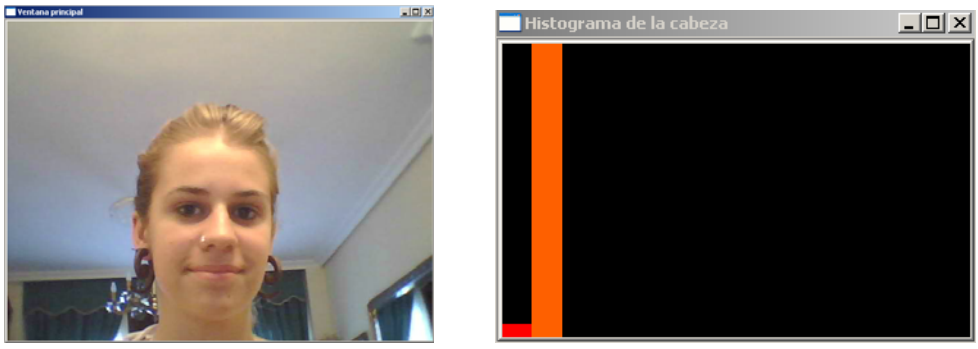


Figura 2.13.- Histograma de la piel de personas distintas (I)

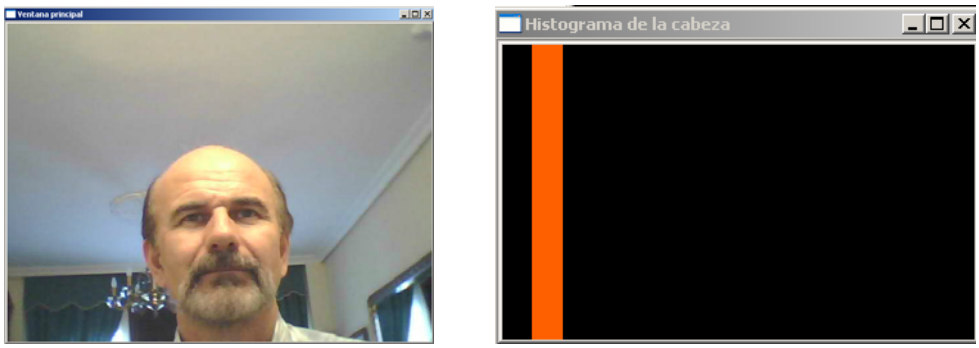


Figura 2.14.- Histograma de la piel de personas distintas (II)

Si analizamos el cono de la figura 2.12, cuando el valor del brillo es bajo, el valor de la saturación también es bajo (por la estructura del cono) haciendo que el plano Hue se vuelva muy ruidoso, ya que el tamaño del hexágono es demasiado pequeño para representar los cambios que se producen en el espacio RGB. Por otro lado, si hay zonas muy brillantes en la imagen, éstas pueden presentar un valor Hue similar al tono de la piel sin serlo. En base a esto, en el plano Hue ignoraremos aquellos píxeles con valores muy bajos o muy altos del brillo.

En un principio, el algoritmo intentó implementarse utilizando un histograma 2D con los valores *r* y *g* (red y green normalizados: r=R/(R+G+B), g = G/(R+G+B)). Sin embargo, este modelado era mucho más sensible a cambios de iluminación, ya que con esta representación el plano S (saturación) no se podía separar del plano H.

#### 2.3.2.4- Funcionamiento general del algoritmo

En este apartado vamos a introducir el diagrama de flujo en el que se basa el funcionamiento del algoritmo, así como todos los conceptos matemáticos necesarios para poder implementarlo. Por otro lado, debido a que el algoritmo Mean Shift es la base del método que estamos implementando, describiremos también de forma detallada en qué consiste.

El algoritmo CAMSHIFT puede resumirse en los siguientes pasos:

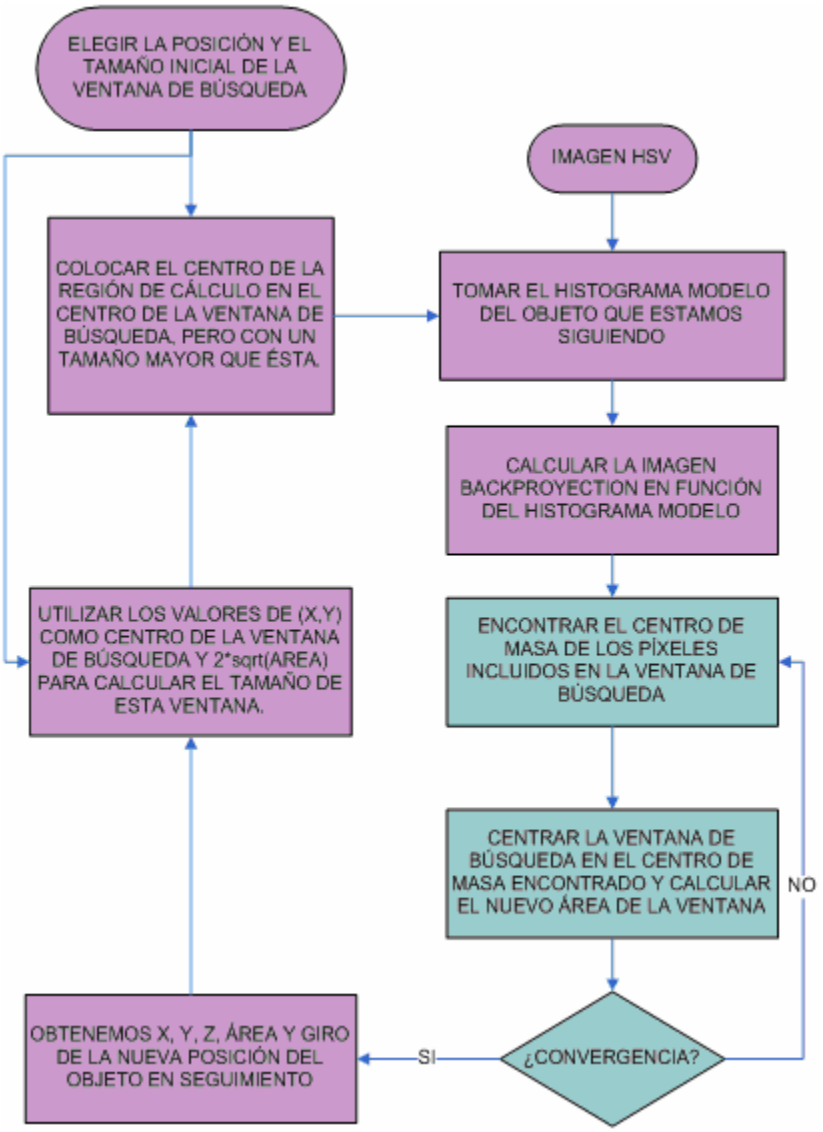


Figura 2.15.- Diagrama de flujo del algoritmo Camshift

Dentro del diagrama anterior, los bloques de color azul se corresponden con el corazón del algoritmo, es decir, el método Mean Shift. Los pasos que se siguen en este método son:

1. Elegir el tamaño de la ventana de búsqueda
2. Elegir la localización inicial de la ventana de búsqueda dentro de la función de distribución de color (backprojection).
3. Calculamos la posición del centroide de la ventana de búsqueda y guardamos el valor del momento de orden cero.
4. Situar la nueva ventana de búsqueda sobre el centroide y calcular su nuevo tamaño en función del momento de orden cero, ambos calculados en el paso 3.

5. Evaluación de la convergencia. Repetir los pasos 3 y 4 hasta que la posición de la ventana de búsqueda se mueva menos que una determinada cantidad preestablecida anteriormente como umbral.

Para imágenes *backprojection* (distribuciones de probabilidad de color) de dos dimensiones, la localización del centroide dentro de la ventana de búsqueda (paso 3 anterior) se calcula como sigue:

En primer lugar calculamos el momento de orden cero:

$$M_{00} = \sum_x \sum_y I(x,y) \qquad [2.15]$$

A continuación calculamos el momento de orden uno, tanto para *x* como para *y*:

$$M_{10} = \sum_x \sum_y x \cdot I(x,y) \qquad M_{01} = \sum_x \sum_y y \cdot I(x,y) \qquad [2.16] - [2.17]$$

Con el valor de estos tres parámetros el centroide dentro de la ventana de búsqueda se podrá computar como:

$$x_c = \frac{M_{10}}{M_{00}}; \quad y_c = \frac{M_{01}}{M_{00}} \qquad [2.18]$$

Donde *I(x,y)* es el valor del píxel en la posición *(x,y)*, es decir, la probabilidad de que dicho píxel forme parte del objeto al que corresponde el histograma patrón.

Al contrario que el método Mean Shift, que fue diseñado para distribuciones estáticas, el algoritmo Camshift se diseñó para distribuciones que pueden cambiar de forma dinámica. Esto ocurre, por ejemplo, cuando tratamos de seguir objetos que se mueven en secuencias de vídeo, de tal forma que el tamaño y la posición de la distribución en la imagen se ven modificadas a lo largo del tiempo. Por esa razón, el algoritmo Camshift ajusta el tamaño de la ventana de búsqueda para cada nuevo frame, basándose en la información extraída de los momentos de orden cero. El tamaño inicial de la ventana puede ser cualquier valor razonable, teniendo en cuenta que para distribuciones discretas, el menor valor permitido para el ancho o el alto de la ventana de búsqueda es tres.

Volviendo al diagrama de flujo de la figura 2.15, los pasos a seguir en el algoritmo Camshift son los siguientes:

1. Asignar la región de cálculo de la distribución de probabilidad a la imagen completa
2. Elegir la posición inicial de la ventana de búsqueda 2D del método Mean Shift.
3. Calcular la distribución de probabilidad de color en la región 2D centrada en la posición de la ventana de búsqueda y con un tamaño algo mayor que esta ventana.
4. Ejecutar el algoritmo Mean Shift para encontrar la nueva posición de la ventana de búsqueda. Almacenar el valor del momento de orden cero (área) y el centroide.
5. Para la siguiente imagen de la secuencia de vídeo, centrar la ventana de búsqueda en la posición del centroide calculado en el paso 4 y asignar como tamaño de esta ventana un valor que sea función del momento de orden cero. Volver al paso 3.

La siguiente figura muestra cómo el algoritmo Camshift encuentra el centroide de una distribución de probabilidad del color de la piel en cuya imagen asociada aparecen un rostro y las manos, y en la figura 2.17 se muestra este mismo ejemplo en la siguiente imagen de la secuencia, cuando la cara y las manos se han movido.

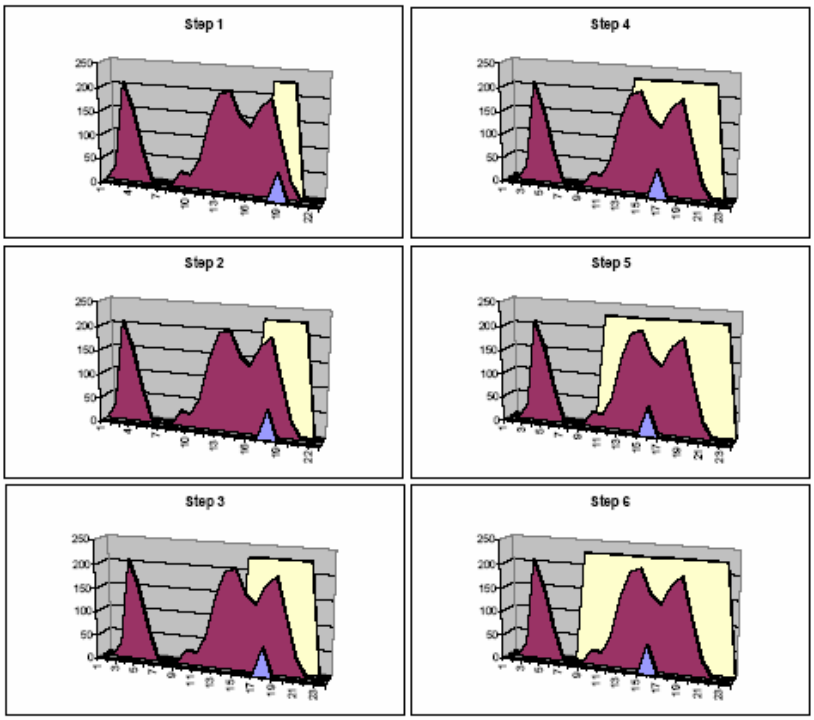


Figura 2.16.- Ejemplo algoritmo Camshift detectando una cara

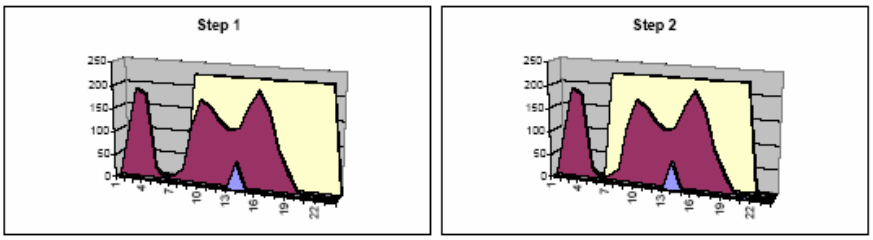


Figura 2.17.- Ejemplo algoritmo Camshift detectando una cara (II)

En la figura anterior, vemos que la convergencia se produce mucho más rápido, ya que comenzamos a partir de la ventana de búsqueda anterior.

Como ya hemos comentado en la breve explicación del algoritmo en el apartado G en el epígrafe 2.2.3.1, con este método no sólo buscamos calcular el área y la posición, sino también el ángulo de inclinación del objeto que estamos siguiendo. Para ello utilizaremos los momentos de segundo orden que se definen como:

$$M_{20} = \sum_x \sum_y x^2 \cdot I(x,y) \qquad M_{02} = \sum_x \sum_y y^2 \cdot I(x,y) \qquad [2.19] - [2.20]$$

Con estos momentos de segundo orden podremos calcular la orientación del objeto bajo seguimiento según la expresión [2.21], la cual nos indica cuál es la orientación del eje mayor.

$$\theta = \frac{\arctan\left(\dfrac{2 \cdot \left(\dfrac{M_{11}}{M_{00}} - x_c y_c\right)}{\left(\dfrac{M_{20}}{M_{00}} - {x_c}^2\right) - \left(\dfrac{M_{02}}{M_{00}} - {y_c}^2\right)}\right)}{2} \qquad [2.21]$$

Por último, los dos primeros autovalores, es decir, la máxima longitud '*l*' (length) y el ancho '*w*' (width) de la distribución de probabilidad del *blob* encontrado por Camshift se calcularán de la siguiente forma:

Si definimos los parámetros a, b y c como sigue:

$$a = \frac{M_{20}}{M_{00}} - {x_c}^2 \qquad b = 2 \cdot \left(\frac{M_{11}}{M_{00}} - x_c y_c\right) \qquad c = \frac{M_{02}}{M_{00}} - {y_c}^2 \qquad [2.22]$$

Entonces la longitud '*l*' y el ancho '*w*' desde el centroide de la distribución seguirán las siguientes expresiones:

$$l = \sqrt{\frac{(a+c)+\sqrt{b^2+(a-c)^2}}{2}} \qquad [2.23]$$

$$w = \sqrt{\frac{(a+c)-\sqrt{b^2+(a-c)^2}}{2}} \qquad [2.24]$$

La implementación de este algoritmo y del encargado del cálculo del flujo óptico que explicaremos a continuación se puede ver en el capítulo 4 de la memoria y el código se puede ver en el anexo C.

## 2.3.3.- Cálculo del flujo óptico.

Para conseguir un análisis más exhaustivo sobre el movimiento existente en una imagen empleando los métodos diferenciales (que son los más adecuados para nuestra aplicación) trabajaremos con algoritmos que calculan el **flujo óptico** entre dos imágenes.

### 2.3.3.1- Definición de flujo óptico

El concepto de flujo óptico es lo que utilizaremos para obtener una representación visual del movimiento de los objetos. En general, esta representación utilizará vectores de desplazamiento que nos indicarán la posición inicial y final de un determinado píxel dentro de dos imágenes consecutivas en una secuencia de vídeo. En un lenguaje más técnico, definiremos el flujo óptico como el movimiento aparente del brillo de una imagen.

Definiremos en primer lugar *I(x,y,t)* como el brillo de una imagen que cambia a lo largo del tiempo. En base a esto, se pueden hacer dos suposiciones:

1. El brillo *I(x,y,t)* depende muy poco de las coordenadas *x, y*, pero presenta una fuerte dependencia con el tiempo.
2. El brillo de cada píxel perteneciente a un objeto estático o en movimiento no cambia a lo largo del tiempo.

Analizando un objeto determinado (o algún punto de ese objeto) dentro de una imagen, transcurrido un tiempo *dt*, la posición del objeto vendrá dada por

*(dx, dy)*. Usando el desarrollo de Taylor para la función *I(x,y,t)* se obtiene lo siguiente:

$$I(x+dx, y+dy, t+dt) = I(x,y,t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt + ... \qquad [2.25]$$

Donde "..." son los términos de orden superior. Según [2.25] y en base a la segunda suposición podremos decir que:

$$I(x+dx, y+dy, t+dt) = I(x,y,t) \qquad [2.26]$$

$$\frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt + ... = 0 \qquad [2.27]$$

Si dividimos [2.27] entre *dt* y definimos:

$$\frac{dx}{dt} = u \qquad \frac{dy}{dt} = v \qquad [2.28]$$

Obtendremos la siguiente ecuación:

$$-\frac{\partial I}{\partial t} = \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v \qquad [2.29]$$

Conocida como **ecuación de restricción del flujo óptico** (o ecuación de restricción del cambio de brillo o problema de apertura del flujo óptico) y donde *u* y *v* son las componentes *x* e *y* respectivamente del campo del flujo óptico. Teniendo en cuenta que la ecuación [2.29] tiene más de una solución necesitaremos aplicar otras restricciones. Los distintos algoritmos existentes en la actualidad para el cálculo del flujo óptico se diferencian entre sí en base a las restricciones extras que aplican a la ecuación anterior para darle solución.

Algunos de estos métodos son: Técnica de Lucas & Kanade, técnica de Horn & Schunck o la técnica de concordancia de bloques (Block matching). De todos ellos hemos decidido utilizar la técnica de Lucas & Kanade por ser actualmente el método diferencial para el cálculo de flujo óptico más utilizado.

#### 2.3.3.2- Algoritmo de Lucas & Kanade

Si utilizamos la ecuación del flujo óptico [2.29] y la aplicamos sobre un conjunto de píxeles adyacentes asumiendo que todos ellos se mueven a la

misma velocidad, entonces la tarea del cálculo del flujo óptico queda reducida a la resolución de un sistema lineal. Veámoslo de forma matricial:

Extendemos primero el resultado obtenido a un caso general de cuatro dimensiones donde el brillo viene expresado por *I(x,y,z,t)*, y a continuación expresaremos la ecuación del flujo óptico como:

$$I_x V_x + I_y V_y + I_z V_z = -I_t \qquad [2.30]$$

O lo que es igual:

$$\nabla I \cdot \vec{V} = -I_t \qquad [2.31]$$

Como ya hemos dicho, la restricción que aplicaremos para resolver el problema es asumir que el flujo óptico es constante en una ventana de tamaño $m \times m \times m$, donde $m$>1 y cuyo centro viene dado por las coordenadas (x,y,z). Si enumeramos los píxeles de 1..*n*, nos encontramos con el siguiente conjunto de ecuaciones:

$$\begin{aligned} &I_{x1} V_x + I_{y1} V_y + I_{z1} V_z = -I_{t1} \\ &I_{x2} V_x + I_{y2} V_y + I_{z2} V_z = -I_{t2} \\ &\vdots \\ &I_{xn} V_x + I_{yn} V_y + I_{zn} V_z = -I_{tn} \end{aligned} \qquad [2.32]$$

Como podemos ver, hay más de tres ecuaciones para encontrar las tres incógnitas *V*, lo que significa que el sistema está sobredeterminado. Expresado matricialmente:

$$\begin{bmatrix} I_{x1} & I_{y1} & I_{z1} \\ I_{x2} & I_{y2} & I_{z2} \\ \vdots & \vdots & \vdots \\ I_{xn} & I_{yn} & I_{zn} \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = \begin{bmatrix} -I_{t1} \\ -I_{t2} \\ \vdots \\ -I_{tn} \end{bmatrix} \qquad [2.33]$$

O lo que es igual:

$$A\vec{v} = -b \qquad [2.34]$$

Para resolver sistemas de ecuaciones sobredeterminados utilizaremos el método de los **mínimos cuadrados**, cuya solución se expresa como:

$$A^T A\vec{v} = A^T(-b) \qquad \Rightarrow \qquad \vec{v} = (A^T A)^{-1} A^T (-b) \qquad [2.35]$$

Y matricialmente:

$$\begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = \begin{bmatrix} \sum I_{x_i}{}^2 & \sum I_{x_i} I_{y_i} & \sum I_{x_i} I_{z_i} \\ \sum I_{x_i} I_{y_i} & \sum I_{y_i}{}^2 & \sum I_{y_i} I_{z_i} \\ \sum I_{x_i} I_{z_i} & \sum I_{y_i} I_{z_i} & \sum I_{z_i}{}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum I_{x_i} I_{t_i} \\ -\sum I_{y_i} I_{t_i} \\ -\sum I_{z_i} I_{t_i} \end{bmatrix} \quad [2.36]$$

El concepto anterior puede aplicarse también para dos píxeles únicamente, obteniendo una solución única para el sistema. Sin embargo, combinando ecuaciones asociadas a más de dos píxeles el algoritmo es más efectivo y se obtienen mejores resultados.

Dentro de la ventana que agrupa los píxeles, se suele dar mayor importancia al píxel del centro de la ventana (se ponderan los píxeles), para lo cual utilizaremos las funciones *W(x,y)*. Estas funciones pueden ser de muy diversa naturaleza, sin embargo las más populares son las distribuciones Gaussianas. A continuación mostramos las ecuaciones para un sistema linear 2x2:

$$\sum_{x,y} W(x,y) I_x I_y u + \sum_{x,y} W(x,y) I_y{}^2 v = -\sum_{x,y} W(x,y) I_y I_t \quad [2.37]$$

$$\sum_{x,y} W(x,y) I_x{}^2 u + \sum_{x,y} W(x,y) I_x I_y v = -\sum_{x,y} W(x,y) I_x I_t \quad [2.38]$$

Uno de los principales inconvenientes del algoritmo Lucas-Kanade es que no es capaz de gestionar una densidad de vectores de flujo muy grande, haciendo que la información extraída del flujo se disipe rápidamente en las zonas límites del área en movimiento y haciendo que en las partes interiores de áreas grandes y homogéneas se muestren desplazamientos muy pequeños, es decir, que no se detecte el movimiento bien.

Por otro lado, su principal ventaja es que es el algoritmo más robusto de todos en presencia de ruido.