UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

I.T.Telecomunicación. Sonido e Imagen

Proyecto Fin de Carrera

TRACKING AUTOMÁTICO DE OBJETOS EN SECUENCIAS DE IMÁGENES USANDO EL FILTRO DE KALMAN

Autor: María Isabel Menor Flores

Fdo.:

Director: Pedro M. Núñez Trujillo

Fdo.:

Tribunal Calificador

Presidente:

Fdo.:

Secretario:

Fdo.:

Vocal:

Fdo.:

CALIFICACIÓN:

FECHA:

A mis padres, Luis y Juana
y a mi hermano Manuel
por su apoyo incondicional
y por sus constantes muestras de cariño.

Índice General

Capítulo 5: Conclusiones y trabajo futuro

Capítulo 6: Anexo

Índice de figuras

Agradecimientos

Resumen

La visión artificial, también conocida como visión por computador, es un subcampo de la inteligencia artificial, cuyo propósito es la extracción automática de información del mundo físico a partir de imágenes.

Hoy por hoy, es una disciplina ampliamente utilizada en múltiples aplicaciones, debido no sólo a los avances tecnológicos que se han experimentado, sino también a su gran expansión en la vida cotidiana. Se trata, por tanto, de un área multidisciplinar relacionada con tratamiento de imágenes, reconocimiento de patrones, fotogrametría, gráficos por ordenador, inteligencia artificial, geometría proyectiva, teoría de control, etc.

En este proyecto nos centraremos en el seguimiento o *tracking* de objetos en movimiento, utilizando para ello secuencias de imágenes reales; ya sean recogidas en un archivo audiovisual o capturadas directamente con algún tipo de sensor. Para llevar a cabo el tratamiento de estas imágenes y, por tanto, poder realizar el seguimiento de los objetos en movimiento, implementaremos un algoritmo utilizando el software Visual C++ y las funciones para el tratamiento de imágenes que proporciona la librería OpenCv (Intel Open Source Computer Vision Library).

Por otro lado, esta aplicación puede ser desarrollada utilizando distintos métodos, como son el *Filtro de Kalman* o el *Filtro de Partículas*.

El *Filtro de Kalman* está definido por un conjunto de ecuaciones matemáticas que proporcionan una solución recursiva eficiente del método de mínimos cuadrados; mientras que el *Filtro de Partículas* está basado en el método secuencial de Monte Carlo, el cual se puede emplear en cualquier transición de estados o modelo de medida.

En definitiva, el objetivo de este proyecto es la implementación del *Filtro de Kalman* aplicado al seguimiento de objetos en movimiento en secuencias de imágenes en 2D. Este filtro es una técnica recursiva que nos permite determinar correctamente los parámetros de un sistema que evoluciona con el tiempo. De este modo, dados unos estimadores iniciales y

los propios parámetros del sistema dinámico, el filtro va prediciendo y autoajustándose con cada nueva medida.

Otro objetivo añadido es la evaluación de los resultados obtenidos en distintas condiciones experimentales, como por ejemplo, oclusión, modificación de parámetros del filtro, movimientos no plano paralelos, etc.

Además de estudiar su funcionamiento bajo distintas condiciones experimentales, también se implementará la aplicación para que realice el seguimiento de varios objetos móviles. Otra particularidad añadida es que el sistema permite seleccionar el objeto a seguir mediante clickeo del ratón o por sustracción de fondo.

Finalmente, tras los resultados obtenidos bajo distintas condiciones experimentales, así como en el seguimiento de varios objetos móviles se comprueba que el rendimiento del filtro es el adecuado, teniendo especial interés los resultados obtenidos en tiempo real, puesto que se trata de aplicaciones reales y poseen mayor funcionalidad.

Capítulo 1: Introducción

El tema principal de este proyecto es la visión artificial y su aplicación en el seguimiento de objetos en movimiento en secuencias de imágenes en 2D, mediante la utilización del *Filtro de Kalman*.

Como ya he mencionado, la visión artificial es una disciplina cuyo propósito es la extracción e interpretación automática de información del mundo físico a partir de imágenes, utilizando para ello un computador. Es decir, esta disciplina extrae información sobre brillo, colores, formas, etc. de las imágenes sobre las que actúa, las cuales pueden ser imágenes estáticas, escenas tridimensionales o imágenes en movimiento.

En definitiva, la visión artificial permite deducir automáticamente las estructuras y propiedades de un mundo tridimensional, posiblemente dinámico, a partir de una o varias imágenes bidimensionales de él. Su objetivo es el de proveer el sentido de la vista a una máquina o computador a partir de imágenes tomadas con algún tipo de sensor, que junto a otros mecanismos como el aprendizaje sea capaz de detectar y ubicar objetos en el mundo real. [1]

Además la gran ventaja de los sistemas basados en la visión artificial es que actualmente el coste de los componentes requeridos para desarrollarlos (cámaras, procesadores, etc.) se ha reducido drásticamente; mientras que las prestaciones han ido aumentando de manera espectacular. De esta forma podemos disponer de un sistema de alta velocidad y resolución capaz de conseguir elevados ratios de inspección, requiriendo para ello una baja inversión.

Por otro lado, un sistema de visión artificial [2] consta generalmente de cuatro fases o etapas:

- Adquisición de la imagen: se trata de una fase en la cual se realiza la captura de las imágenes mediante algún tipo de sensor.
- Preproceso: en esta fase se realiza el tratamiento digital de las imágenes, con el objetivo de mejorar la calidad para ser utilizadas en etapas posteriores. Es decir, mediante filtros o transformaciones geométricas se

eliminan partes indeseables o se realzan partes interesantes de las imágenes.

- Segmentación: es la etapa donde la imagen se divide en regiones homogéneas que se corresponden con los objetos contenidos en ellas. De esta forma se aislan los objetos que interesan de la imagen.

- Extracción de características: en esta etapa se obtienen medidas de características de los objetos segmentados, como por ejemplo, color, textura, forma, etc.

- Reconocimiento o clasificación: consiste en clasificar los objetos a partir de las características extraídas para distinguirlos.

El siguiente diagrama muestra las distintas etapas en las que se basa un sistema de visión artificial, anteriormente citadas:

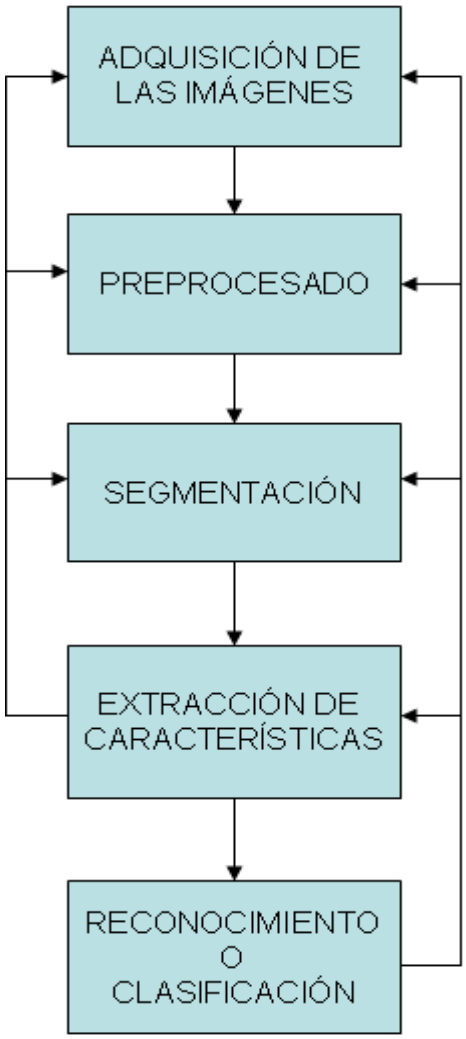

Figura 1. Etapas de la visión artificial.

Como se puede apreciar en el diagrama anterior, si en algún caso una de las etapas falla se vuelve a la etapa anterior; es decir, se realimenta hacia atrás, volviendo a repetirse la fase anterior y siguiendo después el proceso secuencialmente con las etapas posteriores.

Algunas de las aplicaciones más importantes de la visión artificial son:

- Sistemas de inspección visual automática.
- Aplicaciones médicas.
- Reconocimiento biométrico.
- Tracking o seguimiento de objetos.
- Análisis de terrenos.
- Robótica.
- Videovigilancia/ Aplicaciones militares.
- Recuperación de imágenes por contenido.
- Domótica.

1.1 Organización de la memoria

El presente proyecto se ha estructurado por capítulos, y éstos a su vez, en sucesivos apartados. Concretamente, en este apartado, se expone un pequeño resumen sobre el contenido de cada uno de ellos.

En el capítulo primero, se realiza una introducción sobre la visión artificial, comentando su utilidad y principales aplicaciones, para luego profundizar en la aplicación de tracking o seguimiento de objetos en movimiento, puesto que es la base de este proyecto. También se indican los objetivos que se pretenden alcanzar, así como una descripción detallada del *Filtro de Kalman*, estimador que utilizamos para llevar a cabo el tracking. Por último se incluyen las distintas herramientas y librerías utilizadas para la implementación de la aplicación.

En el segundo capítulo, se exponen brevemente, de forma descriptiva, cada una de las etapas que se deben seguir para el desarrollo de esta aplicación. Se indican las técnicas de procesado utilizadas y el método para

determinar la posición observada del objeto de interés, a la que se aplicará luego el algoritmo del *Filtro de Kalman*.

En el tercer capítulo, se describe cómo se ha llevado a cabo la implementación de esta aplicación. Se explican cada una de las clases creadas, así como el programa principal. También se indican los métodos que incluyen y los parámetros y variables que utilizan cada uno de ellos.

En el cuarto capítulo, se muestran las pruebas y experimentos realizados. Es el capítulo en el que se comprueba el rendimiento del sistema bajo distintas condiciones experimentales. Para cada prueba se muestran, de forma ilustrativa, los resultados obtenidos y posteriormente se añaden las conclusiones obtenidas a partir de ellos.

El quinto capítulo incluye las conclusiones extraídas tras la finalización del proyecto realizado y las posibles mejoras de la aplicación como trabajo futuro.

En el sexto y último capítulo se menciona el conjunto de funciones de OpenCv que se han utilizado y las funciones que se han desarrollado durante la implementación del sistema. Para cada una de ellas se realiza una breve descripción. También contiene las referencias seguidas en la elaboración de este proyecto fin de carrera y el procedimiento que se ha seguido en la instalación de la librería de OpenCv.

1.2 Tracking o seguimiento de objetos en movimiento

El tracking o seguimiento de objetos en movimiento es una de las aplicaciones comunes del análisis de múltiples imágenes, pertenece pues al campo de aplicación de la visión artificial. [3]

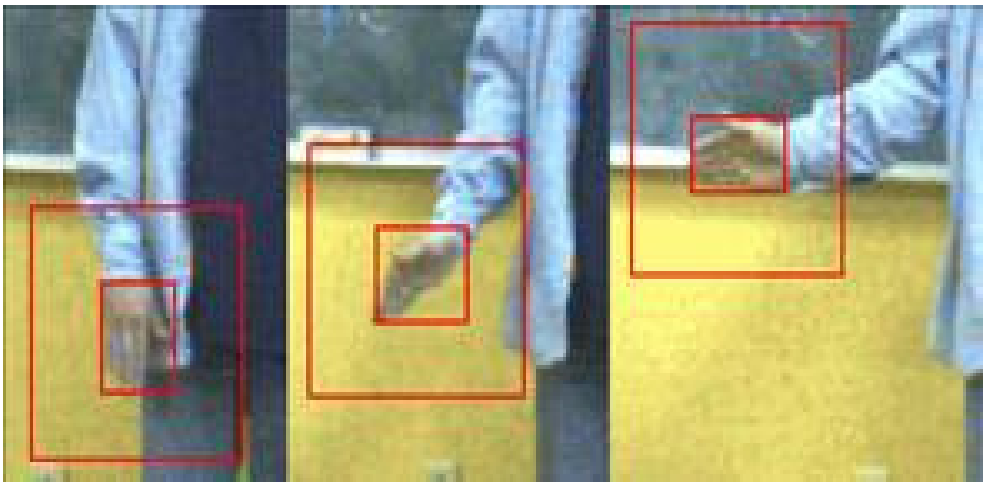

Figura 2. Ejemplo de tracking de una mano en movimiento.

En la actualidad, debido al creciente número de vehículos en las carreteras y autopistas, se hace cada vez más importante controlar el flujo de vehículos en movimiento mediante la aplicación de sistemas informáticos que utilicen algoritmos de visión y de procesamiento de imágenes. Se trata de un claro ejemplo de aplicación actual de este tipo de sistemas.

Además estos sistemas, como ya he citado con anterioridad, tienen la gran ventaja de que actualmente el coste de los componentes requeridos para desarrollarlos se ha reducido mucho; mientras que las prestaciones han ido aumentando exponencialmente, lo que pone a nuestro alcance sistemas con una elevada potencia de bajo coste.

Este tipo de aplicaciones consisten básicamente en determinar la posición y la velocidad de un punto, región, objeto o cualquier parte de interés de la imagen, dada su posición y velocidad en una secuencia anterior de imágenes. [4] [5]

El seguimiento se puede realizar en base a modelos tridimensionales de objetos, características significativas de la imagen, modelos deformables, regiones, etc.

El problema básico es integrar la información de varias imágenes dentro de una secuencia. Normalmente se considera que pasa poco tiempo entre las imágenes, de forma que imágenes consecutivas son similares.

Por este motivo, para llevar a cabo este seguimiento se precisa de estimadores eficientes como pueden ser, por ejemplo, el *Filtro de Kalman* o el *Filtro de Partículas*.

1.2.1 Diferencias entre el *Filtro de Kalman* y el *Filtro de Partículas*

Una de las diferencias entre ambos filtros es que el *Filtro de Partículas* [10] es una poderosa herramienta para el tratamiento de procesos del mundo real, sin la necesidad de hacer suposiciones sobre las características intrínsecas del proceso, tal y como hace el *Filtro de Kalman*. [9]

Por otro lado, el *Filtro de Kalman* está definido por un conjunto de ecuaciones matemáticas que proporcionan una solución recursiva eficiente del método de mínimos cuadrados; mientras que el *Filtro de Partículas* está basado en el método secuencial de Monte Carlo, el cual se puede emplear en cualquier transición de estados o modelo de medida. [13]

Además, el *Filtro de Kalman* [8] consta de 2 fases significativas: predicción del estado y corrección, aunque previamente es necesario inicializar los parámetros del filtro y después de la corrección se realiza una actualización para volver a repetir el proceso. En cambio, el *Filtro de Partículas* [11] consta de 5 fases: inicialización o lanzamiento de partículas aleatorias, ponderación, estimación, selección y difusión, que son distintas a las del *Filtro de Kalman.*

También podría decir que la gran ventaja del *Filtro de Kalman* es que evita la influencia de posibles cambios estructurales en la estimación, ya que intenta estimar una trayectoria estocástica de los coeficientes. Se distingue, por tanto, por su habilidad para predecir el estado de un modelo en el pasado, presente y futuro, incluso cuando la naturaleza del sistema modelado es desconocida. En contraposición, algunas desventajas son que requiere de condiciones iniciales de la media y varianza del estado para iniciar el algoritmo recursivo y cuando se emplea en modelos autorregresivos, al depender de la información de instantes pasados, es eficiente a corto plazo. [13]

En cuanto al *Filtro de Partículas*, una de sus ventajas es que permite centrar la atención en una región de la imagen donde es probable encontrar al objeto, despreciando la exploración del resto. Por el contrario, tiene la desventaja de que proporciona peores resultados que el *Filtro de Kalman* en situaciones óptimas. [11]

En definitiva, la principal diferencia radica en que el *Filtro de Kalman* sólo puede representar la estimación del estado por una gaussiana unimodal, frente al *Filtro de Partículas* que puede representar densidades multimodales complejas empleando una gran cantidad de partículas aleatorias.

1.3 Objetivos

El objetivo principal de este proyecto es implementar un algoritmo en Visual C++ que nos permita aplicar el *Filtro de Kalman* al tracking o seguimiento de objetos en movimiento en imágenes de 2D. El *Filtro de Kalman*, como ya he descrito anteriormente, es una técnica recursiva que nos permitirá determinar los parámetros de un sistema que evoluciona con el tiempo. De esta forma, a partir de unos estimadores iniciales y los propios parámetros del sistema, el filtro irá prediciendo y actualizándose con cada nueva medida.

Por otro lado, también se evaluarán los resultados obtenidos bajo diferentes condiciones experimentales, como son oclusión, modificación de parámetros del filtro, movimientos no plano paralelos, etc. Además, se implementará la aplicación para el seguimiento de varios objetos móviles, analizando también la respuesta del filtro y los resultados obtenidos.

Para llevar a cabo nuestro propósito, se utilizarán secuencias de imágenes reales, recogidas en un archivo audiovisual o capturadas con algún tipo de sensor y las funciones disponibles en la librería OpenCv (Intel Open Source Computer Vision Library) [6] para el tratamiento de imágenes.

Por lo tanto, para llegar a la consecución de estos objetivos deberán cumplirse previamente los siguientes pasos:

1. Familiarizarse con el entorno Visual C++, estudiar su funcionamiento y comprender la programación orientada a objetos en C++.
2. Estudiar y esquematizar cada una de las fases que se deben llevar a cabo para realizar tracking de un objeto en movimiento. [1] [2] [3] [4] [5]
3. Comprender el funcionamiento del *Filtro de Kalman*, estudiar sus ecuaciones y distintos parámetros. [8] [13]
4. Incluir la librería OpenCv en el proyecto de Visual C++, familiarizarse con ella y comprender las distintas funciones que permiten el tratamiento de imágenes. [7]
5. Coleccionar distintas secuencias de imágenes para evaluar la respuesta del sistema bajo diferentes condiciones experimentales.
6. Implementar el algoritmo en C++, mediante la programación orientada a objetos. Este algoritmo debe desarrollar cada una de las fases del tracking, así como la aplicación del *Filtro de Kalman*.
7. Otros objetivos añadidos son que la implementación del algoritmo también permita el seguimiento de varios objetos móviles y dé la posibilidad de seleccionar el objeto a seguir mediante clickeo del ratón o mediante sustracción de fondo.
8. Estudiar y analizar los resultados obtenidos en los distintos y casos y bajo las diferentes condiciones experimentales.

A continuación muestro un diagrama en el que se puede comprobar el tiempo empleado en cada una de las fases hasta la finalización del proyecto:

AÑADIR DIAGRAMA DE GANTT.

1.4 Estado del arte

En este apartado se explicará de forma descriptiva el funcionamiento del *Filtro de Kalman*, indicando cada una de sus fases y la función que desempeña cada una de ellas. [8]

1.4.1 Filtro de Kalman

El *Filtro de Kalman* es un algoritmo recursivo que fue desarrollado por Rudolf E. Kalman en 1960 en el contexto de la teoría del control [12]. A partir del documento de Kalman, en el que se describe una solución recursiva para el problema de filtrado lineal de datos discretos, este algoritmo ha presentado una extensa investigación y aplicación.

Una de las aplicaciones más destacadas es la que se desarrolla en el presente proyecto, basada en el tracking o seguimiento de objetos en movimiento en imágenes de 2D.

Este filtro es un procedimiento matemático que opera de acuerdo a un mecanismo de predicción y corrección. Es decir, es un algoritmo que pronostica el estado siguiente a partir de su estimación previa y le añade una corrección que es proporcional al error de predicción, de tal forma que este último es minimizado estadísticamente. [13]

Se trata, por tanto, de una técnica de estimación Bayesiana utilizada para seguir sistemas estocásticos dinámicos observados mediante sensores ruidosos.

Aunque se basa en dos fases principales de predicción y corrección, previamente es necesario inicializar sus parámetros y después de hacer la corrección hay que actualizar para volver a predecir, y así sucesivamente.

A continuación explico brevemente cada una de estas fases:

- Inicialización de los parámetros: en esta fase las matrices que intervienen en cada una de las ecuaciones del algoritmo se inicializan con sus valores correspondientes. El estado actual toma inicialmente el valor del

centro de masas del objeto de interés y la matriz covarianza del error inicial se toma como matriz identidad.

- Predicción: realiza la proyección del estado actual hacia el futuro, tomando como referencia el estado anterior. También se hace lo mismo con la matriz de covarianza del error.
- Corrección: incorpora nueva información a la estimación a priori para mejorar la estimación del estado a posteriori, disminuyendo el ruido que se produce en la medida.

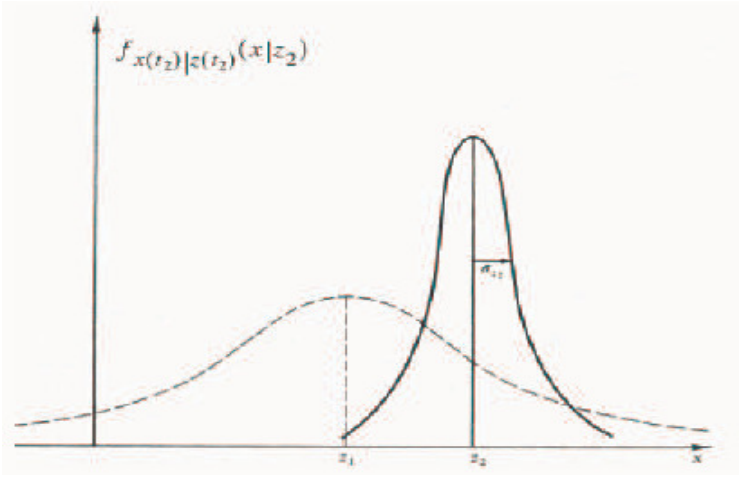

Figura 3. Etapa corrección. Filtro de Kalman.

- Actualización: las estimaciones del estado y la covarianza obtenidas a posteriori se incorporan a la fase de predicción, para volver a realizar el proceso de nuevo. De esta forma se crea un proceso realimentado.

El diagrama de bloques que resume todo el proceso se expone a continuación:

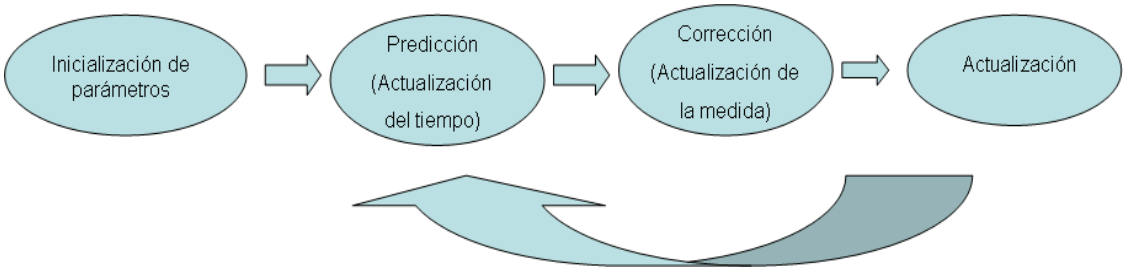

Figura 4. Fases del Filtro de Kalman básico.

1.5 Herramientas y librerías utilizadas

Las herramientas y librerías que se han utilizado para el desarrollo de este proyecto son:

- Visual C++.
- Librería OpenCv (Intel Open Source Computer Vision Library).
- Cámara digital Sony DCR/HC18E.

1.5.1 Visual C++

Visual C++ es un entorno de desarrollo de software, destinado para crear programas para sistemas operativos Windows en C++. Soporta tanto el lenguaje de programación C++ como el C. Además este entorno permite la utilización de múltiples herramientas como los lenguajes citados anteriormente, la programación orientada a objetos (POO) conjuntamente con el sistema de desarrollo SDK (también denominado API), biblioteca de clases, tecnología de componentes, etc.

Incluye las siguientes herramientas de desarrollo:

- Editor de texto.
- Compilador/ Enlazador.
- Depurador.
- Visor de datos y dependencias (browser).

Uno de sus principales objetivos es programar para Windows en C++ utilizando MFC (Microsoft Foundation Class Library), que consiste en una librería de clases de C++ que permiten crear y gestionar de manera intuitiva componentes típicos de Windows. Luego la MFC es una implementación que utiliza el API encapsulando todas las estructuras y llamadas a funciones en objetos fáciles de utilizar.

El objetivo del presente proyecto es utilizar este entorno para crear un algoritmo mediante la programación orientada a objetos en C++.

El lenguaje de programación C++ es un superconjunto de C orientado a objetos, luego permite la creación y manipulación de estos objetos. En esto último se basa la programación orientada a objetos y los mecanismos básicos que utiliza son:

- Objetos: entidad con atributos o datos particulares sobre los que se puede operar mediante métodos o funciones miembro.
- Mensajes: se corresponden con el nombre de los métodos de un objeto. Al aplicarlos a los objetos responden con la ejecución del código de la función asociada.
- Métodos: se implementan dentro de los objetos y determinan cómo tienen que actuar cuando se produce el mensaje asociado. Son la definición de la función miembro del objeto.
- Clases: definición de un tipo de objetos.

1.5.2 Librería OpenCv

OpenCv significa Open Source Computer Vision Library, o lo que es lo mismo, librería de visión por ordenador de código abierto. Fue desarrollada por Intel Corporation y contiene un conjunto de utilidades de procesamiento de imágenes, visión artificial, captura de video y visualización de imágenes. Es de código abierto, gratuita, rápida, fácil de usar y está en continuo desarrollo.

Esta librería es independiente de la plataforma en la cual se ejecute, aunque está optimizada para los procesadores de Intel, su patrocinador.

Además es una librería que incluye más de 350 algoritmos que cubren las siguientes áreas:

- Análisis de movimiento y seguimiento de objetos.

- Análisis de imágenes.
- Análisis de estructuras.
- Reconocimiento de objetos.
- Calibración de cámara y reconstrucción en 3D.
- Funciones experimentales.
- Estructuras y operadores básicos.
- Interfaz gráfica y adquisición.

También contiene una gran cantidad de funciones en C [7] y clases en C++ y puede utilizar ciertas librerías externas en tiempo de ejecución, aunque sea independiente.

Es una librería que implementa una gran variedad de herramientas para la interpretación de la imagen. Por este motivo, se ha utilizado en la implementación de este proyecto, ya que es una librería de algoritmos aplicables al seguimiento o tracking, así como a la segmentación y reconocimiento de objetos.

1.5.3 Cámara Digital Sony DCR/HC18E

El propósito de utilizar esta cámara es el de conseguir videos con una mayor calidad, proporcionando de esta manera, una mayor facilidad en la interpretación de los resultados y una mejor respuesta del sistema.

Capítulo 2: Descripción del sistema

El presente capítulo describirá el sistema utilizado para realizar el tracking o seguimiento de un objeto de interés. Este tracking será realizado en este proyecto fin de carrera usando un proceso estocástico comúnmente utilizado en la comunidad científica, el *Filtro de Kalman*. Como ya se ha comentado en el capítulo anterior, el Filtro de Kalman aplicado al seguimiento de objetos en movimiento en imágenes reales permite obtener una buena estimación de la posición del objeto de interés. Pero, para ello, es necesario realizar una serie de procesos sobre la imagen para extraer características de la misma que nos indiquen dónde se encuentra el objeto, lo que se tomará como la posición observada del objeto. Para poder determinar esta posición, que denominaremos centros de masas se han utilizado dos técnicas de seguimiento:

- Segmentación basada en umbralizado.
- Sustracción de fondo.

Ambas técnicas serán explicadas a continuación.

2.1 Segmentación basada en umbralizado

La segmentación es el proceso que permite dividir la imagen en regiones homogéneas que comparten propiedades de brillo o color, por ejemplo, con el objetivo de facilitar su posterior análisis y reconocimiento. [3]

Se trata de una técnica que no sigue unas reglas estrictas, sino que depende de cada problema particular; siendo necesario en ocasiones, crear técnicas a medida. Además, puede resultar compleja si no se tiene información adecuada sobre la imagen a procesar, la cual además suele estar influenciada por el ruido. [2]

Concretamente, la segmentación por umbralizado, tiene en cuenta el valor de intensidad de los píxeles para determinar si éstos pertenecen al objeto de interés o no. Es un proceso que nos permite convertir una imagen en escala de grises o en color en una imagen binaria, de 0 y 1, etiquetando el objeto de interés con un valor distinto al fondo. [14]

En el caso de una imagen en escala de grises para conseguir este objetivo se realiza un análisis del histograma de la imagen; es decir, se realiza una búsqueda de niveles de intensidad de los píxeles en dicho histograma.

Si los valores de intensidad de los píxeles del objeto están claramente diferenciados con respecto a los del fondo, al establecer un umbral determinado, obtendremos un histograma bimodal con dos máximos distintos, separados por una región vacía. En este caso se conseguiría una separación perfecta del objeto respecto del fondo.

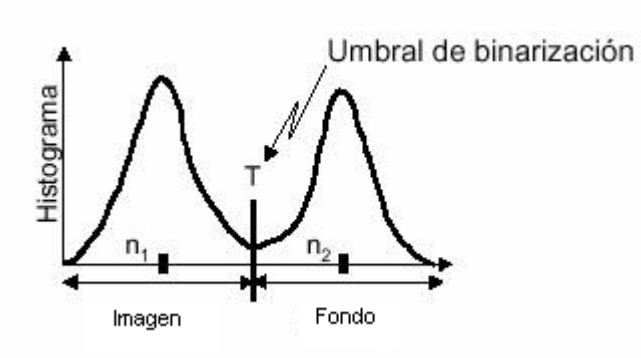

Figura 5. Histograma bimodal.

La técnica explicada anteriormente se conoce como umbralización binaria o binarización. El éxito de este método estará en cómo se divida el histograma y se basa en establecer un umbral simple T, que generalmente se selecciona a partir de dicho histograma.

Matemáticamente se puede expresar de la siguiente manera:

$$g(x,y) = \begin{cases} 1 & \text{si } f(x,y) > T \quad \text{Objeto} \\ 0 & \text{si } f(x,y) \leq T \quad \text{Fondo} \end{cases}$$

En el caso de que el objeto sea oscuro respecto del fondo la asignación anterior se haría a la inversa.

Sin embargo, la selección automática del umbral es una tarea complicada, ya que el histograma de la imagen no siempre es bimodal. En ese caso sería necesario combinar la información espacial presente en la imagen con la información referente al nivel de gris. Se trata de una tarea decisiva para conseguir una adecuada segmentación del objeto de interés.

Para las imágenes en color se puede aplicar la técnica anterior, con la diferencia de que tendremos tres canales, uno para cada componente de color, en lugar de uno, como ocurre en escala de grises. El procesado también se realiza teniendo en cuenta el histograma de la imagen para evaluar el nivel de los píxeles y poder determinar el umbral que diferenciará el objeto del resto de la imagen.

En cuanto a nivel de procesado resulta más eficiente evaluar sólo un canal a evaluar tres canales, ya que el tiempo requerido para ello será menor. Esta técnica será utilizada en la sustracción de fondo, explicada con detalle en el apartado siguiente.

En nuestro proyecto utilizamos imágenes en color RGB, por lo que aplicaremos la técnica de segmentación basada en umbralizado para imágenes en color. Para ello hemos tenido en cuenta un rango de valores de sensibilidad para el color a umbralizar. Se tendrá un valor de sensibilidad para cada componente RGB de la imagen.

A la hora de determinar los valores de sensibilidad para cada color se ha establecido mayor rango para la componente RGB correspondiente al color que se desea umbralizar y menos rango para las componentes que no aparecen o que no se desean segmentar.

Luego para aplicar esta técnica lo que hacemos es recorrer la imagen y comprobar que las componentes RGB de cada píxel están dentro de unos rangos de sensibilidad. Si es así, el píxel se etiqueta con valor 1 sino con valor 0. Además se ha desarrollado de tal forma que sea posible seleccionar con el ratón el color del objeto que se desea umbralizar.

Un ejemplo de aplicación de esta técnica se muestra a continuación:

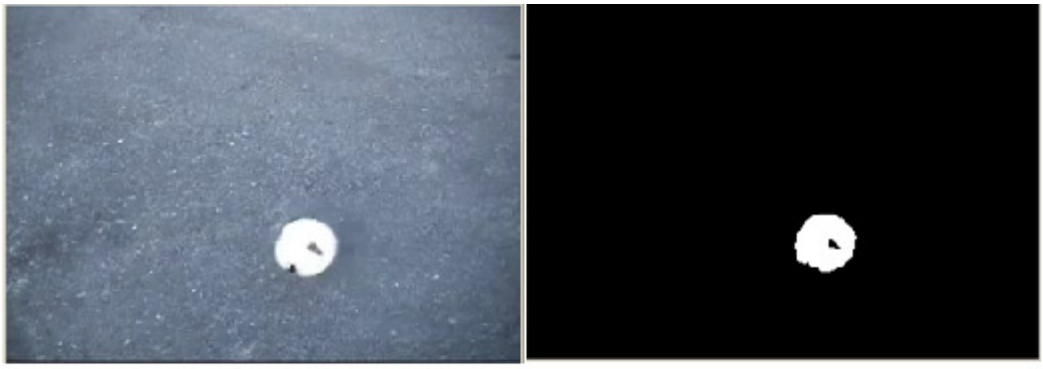

Figura 6. Ejemplo de segmentación por umbralización del color blanco sobre fondo oscuro.

2.2 Sustracción de fondo

Para identificar los objetos que han experimentado movimiento con respecto al resto de la imagen (fondo), se puede aplicar la técnica de sustracción de fondo.

Esta técnica consiste en restar una imagen inicial, considerada fondo, con el resto de imágenes sucesivas de la secuencia. El resultado será una imagen en la que sólo aparecerá el objeto que haya experimentado movimiento sobre fondo negro.

Matemáticamente, la operación anterior se puede expresar de la siguiente forma:

$$d_{t,f}(x,y) = \begin{cases} 1 & \text{si } |f_t(x,y) - f_f(x,y)| > T \quad \text{Objeto} \\ 0 & \text{en otro caso} \quad \text{Fondo} \end{cases}$$

Para aplicar esta técnica, previamente, es necesario convertir tanto la imagen fondo, como el resto de imágenes de la secuencia a escala de grises. De esta forma, resulta más eficiente realizar la diferencia entre imágenes que sólo tienen un canal, en lugar de tres como las imágenes en color. En ese caso, habría que realizar la misma operación para cada canal por separado.

Después de realizar la diferencia entre el fondo y las imágenes sucesivas, se aplica un umbral, de forma que los píxeles del objeto segmentado se encuentren por encima de él, para así etiquetarlo con valor 1. Por el contrario, el fondo quedaría etiquetado con valor 0 y el resultado sería una imagen binarizada, con el objeto segmentado en blanco y el fondo en negro.

En la siguiente imagen se muestra todo el proceso y el resultado final:

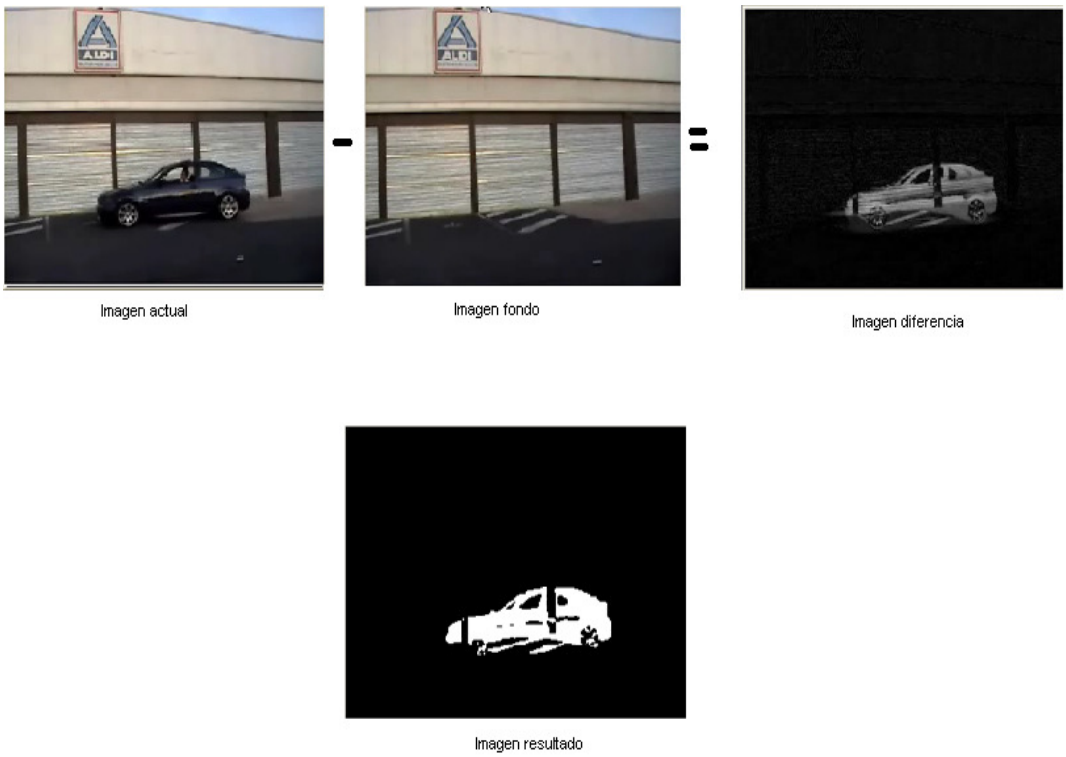

Figura 7. Ejemplo de sustracción de fondo.

Hay que indicar que como esta técnica depende del conocimiento previo del fondo sobre el que se mueve el objeto, cuando las condiciones meteorológicas no sean favorables (lluvia, viento, nieve, etc.) o aparezcan objetos inesperados en la escena (personas andando, coches, puertas que se abren, etc.) esta técnica no será la más eficiente, puesto que en el fondo también habrá movimiento. Por este motivo, se ha implementado la aplicación con la doble opción de poder segmentar el objeto en movimiento mediante umbralizado del color, con la posibilidad de seleccionar el color deseado con el ratón del ordenador.

2.3 Determinación de la posición

Como ya he mencionado antes, para poder realizar el seguimiento de un objeto es necesario extraer características del mismo que nos indiquen dónde se encuentra; es decir, necesitamos determinar la posición observada

del objeto. En este proyecto dicha posición se corresponderá con su centro de masas.

Por este motivo, previamente a la determinación de la posición, debemos realizar un procesado sobre la imagen para extraer de ella el objeto de interés. Para ello se han implementado dos posibilidades de seguimiento, descritas en el apartado anterior. La primera nos permite seleccionar el objeto de un color determinado con el ratón del ordenador. Con esa información del píxel seleccionado se aplicará la técnica de binarización y se pondrán en blanco todos los píxeles que tengan el mismo color que el seleccionado. La segunda posibilidad extrae el objeto en movimiento respecto del fondo de la imagen que permanece estático. Esta técnica se conoce como sustracción de fondo y también binariza la imagen con el objeto en blanco y el resto en negro.

Una vez que hemos extraído el objeto en blanco mediante alguna de las dos posibilidades, para llevar a cabo la determinación del centro de masas se ha utilizado un método en el cual se recorre la imagen umbralizada y se obtienen el primer y último punto en blanco, recorriendo por filas y el primer y último punto en blanco, recorriendo por columnas. De esta forma se obtienen las cotas superior, inferior, izquierda y derecha del objeto, respectivamente. Con estas cotas se puede determinar el centro de masas del objeto. La operación a seguir es calcular la mitad de la suma de las cotas superior e inferior y haríamos lo mismo con las cotas izquierda y derecha. El primer resultado nos proporcionará la coordenada y, el segundo nos dará la coordenada x del centro buscado. Además, también es posible englobar al objeto en una región cuadrada, cuyos extremos vienen dados por las cotas citadas anteriormente. El extremo superior izquierdo se encontrará en la posición (cota izquierda, cota superior) y el extremo inferior derecho se encontrará en la posición (cota derecha, cota inferior). Con estos puntos se crearía la región cuadrada.

La siguiente figura muestra de forma ilustrativa el proceso anteriormente explicado:

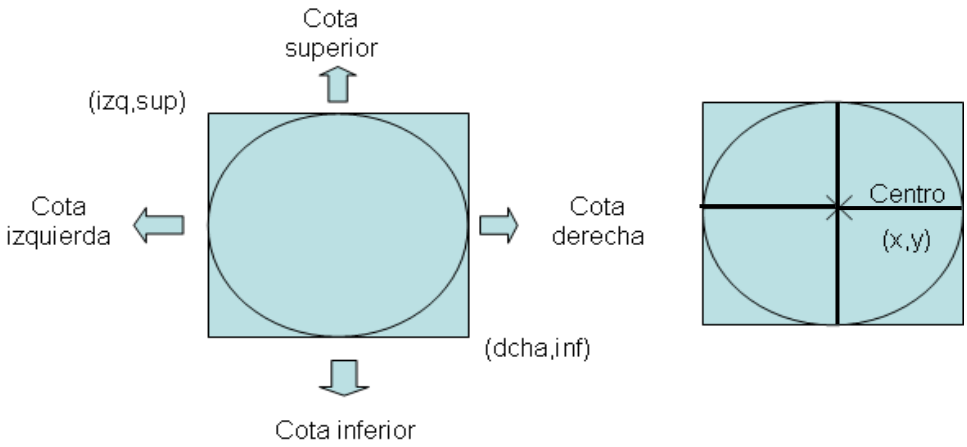

Figura 8. Cálculo del centro de masas.

A continuación muestro un ejemplo de una imagen en la que aparece una pelota de tenis en movimiento, utilizada para experimentar con la aplicación. El resultado muestra la imagen con la región y el centro de masas del objeto pintados en amarillo. Tanto en la sustracción de fondo como en la segmentación por umbralización el resultado de la región y el centro de masas se mostrarán de la misma forma que se muestra a continuación:

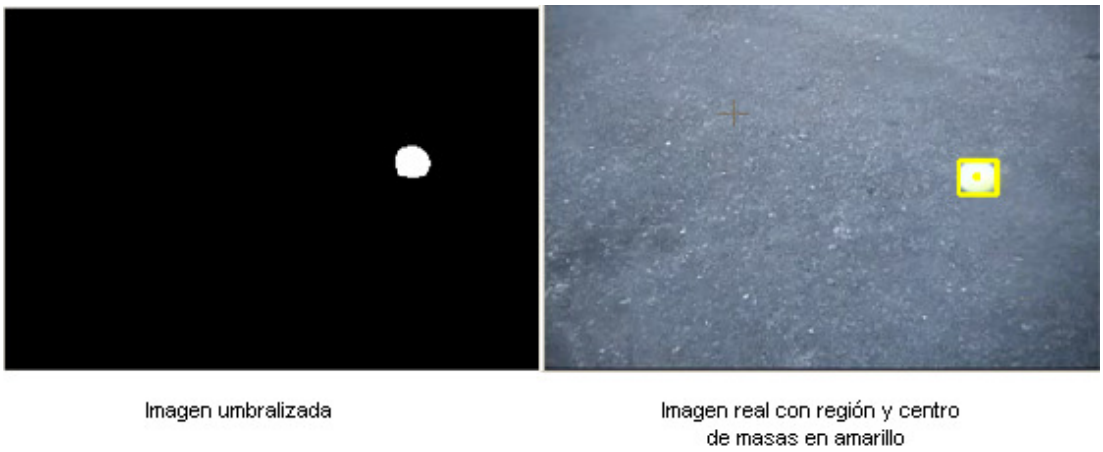

Figura 9. Ejemplo región y centro de masas en amarillo.

2.4 Filtro de Kalman

Una vez que hemos conseguido separar el objeto de interés del resto de la imagen y hemos determinado su posición observada, procedemos a la aplicación de algún estimador que nos permita realizar su seguimiento a lo largo de la secuencia de imágenes.

En este proyecto el estimador utilizado es el *Filtro de Kalman* [8], que como ya mencioné en el *Estado del arte*, se trata de un algoritmo que pronostica el estado siguiente a partir de su estimación previa y le añade una corrección que es proporcional al error de predicción, de forma que éste es minimizado estadísticamente. [13]

Concretamente, tiene como objetivo la estimación del estado $x \in R^n$ de un proceso estocástico dinámico, de tiempo discreto, modelado por la siguiente ecuación:

$$x_{k+1} = Ax_k + Bu_k + w_k \qquad (1)$$

Además, está sujeto a medidas de la observación *z*, pertenecientes a R^m, mediante la siguiente ecuación:

$$z_k = Hx_k + v_k \qquad (2)$$

Las variables w_k y v_k son matrices de dimensión $nx1$ que corresponden al ruido del proceso y al ruido de la medida, respectivamente. En ambos casos se consideran que son independientes y ruido blanco o gaussiano; es decir, con media cero y desviación igual a la matriz de covarianza del ruido del proceso (*Q*) o a la matriz de covarianza del ruido de la observación (*R*), según corresponda en cada caso:

$$p(w) \sim N(0, Q)$$
$$p(v) \sim N(0, R)$$

Por otro lado, el estado *x* se corresponde con una matriz que contiene las coordenadas de posición (x_x , x_y) del objeto, así como sus coordenadas

de velocidad *(v_x , v_y)*, la cual se considerará constante. El resultado es una matriz de dimensión $nx1$. En cambio, la observación *z* se define mediante una matriz que contiene las coordenadas de la posición observada del objeto de interés *(z_x , z_y)*; es decir, será una matriz de dimensión $mx1$.

La matriz *A* es una matriz cuadrada de dimensiones nxn que relaciona el estado en el instante *k* con el estado en el instante *k+1*, de la siguiente forma:

$$x_{x_{k+1}} = x_{x_k} + v_{x_k} t$$

$$x_{y_{k+1}} = x_{y_k} + v_{y_k} t$$

$$v_{x_{k+1}} = v_{x_k}$$

$$v_{y_{k+1}} = v_{y_k}$$

, dando como resultado la siguiente matriz:

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

La matriz B tiene dimensiones $nx1$ y refleja la influencia del input de control o función externa $u \in R^l$ en el estado x. En nuestro caso esta matriz es nula, puesto que no tenemos influencia por parte de ningún control externo.

La matriz *H* posee dimensiones mxn y relaciona el estado con la observación $z \in R^m$. Su forma es la siguiente:

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

Las matrices *Q* y *R* corresponden a la matriz de covarianza del ruido del proceso y a la matriz de covarianza del ruido de la observación, respectivamente. Sus dimensiones son de nxn y se trata de matrices identidad multiplicadas por un escalar, el cual indica el valor de la desviación estándar del ruido, elevado al cuadrado. Es decir, su forma es la siguiente:

$$Q = \sigma^2 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad R = \sigma^2 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Sus valores pueden variar en el tiempo, aunque no es el caso habitual. Son matrices críticas para el funcionamiento del filtro, ya que es necesario un compromiso entre sus valores para el adecuado funcionamiento de éste. En este proyecto se inicializarán con distintos valores y se evaluarán los distintos resultados obtenidos en cada caso.

El objetivo inicial del *Filtro de Kalman* es calcular el estimador a posteriori $\hat{x}_k$ como combinación lineal del estimador a priori $\hat{x}_k^-$ y el error en la predicción de la observación. Esto se define mediante la siguiente ecuación:

$$\hat{x}_k = \hat{x}_k^- + K\left(z_k - H\hat{x}_k^-\right) \qquad (3)$$

, donde $\left(z_k - H\hat{x}_k^-\right)$ es el residual o innovación en la observación que refleja la discrepancia entre la predicción de la medida $H\hat{x}_k^-$ y la observación actual z_k.

La matriz *K* es de dimensiones nxn y se conoce como *ganancia de Kalman* o factor de mezcla, la cual indica la cantidad de influencia del error entre nuestra estimación y la medida. Se obtiene mediante la siguiente expresión:

$$K_k = P_k^- H^T \left(HP_k^- H^T + R\right)^{-1} \qquad (4)$$

, donde P_k^- es el estimador de la covarianza del error a priori y *R* y *H*, tal y como ya he definido anteriormente, son la matriz de covarianza del ruido de la observación y la matriz que relaciona el estado con la observación *z*, respectivamente.

A partir de esta expresión podemos comprobar que si *R* se aproxima a 0, el valor de la *ganancia de Kalman* es mayor, luego la influencia del error será más grande. En cambio, cuando P_k^- se aproxime a 0, el valor de esta ganancia será menor y el error influirá menos en la estimación.

Por otro lado, cuando *R* se aproxime a 0, estaremos dando mayor confianza a la observación actual z_k y, por lo tanto, menor confianza a la predicción de la medida $H\hat{x}_k^-$. Por el contrario, cuando P_k^- sea próximo a 0 se producirá el efecto contrario al anterior.

En resumen, el Filtro de Kalman realiza estimaciones del estado de un proceso con realimentación. Es decir, estima el estado del proceso en algún instante de tiempo y con la información obtenida de esa medida se realimenta [8].

Se puede decir que este proceso iterativo se descompone en dos etapas importantes: en primer lugar, la predicción del estado a partir del estado anterior y las ecuaciones dinámicas; en segundo lugar, la corrección de la predicción usando la observación actual.

De esta forma, las ecuaciones del *Filtro de Kalman* se pueden dividir en dos grupos: las que actualizan el tiempo o ecuaciones de predicción y las que actualizan los datos observados o ecuaciones de actualización.

Las ecuaciones del primer grupo se encargan de la proyección del estado actual hacia el futuro tomando como referencia el estado pasado, así como de la actualización intermedia de la matriz de covarianza del error. Las del segundo grupo, en cambio, son responsables de la realimentación, incorporando nueva información a la estimación a priori para mejorar la estimación del estado a posteriori. Por este motivo, las ecuaciones de actualización del tiempo se pueden interpretar como ecuaciones de predicción y las ecuaciones de actualización de la medida como ecuaciones de corrección.

El siguiente esquema muestra gráficamente cómo se realizaría el proceso:

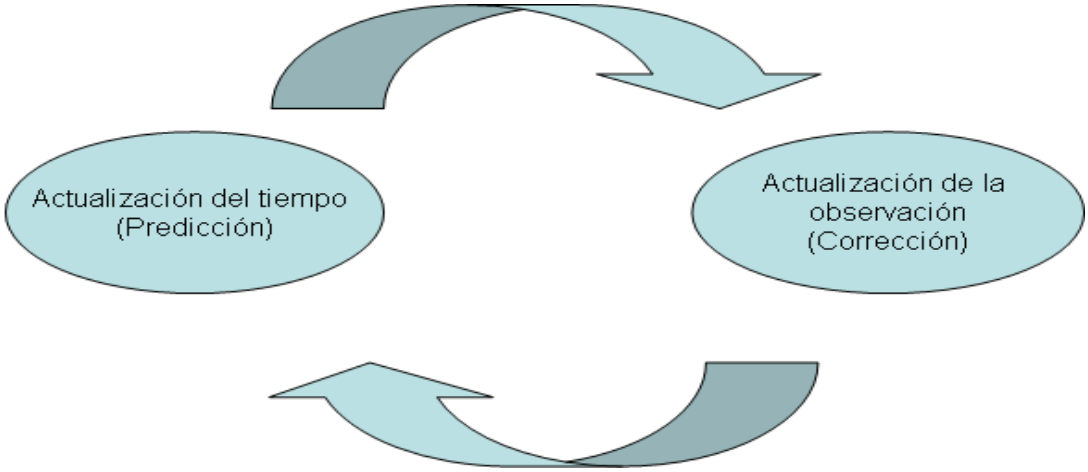

Figura 10. Ciclo del Filtro de Kalman.

Las ecuaciones específicas para la predicción y corrección del estado se detallan a continuación:

$$\hat{x}_{k+1}^{-} = A\hat{x}_k + Bu_k + w_k \quad (5)$$

$$\hat{P}_{k+1}^{-} = A\hat{P}_k A^T + Q \quad (6)$$

$$K = P_k^{-} H^T \left(HP_k^{-} H^T + R\right)^{-1} \quad (7)$$

$$\hat{x}_k = \hat{x}_k^{-} + K\left(z_k - H\hat{x}_k^{-}\right) \quad (8)$$

$$P_k = (I - KH)P_k^{-} \quad (9)$$

A continuación muestro una descripción detallada de las etapas del *Filtro de Kalman* con sus ecuaciones:

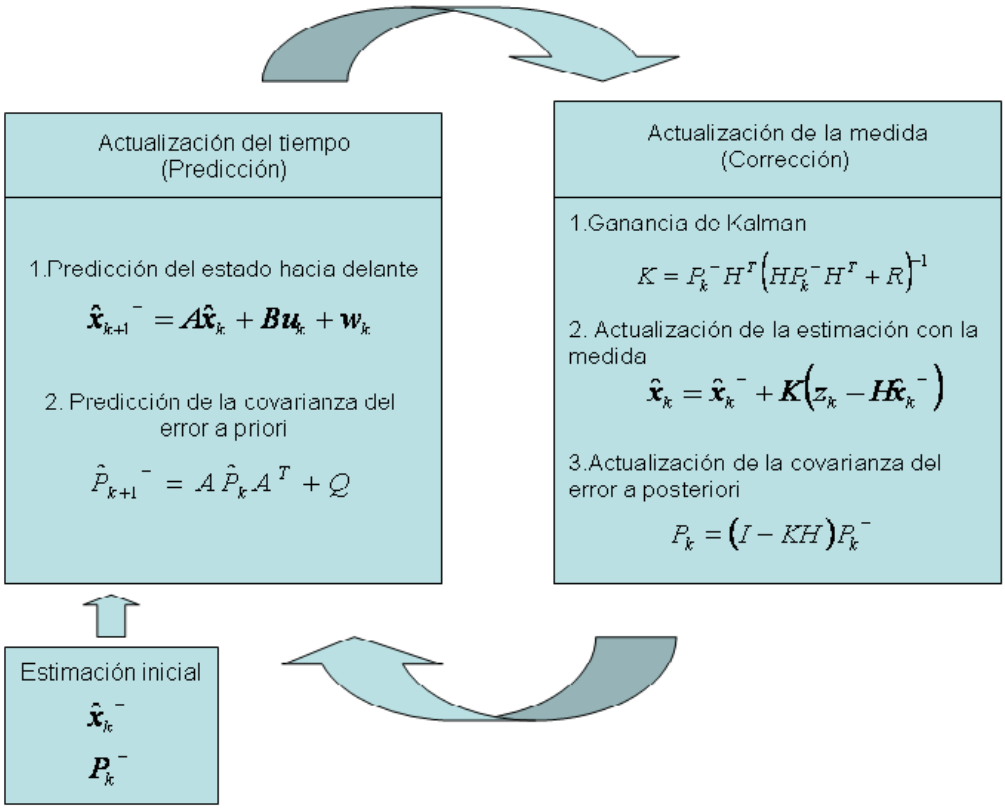

Figura 11. Descripción detallada del Filtro de Kalman.

La estimación inicial del estado, en nuestro caso, se corresponde con el centro de masas del objeto a estudiar y la matriz de covarianza del error inicial puede ser la matriz identidad.

Otro aspecto importante radica en las matrices *Q* (ruido del proceso) y *R* (ruido de la observación), que tal y como ya he mencionado, son críticas para el adecuado funcionamiento del filtro. Si son constantes y están bien estimadas, las matrices de covarianza del error a posteriori P_k y a priori P_k^- convergen rápidamente y son constantes. Por este motivo es necesario un compromiso entre los valores de ambas matrices.

Como ya he comentado anteriormente se modificarán los valores de *R* y *Q* para comprobar cómo responde el sistema en cada caso.

Por otro lado, también es importante indicar que el *Filtro de Kalman* se ha aplicado no sólo a un objeto en movimiento, sino que también se ha implementado la aplicación con la posibilidad de realizar el seguimiento de dos objetos móviles. Para ello se aplica el mismo filtro dos veces, la primera

vez para el primer objeto que se selecciona con el ratón y la segunda vez para el segundo objeto seleccionado.

Un ejemplo del resultado que se obtiene es el siguiente:

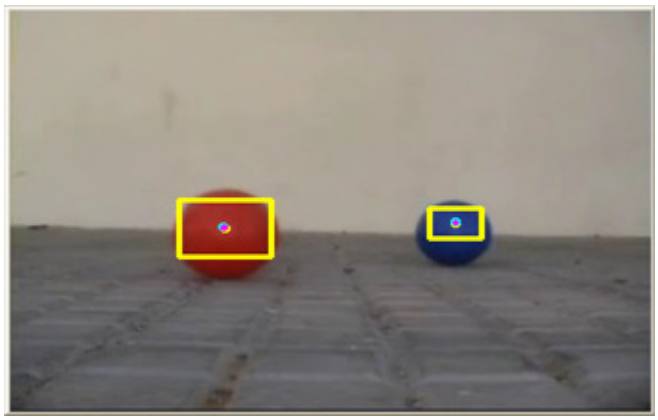

Figura 11. Ejemplo de tracking sobre dos objetos.

Capítulo 3: Implementación del sistema

En este capítulo se explica cómo se ha llevado a cabo la implementación del sistema. Para realizar la segmentación, determinación de la posición del objeto de interés y aplicación del *Filtro de Kalman* se ha creado un algoritmo mediante la programación orientada a objetos en Visual C++. Este algoritmo dispone de una serie de clases con sus atributos y métodos y, de un programa principal, en el cual se llama a esas clases para realizar cada una de las fases del tracking sobre el objeto.

Por este motivo se ha diseñado una clase para cada fase del tracking: una clase llamada *Procesado*, otra clase llamada *Región* y, por último, una clase llamada *FiltroKalman*.

A continuación, hago una descripción del código que contiene cada una de ellas.

3.1 Clase Procesado

Como he indicado en el capítulo anterior, para poder determinar la posición observada del objeto de interés es necesario que previamente se separe el objeto a seguir del resto de la imagen. Para ello se han utilizado dos técnicas distintas de seguimiento:

- Segmentación basada en umbralizado
- Sustracción de fondo.

Esta clase desarrollará cada una de estas técnicas, incluyendo un método distinto para cada una de ellas.

3.1.1 Método de segmentación basada en umbralizado:

Para la segmentación basada en umbralizado se ha creado el siguiente método:

►void CvFindColorRGB(const IplImage* src, IplImage* dst, int B, int G, int R, int BSensibility, int GSensibility, int RSensibility)

Entradas: imagen capturada, imagen de salida umbralizada, componentes RGB del color del píxel y valores de sensibilidad de ese color.
Salida: no devuelve nada, sólo hace la conversión de la imagen de entrada a una imagen binarizada.
Objetivo: etiquetar los objetos que tengan las componentes RGB de entrada con valor 1 y el resto de la imagen con valor 0.
Descripción: recorre la imagen de entrada comprobando las componentes RGB de cada píxel, si coinciden con las de entrada dentro de un rango de sensibilidad, etiqueta ese píxel como blanco, sino lo etiqueta como negro. Para ello se ha diseñado el siguiente algoritmo:

```
int i, j;
int fondo=0;
int objeto=255;

for (i=0; i<src->height; i++)
        for (j=0; j<src->width; j++)
        {
                CvScalar s = cvGet2D(src, i, j);

                if ((s.val[0]>(B-BSensibility) && s.val[0]<(B+BSensibility)) &&
                                (s.val[1]>(G-GSensibility) && s.val[1]<(G+GSensibility)) &&
                                (s.val[2]>(R-RSensibility) && s.val[2]<(R+RSensibility)))
                {
                        s.val[0]=objeto;
                        s.val[1]=objeto;
                        s.val[2]=objeto;
                        cvSet2D(dst, i, j, s);
                }
                else
                {
                        s.val[0]=fondo;
                        s.val[1]=fondo;
                        s.val[2]=fondo;
                        cvSet2D(dst, i, j, s);
                }
        }
```

Figura 12. Código del método cvFindColorRGB.

Además se ha creado otro método denominado *Umbralizar* para llamar dentro de él al método anterior y que nos devuelva la imagen umbralizada. Este método se mostrará a continuación.

3.1.2 Método para realizar la umbralización:

►IplImage* Umbralizar(IplImage* image, int x, int y)

Entradas: imagen capturada, coordenada x del píxel seleccionado, coordenada y del píxel seleccionado.
Salida: imagen umbralizada.
Objetivo: convertir la imagen de entrada en una imagen binarizada, con el objeto de interés en blanco y el resto en negro.
Descripción:
- Se obtienen las componentes RGB del píxel seleccionado de la imagen de entrada mediante la función cvGet2D.
- Se llama al método cvFindColorRGB para que realice la umbralización.
- Se aplica un filtro paso-bajo para eliminar el posible ruido mediante la función cvSmooth.

En resumen, el código utilizado es el siguiente:

```
pixel=cvGet2D(image,x,y);
CvFindColorRGB(image,image_umbralizada,pixel.val[0],pixel.val[1],pixel.val[2],20,20,20);
cvSmooth(image_umbralizada,image_filtrada,CV_MEDIAN,3,0,0,0);
```

Figura 13. Código del método Umbralizar.

3.1.3 Método para realizar la sustracción de fondo:

Para aplicar esta técnica se ha creado el siguiente método:

►IplImage* SustraccionFondo(IplImage* fondo,IplImage* image)

Entradas: primer frame considerado fondo y siguiente frame de la secuencia de imágenes.
Salida: imagen umbralizada.
Objetivo: separar el objeto en movimiento del resto de la imagen (fondo).
Descripción:
- Se convierten las imágenes de entrada a escala de grises mediante la función cvCvtColor.
- Se realiza la diferencia de las imágenes de entrada mediante la función cvAbsDiff.
- Se aplica un umbral para convertir la imagen diferencia en una imagen binarizada, con el objeto de interés en blanco y el fondo en negro. Utiliza la función cvThreshold y un umbral que depende de las características del objeto en movimiento.
- Se añade un filtrado paso-bajo para eliminar la información indeseada mediante la función cvSmooth.

El código utilizado para ello se muestra a continuación:

```
cvCvtColor(t1,bn,CV_BGR2GRAY);
cvCvtColor(t2,bn2,CV_BGR2GRAY);

cvAbsDiff(bn2,bn,diferencia);
cvThreshold(diferencia,umbralizada,50,255,CV_THRESH_BINARY);
cvSmooth(umbralizada,filtrada,CV_MEDIAN,3,0,0,0);
```

Figura 14. Código método sustracción de fondo.

3.2 Clase Región

Esta clase se crea para determinar la posición observada del objeto de interés, que en este caso se corresponde con su centro de masas.

Contiene como atributos las cotas que nos permiten encerrar al objeto en una región, la cual nos permitirá posteriormente determinar el centroide.

Para conseguir este propósito se han creado los siguientes métodos.

3.2.1 Método para crear la región:

Este método se muestra a continuación:

►void CrearRegion(IplImage* imagenUm)

Entrada: imagen umbralizada que se obtiene del procesado previo.
Salida: no devuelve nada. Calcula las cotas del objeto.
Objetivo: calcular las cotas superior, inferior, derecha e izquierda del objeto.
Descripción:
- Se recorre la imagen umbralizada buscando píxeles en blanco correspondientes al objeto de interés. De esta forma obtiene los extremos del objeto. Para ello se ha diseñado el siguiente código:

```
x=9999;
y=9999;
xFinal=-1;
yFinal=-1;

CvScalar pixel;

int i,j;

for(i=0;i<imagenUm->height-1;i++)
{
        for(j=0;j<imagenUm->width-1;j++)
        {
                pixel=cvGet2D(imagenUm,i,j);

                if(pixel.val[0]==255 && i<x)
                        x=i;
                if(pixel.val[0]==255 && j<y)
                        y=j;
                if(pixel.val[0]==255 && i>xFinal)
                        xFinal=i;
                if(pixel.val[0]==255 && j>yFinal)
                        yFinal=j;
        }
}
```

Figura 15. Código del método CrearRegion.

3.2.2 Método para pintar la región sobre la imagen:

A continuación, se muestra el método creado para pintar la región a partir de las cotas obtenidas por el método anterior:

►void PintarRegion(IplImage* imagenP)

Entrada: imagen copia de la imagen que se captura. Se hace esta copia para pintar los resultados sobre la imagen real, aunque el procesado de la imagen se realiza sobre la umbralizada.
Salida: no devuelve nada, se ocupa de pintar la región.
Objetivo: dibujar de color amarillo la región que encuadra al objeto.
Descripción:
- A partir de las cotas calculadas en el método anterior, se obtienen los puntos extremos de la región.
- Se pinta la región utilizando la función cvRectangle.

Este método emplea el siguiente método:

```
p1.x=cota izquierda;
p1.y=cota superior;
p2.x=cota derecha;
p2.y=cota inferior;

cvRectangle(imagenP,p1,p2,CV_RGB(255,255,0),2,8,0);
```

Figura 16. Código del método PintarRegion.

3.2.3 Método para pintar el centro de masas:

Para finalizar, se crea un método que nos permite calcular el centro de masas del objeto y lo pinta en la imagen capturada. Este método es el siguiente:

►void PintarCentro(IplImage* imagenP)

Entrada: imagen copia de la imagen capturada para pintar sobre ella.
Salida: no devuelve nada, se encarga de pintar el centro de masas.
Objetivo: calcular y pintar el centroide a seguir del objeto de interés.
Descripción:
- Calcula la coordenada (x , y) del centro de masas a partir de los lados de la región. Para ello calcula la mitad de dos de sus lados, puesto que será una región cuadrada o rectangular.
- Pinta el centro de masas utilizando la función cvCircle.

El código utilizado es el siguiente:

```
CvPoint centro;
centro.x=(cotaizquierda+cotaderecha)/2;
centro.y=(cotasuperior+cotainferior)/2;

cvCircle(imagenP,centro,1,CV_RGB(255,255,0),2,8,0);
```

Figura 17. Código del método PintarCentro.

3.3 Clase FiltroKalman

En esta clase se lleva a cabo la implementación del algoritmo del *Filtro de Kalman*. Tiene como atributos las distintas matrices que intervienen en sus ecuaciones, para que así se puedan utilizar en cualquier método. Estas matrices se crean mediante la función cvCreateMat, en el constructor. Luego es necesario liberarlas mediante la función cvReleaseMat, en el destructor.

Se ha diseñado un método para cada una de las fases que componen su algoritmo.

3.3.1 Método inicialización del Filtro de Kalman:

El primer método recibe el nombre de *Inicialización* y se muestra a continuación:

►**void Inicializacion(CvPoint centro)**

Entrada: el centro de masas calculado en la clase anterior. Es una variable de tipo CvPoint, ya que se trata de un punto con dos coordenadas.
Salida: no devuelve nada. Se encarga de dar un valor inicial a las matrices creadas.
Objetivo: inicializar las distintas matrices que intervienen en el algoritmo del filtro.
Descripción:
- Se inserta un valor inicial en cada una de las matrices creadas mediante la función cvmSet. El centro se introduce en la matriz correspondiente al estado inicial.

3.3.2 Método para realizar la fase de predicción:

Se ha creado un método que se encarga de realizar la fase de predicción. Este método es el siguiente:

►**void Prediccion( )**

Entrada: ninguna, ya que se utilizan las matrices que son atributos de la clase.
Salida: no devuelve nada, se encarga de realizar la predicción.
Objetivo: proyectar hacia el futuro el estado inicial, obteniendo una estimación del estado siguiente a partir del actual.
Descripción:
- Se genera ruido gaussiano. Para ello se utiliza el siguiente código:

```
CvRandState rng;
cvRandInit(&rng,desviacion,media,primervalor,CV_RAND_NORMAL);
cvRand(&rng,RuidoProceso);
```

Figura 18. Código crear ruido gaussiano.

- Se aplica la ecuación de predicción del estado a priori. Para ello se utiliza la función cvMatMulAdd.
- Se traspone A mediante la función cvTranspose.
- Se mutiplica A*P mediante la función cvMatMul.
- Se aplica la ecuación de predicción de la covarianza del error a priori. Para ello se utiliza la función cvMatMulAdd.

El código utilizado para implementar esta fase sería:

$\hat{x}_{k+1}^{-} = A\hat{x}_k + Bu_k + w_k$

```
cvMatMulAdd(A,estado,RuidoProceso,estadopriori);
```

$\hat{P}_{k+1}^{-} = A\hat{P}_k A^T + Q$

```
cvTranspose(A,traspuesta);
cvMatMul(A,P,multiplicacion);
cvMatMulAdd(multiplicacion,traspuesta,Q,Ppriori);
```

Figura 19. Código del método Predicción.

3.3.3 Método para realizar la fase de corrección:

A continuación se describe el método que se encarga de realizar la fase de corrección del *Filtro de Kalman*:

►void Correccion()

Entrada: ninguna. Utiliza los atributos de la clase.
Salida: no devuelve nada. Su función es realizar la fase de corrección.
Objetivo: añadir información a la estimación a priori para mejorar la estimación a posteriori. Minimiza la influencia del ruido.
Descripción:
- Se calcula la ganancia de Kalman. Para ello es necesario realizar operaciones con matrices. Se utilizan las funciones cvMatMul, cvTranspose, cvMatMulAdd y cvInvert.

- Se calcula la estimación a posteriori del estado. En este caso también es necesario realizar operaciones con matrices. Se utilizan las funciones cvMatMul, cvSub y cvMatMulAdd.
- Calulamos al estimación a posteriori de la covarianza del error. Las funciones utilizadas han sido cvMatMul y cvSub.

El código implementado para desarrollar esta fase sería:

$$K = P_k^- H^T \left(H P_k^- H^T + R\right)^{-1}$$

```
cvMatMul(H,Ppriori,multiplicacion);
cvTranspose(H,traspuesta);
cvMatMulAdd(multiplicacion,traspuesta,R,multiplicacion2);
cvInvert(multiplicacion2,invertida);
cvMatMul(Ppriori,traspuesta,multiplicacion3);
cvMatMul(multiplicacion3,invertida,K);
```

$$\hat{x}_k = \hat{x}_k^- + K\left(z_k - H\hat{x}_k^-\right)$$

```
cvMatMul(H,estpriori,multiplicacion4);
cvSub(estobservacion,multiplicacion4,residuo);
cvMatMulAdd(K,residuo,estpriori,estPosteriori);
```

$$P_k = \left(I - KH\right)P_k^-$$

```
cvMatMul(K,H,multiplicacion6);
cvSub(identidad,multiplicacion6,resta);
cvMatMul(resta,Ppriori,Pposteriori);
```

Figura 20. Código del método Corrección.

3.3.4 Método para pintar la estimación a posteriori:

Para mostrar la estimación que hace el filtro se crea un método con las siguientes características:

►void PintarEstimacion(IplImage* imagenP)

Entrada: imagen copia de la imagen capturada para pintar sobre ella los resultados.
Salida: no devuelve nada, únicamente se encarga de dibujar.

Objetivo: mostrar por pantalla el punto estimado por el filtro.

Descripción:

- Se obtienen las coordenadas del punto estimado con la función cvmGet.
- Se pinta el punto estimado de color azul, usando la función cvCircle.

Para este método el código utilizado es:

```
float x=cvmGet(estPosteriori,0,0);
float y=cvmGet(estPosteriori,1,0);

punto.x=int(x);
punto.y=int(y);

cvCircle(imagenP,punto,1,CV_RGB(0,255,255),3,8,0);
```

Figura 21. Código del método PintarEstimacion.

3.3.5 Método para realizar la fase de actualización:

Por último, se diseña un método que realice la fase de actualización, el cual se describe a continuación:

►void Actualizar( )

Entrada: ninguna, utiliza los atributos de la clase para realizar las operaciones.

Salida: no devuelve nada, sólo se encarga de realizar la fase de actualización.

Objetivo: incluir los valores del estado y la covarianza del error estimados a posteriori en la fase de predicción para que empiece de nuevo el proceso.

Descripción:

- Se obtienen las coordenadas del estado estimado y se introducen en la matriz estado actual para que empiece de nuevo la predicción. Para ello se utilizan las funciones cvmGet y cvmSet, respectivamente.

- Se realiza la misma operación que en el caso anterior, pero con la covarianza estimada y la actual. También se utilizan las funciones cvmGet y cvmSet.

El código que se utiliza es el siguiente:

```
float valor=cvmGet(estPosteriori,0,0);
cvmSet(estado,0,0,valor);

float valor2=cvmGet(Pposteriori,0,0);
cvmSet(P,0,0,valor2);
```

Figura 22. Código del método Actualizar.

3.4 Programa principal

En el programa principal es donde se gestionan cada una de las clases y donde se da forma a cada una de las fases del tracking.

3.4.1 Fase de captura de imágenes:

En primer lugar se capturan las imágenes almacenadas en un archivo audiovisual .avi o directamente con la cámara digital. Para elegir una opción u otra se ha diseñado el siguiente código:

```
CvCapture* capture = 0;
input_name ="Fichero.avi";

if (!input_name)
        capture = cvCaptureFromCAM(-1);
else
        capture = cvCaptureFromAVI(input_name);
```

Figura 23. Código captura de imágenes.

3.4.2 Fase de procesado de las imágenes:

Se realiza el procesado de cada frame. Este procesado se divide en las siguientes partes:

El primer frame capturado se toma como fondo. Para ello se utiliza el código siguiente:

```
bool primerframe=true;

if(primerframe)
{
        cvCopy(frame_copy,frame_fondo,0);
        primerframe=false;
}
```

Figura 24. Código para tomar frame fondo.

A continuación, se decide si utilizar segmentación basada en umbralizado o bien sustracción de fondo. El código que se muestra realiza esta función:

```
if(sustraccion)

        frame_umbralizado=p->SustraccionFondo(frame_copy,frame_fondo);

else
{
        frame_umbralizado=p->Umbralizar(frame_copy,xSelect,ySelect);

        if(elegido2)
                frame_umbralizado2=p2->Umbralizar(frame_copy,xSelect,ySelect);

}
```

Figura 25. Código para elegir sustracción de fondo o umbralización por color.

Las variables p y p2 son variables dinámicas que apuntan a la dirección de memoria de la clase *Procesado*.

Por otro lado, se puede apreciar en el código anterior que se incluye la posibilidad de que en el caso que hubiera un segundo objeto en

movimiento, también se podría realizar su segmentación mediante umbralizado. Esta opción se incluye aquí para poder aplicar también sobre ese objeto el *Filtro de Kalman* posteriormente.

En la siguiente fase se procede a la determinación de la región y del centro de masas del primer objeto seleccionado, o bien, del objeto al que se le ha aplicado la sustracción de fondo. Para ello se emplea el código siguiente:

```
r->CrearRegion(frame_umbralizado);
r->PintarRegion(frame_pintar);
r->PintarCentro(frame_pintar);
```

Figura 26. Código para crear la región y pintar el centro de masas.

El frame denominado frame_pintar es una copia de frame_copy, o lo que es lo mismo, del frame capturado. Esto se hace para dibujar sobre la imagen que se captura, aunque el procesado se realice sobre la imagen umbralizada.

La variable r es una variable dinámica que apunta a la dirección de memoria de la clase *Región*.

A continuación, se aplica el *Filtro de Kalman* al centro de masa calculado anteriormente. Para ello utilizo:

```
if (primero)
{
        Fk->Inicializacion(r->getcentro());
        primero=false;
}

Fk->setcentro(r->getcentro());
Fk->Prediccion();
Fk->Correccion();
Fk->PintarEstimacion(frame_pintar);
Fk->Unir(frame_pintar,r->getcentro(),Fk->getPuntoEstimado());
Fk->Actualizar();
```

Figura 27. Código Filtro de Kalman.

Como se puede apreciar en el código anterior, la inicialización del filtro sólo se realiza una vez, ya que en la fase actualizar se le dará un valor nuevo a algunos parámetros. Los parámetros que no cambian al ser

atributos de la clase siempre van a tomar el mismo valor, a menos que se modifiquen manualmente.

La variable Fk es una variable dinámica que apunta a la dirección de memoria de la clase *FiltroKalman*.

Por último, para permitir que el programa realice el seguimiento de dos objetos a la vez, a continuación del código expuesto anteriormente, se volvería a inicializar el filtro y se aplicarían sus fases sobre un segundo objeto, previamente seleccionado. El código a utilizar sería el siguiente:

```
if(elegido2)
{
        r2->CrearRegion(frame_umbralizado2);
        r2->PintarRegion(frame_pintar);
        r2->PintarCentro(frame_pintar);

        if(primero2)
        {
                Fk2->Inicializacion(r2->getcentro());
                primero2=false;
        }

        Fk2->setcentro(r2->getcentro());
        Fk2->Prediccion();
        Fk2->Correccion();
        Fk2->PintarEstimacion(frame_pintar);
        Fk2->Unir(frame_pintar,r2->getcentro(),Fk2->getPuntoEstimado());
        Fk2->Actualizar();
}
```

Figura 28. Código seguimiento de segundo objeto.

Como se puede apreciar, el filtro también se inicializa sólo la primera vez.

Todo este procesado está incluido dentro de un bucle infinito, de forma que se aplique a cada uno de los frames capturados y finalice cuando termine la secuencia de imágenes.

Para finalizar este capítulo, muestro el diagrama de bloques que representa el esquema que se ha seguido para implementar el programa principal, lo que facilitará la comprensión del código:

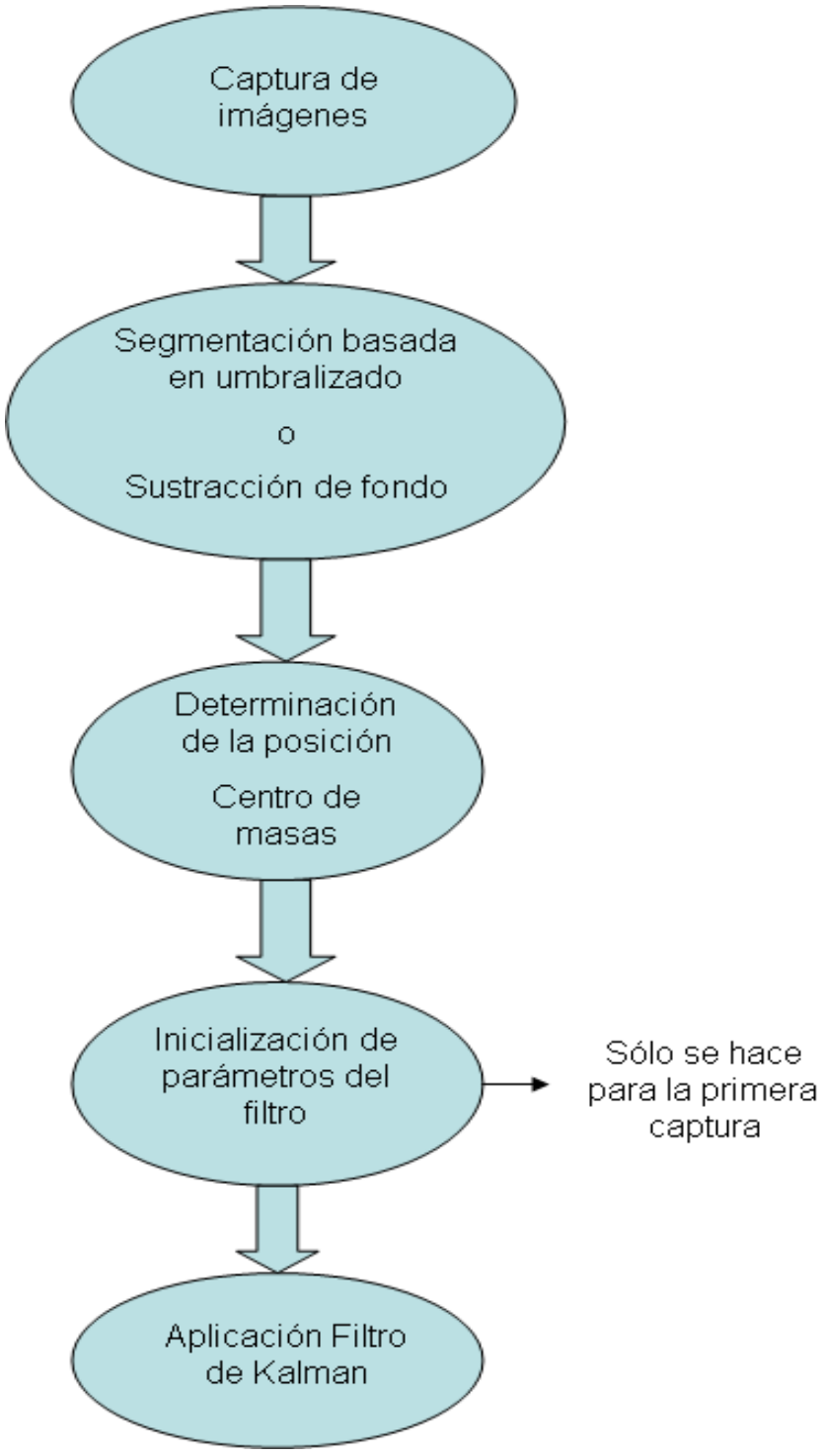

Figura 29. Esquema global del programa principal.

Capítulo 4: Experimentos y resultados

En este capítulo se procederá a evaluar la respuesta del sistema implementado bajo distintas condiciones experimentales. Se utilizarán dos técnicas distintas para realizar la segmentación del objeto, sustracción de fondo o segmentación basada en umbralizado. Además se realizará el seguimiento a dos objetos móviles y se le aplicarán también las mismas experiencias.

Las condiciones a las que he sometido al sistema en las distintas experiencias son:

- Oclusión: el objeto de interés desaparece por un momento de la escena, pero luego vuelve a aparecer.
- Movimientos plano paralelos y no plano paralelos: en el primer caso el objeto a seguir es un coche en movimiento y en el segundo una pelota que se lanza y bota.
- Modificación de parámetros del filtro: se modifican los valores del ruido del proceso y de la medida para evaluar la respuesta del *Filtro de Kalman*.
- Modificación del rango de sensibilidad del color: se cambian los valores de sensibilidad para comprobar cómo realiza la umbralización del color seleccionado mediante el ratón.
- Modificación del umbral en sustracción de fondo: al igual que antes, se cambia el valor del umbral para ver cómo realiza la binarización del objeto en movimiento con respecto al fondo.
- Se realizarán pruebas en espacios exteriores e interiores para comprobar cómo afecta el cambio de iluminación en la respuesta del sistema.

En cada prueba se realizará un análisis de los resultados obtenidos y se evaluará el comportamiento del filtro.

Finalmente se realizará una comparación con los resultados obtenidos por otro estimador, como es el *Filtro de Partículas*.

4.1 Pruebas utilizando sustracción de fondo

En estos experimentos ha sido necesario tener conocimiento previamente sobre el fondo de la escena (fondo controlado). Por este motivo se ha tenido especial cuidado en que el fondo no experimentara ningún tipo de movimiento, ya que produciría resultados indeseables y no se podrían evaluar los resultados del sistema bajo la aplicación de esta técnica.

4.1.1 Prueba 1 de sustracción de fondo modificando el umbral:

En esta prueba vamos a analizar cómo responde el sistema cuando cambiamos el valor del umbral en la sustracción de fondo.

Este umbral es decisivo para que segmente de forma adecuada al objeto de interés. Además esto también influirá a la hora de determinar la región que encuadra al objeto, ya que ésta depende de la cantidad de píxeles blancos que haya en la escena. Por lo tanto, si el objeto no se binariza a blanco perfectamente, no será posible determinar de forma correcta la región ni tampoco el centro de masas.

Experimento 1:

Se trata de un movimiento plano paralelo de un coche. El umbral establecido es de 50 y los valores del ruido de la medida R y del ruido del proceso Q son 0.001 y 0.01, respectivamente.

Los resultados obtenidos se muestran a continuación:

Figura 30. Prueba 1 de sustracción de fondo. Experimento 1. Umbral 50.

Se puede comprobar a partir de las fotografías que la región encuadra bien al objeto. En algunas ocasiones es algo más grande que éste debido a la aparición de sombras, que como se mueven, también son umbralizadas.

Por otro lado, vemos que la estimación que realiza el *Filtro de Kalman* es buena, ya que es próxima en todo momento al centro de masas del objeto. Luego podemos decir que los valores establecidos para Q y R son adecuados.

Experimento 2:

En este caso el umbral establecido es 20, el resto de valores se han mantenido iguales a los del experimento anterior.

Figura 31. Prueba 1 de sustracción de fondo. Experimento 2. Umbral 20.

A la vista de las fotografías se aprecia que antes de que el objeto aparezca en la escena empieza a umbralizar partes de ésta que se encuentren por encima del valor del umbral. Esto es debido a que aparecen algunas sombras, de modo que al establecer un umbral más bajo se van a

umbralizar algunas de esas zonas más oscuras. Por este motivo, aunque el color azul se encuentra un poco por debajo de la mitad del espectro, en el primer experimento se utilizó un umbral de 50 para evitar que se umbralicen las sombras más oscuras.

A continuación muestro un ejemplo del resultado de la umbralización en cada uno de los experimentos para comprobar el efecto comentado anteriormente:

Figura . Segmentación con umbral = 50.

Figura . Segmentación con umbral = 20.

Se puede comprobar que efectivamente con un umbral más bajo se dejan pasar sombras oscuras que se producen por el movimiento del coche. Por lo tanto, se demuestran las conclusiones extraídas.

Por otro lado, debido a la mala umbralización en el segundo caso, no es posible determinar de forma correcta la región del objeto, ya que en la mayoría de los casos es mucho mayor que el tamaño de éste.

En conclusión, se demuestra que la determinación del umbral es decisiva para una adecuada segmentación del objeto.

4.1.2 Prueba 2 de sustracción de fondo modificando los parámetros del filtro:

En esta ocasión probaremos con el movimiento no plano paralelo de una pelota de tenis y veremos cómo le afecta el cambio en los valores del ruido del proceso *Q* y de la medida *R* que intervienen en el *Filtro de Kalman*.

Luego este experimento no tendrá tanto que ver con la parte de segmentación, sino que está orientado a evaluar la respuesta del *Filtro de Kalman* en función de cómo varían sus parámetros. Para ello estableceremos un umbral adecuado, por ejemplo, un umbral de 50.

Experimento 1:

Se tomará un valor de Q=0.01 y un valor de R=0.001. El umbral en este caso también será 50, para evitar que se umbralicen sombras oscuras. Además el color amarillo está un poco por encima de la mitad del espectro, por lo que este umbral es adecuado para binarizar ese color.

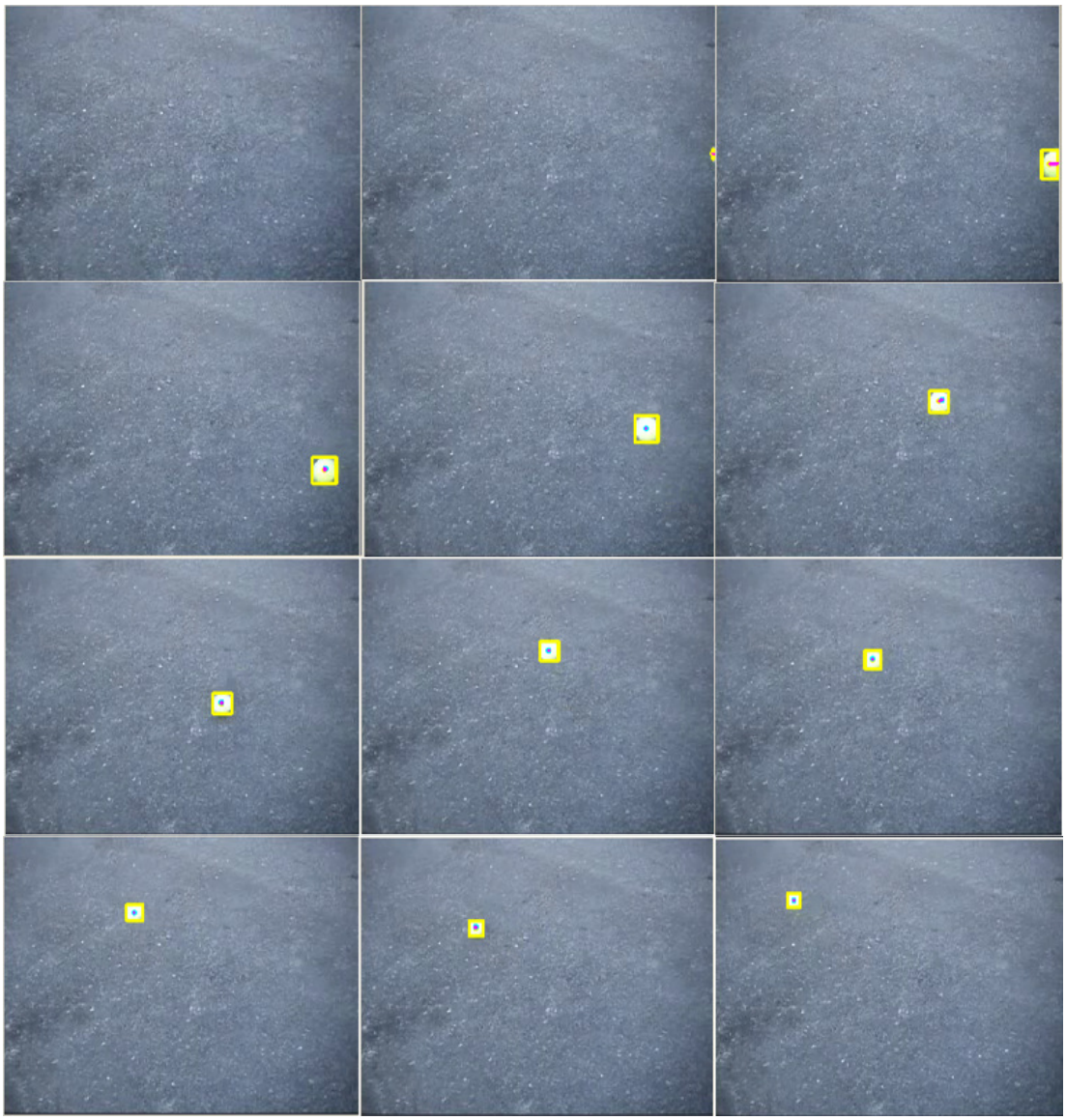

Figura 32. Prueba 2 de sustracción de fondo. Experimento 1. R=0.001 Q=0.01.

A partir de los resultados obtenidos se observa que la estimación que realiza el filtro (en azul) y el centro de masas del objeto (en amarillo) son en todos los casos prácticamente el mismo punto. El color rosa que aparece corresponde a la línea que une los dos puntos.

Por este motivo, podríamos decir que la estimación que realiza el filtro es buena. Luego los valores establecidos serían adecuados.

Experimento 2:

Para esta experiencia se aumentará el valor de *R* a 0.1 y el valor de Q se quedará igual al utilizado en el experimento anterior.

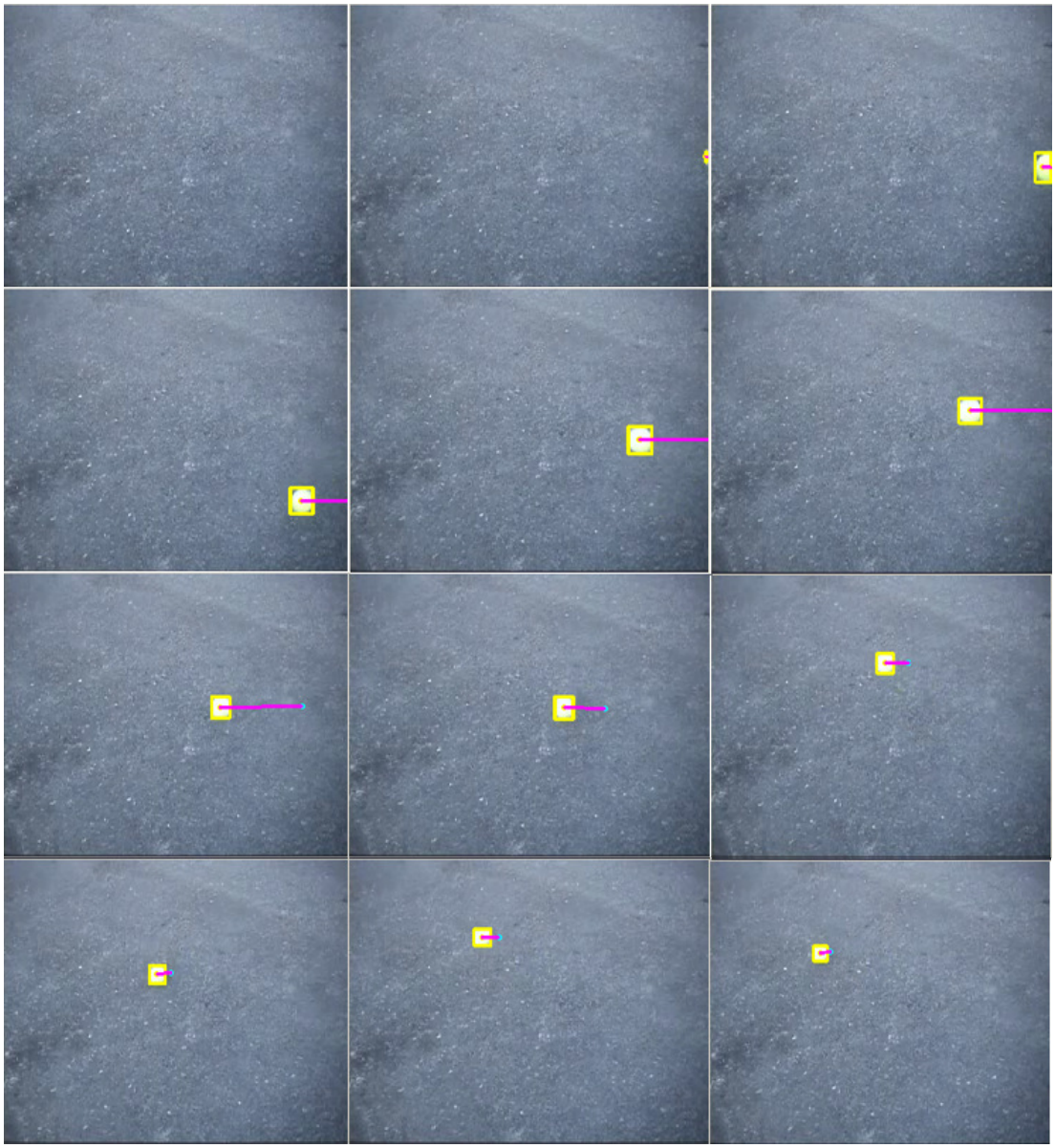

Figura 33. Prueba 2 de sustracción de fondo. Experimento 2. R=0.1 Q=0.01.

En este caso se aprecia que el filtro tarda mucho más en encontrar al objeto, luego realiza una peor estimación. Se puede observar que hasta el final de la secuencia no se aproxima más al centro del objeto.

En consecuencia, podemos determinar que a medida que el valor de *R* aumenta, el valor de la estimación es peor. Esto es debido a que se da menor confianza a la medida de la observación que a la predicción.

Por un lado tenemos que:

$$z_k = Hx_k + v_k$$

, donde v_k es el ruido de la medida que depende de *R*. Por consiguiente, cuanto mayor sea este valor, mayor será el valor obtenido en la observación.

Por otro lado tenemos que la ecuación de estimación a posteriori del estado utilizada en la fase de corrección es la siguiente:

$$\hat{x}_k = \hat{x}_k^- + K\left(z_k - H\hat{x}_k^-\right)$$

A partir de ella se puede observar que cuanto mayor es el valor de la observación, la ganancia de *Kalman* aumenta. Luego la influencia del ruido es mayor, proporcionando una estimación peor.

De esta forma, queda demostrado el motivo por el cual al aumentar el valor de *R* se produce una peor respuesta del filtro.

Por lo tanto, se puede llegar a la conclusión de que al aumentar R, se da menos confianza a la observación, lo que proporciona una peor estimación. Al realizar la operación contraria; es decir, disminuyendo R se mejorará la estimación, dando mayor confianza a la obsevación.

En el caso del valor de *Q*, tenemos que:

$$\hat{x}_{k+1}^- = A\hat{x}_k + Bu_k + w_k$$

, donde $\hat{x}_{k+1}^-$ es la predicción del estado y w_k es el ruido del proceso que depende de *Q*. Entonces al aumentar el valor de *Q* aumentará también el valor del ruido del proceso, proporcionando un valor mayor en la predicción.

Según la ecuación de estimación del estado a posteriori tenemos:

$$\hat{x}_k = \hat{x}_k^- + K\left(z_k - H\hat{x}_k^-\right)$$

Se puede comprobar que al aumentar el valor de la predicción se disminuye el valor de la ganancia de *Kalman*, con lo que se mejorará el valor estimado.

Por lo tanto, se llega a la conclusión de que al aumentar *Q* se favorece a la estimación, ya que se da menos confianza a la predicción y más confianza a la observación. Al disminuir su valor se producirá el efecto contrario.

Por consiguiente, para conseguir un adecuado rendimiento del *Filtro de Kalman* será necesario llegar a un equilibrio entre los valores de *Q* y *R*, tal y como indiqué en capítulos anteriores. Se trata, por tanto, de factores decisivos para realizar el seguimiento del objeto de forma eficaz.

Para la técnica de la segmentación basada en umbralizado no se han realizado pruebas de este tipo, ya que los resultados obtenidos serán similares a los explicados en la prueba anterior. Los valores de *Q* y *R* afectarán de la misma forma a la respuesta del filtro para las dos técnicas de segmentación.

4.1.3 Prueba de sustracción de fondo en interiores:

Como las dos pruebas anteriores se han realizado en exteriores, a continuación muestro un experimento de sustracción de fondo realizado en interiores para ver cómo afecta el cambio de iluminación.

Experimento 1:

El objeto a seguir es un tapón azul sobre fondo blanco. En este caso se ha utilizado un umbral de 30, debido a que el azul se encuentra en esta zona del espectro. En cuanto a los valores de R y Q se tomarán 0.001 y 0.01, respectivamente para que el filtro realice una buena estimación.

Los resultados obtenidos son los siguientes:

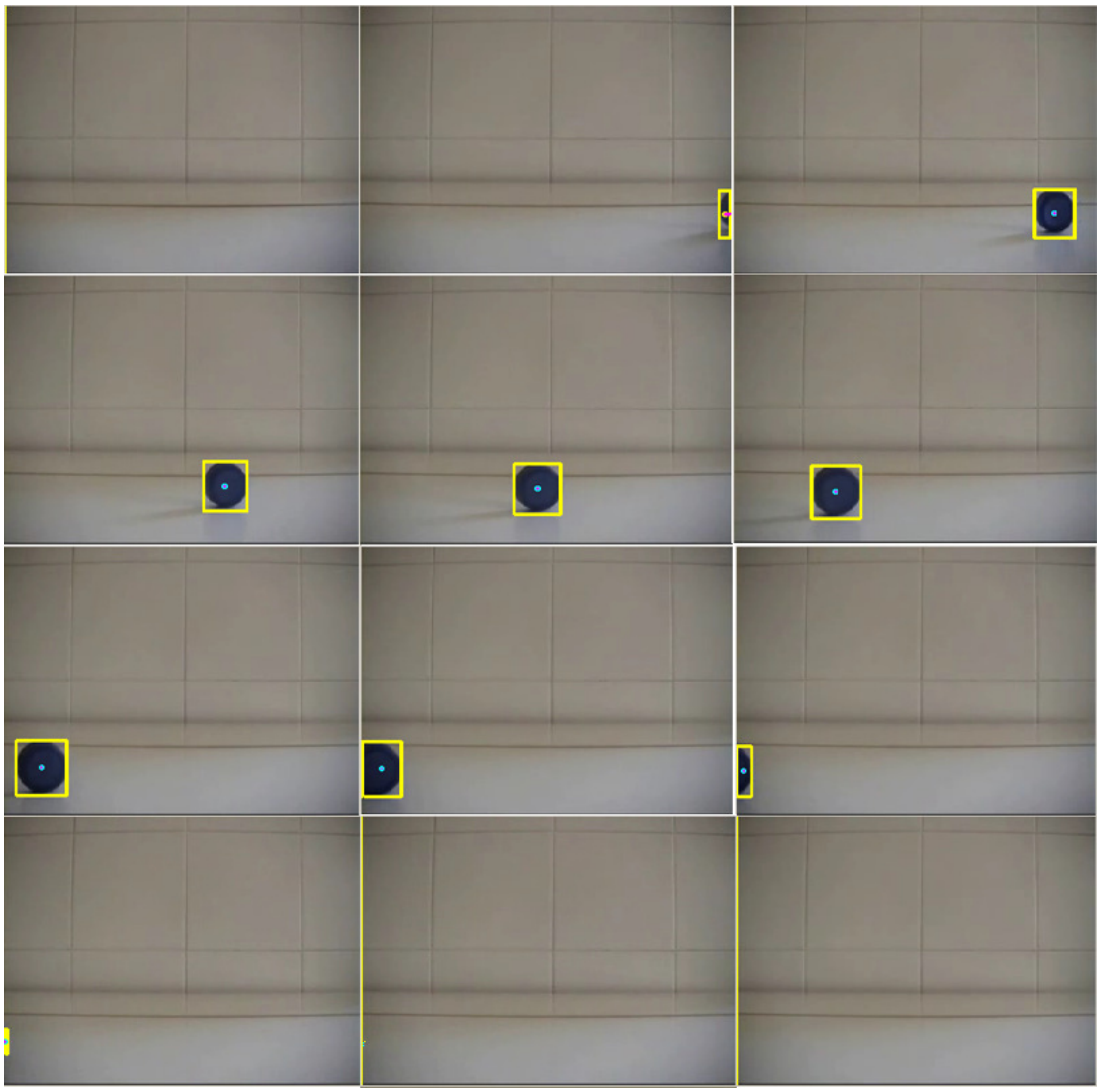

Figura . Prueba 3 de sustracción de fondo. Experimento 1. Umbral 30.

A partir de las fotografías se puede comprobar que el umbral establecido es correcto, puesto que en todos los casos se encuadra perfectamente al objeto dentro de la región y es posible determinar su centro de masas. El filtro por su parte también realiza una correcta estimación, ya que encuentra en todo momento al objeto.

Hay que indicar que aunque aparecen sombras, éstas no se umbralizan porque al ser oscuras están por debajo del umbral establecido. Por lo tanto, podemos decir que tanto en interiores como en exteriores se debe tener especial cuidado al establecer el umbral para evitar que las sombras que se producen por el movimiento del objeto sean umbralizadas.

A continuación muestro un ejemplo de la imagen umbralizada, para comprobar que efectivamente este valor del umbral proporciona una correcta umbralización:

Figura .Umbralización de tapón azul. Umbral 30.

Se puede observar que la umbralización es correcta, ya que el objeto queda bien definido. Luego las conclusiones extraídas son acertadas.

Por otro lado, también se puede comentar que existe una situación de oclusión. Es decir, que hay un momento en el que el objeto desaparece de la escena. Se observa que la respuesta del sistema es la adecuada, ya que cuando no encuentra al objeto desaparece la región.

4.2 Pruebas utilizando segmentación basada en umbralizado mediante selección del color

En este caso se han realizado pruebas en las que la segmentación del objeto se lleva a cabo mediante umbralización del color del objeto seleccionado con el ratón.

Las condiciones experimentales que se han tenido en cuenta son la modificación del rango de sensibilidad del color seleccionado y también se realizará el seguimiento de varios objetos móviles de distinto color.

4.2.1 Prueba 1 de segmentación basada en umbralizado modificando los valores de sensibilidad:

En primer lugar, veremos cómo afecta el cambio del rango de sensibilidad para el color seleccionado a la hora de realizar su umbralización. Tal y como hemos observado tras las experiencias realizadas mediante sustracción de fondo, cuando el objeto no es umbralizado correctamente no se puede determinar de forma eficaz la región que engloba al objeto y tampoco su centro de masas. Luego, al igual que ocurre con el umbral utilizado en la técnica anterior, también estos valores de sensibilidad serán decisivos para conseguir una adecuada separación del objeto respecto del resto de la imagen.

En este caso no es necesario tener un conocimiento previo del fondo de la escena (fondo no controlado), con lo cual cuenta con esa ventaja. Sin embargo, resulta un método menos eficiente que la sustracción de fondo, ya que es necesario recorrer toda la imagen comprobando las componentes RGB de cada uno de sus píxeles para determinar si éstas coinciden con las del color seleccionado mediante el ratón. En el caso de la sustracción de fondo, la imagen fondo y el resto de imágenes de la secuencia se convierten a escala de grises, por lo que sólo será necesario comprobar los niveles de un canal para cada píxel.

Experimento 1:

Para esta experiencia se utiliza un video de una persona en bicicleta que se encuentra en movimiento. El objetivo es clickear sobre el color rojo de la camiseta para umbralizarla y así realizar el seguimiento de la persona.

Para ello es necesario utilizar un rango de sensibilidad mayor para el rojo que para el resto de componentes. Podrían ser válidos los valores R=80, G=20 y B=20, ya que de esta forma se dejarían pasar distintas tonalidades del rojo y pocas del verde y del azul.

En cuanto al filtro se tomarán los siguientes valores R=0.001 y Q=0.01, ya que para dichos valores el filtro realizaba una buena estimación, tal y como vimos en un experimento anterior.

El resultado de la experiencia es el siguiente:

Figura 34. Prueba 1 de segmentación basada en umbralizado. Experimento 1. R=80, G=20, B=20.

Como se puede apreciar en los resultados, debido a que los valores de sensibilidad son correctos, la región encuadra perfectamente al objeto de color rojo. Además, la estimación del filtro (de color azul) es muy buena, ya que encuentra al objeto en todo momento, incluso se solapa con el centro de masas de éste.

Otro dato de interés es que en ocasiones la región cambiaba de tamaño, debido a que el color rojo se oscurecía demasiado en algunas zonas cuando el objeto se alejaba. En este caso la región sólo englobará a los píxeles que mantengan las mismas componentes que el píxel seleccionado al principio, dentro de un rango de sensibilidad. Si estas componentes se salen del rango entonces la región no las englobará, ya que no las identifica como parte del objeto.

Experimento 2:

En este caso se modificarán los valores de sensibilidad de la siguiente manera: R=60, G=60, B=60. Como se puede observar se aplica una sensibilidad igual para cada una de las componentes, por lo que se van a umbralizar otros colores a parte del rojo.

El resultado experimental se muestra a continuación:

Figura 35. Prueba 1 de segmentación basada en umbralizado. Experimento 2. R=G=B=60.

A la vista de los resultados se verifica lo que he comentado anteriormente. Al establecer esos valores de sensibilidad no sólo se umbraliza el color rojo, sino que también umbraliza otras zonas de la escena que tengan tonalidades de verde y azul.

Se puede comprobar que incluso cuando no hay objeto sigue marcando una región. Esto es debido a que umbraliza otras zonas de la escena, como ya he dicho.

A continuación muestro un ejemplo de la imagen umbralizada para comprobar por qué se produce el efecto anterior:

Figura . Umbralización bicicleta R=G=B=60.

A la vista de la imagen anterior, se comprueba que es cierto que no sólo umbraliza el color rojo de la camiseta, sino que también umbraliza zonas del suelo. El motivo es que el suelo tiene un tono amarillo y este color se encuentra próximo a la mitad del espectro, por lo que estaría dentro del rango de sensibilidad.

En conclusión, será primordial establecer correctamente el rango de sensibilidad para cada componente, en función del color que se desee umbralizar, de forma que se consiga una adecuada segmentación del objeto de interés.

4.2.2 Prueba 2 de segmentación basada en umbralizado seleccionando dos objetos en movimiento:

En esta prueba se realizará el seguimiento de dos objetos móviles. Para los dos objetos se utilizará la técnica de segmentación basada en umbralizado con la posibilidad de clickear sobre ellos. La sustracción de fondo no ha sido implemetada para el seguimiento de dos objetos, por este motivo no se han realizado pruebas de este tipo utilizando esta técnica.

El rango de sensibilidad establecido para cada componente será R= 40, G=10, B=40. Como se puede observar se da mayor sensibilidad a la componente roja y a la azul, ya que los objetos utilizados tienen estos colores. En cambio, al verde se le da menos sensibilidad porque no se desea umbralizar este color.

En cuanto al filtro se sigue manteniendo el valor de R=0.001 y Q=0.01, para que el filtro proporcione una buena estimación.

Los resultados obtenidos son los siguientes:

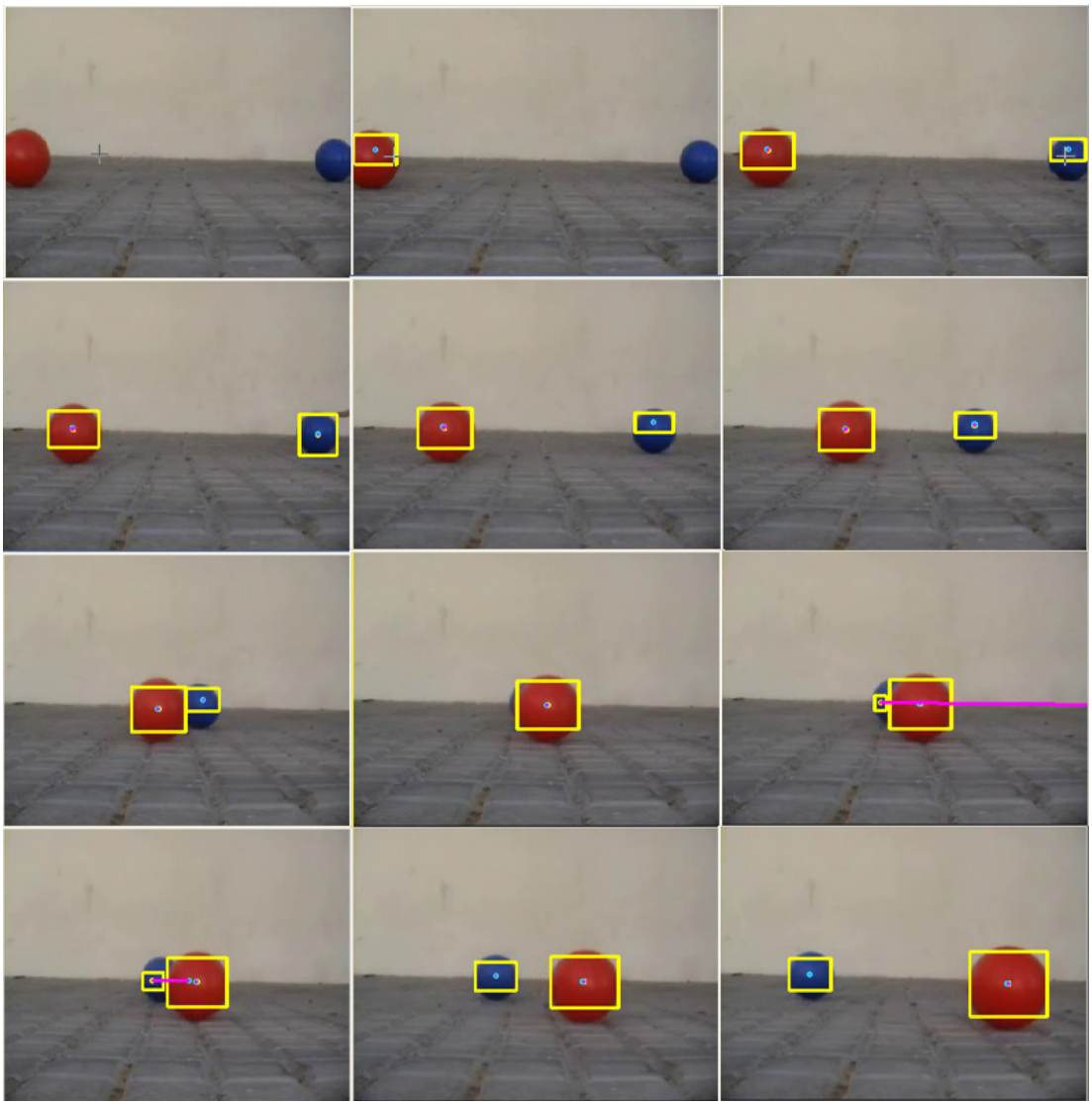

Figura . Prueba 2 de segmentación basada en umbralizado. Experimento 1. R=40, G=10, B=40.

A partir de las imágenes anteriores se puede comprobar que el rango de sensibilidad establecido para cada componente es el adecuado, ya que la región engloba al objeto de interés en todo momento. Hay ocasiones en las que la región cambia de tamaño. Esto se debe a que la zona de la pelota próxima al suelo tiene un color más oscuro que la zona de la parte superior, por lo que al tener un color distinto al seleccionado con el ratón, no la umbraliza y, por tanto, no forma parte de la región.

Por otro lado, se puede apreciar un claro ejemplo de oclusión. Vemos cómo al cruzarse las pelotas una de ellas se oculta y se pierde por un momento. El sistema responde de forma adecuada, ya que la región correspondiente al objeto que se oculta desparece y cuando el objeto vuelve a aparecer la región vuelve a englobarlo. Además también se aprecia cómo el filtro vuelve a encontrar al objeto rápidamente, lo que demuestra, una vez más que para esos valores de *Q* y *R* realiza una buena estimación. Se aprecia una línea rosa que une los dos puntos, para comprobar cómo sigue la estimación al centro de masas del objeto de interés.

Tras la consecución de las pruebas experimentales se ha medido el tiempo de ejecución con el objetivo de comprobar la eficiencia de cada una de las fases de nuestro sistema.

A partir de los resultados obtenidos se ha podido comprobar que la técnica de segmentación por umbralizado es menos eficiente que la técnica de sustracción de fondo. Esto se debe a que en la primera es necesario recorrer toda la imagen para comprobar las componentes RGB de cada píxel. En cambio, la segunda técnica convierte la imagen a escala de grises antes de umbralizarla, lo que reduce el coste computacional.

Por otro lado, se ha comprobado que también requiere mucho tiempo para determinar la región que encuadra al objeto. El motivo es el mismo que para la técnica de umbralizado, puesto que se recorre toda la imagen para buscar los píxeles que quedan etiquetados en blanco después de la umbralización.

Por último, se ha podido observar que el *Filtro de Kalman* es un algoritmo muy eficiente, pues requiere poco tiempo para llevar a cabo cada una de sus fases.

4.3 Comparación con el Filtro de Partículas

En este apartado se indicarán las diferencias existentes entre el *Filtro de Kalman* y el *Filtro de Partículas* a partir de los resultados experimentales obtenidos en cada caso.

Para poder comparar ambos filtros se han empleado las mismas imágenes en los experimentos realizados para cada uno de ellos. Además los dos filtros utilizan las mismas técnicas de procesado sobre la imagen para extraer el objeto de interés del resto de la escena. Estas técnicas son: detección de movimiento o segmentación basada en umbralizado.

También se ha implementado la aplicación para realizar el seguimiento de varios objetos móviles para los dos filtros. Luego la única diferencia radica en la aplicación de un estimador diferente en cada caso.

4.3.1 Comparación respecto al procesado de la imagen:

En cuanto al procesado, como ya he indicado anteriormente, en ambos casos se utilizan las mismas técnicas para extraer el objeto de interés del resto de la escena. Es decir, los dos filtros segmentan al objeto por sustracción de fondo o mediante umbralización.

Tras someter al sistema bajo diferentes condiciones experimentales (oclusión, cambios de iluminación, cambios de sensibilidad, cambios de umbral, etc.) en cada uno de los casos, podemos llegar a la conclusión de que el *Filtro de Partículas* es menos restrictivo a la hora de realizar la segmentación del objeto que el *Filtro de Kalman*. Esto se debe a que el Filtro de Partículas se basa en probabilidades, sin necesidad de extraer ninguna característica intrínseca del objeto de interés. En cambio, el *Filtro de Kalman* necesita conocer exactamente dónde se encuentra el objeto de interés para poder aplicar a la posición observada de dicho objeto las ecuaciones que representan su algoritmo.

Hemos comprobado que el *Filtro de Partículas* realiza el seguimiento del objeto de interés de forma adecuada, incluso cuando se umbralizan más

zonas de la escena además del objeto a seguir. El motivo radica en que este filtro se quedará con la zona donde haya mayor densidad de partículas, así funcionará adecuadamente aunque en la imagen umbralizada halla algunos puntos en blanco dispersos. Por el contrario, el *Filtro de Kalman* falla cuando se umbralizan más partes de la imagen junto con el objeto, ya que para determinar la región se tendrán en cuenta todos los píxeles etiquetados en blanco después de la umbralización. En este caso la región englobará no sólo al objeto en movimiento, sino también al resto de zonas que se hayan umbralizado, impidiendo determinar cuál es la posición observada de dicho objeto.

Por lo tanto, en el caso del *Filtro de Kalman* ha sido necesario tener mayor cuidado a la hora de establecer los valores umbrales y los valores de sensibilidad, ya que era necesario umbralizar exactamente al objeto de interés.

A continuación muestro un ejemplo del resultado obtenido para el *Filtro de Kalman* y para el *Filtro de Partículas* cuando se aplica la técnica de sustracción de fondo sobre una imagen con una pelota de tenis en movimiento. El valor umbral establecido en ambos casos ha sido de 20.

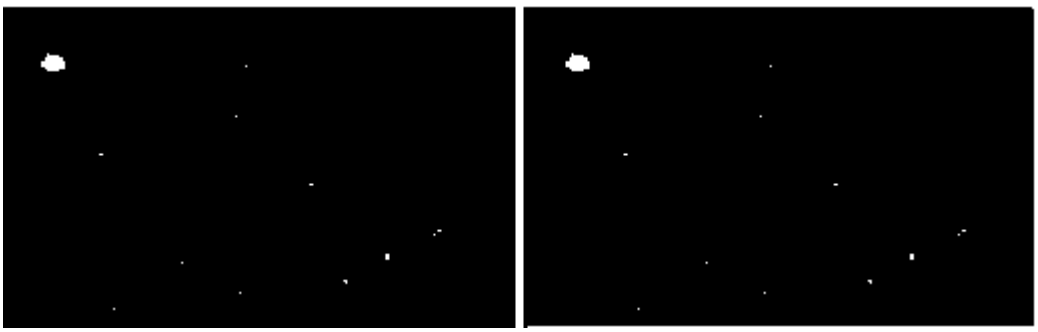

Figura .Umbralización de la pelota para el Filtro de Kalman y el Filtro de Partículas. Umbral=20.

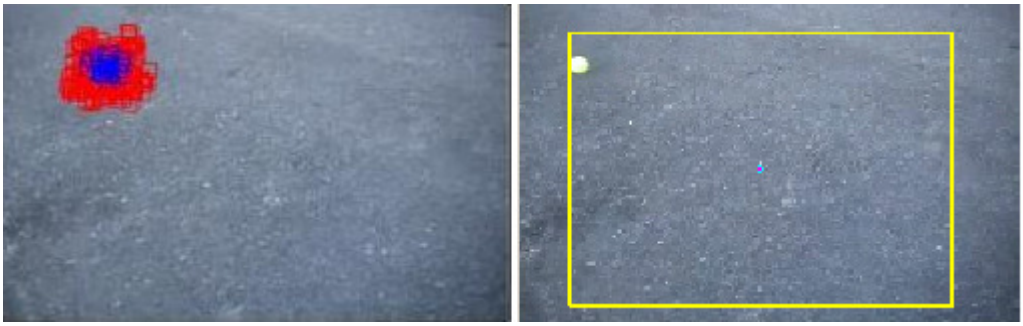

Figura .Resultado al aplicar Filtro de Partículas y Filtro de Kalman mediante sustracción de fondo. Umbral=20.

Como se puede comprobar a la vista de las fotografías, cuando se establece un umbral muy pequeño y se binarizan más zonas de la imagen además del objeto de interés, el *Filtro de Partículas* funciona correctamente porque se queda con la zona donde hay mayor densidad de puntos blancos. En cambio, el *Filtro de Kalman* encuadra en una región a todos los píxeles blancos, con lo que es imposible determinar la posición observada del objeto y realizar su seguimiento.

Para el caso del Filtro de Kalman sería necesario aplicar un umbral mayor para eliminar todos aquellos puntos dispersos que se umbralizan en blanco.

4.3.2 Comparación con respecto al tiempo de ejecución:

Para cada uno de los filtros se ha medido el tiempo de ejecución empleado en realizar cada una de sus fases. Tras esta comprobación podemos llegar a la conclusión de que el *Filtro de Partículas* es menos eficiente en cuanto a tiempo de ejecución, ya que tarda más tiempo en realizar cada uno de los procesados sobre la imagen. Esto puede deberse a que se trata de un filtro que consta de muchas más etapas que el *Filtro de Kalman*.

El *Filtro de Partículas* consta de 5 etapas y el *Filtro de Kalman* sólo tiene 2 etapas que se realimentan.

4.3.3 Comparación con respecto a cómo influye la velocidad:

Para realizar esta comparación hemos utilizado un video de un coche que se mueve a gran velocidad. Los resultados que se obtienen en la aplicación de cada uno de los filtros son los siguientes:

Figura : Influencia de la velocidad en el Filtro de Partículas y en el Filtro de Kalman

A la vista de las imágenes se puede comprobar que el *Filtro de Partículas* es más sensible a la velocidad que el *Filtro de Kalman*. En el primer caso, tenemos que cuando la velocidad es grande, las partículas no se encuentran sobre el centro del objeto sino que siguen su movimiento a corta distancia. Por el contrario, cuando los valores de *Q* y *R* (ruido del proceso y ruido de la medida) están correctamente ajustados para el *Filtro de Kalman*, se comprueba que la estimación es buena incluso cuando el objeto se mueve con gran rapidez.

Luego el *Filtro de Kalman* es más eficiente en este sentido.

4.3.4 Comparación con respecto a la complejidad del filtro:

Podríamos decir que el *Filtro de Partículas* consta de mayor número de fases que el *Filtro de Kalman*. En este sentido requiere un mayor esfuerzo computacional.

En cambio, el *Filtro de Kalman* es un filtro más abstracto, debido al conjunto de ecuaciones que componen su algoritmo.

4.3.5 Comparación con respecto a la eficiencia en la estimación de la posición del objeto a seguir:

Debido a que el *Filtro de Partículas* se basa en probabilidades para determinar la posición del objeto sobre el que se quiere realizar tracking, resulta menos eficiente que el *Filtro de Kalman* en este sentido, ya que éste último necesita una determinación exacta de la posición del objeto para poder aplicar las ecuaciones de su algoritmo. El resultado de estas ecuaciones proporcionan una estimación a la que se le aplica una corrección, la cual minimiza la influencia del ruido en dicha estimación.

Capítulo 5: Conclusiones y trabajo futuro

En este capítulo se exponen las conclusiones a las que se ha llegado tras la realización de este proyecto fin de carrera. Seguidamente se indicarán las posibles mejoras que se podrían realizar sobre este proyecto en un trabajo futuro.

5.1 Conclusiones

En este proyecto fin de carrera se ha diseñado la implementación de una de las aplicaciones con mayor desarrollo y crecimiento en el campo de la visión artificial. Esta aplicación consiste en realizar tracking o seguimiento de un objeto en movimiento en imágenes de 2D, aplicando para ello el *Filtro de Kalman.*

La implementación se ha realizado empleando la programación orientada a objetos mediante una estructura jerárquica de clases, dotando al programa de mayor flexibilidad y facilidad para relacionarlo con el mundo real y además fomenta su reutilización y extensión del código.

Llegados a este punto, podríamos extraer las siguientes conclusiones.

Una primera conclusión hace referencia a la importancia de seleccionar una buena técnica de segmentación de los objetos, puesto que es una tarea decisiva a la hora de realizar el seguimiento de dichos objetos de forma adecuada. Se ha comprobado experimentalmente que si no es posible conseguirlo no se alcanzarán los objetivos esperados.

En segundo lugar, hemos comprobado que la respuesta del filtro es adecuada bajo distintas condiciones experimentales, aplicando especial cuidado en la influencia de la iluminación del entorno y considerando rangos de sensibilidad correctos en función del color del objeto que se desee segmentar.

En definitiva, podemos afirmar que el *Filtro de Kalman* constituye un buen método para realizar predicciones en sistemas que evolucionan con el tiempo, pues se ha comprobado experimentalmente que se trata de una

técnica de gran eficiencia. Algunos de estos sistemas pueden ser sistemas de control del tráfico de coches, cámaras de vigilancia, robots, etc.

5.2 Trabajos futuros

Aunque se han conseguido todos los objetivos propuestos para la realización de este proyecto fin de carrera probablemente queden cosas que se puedan mejorar.

Como la implementación de la aplicación se ha llevado a cabo mediante una estructura jerárquica de clases basada en la programación orientada a objetos, resulta muy sencillo modificar o extender las prestaciones de esta aplicación.

Una mejora interesante sería ampliar las técnicas de segmentación utilizadas, para conseguir una determinación más exacta del objeto de interés. Una posibilidad sería reducir los espacios de búsqueda en la escena hasta llegar a la región de interés, realizando en ella todo el procesado. Esto reduciría aún más el coste computacional.

Otro estudio de interés sería utilizar un espacio de color menos sensible a los cambios de iluminación que se producen en la escena. De esta forma se mejoraría la umbralización.

Capítulo 6: Anexo

En este capítulo se explica el proceso de instalación de la librería OpenCv en el entorno de Visual Studio, se indican las funciones más importantes que se han utilizado de esta librería y las funciones creadas en la implementación del sistema. También se incluyen las referencias que se han seguido para la elaboración de este proyecto fin de carrera.

6.1 Instalación de OpenCv

Para instalar OpenCv en Windows e incluir la librería en Visual Studio se deben seguir los siguientes pasos:

1. Descargar la librería OpenCv de la dirección:
   http://sourceforge.net/projects/opencvlibrary [12]
2. A continuación descargar el instalador de OpenCv denominado OpenCv_1.0.exe y ejecutarlo.
3. Para incluir la librería en Visual Studio es necesario instalar previamente este entorno. Luego se abre dicho entorno y se procede a la creación de un nuevo proyecto. Para ello se selecciona el menú *Archivo* y, dentro de él, se elige la opción *Nuevo* → *Proyecto*.
4. Se realiza la configuración de las librerías. Para ello se clickea en el nombre del proyecto creado con el botón derecho y dentro del menú que se abre se selecciona la opción *Propiedades*.
5. Dentro de la ventana que aparece al seleccionar la opción *Propiedades* se procede a la configuración de los directorios de inclusión de C/C++. Esta operación se realiza seleccionando en *Propiedades de configuración* → *C/C++* → *General*. Seguidamente en *Directorios adicionales de inclusión* se introduce lo siguiente:

   C:\Archivos de programa\OpenCv\cv\include\
   C:\Archivos de programa\OpenCv\cvaux\include\
   C:\Archivos de programa\OpenCv\cxcore\include\

C:\Archivos de programa\OpenCv\otherlibs\highgui\
C:\Archivos de programa\OpenCv\otherlibs\cvcam\include\

6. A continuación dentro de *Propiedades de configuración → Vinculador → Entrada* se realiza la configuración de los directorios de bibliotecas adicionales. Para ello se introduce en *Dependencias adicionales* lo que se muestra a continuación:

   "C:\Archivos de programa\OpenCv\lib\cv.lib"
   "C:\Archivos de programa\OpenCv\lib\cvaux.lib"
   "C:\Archivos de programa\OpenCv\lib\cxcore.lib"
   "C:\Archivos de programa\OpenCv\lib\cvcam.lib"
   "C:\Archivos de programa\OpenCv\lib\highgui.lib"

Éstos serían los pasos a seguir para la inclusión de la librería OpenCv en un proyecto creado en Visual Studio. A partir de aquí ya se pueden utilizar las distintas funciones que esta librería ofrece.

6.2 Funciones de OpenCv utilizadas

En este apartado se indicarán las distintas funciones de OpenCv que se han utilizado para la implementación de esta aplicación. Para cada una de ellas se indicará la operación que realiza y se mostrará su cabecera con los parámetros que incluye.

Las funciones de OpenCv utilizadas se muestran a continuación:

►int cvNamedWindow ("const char* name", unsigned long flags)

Esta función se encarga de crear una ventana para mostrar las imágenes y los resultados obtenidos tras la manipulación de las mismas.

Los parámetros que incluye su cabecera son:

- name: nombre de la ventana que se usa para identificarla y que aparece en el título de la ventana de captura.
- flags: indica el tamaño de la ventana. Con CV_WINDOW_AUTOSIZE se ajusta automáticamente el tamaño de la ventana a lo que se haya capturado.

►Void cvDestroyWindow (const char* name)

Esta función se utiliza para destruir la ventana cuyo nombre es el que se indica en su cabecera.

El parámetro que aparece en su cabecera es el siguiente:

- name: indica el nombre de la ventana que se quiere destruir.

►void cvSetMouseCallback (const char* window_name, CvMouseCallback on mouse)

Esta función introduce la llamada a los acontecimientos producidos por el ratón dentro de la ventana especificada.

Sus parámetros son los siguientes:

- windows_name: nombre de la ventana donde se producen los acontecimientos del ratón.
- on mouse: es un puntero a la función llamada cada vez que se produce un acontecimiento del ratón en la ventana especificada. El prototipo de esta función es el siguiente:
  void Foo (int event, int x, int y, int flags).
  Esta función se incluirá en el apartado de funciones creadas.

►IplImage* cvLoadImage (const char* filename, int iscolor CV_DEFAULT(1))

Esta función carga la imagen desde el archivo especificado y devuelve un puntero a la imagen cargada.

Los parámetros de su cabecera son los siguientes:

- filename: nombre del archivo a cargar.
- iscolor: indica el número de canales de la imagen cargada.

  Si es mayor que 0, la imagen cargada tendrá 3 canales; si es 0 tendrá un solo canal y si es menor que 0 el número de canales dependerá del archivo que sea.

►void cvShowImage (const char* name, const CvArr* image)

Esta función tiene como objetivo mostrar la imagen de entrada en la ventana especificada. Si la ventana fue creada con un tamaño CV_WINDOW_AUTOSIZE, la imagen se mostrará con su tamaño original, en caso contrario se ajustará al tamaño de la ventana especificado.

Los parámetros de su cabecera son los siguientes:

- name: nombre de la ventana en la que se muestra la imagen.
- Image: imagen que se quiere mostrar.

►CvCapture* cvCaptureFromAVI (const char* filename)

El objetivo de esta función es asignar e inicializar la estructura CvCapture para grabar la secuencia de video desde el archivo especificado.

El parámetro que aparece en su cabecera es el siguiente:

- filename: nombre del archivo con formato .avi.

►CvCapture* cvCaptureFromCAM (int index)

Esta función asigna e inicializa la estructura CvCapture para grabar la secuencia de video desde una cámara.

El parámetro de su cabecera es el siguiente:

- index: índice de la cámara que se utiliza. En el caso de que haya una sola cámara se utiliza índice igual a -1.

►void cvReleaseCapture (CvCapture capture)**

El objetivo de esta función es liberar la estructura CvCapture asignada por las funciones cvCaptureFromCAM y cvCaptureFromAVI.

El parámetro que incluye en su cabecera es el siguiente:

- capture: puntero a la estructura CvCapture que se quiere liberar.

►int cvGrabFrame (CvCapture* capture)

Esta función almacena los frames desde la cámara o un archivo .avi. Estos frames son almacenados internamente en un formato comprimido. Para acceder a estos frames esta función suele ir acompañada de la función cvRetrieveFrame, que se explicará a continuación.

El parámetro que contiene su cabecera es:

- capture: puntero a la estructura CvCapture en la que se almacena la captura del archivo .avi o de la cámara.

►IplImage* cvRetrieveFrame (CvCapture* capture)

Esta función retorna un puntero a la imagen almacenada con la función cvGrabFrame. Esta imagen no debe ser liberada.

El parámetro que incluye su cabecera es el siguiente:

- capture: puntero a la estructura CvCapture en la que se almacena la captura del archivo .avi o de la cámara.

►int cvWaitKey (int delay CV_DEFAULT (0))

El objetivo de esta función es introducir una pausa.

El parámetro que aparece en su cabecera es:

- delay: retardo o pausa en milisegundos.

►IplImage* cvCreateImage (CvSize size, int depth, int channels)

Esta función crea una imagen con un determinado tamaño y número de canales.

Los parámetros que aparecen en su cabecera son:

- size: viene determinado por la función cvSize (width, heigth). Esta función proporciona las dimensiones a la imagen, el ancho y el alto.
- depth: profundidad del píxel en bits (IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16U, IPL_16S, IPL_DEPTH_32S, IPL_DEPTH_32F, IPL_DEPTH_64F).
- channels: número de canales de la imagen.

►const cvCopy (const CvArr* A, CvArr* B, const CvArr* mask=0)

Esta función se encarga de copiar los elementos de un array de entrada a uno de salida.

Los parámetros que aparecen en su cabecera son:

- A: array de entrada del que se quiere realizar la copia.
- B: array de destino.

- mask: máscara que especifica los elementos del array de destino que van a ser modificados.

►void cvFlip (const CvArr* A, CvArr* B, int flip_mode)

Esta función se encarga de girar una array.

Tiene como parámetros:

- A: array fuente.
- B: array destino.
- flip_mode: indica el modo de giro. Si vale 0 gira con respecto a x, si es mayor que 0 gira con respecto a y, si es menor que 0 gira con respecto a x e y.

►void cvReleaseImage (IplImage image)**

Esta función libera la cabecera de la imagen especificada.

El parámetro que utiliza es:

- image: puntero doble a la cabecera de la imagen especificada.

►void cvCvtColor (const CvArr* src, CvArr* dst, int code)

Esta función transforma la imagen de entrada de un espacio de color a otro.

Los parámetros que utiliza son:

- src: imagen fuente a convertir.
- dst: imagen destino que almacenará la imagen convertida.
- Code: código de conversión CV_<src_color_space>2<dst_color_space>.

►void cvAbsDiff (const CvArr* src1, const CvArr* src2, CvArr* dst)

Esta función realiza el valor absoluto de la resta de dos arrays.

Los parámetros de su cabecera son:

- src1: primer array fuente.
- src2: segundo array fuente.
- dst: array destino que almacenará el resultado de la operación.

►double cvThreshold (const CvArr* src, CvArr* dst, double threshold, double max value, int threshold_type)

Esta función realiza la umbralización del array de entrada teniendo en cuenta un cierto umbral.

Los parámetros que utiliza son los siguientes:

- src: array fuente o de entrada.
- dst: array destino o de salida.
- threshold: valor del umbral.
- max value: máximo valor que se puede alcanzar.
- threshold_type: tipo de umbralización (CV_THRESH_BINARY, CV_THRESH_BINARY_INV, CV_THRESH_TRUNC, etc.).

►void cvSmooth (const CvArr* src, CvArr* dst, int smoothtype, int param1, int param2, int param3, int param4)

Esta función realiza un filtrado de la imagen de entrada del tipo que se especifique.

Los parámetros que utiliza son:

- src: array fuente o de entrada.
- dst: array destino o de salida.
- smoothtype: tipo de filtrado CV_<tipo de filtrado>.
- param1, param2, param3 y param4: indican el tamaño de la máscara.

►CvScalar cvGet2D (IplImage* image, int i, int j)

Esta función obtiene el píxel situado en la posición (i, j) de la imagen especificada.

Los parámetros que utiliza son:

- image: imagen especificada.
- i: fila.
- j: columna.

►void cvSet2D (IplImage* image, int i, int j, CvScalar s)

Esta función tiene como objetivo introducir un valor en la posición (i, j) de la imagen especificada.

Los parámetros que utiliza son:

- image: imagen especificada.
- i: fila.
- j: columna.
- s: escalar a introducir en la imagen.

►void cvRectangle (CvArr* array, CvPoint pt1, CvPoint pt2, double color, int thickness CV_DEFAULT(1))

Esta función dibuja un rectángulo o un cuadrado sobre la imagen de entrada.

Utiliza los siguientes parámetros:

- array: imagen sobre la que se quiere pintar.
- pt1: punto de la esquina superior.
- pt2: punto de la esquina contraria.
- color: color de la figura. Se indica como CV_RGB().
- thickness: grosor de la línea.

►void cvCircle (CvArr* array, CvPoint center, int radius, double color, int thickness CV_DEFAULT(1))

Esta función dibuja un círculo sobre la imagen especificada.

Utiliza los siguientes parámetros:

- array: imagen sobre la que se dibuja el círculo.
- center: centro del círculo.
- radius: radio del círculo.
- color: color del círculo.
- thickness: grosor de la línea.

►CvMat* cvCreateMat (int rows, int cols, int type)

Esta función crea una matriz con un determinado número de filas y columnas.

Los parámetros que utiliza son:

- rows: número de filas.
- cols: número de columnas.
- type: tipo de los elementos de la matriz. Se indica con CV_<bit_depth> (S|U|F) C<number_of_channels>. Un ejemplo sería CV_8UC1.

►void cvmSet (CvMat* mat, int i, int j, CvScalar s)

Esta función introduce un escalar en la posición (i, j) de una matriz.

Los parámetros que aparecen en su cabecera son:

- mat: matriz en la que se quiere introducir un escalar.
- i: fila.
- j: columna.
- s: escalar que se quiere introducir.

►CvScalar cvmGet (CvMat* mat, int i, int j)

Esta función se encarga de extraer el valor del píxel en la posición (i, j) de la matriz especificada.

Utiliza los siguientes parámetros:

- mat: matriz de la que se quiere extraer el valor.
- i: número de la fila
- j: número de la columna.

►void cvRandInit (CvRandState* state, double lower, double upper, int seed)

Esta función inicializa un generador de valores aleatorios.

Utiliza los siguientes parámetros:

- state: puntero al estado de la estructura CvRandState que inicializa el rango de valores aleatorios.
- lower: cota inferior de valores aleatorios.
- upper: cota superior de valores aleatorios.
- seed: valor de 32 bits que inicializa el rango de valores aleatorios.

►void cvRand (CvRandState* state, CvArr* arr)

Esta función introduce números aleatorios en un array e inicializa el estado de valores aleatorios.

Utiliza los siguientes parámetros:

- state: puntero al estado de la estructura CvRandState que inicializa el rango de valores aleatorios.
- arr: array en el que se quieren introducir valores aleatorios.

►void cvMatMulAdd (const CvArr* A, const CvArr* B, const CvArr* C, CvArr* D)

Esta función calcula el producto de dos array y al resultado le suma otro array.

Los parámetros que incluye en su cabecera son:

- A: primer array a multiplicar.
- B: segundo array a multiplicar.
- C: tercer array a sumar.
- D: array de destino.

►void cvTranspose (const CvArr* src, CvArr* dst)

Esta función realiza la trasposición de un array.

Cuenta con los siguientes parámetros:

- src: array que se quiere trasponer.
- dst: array destino o de salida.

►int cvInvert (const CvArr* src, const CvArr* dst)

Esta función calcula el array inverso del array de entrada.

Utiliza los siguientes parámetros:

- src: array fuente o de entrada.
- dst: array destino o de salida.

►void cvSub (const CvArr* srcA, const CvArr* srcB, const CvArr dst)

Esta función realiza la resta de dos arrays.

Los parámetros que utiliza son los siguientes:

- srcA: primer array.
- srcB: segundo array.
- dst: array destino o de salida.

►void cvLine(IplImage* image, CvPoint pt1, CvPoint pt2, int color, int thickness)

Esta función sirve para unir con una línea dos puntos.

Los parámetros que incluye en su cabecera son los siguientes:

- image: imagen en la que se pinta la línea.
- pt1: primer punto (origen).
- pt2: segundo punto (destino).
- color: color de la línea. Se indica como CV_RGB().
- thickness: grosor de la línea.

►void cvReleaseMat (CvMat mat)**

Esta función se utiliza para liberar el contenido de la matriz especificada.

El parámetro que utiliza es el siguiente:

- mat: puntero doble a la cabecera de la matriz especificada.

►void cvMatMul (const CvArr* A, const CvArr* B, const CvArr* C)

Esta función realiza la multiplicación de dos matrices.

Utiliza los siguientes parámetros:

- A: primer array.
- B: segundo array.
- C: array destino donde se guarda el resultado.

6.4 Funciones creadas

En este apartado se incluyen las funciones que se han creado en la implementación de esta aplicación.

A continuación se describirán cada una de ellas.

Podemos considerar como funciones creadas cada uno de los métodos implementados en cada una de las clases. Estos métodos ya han sido incluidos en el capítulo 3 de este proyecto fin de carrera, donde se han explicado con detalle.

Otra función que fue creada durante la implementación es la siguiente:

►void on_mouse(int event, int x, int y, int flags, void* param)

Es una función que no pertenece a ninguna clase. Se crea fuera del programa principal para luego llamarla dentro de la función cvSetMouseCallback y asignar los eventos del ratón a la ventana que muestra las imágenes.

Su código se muestra a continuación:

```
void on_mouse(int event, int x, int y, int flags, void* param)
{

	if(event==CV_EVENT_RBUTTONDOWN)
	{
		sustraccion=true;
		elegido=true;

	}
	else if(event==CV_EVENT_LBUTTONDOWN){

		xSelect=x;
		ySelect=y;

		if(elegido==false)
		{
			elegido=true;
			printf("Primer objeto\n");
		}
		else
		{
			elegido2=true;
			printf("Segundo objeto\n");
		}
	}

}
```

Figura . Código de la función on_mouse.

Esta función nos permite seleccionar con el ratón sobre la ventana que nos muestra la imagen. En particular, cuando se clickea con el botón derecho, se realiza sustracción de fondo; en cambio, cuando se clickea con el botón izquierdo, se realiza segmentación basada en umbralizado, permitiendo extraer el valor de las coordenadas del píxel seleccionado.

Los parámetros que utiliza son los siguientes:

- event: indica qué acción realiza el ratón. Son de la forma CV_EVENT_<acción del ratón>.
- x: coordenada x del píxel seleccionado.
- y: coordanda y del píxel seleccionado.
- flags: se utilizan para realizar combinaciones de movimientos del ratón. Son de la forma CV_EVENT_<combinación de movimientos>.

- param: parámetro que se pasa a la función cvSetMouseCallback para indicar si se llama dentro de ella a la función on_mouse.

6.4 Bibliografía

A continuación se muestran las referencias que se han utilizado para la elaboración de este proyecto fin de carrera.

[1] R.C. González y R. E. Woods. *Digital Image Processing*. Prentice Hall, 2ª Edición, 2002.

[2] J. Vélez, A. Sánchez, A.B. Moreno y J.L. Esteban. *Visión por Computador*. Ed. Dyckinson y Serv. Public URJC, 2003.

[3] Javier González Jiménez. Visión por Computador. Ed. Paraninfo 2000.

[4] E. Trucco y A. Verri (1998). *Introductory Techniques for 3D Computer Vision*. Prentice Hall.

[5] L-Q Xu J.L. Landabaso and B Lei. Segmentation and tracking of multiple moving for intelligent video analisys. BT Technology Journal Vol. 22, Nº. 3, July 2004.

[6] Gary Rost Bradski, Adrian Kaehler. *Learning OpenCv, Computer Vision with the OpenCv Library*. O'Reilly, Septiembre, 2008.

[7] SourceForge.net: Open Computer Vision Library: http://sourceforge.net/projects/opencvlibrary/

[8] G. Welch, G. Bishop. *An Introduction to the Kalman Filter* (2002).

[9] http://es.wikipedia.org/wiki/Filtro_de_Kalman

[10] http://es.wikipedia.org/wiki/Filtro_de_partículas

[11] Arulampalam, M. (2002). A Tutorial on Particle Filter for Online Nonlinear/Non-Gaussian Bayesian Tracking. IEE Trans. On Signal Processing.

[12] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. Trans. ASME, J. Basic Engineering. Vol. 82, March 1960, págs. 94-35.

[13] Carlos Segura. Bayesian Filtering. Presentación en UPC-TALP.

[14] Trussel H. J., *Comment on Picture Thresholding using an Iterative Selection Method*, IEEE Transactions on Systems, Man and Cybernetics. Vol. SMC-9, pág. 311, 1979.