UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

I.T. Telecomunicación. Sonido e Imagen

Proyecto Fin de Carrera

"Tracking automático de objetos en secuencias de imágenes usando Filtro de Partículas"

Autor: Eva Chaparro Laso

Fdo.:

Director: Pedro M. Núñez Trujillo

Fdo.:

Tribunal Calificador:

Presidente:

Fdo:

Secretario:

Fdo.:

Vocal:

Fdo.:

CALIFICACIÓN:

FECHA:

A mis padres y a mi hermana,
por sus muestras de cariño
y por su constante apoyo

Índice General

Índice de figuras

Índice de Ecuaciones

Agradecimientos

Resumen

En los últimos años, la inteligencia artificial ha experimentado un desarrollo exponencial, gracias a los avances tecnológicos y a su expansión en muchos ámbitos de la vida cotidiana. Este progreso, hace que se esté investigando continuamente en cada una de sus aplicaciones como son la visión artificial, robótica, mundos virtuales, lingüística computacional...

La visión artificial tiene como finalidad, la extracción automática de información del mundo físico a partir de imágenes, utilizando para ello un computador. Una de las aplicaciones de la visión artificial es el seguimiento de objetos en secuencia de imágenes y se puede estudiar mediante varios métodos , como pueden ser el Filtro de Kalman o el Filtro de Partículas.

Por un lado el Filtro de Kalman define un conjunto de ecuaciones matemáticas que proveen una solución recursiva computacionamente eficiente del método de mínimos cuadrados.

En el otro lado, se encuentra el Filtro de Partículas que es un método secuencial de Monte Carlo aplicable a cualquier transición de estados, en el que se dispone de un modelo de medida.

El presente Proyecto Fin de Carrera consiste en el desarrollo de una aplicación que permita, como su propio nombre indica, realizar el seguimiento de objetos 2D en secuencia de imágenes utilizando la técnica del Filtro de Partículas. Para conseguir este objetivo, hemos utilizado las distintas técnicas de análisis de imágenes que nos proporcionan la librearía de procesado de

imágenes OpenCV (Open Source Computer Vision Library), además de las instrucciones necesarias para la implementación de un algoritmo que nos permita estimar el estado de un objeto que cambia a lo largo del tiempo.

Para ello, lo primero que tenemos que hacer es extraer puntos característicos con propiedades singulares captados por una cámara. Posteriormente, no hace falta ningún elemento artificial para marcar la búsqueda, simplemente aplicaremos a estos puntos el filtro de partículas, calculando así el estado de esos puntos en el instante siguiente.

Tras los resultados obtenidos con uno y varios móviles, movimientos no plano paralelos...se comprueba que el rendimiento del filtro cumple con las especificaciones propuestas.

Además de estudiar bajo diferentes condiciones experimentales el Filtro de Partículas aplicado al seguimiento de objetos, y a pesar de que no era un objetivo fundamental del proyecto, se ha conseguido implementar una aplicación del Filtro de Partículas que es capaz de seguir uno o varios móviles en secuencias de imágenes.

Capítulo 1
Introducción

La visión artificial, visión por computador o visión técnica es una disciplina que tiene como finalidad, reproducir artificialmente el sentido de la vista mediante el procesamiento e interpretación de imágenes, capturadas con distintos tipos de sensores (en la mayoría de los casos, cámaras), y utilizando para ello las prestaciones de los ordenadores [1]. Es una tarea muy difícil, que para los humanos parece trivial pero es infinitamente compleja para los computadores.

La visión artificial es una gran herramienta para establecer la relación entre el mundo tridimensional y sus vistas bidimensionales. Por medio de esta teoría se pueden hacer, por una parte, una reconstrucción del espacio tridimensional a partir de sus vistas, y por otra parte, llevar a cabo una simulación de una proyección de una escena tridimensional en la posición deseada a un plano bidimensional.

Algunas de las aplicaciones de la visión artificial son las siguientes:

- Fotogrametría: en ella se persigue la realización de mediciones del espacio 3D a partir de fotografías tomadas en él. Utilizando, la visión artificial es posible medir superficies, construcciones, objetos...

- Rectificación métrica: mediante visión artificial, se pueden hacer correcciones de perspectiva.
- Reconstrucción 3D: a partir de las vistas y mediante la técnica de triangulación se puede obtener un modelo 3D del objeto proyectado en las vistas.
- Matching y Tracking: por medio de estas dos técnicas es posible encontrar la correspondencia entre puntos de varias imágenes. Los puntos correspondientes son aquellos que representan una proyección del mismo punto físico en el espacio 3D.
- Estimación del movimiento: mediante una cámara que toma imágenes de un objeto en movimiento es posible estimar el movimiento del objeto a partir de los puntos de correspondencia en la secuencia de imágenes.

La visión artificial, en un intento de reproducir el comportamiento del ser humano, define tradicionalmente cuatro etapas principales, las cuales son detalladas en la figura 1.1:

- La primera fase, consiste en la *Captura* o *Adquisición* de las imágenes digitales mediante algún tipo de sensor, en nuestro caso, una cámara digital.
- La segunda etapa consiste en el tratamiento digital de las imágenes. En esta etapa de *Preprocesamiento*, es donde, mediante filtros y transformaciones geométricas, se eliminan partes indeseables de la imagen o se realzan partes interesantes de la misma.
- La siguiente fase se conoce como *Segmentación*, y consiste en la extracción de los elementos que nos interesan de la imagen.
- Por último, se llega a la etapa de *Reconocimiento* o *Clasificación*. En ella se pretende distinguir los objetos

segmentados, gracias al análisis de ciertas características que se establecen previamente para diferenciarlos.

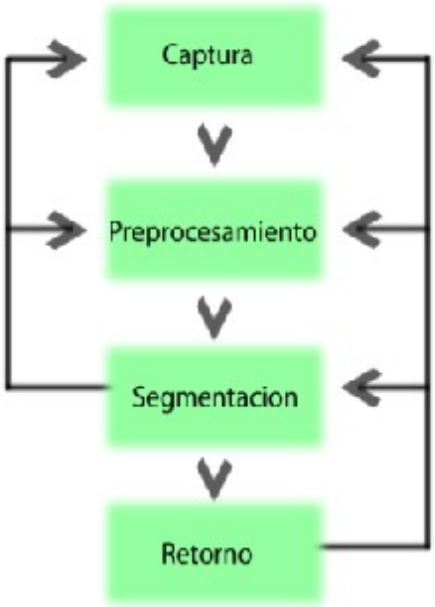


Figura 1.1. Diagrama de bloques de las etapas típicas en un sistema de Visión Artificial

Lo normal, es que estas etapas no se desarrollen de forma secuencial, aplicando en este caso una realimentación. Por ejemplo de la etapa de reconocimiento se puede volver a la etapa de segmentación, a la de preprocesamiento o incluso a la de captura si falla alguna de ellas.

1.1. Objetivos del proyecto

Como hemos visto anteriormente, una de las aplicaciones de la visión artificial, es el seguimiento de objetos en secuencia de imágenes. La complejidad de esta aplicación aparece cuando el seguimiento debe funcionar en tiempo real. En este caso, se utilizan métodos probabilísticos como son el Filtro de Partículas o el

Filtro de Kalman.

El objetivo fundamental del presente Proyecto Fin de Carrera es la implementación de un algoritmo que realice el seguimiento de objetos en secuencia de imágenes 2D utilizando el Filtro de Partículas y ver el funcionamiento del mismo bajo diversas condiciones experimentales.

En nuestro caso y aunque no era objetivo inicial del proyecto, se ha implementado un algoritmo que permite realizar el seguimiento de hasta dos objetos móviles, además de que existen dos formas de realizar la umbralización de la imagen, la primera es extrayendo alguna característica física del objeto del que queremos realizar el seguimiento (color), a esto se le conoce como segmentación por umbralización y la otra es utilizando el método de sustracción de fondo, que explicaremos con más detalle en capítulos posteriores.

Para llegar al objetivo final del proyecto, hemos tenido que desarrollar en primer lugar los siguientes sub-objetivos:

1. Aprender a utilizar el entorno de programación Visual C++
2. Familiarizarse con la librería OpenCV
3. Crear nuevos proyectos en Visual C++ añadiendo la librería OpenCV
4. Aprender técnicas de procesamiento y tratamiento digital de imágenes
5. Aprender el funcionamiento del Filtro de Partículas
6. Implementar cada una de las partes del Filtro de Partículas básico
7. Coleccionar datos para evaluar el funcionamiento del filtro en diferentes condiciones
8. Aplicar el Filtro de Partículas básico a una imagen 1D y a secuencia de imágenes 2D
9. Desarrollo del Filtro de Partículas utilizando la segmentación

basada en umbralizado

10. Aplicar Filtro de Partículas que utiliza la segmentación basada en umbralizado en imagen 1D y en secuencia de imágenes 2D
11. Desarrollo del Filtro de Partículas empleando sustracción de fondo
12. Aplicar Filtro de Partículas que emplea sustracción de fondo sobre imagen 1D y secuencia de imágenes 2D
13. A la vista de los resultados obtenidos, proponer alternativas para mejorar el rendimiento del filtro.

Para la consecución principal del proyecto ha sido necesario abordar por separado cada uno de los sub-objetivos descritos anteriormente. El tiempo que hemos necesitado para cada uno de ellos se muestran en el siguiente diagrama de Gantt:

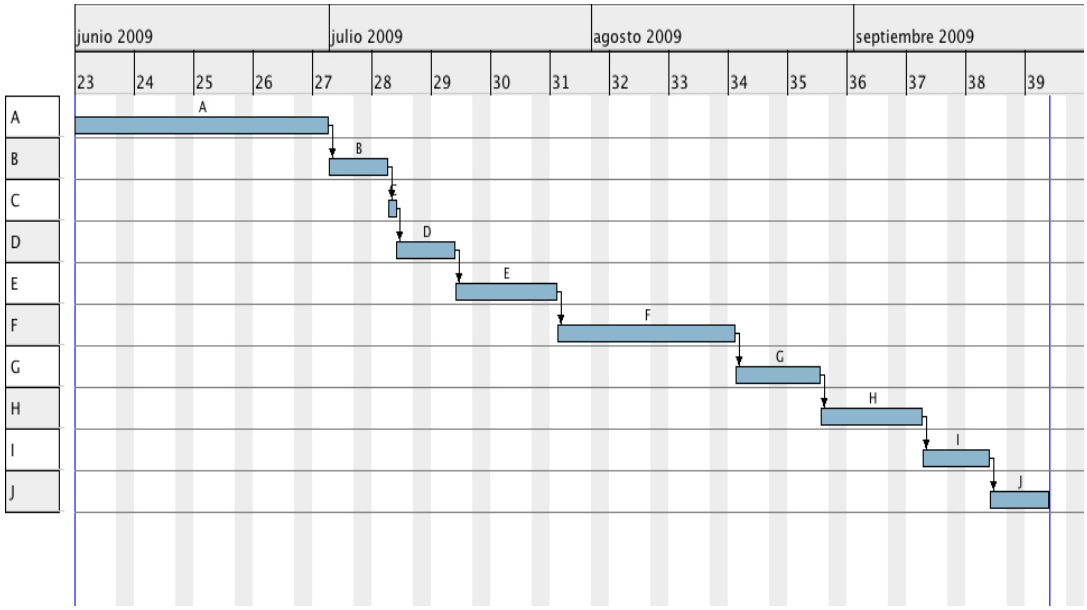


Figura 1.2. Diagrama de Gantt del proyecto

A = Aprender el entorno Visual C++ (30 días)

B= Familiarizarse con OpenCV(5días)

C=Aprender a usar Visual C++ incluyendo OpenCV (1día)

D=Aprender técnicas de tratamiento de imágenes sintéticas y reales (7 días)
E = Comprender el Filtro de Partículas (10días)
F = Implementar el Filtro de Partículas Básico (20días)
G = Segmentación por Umbralizado (10días)
H=Sustracción de Fondo (10días)
I = Desarrollar las pruebas correspondientes (7 días)
J= Elaboración de la memoria del proyecto (10 días)

1.2. Organización de la memoria

La memoria del presente Proyecto Fin de Carrera va a ser desarrollada en cinco capítulos y un anexo.

En el primer capítulo se introduce al lector en el mundo de la visión artificial, comentando sus principales aplicaciones en el mundo actual y cada una de las partes en las que se divide. Más concretamente, se explica la motivación de este proyecto, que no es otra que la realización de un algoritmo que realice el seguimiento de objetos en secuencia de imágenes utilizando el Filtro de Partículas. De ahí, que se explique con detenimiento en qué consiste el seguimiento de objetos y el Filtro de Partículas. Además, se comentan cada una de las herramientas utilizadas para la realización del proyecto.

A continuación el capítulo 2 de este proyecto lo dedicaremos a definir de forma descriptiva en qué consiste la aplicación desarrollada. Explicaremos cada uno de los métodos de umbralización empleados, segmentación por umbralizado y sustracción de fondo, además de cada una de las fases de las que se compone el Filtro de Partículas, inicialización, ponderación, estimación, selección y difusión. Finalmente, explicaremos como hemos aplicado el Filtro de Partículas sobre dos objetos

conjuntamente.

En el tercer capítulo, se explica como se ha desarrollado la implementación de la aplicación para que realice seguimiento de objetos en secuencia de imágenes, usando el método de Filtro de Partículas. Se explica cada clase implementada, con sus métodos...además del programa principal.

El cuarto capítulo, lo hemos reservado para describir los resultados de las pruebas realizadas, al aplicar el programa desarrollado en ellas. Estas pruebas han sido desarrolladas en diferentes entornos, lugares abiertos con cambios de iluminación y lugares cerrados. Además, para comprobar la eficiencia del filtro, hemos hecho pruebas cambiando los diferentes parámetros del filtro, así como los umbrales y la sensibilidad en los métodos de umbralización.

En el capítulo 5 exponemos las conclusiones extraídas tras la finalización del presente Proyecto Fin de Carrera, además de enfocar distintas líneas de trabajo futuro.

Por último, en el capítulo 6 se explicará de una forma de detallada los pasos necesarios para la instalación de OpenCV en Windows. Además, se explicarán cada una de las funciones y estructuras que hemos utilizado de esta librería a lo largo de la implementación de nuestro programa.

1.3. Tracking o seguimiento de objetos

El *seguimiento de objetos* es una de las muchas aplicaciones de la visión artificial, que consiste en determinar el estado de un objeto que esta en movimiento [2]. A continuación, mostramos un ejemplo de *seguimiento de objetos*:

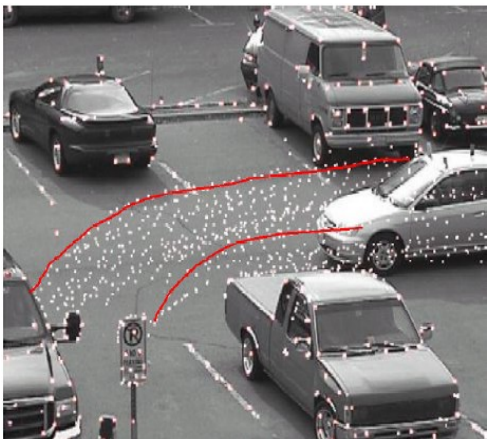


Figura 1.3. Trayectoria del seguimiento de un coche

En las últimas décadas, este fenómeno se ha estudiado en los diferentes ámbitos de la vida, como pueden ser: vigilancia y monitorización, para la interacción con personas ya sea por reconocimiento de gestos o simplemente identificación;terapias médicas, para mejorar la calidad de vida de las personas; control de tráfico, ya sea para gestionarlo o detectar accidentes;ámbito militar, para ver la trayectoria de un proyectil; juegos interactivos, como podría ser el Eye Toy de la PlayStation.

Aunque el seguimiento de objetos es una aplicación estudiada desde hace mucho tiempo, sigue siendo un problema para la investigación, debido a las múltiples complicaciones que conlleva, a causa de fenómenos tales como la pérdida de información debida a la proyección del mundo 3D en una imagen 2D, el ruido de las imágenes, el movimiento de los objetos, cambios en la iluminación de la escena, forma de los objetos...

El principal problema del seguimiento de objetos reside en cómo definir al objeto. Una vez seleccionada alguna característica física del objeto,por ejemplo, el color, es sencillo llevar un registro de la trayectoria del objeto.

En el caso, de que el movimiento del objeto sea lo suficientemente alto como para que su localización entre una imagen y la siguiente de la secuencia sea muy diferente,

utilizaremos algoritmos eficientes de predicción como lo son el Filtro de Partículas y el Filtro de Kalman.

1.3.1. Diferencias Filtro de Partículas y Kalman

El Filtro de Kalman es un algoritmo definido por un conjunto de ecuaciones matemáticas que sirven para calcular el estado de un sistema dinámico lineal, estando éste sometido a un ruido blanco o ruido gaussiano. El filtro tiene dos fases iterativas, predicción y corrección. A partir de unos estimadores iniciales, el filtro va prediciendo y ajustándose con cada nueva medida.

El Filtro de Kalman es óptimo cuando el sistema puede ser descrito mediante ecuaciones lineales. No obstante, ya se ha conseguido implementar un Filtro de Kalman que aborde problemas no descritos mediante ecuaciones lineales.

El Filtro de Partículas es un método secuencial de Monte Carlo que sirve para estimar el estado de un sistema que cambia a lo largo del tiempo. El Filtro de Partículas es muy utilizado en visión artificial para el seguimiento de objetos. El filtro consta de cinco fases, inicialización, ponderación, estimación, selección y difusión. Excepto la de inicialización, el resto son fases son iterativas.

El Filtro de Partículas no ofrece unos resultados tan robustos como los que ofrece el Filtro de Kalman ante situaciones óptimas, aunque los resultados siguen siendo en general muy buenos. No obstante, la principal diferencia entre ambos filtros radica en que el Filtro de Kalman puede representar solamente la estimación del estado por una gaussiana uni-modal, mientras que el Filtro de Partículas pueden representar densidades multi-modales complejas empleando una gran cantidad de partículas aleatoriamente muestreadas.

1.4. Estado del arte

En este apartado, vamos a centrarnos en la explicación del método de Monte Carlo y del Filtro de Partículas.

1.4.1. Método de Monte Carlo

El método de Monte Carlo es una técnica numérica para calcular probabilidades utilizando una secuencia de números aleatorios[8]. Debe su nombre a su clara analogía con los juegos de ruleta de los casinos, el más célebre el Casino de Montecarlo (Principado de Mónaco), al tomar una ruleta como generador simple de números aleatorios.

El uso real del método de Monte Carlo como herramienta de investigación, viene de los trabajos realizados para la elaboración de la bomba atómica durante la Segunda Guerra Mundial [8].

La importancia actual del método de Monte Carlo se basa en la existencia de problemas que tienen una difícil solución usando métodos puramente analíticos, pero que dependen de factores aleatorios. Con el surgimiento de las computadoras, el método de Monte Carlo experimentó un gran desarrollo, ya que con ellas se podían llegar a hacer cálculos que hasta entonces eran inconcebibles.

El método de Monte Carlo es aplicable a cualquier tipo de problema, ya sea de azar o no. A diferencia de otros métodos que se basan en evaluaciones de N puntos en un espacio para producir una solución aproximada, el método de Monte Carlo tiene un error absoluto que decrece como $\sqrt{\frac{1}{N}}$ en virtud del teorema del límite central.

" El teorema del límite centra indica que, en condiciones muy generales, la distribución de la suma de variables aleatorias tiende a una distribución normal cuando la cantidad de variables es muy grande"

1.4.2. Filtro de partículas

El Filtro de Partículas es un método secuencial de Monte Carlo usado muy a menudo en visión artificial para seguimiento de objetos en secuencia de imágenes [7]. El objetivo para el que se usa este método es poder estimar el estado de un sistema cuyo movimiento es tan alto, que no se puede seguir su trayectoria de una imagen a otra.

El Filtro de Partículas ha sido denominado de varias formas a lo largo del tiempo. Algunos ejemplos son, algoritmo de Condensation [11] recogido en ICONDESATION [12], presentado por Michael Isard y Andrew Brake para seguir contornos, Método Secuencial de Monte Carlo (Sequencial Monte-Carlo Methods) [9], Filtro Bootstrap(Bootstrap Filter) propuesto por Gordon y Filtro de Supervivencia del Más Apto. No obstante, a todos estos filtros en la actualidad se conocen como Filtro de Partículas. Todos estos filtros, son algoritmos similares que se utilizan para propagar las partículas, utilizando el modelo de movimiento y el modelo de verosimilitud.

- Modelo de movimiento, se usa para predecir el estado del sistema en el instante siguiente
- Modelo de verosimilitud, se usa para calcular el peso asociado al estado.

El Filtro de Partículas fue propuesto en 1993 por N.Gordon, D.Salmond y A.Smith, para la implementación de filtros bayesianos

recursivos [6]. Éstos filtros se basan en la aplicación del Teorema de Bayer.

> *"El Teorema de Bayes, enunciado por Thomas Bayes, en la teoría de la probabilidad, es el resultado que da la distribución de la probabilidad condicional de un evento aleatorio A dado B en términos de la distribución de probabilidad condicional del evento B dado A y la distribución de probabilidad marginal de sólo A"*

Las primeras ideas sobre el Filtro de Partículas en forma de muestreo secuencial se iniciaron en los años 50, pero estas ideas se ignoraron debido a que, para estas fechas el Filtro de Partículas no resolvía correctamente los problemas para los que se aplicaba. Las mayores contribuciones al filtro surgen con la aparición del resampling (remuestreo) y el desarrollo de la velocidad de los procesadores.

Desde entonces, la actividad investigadora en este campo ha aumentado produciendo numerosas mejoras en el funcionamiento del filtro y aumentando sus numerosas aplicaciones.

Los Filtros de Partículas son una alternativa a los Filtros de Kalman, con la ventaja, de que para un número adecuado de muestras, llegan a la estimación Bayesiana óptima, siendo más exactos que los Filtros de Kalman.

Desde un aspecto más general, el Filtro de Partículas es un método recursivo, en el que se diferencian cuatro etapas fundamentales:

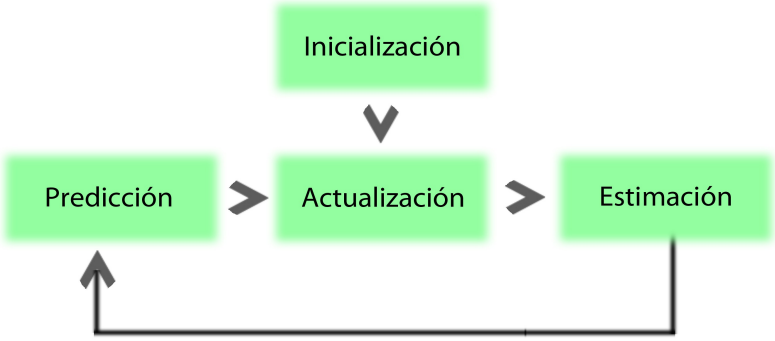


Figura 1.4.Etapas de un Filtro de Partículas

- Inicialización: en esta etapa se generan aleatoriamente valores para las variables que definen el estado de cada partícula.
- Actualización: en esta etapa se asigna un valor (peso) a cada una de las partículas anteriormente creadas que expresa la calidad de la predicción realizada por cada una de ellas.
- Estimación: se estima un estado del sistema en cada instante. Usualmente es el estado de la partícula con más peso, o un promedio ponderado de todas las partículas.
- Predicción: a partir de este conjunto de partículas, se selección aquellas de mayor calidad y sobre ellas se aplica el modelo de movimiento, con el fin de estimar el estado del sistema en el instante siguiente.

Después de esta etapa de predicción, se obtiene un nuevo conjunto de partículas al que se le vuelve a aplicar la etapa de actualización, hasta que termine la secuencia de datos.

A continuación, se explica detalladamente como funcionan cada una de estas etapas.

1. Etapa de Inicialización

   El Filtro de Partículas se compone de un conjunto de

muestras en el espacio del problema a los cuales se les asigna un peso de acuerdo con ciertas características de medidas. Una muestra o partícula se define por medio de un estado x y un peso Π.

$$p = \{x, \pi\} \mid x \in X, \pi \in [0,1] \qquad (1.1)$$

En la ecuación (1.1), la x son posibles estados del proceso, y Π son los pesos o verosimilitud asociada a cada estado x y normalizados entre [0,1].

La idea clave del Filtro de Partículas consiste en la representación y la generación recursiva de los estados de la función de densidad de probabilidad (FDP). Para ello, la FDP la vamos a representar como un conjunto de partículas, dadas por:

$$S_n = \{(x_1, \pi_1) \dots (x_n, \pi_n)\}$$

$$\sum_{i=1}^{n} \pi_i = 1 \qquad (1.2)$$

En la siguiente figura se representa un muestreo continuo de una función de densidad de probabilidad por partículas, cuyo tamaño hace referencia al peso o probabilidad de la misma.

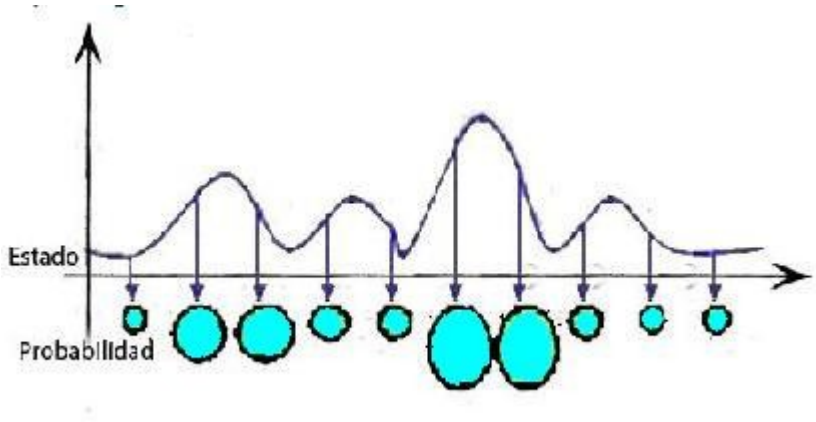


Figura 1.5.Muestreo de una función de distribución

Para un conjunto muy grande de muestras, se puede demostrar que la FDP sería muy exacta, de hecho para un numero infinito de partículas la FDP estimada coincidiría con la FDP real.

Inicialmente, el conjunto de partículas se escoge de la distribución a priori $p(x_0)$. Si no existe esta información, las partículas se distribuyen uniformemente por el espacio de estados.

2. Etapa de Actualización

Una vez que tenemos este conjunto de partículas, las tenemos que multiplicar por una función para modificar el peso de cada una de ellas. Por lo tanto, sea un conjunto de partículas como el especificado en la ecuación (1.2) y una función h(x), continua, no negativa y acotada lejos de 0. Al multiplicarlas, obtendremos un nuevo conjunto de partículas que estarán en la misma posición que el antiguo conjunto, pero su peso habrá cambiado siendo ahora proporcional a la función que se le ha aplicado, h(x).

3. Etapa de estimación

Posteriormente, aplicamos a las N partículas el modelo de movimiento, obteniéndose así un nuevo conjunto de partículas. Esto se expresa como una distribución de probabilidades.

Por lo tanto, sea el conjunto de partículas y la distribución de

probabilidades siguientes:

$$S = \{(x_i, \pi_i)_{i=1}^{n}, n = 1,2, \dots\}$$

$$p(x'|x) \tag{1.3}$$

al aplicar la función de probabilidad se obtiene otro conjunto de partículas que cumple:

$$S' = \{(x'_i, \pi'_i)_{i=1}^{n}, n = 1,2, \dots\}$$

$$x'_i \; \alpha \; p(x'_i|x_i) \tag{1.4}$$

$$\pi'_i = \frac{1}{N}$$

Las nuevas partículas representan la predicción de la variable de estado, sin considerar la observación, se obtiene el peso asociado a cada partícula.

4. Etapa de Predicción

Por último, se vuelve a muestrear el conjunto de partículas, extrayendo N partículas del conjunto actual, proporcional al peso de cada una. En este nuevo conjunto tienen más posibilidades de desaparecer las partículas de menor peso. Una vez construido el nuevo conjunto de partículas, según la probabilidad de cada una, se les asocia un peso. Este nuevo conjunto de partículas constituye una representación muestral de la probabilidad a posteriori.

En la siguiente figura, podemos ver representado, las operaciones explicadas anteriormente:

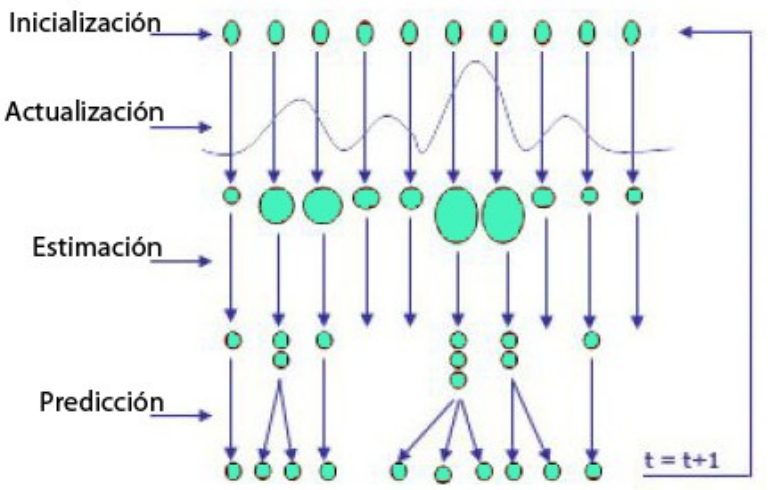


Figura 1.6. Funcionamiento de un Filtro de Partículas

1.5. Herramientas y librerías

Para la realización de este proyecto han sido necesaria la utilización de las siguientes herramientas y librerías:

- Visual C++
- Librería de tratamiento de imágenes OpenCV
- Digital Video Camera Recorder (Sony DCR-HC18E)
- GanttProject

1.5.1. Visual C++

Visual C++ es un entorno de desarrollo para escribir programas en diferentes lenguajes de programación como son C, C++ o C#. Además, permite la programación orientada a objetos (POO) conjuntamente con el sistema de desarrollo SDK de Windows.

Al ser un entorno de desarrollo integrado incluye las siguientes herramientas de desarrollo:

- editor de texto
- compilador/enlazador
- depurador
- visor de datos y dependencias

El sistema SDK esta compuesto por un conjunto complejo de funciones, para hacer más fácil la labor, Visual C++ incluye la librería de clases MFC (Microsof Foundation Classes) que permite crear y gestionar de manera intuitiva componentes típicos de Windows.

Para la elaboración de este proyecto, hemos utilizado el lenguaje C++ y el paradigma de programación orientada a objetos (POO), con el objetivo de obtener la meta de cualquier modelo de programación estructurada convencional, que no es otro que el de establecer una serie de normas de desarrollo que aseguren y faciliten la mantenibilidad y reusabilidad del código.

En la programación orientada a objetos es muy importante conocer los conceptos de objeto, mensaje, método y clase.

- Objeto: entidad que tiene unos atributos (variables) y unos métodos para operar con ellos.
- Mensaje: es el nombre de uno de los métodos de un objeto. Cuando se pasa un mensaje a un objeto, éste responde ejecutando el código de la función asociada.
- Método: determina como debe de actuar un objeto cuando se produce el mensaje asociado. En C++ un método es una función miembro del objeto.
- Clases: definición de un tipo de objetos.

1.5.2. OpenCV

Open Source Computer Vision Library (OpenCV) es una librería que surge en 1999 y que fue desarrollada por Intel Corporation por lo que está optimizada para sus procesadores, aunque también es muy eficiente en otros procesadores. Fue publicada bajo licencia BSD, por lo que OpenCV puede ser usada tanto para investigación como para usos comerciales.

Esta librería esta orientada especialmente para el procesamiento y el tratamiento digital de imágenes en tiempo real. Para ello, OpenCV ha sido implementada junto con la librería Intel IPL (Image Processing Library).

Es rápida, de fácil uso y está en continuo desarrollo.

La librería OpenCV es independiente de la plataforma en la que se ejecute, existiendo versiones para Linux, Mac OS y Windows.

OpenCV incluye más de 350 algoritmos y nos proporciona:

- Métodos de análisis de movimiento y seguimiento de objetos
- Métodos de análisis de imágenes
- Métodos de análisis de estructuras
- Métodos de reconocimiento de objetos
- Métodos de reconstrucción en 3D
- Estructuras y operadores básicos
- Interfaz gráfica y adquisición

OpenCV implementa una gran variedad de herramientas para la interpretación de la imagen. Además de funciones para realizar binarización, filtrado, estadísticas de la imagen y pirámides, OpenCV es una librería que contiene algoritmos para las técnicas

de calibración, detección de rasgos, seguimiento o tracking, análisis de la forma, análisis del movimiento, reconstrucción 3D, segmentación de objetos y reconocimiento.

Para la elaboración de este proyecto, hemos utilizado tres módulos de OpenCV, que se muestran en la siguiente figura:

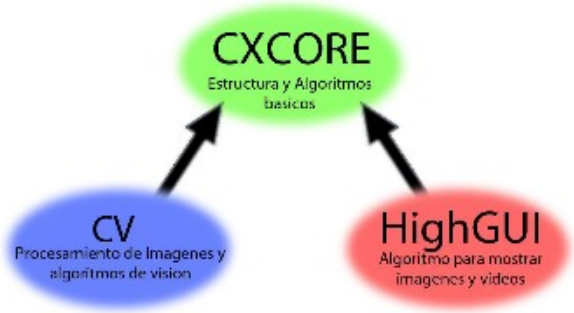


Figura 1.7. Módulos de OpenCV utilizados en este proyecto

1.5.3. Digital Video Camera Recorder (Sony DCR-HC18E)

En la realización de este proyecto, hemos utilizado una cámara de video digital, concretamente, Sony DCR-HC18E, con el objetivo de obtener imágenes con mayor calidad y por lo tanto, obtener mejores resultados.

1.5.4. GanttProject

GanttProject es un programa que se utiliza para planificar, supervisar, controlar y rastrear cada aspecto de un proyecto. En nuestro caso hemos utilizado este programa para la realización del diagrama de Gantt para planificar los objetivos de nuestro proyecto.

Capítulo 2
Descripción de la aplicación

En este segundo capítulo, vamos a explicar de una forma descriptiva en que consiste la aplicación desarrollada para realizar el seguimiento de objetos en secuencia de imágenes 2D, utilizando el método del Filtro de Partículas.

Aunque no era objetivo de este proyecto, esta aplicación cuenta con dos posibles técnicas para extraer información de la localización del o de los objetos, para después poder calcular los pesos de las partículas generadas. Estas técnicas son las siguientes:

- Segmentación por Umbralizado
- Sustracción de Fondo

Además, en este capítulo se explicarán cada una de las fases de las que se compone el Filtro de Partículas, y que hemos implementado para su correcto funcionamiento.

Finalmente, se procederá a la explicación del Filtro de Partículas para dos objetos.

2.1. Segmentación por Umbralizado

La *segmentación* es uno de los pasos más importantes y más complicados en el procesado de imágenes, ya que permite la cuantificación y la visualización de los objetos de interés [14]. La segmentación consiste en la división de la imagen en las partes u objetos que la forman. Esta división de la imagen se hace atendiendo a las características similares que existen entre los píxeles de la misma.

Se pretende simular el funcionamiento del cerebro humano cuando al recibir una imagen, realiza un reconocimiento de cada una de las partes que la forman, dando a cada región un significado físico que corresponde con la realidad.

Existen varias técnicas de segmentación:

- Técnicas de segmentación basadas en los valores del píxel
- Técnicas de segmentación basadas en el área
- Técnicas de segmentación basadas en la extracción de bordes
- Técnicas de segmentación basadas en la física

Como ya hemos dicho anteriormente, la segmentación es una técnica complicada, con diversos problemas, ruido de las imágenes, iluminación de la escena, ambigüedad de los objetos, entre otros. Algunos de ellos se pueden solucionar realizando un procesado de bajo nivel, pero en otras ocasiones habrá que improvisar y realizar técnicas a medida.

Nosotros nos vamos a centrar, en la técnica de segmentación basadas en los valores del píxel, también conocida como

segmentación por umbralizado.

Algunas de las aplicaciones de la segmentación por umbralizado son las siguientes:

- Medicina: detección de células en citología; localización de fracturas;análisis de ecografrías; imágenes de rayos X.
- Industria:supervisión automática de procesos industriales, como control de calidad, vigilancia o detección de fallos.
- Análisis de fotografías áreas o por satélite: clasificación de diferentes tipos de vegetación para el estudio de su degradación; clasificación y control de cultivos de agricultura; detección de objetos.

El método de *segmentación por umbralizado* permite convertir una imagen en color o en escala de grises a binario de forma que los píxeles cuyos niveles de intensidad superen cierto umbral tengan distinto valor al resto.

Existen muchas técnicas para definir el umbral, pero las más empleada es mediante el análisis del histograma de la imagen [6].El umbral puede ser fijo o puede ser variable. La umbralización fija establece de antemano el valor del umbral, mientras que la variable lo calcula en tiempo de ejecución en función de determinadas propiedades de la imagen.

Si los valores de intensidad del objeto y del fondo difieren claramente, el histograma de la imagen presentará un aspecto bimodal, dos grandes máximos con un gran espacio entre ellos, por lo tanto se logrará una separación excelente entre el objeto y el fondo, al poner un umbral T en esta zona del histograma.

Cualquier píxel de la imagen que esté por encima del umbral T será considerado como objeto y cualquier píxel de la imagen que

esté por debajo del umbral será considerado como fondo.

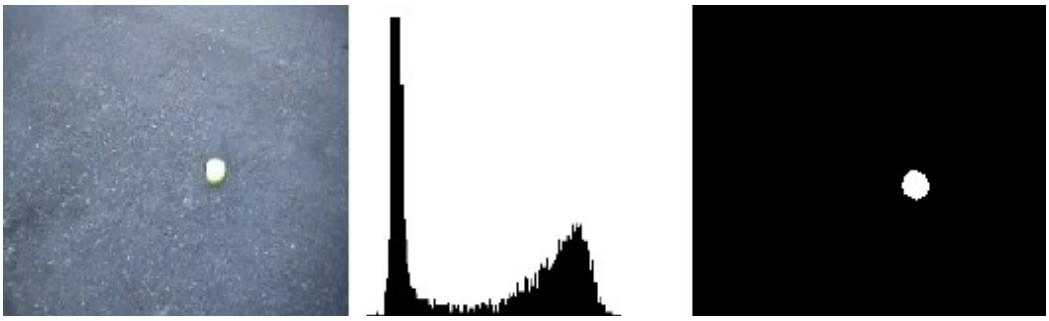
Por lo tanto, podemos definir la umbralización por la siguiente ecuación:

$$g(x,y) = \begin{cases} 1 & \text{si } f(x,y) \geq U \\ 0 & \text{en cualquier otro caso.} \end{cases} \qquad (2.1)$$

Aunque en ocasiones es conveniente, hacer justamente lo contrario, es decir:

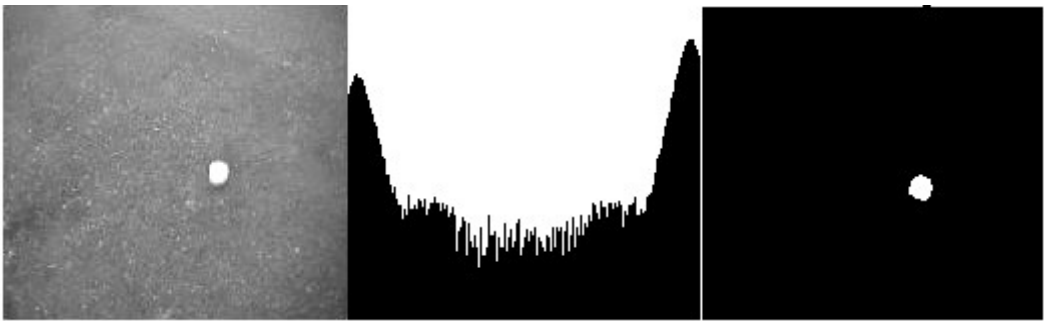
$$g(x,y) = \begin{cases} 1 & \text{si } f(x,y) \leq U \\ 0 & \text{en cualquier otro caso.} \end{cases} \qquad (2.2)$$

A continuación,se muestra dos ejemplo de *segmentación por umbralizado*, el primero de una imagen en color y el segundo de la misma imagen en escala de grises, teniendo en cuenta los histogramas de las imágenes y aplicando la ecuación (2.1) en ambos casos:



(a) (b) (c)

Figura 2.1. (a) Imagen original en color; (b) Histograma bimodal de la imagen(a); (c)Imagen Umbralizada



(a) (b) (c)

Figura 2.2. (a) Imagen original en escala de grises; (b) Histograma bimodal de la imagen en color(a); (c)Imagen Umbralizada

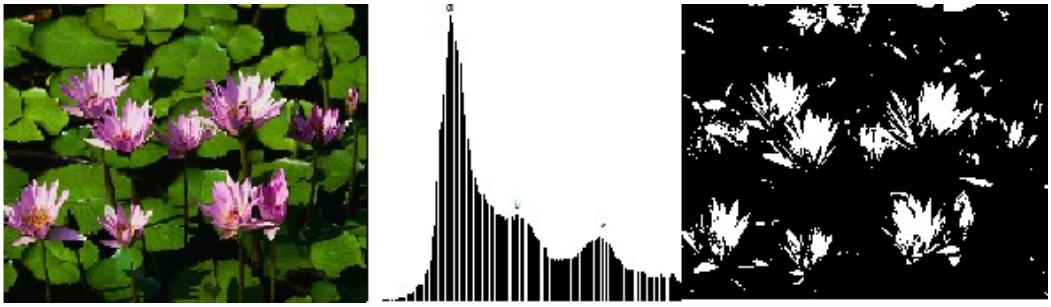
Como podemos ver al aplicar la segmentación por umbralizado a una imagen en color o a una imagen en escala de grises, el resultado es el mismo, pero en cuanto al tiempo, el procesado de la imagen en escala de grises es mucho más rápido que el de la imagen en color, al tener únicamente un canal.

Un caso más general, es cuando el histograma tiene un aspecto trimodal, tetramodal... Ahora, habría que poner varios valores umbrales, por ello estos casos son menos viables, ya que es más difícil la determinación de los umbrales necesarios para determinar los objetos de interés.

En este caso habría que aplicar la siguiente ecuación:

$$g(x,y) = \begin{cases} 1 & \text{si } U1 \leq f(x,y) \leq U2 \\ 0 & \text{en cualquier otro caso.} \end{cases} \qquad (2.3)$$

A continuación, se muestra un ejemplo de *segmentación por umbralización*, en el caso, de que el histograma presente más de dos modos.



(a) (b) (c)

Figura 2.3.(a) Imagen original; (b) Histograma trimodal de la imagen(a); (c)Imagen Umbralizada

Entre las ventajas que proporciona la segmentación por umbralizado, cabe destacar que es una segmentación rápida y con un baje coste computacional [18].

2.2. Sustracción de Fondo

Otra de las técnicas utilizadas para extraer información el objeto de interés en este proyecto es el método de *sustracción de fondo.* Este método, es una tarea crítica y fundamental en muchos usos de visión artificial.

La sustracción de fondo (background subctraction) identifica objetos móviles sobre un fondo estático conocido. Éste fondo debe de ser robusto ante cambios de iluminación. Además debería de evitar descubrir objetos no inmóviles de fondo como puede ser el movimiento de las hojas, lluvia, nieve y sombras de los objetos que están en movimiento.

La técnica de la sustracción de fondo consiste en capturar una imagen que el sistema debe entender como fondo libre de objetos y sustraerla de cada imagen de la secuencia(restando a la imagen fondo cada una de las imágenes de las secuencia). Como resultado de su aplicación se obtiene una matriz (imagen)de ceros excepto en las regiones con objetos que difieran del fondo, donde

su valor es 1.

Es decir, sea una imagen fondo I_f, una imagen en un instante determinado I_t y un umbral U, se obtiene la imagen diferencia I_d a partir de la siguiente ecuación:

$$I_d(x,y) = \begin{cases} 1, \; si \; | I_f(x,y) - I_t(x,y) | > U \\ 0, \; en \; caso \; contrario \end{cases} \qquad (2.4)$$

A continuación, podemos ver un ejemplo de la aplicación del método explicado a una imagen:

Figura 2.4.(a) Fondo; (b) Fondo + Objetos de interés; (c)Imagen Binarizada

Para poder aplicar la sustracción de fondo, en primer lugar hay que tener las imágenes, tanto el fondo como la secuencia de imágenes en escala de grises. En escala de grises cada píxel de la

imagen posee un valor equivalente a una graduación de gris.

En la siguiente figura, podemos ver una imagen original y su correspondiente imagen en escala de grises.

Figura 2.5.(a) Imagen original; (b) Imagen en escala de grises

2.3. Filtro de Partículas

En este apartado se describen cada una de las fases que se han necesitado para la implementación del Filtro de Partículas. Para ello, hemos elaborado el siguiente diagrama de flujo:

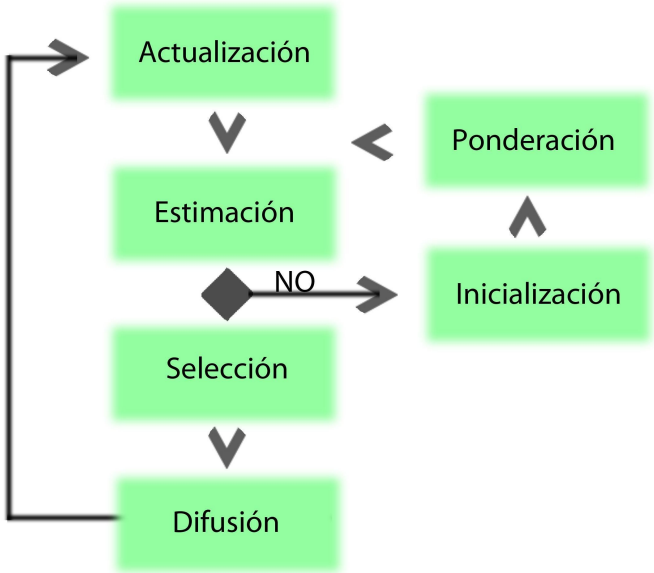


Figura 2.6.Diagrama de Flujo de un Filtro de Partículas

- *Actualización* : esta etapa simplemente se aplica para saber sobre que imagen estamos trabajando.
- *Inicialización:* en esta fase se se genera un vector de N partículas aleatorias y con peso cero, en el espacio de búsqueda. Hay que controlar que cuando se generen los números aleatorios, las partículas no se salgan de la imagen. Esta etapa pertenece a la etapa de *inicialización* del Filtro de Partículas.

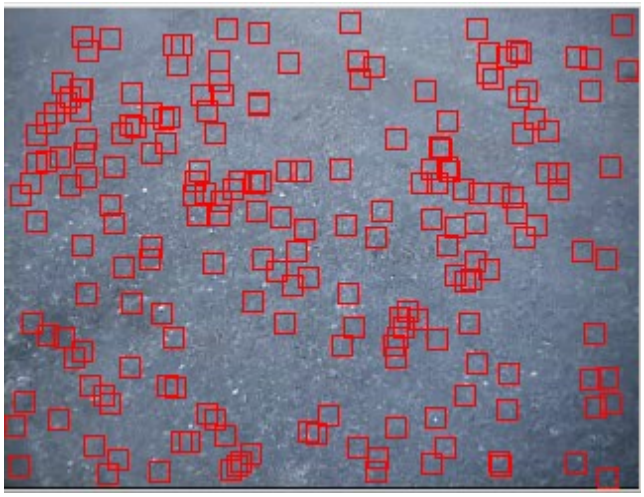


Figura 2.7.Fase de Inicialización

- *Ponderación:* cada partícula generada tiene su peso correspondiente. Para calcular el peso de las partículas vamos a utilizar los niveles de intensidad de la imagen. Aquellos píxeles cuya intensidad tenga un valor igual a la unidad, los vamos a considerar que forman parte del objeto de interés y por lo tanto dicho píxel contribuirá al peso de la partícula en la que se encuentre localizado. Esta etapa pertenece a la fase de *actualización* del Filtro de Partículas.

  En la siguiente imagen, podemos ver un ejemplo de aplicación de esta fase. En ella, las partículas rojas no tienen peso, por el contrario las azules si lo tienen. En la fase siguiente, se estimará cuál es la partícula de mayor peso.

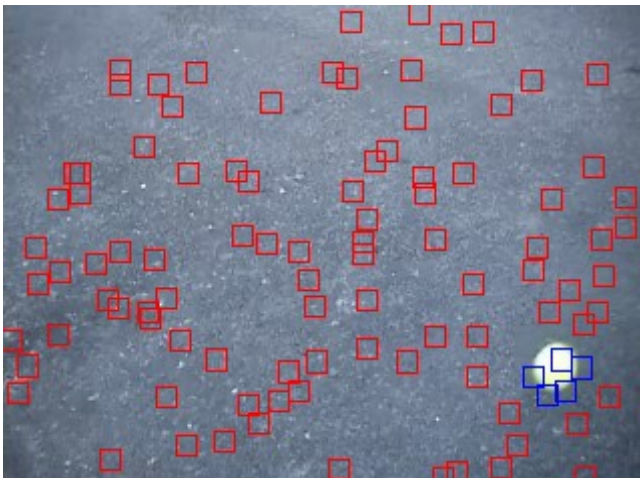


Figura 2.8.Fase de Ponderación

- *Estimación:* en esta fase seleccionamos la partícula de mayor peso (partícula verde en la figura 2.9), que es la que mayor cantidad de niveles de intensidad con valor unidad tiene, y por lo tanto, es la partícula que está más cerca del objeto de interés. Esta fase pertenece a la etapa de *estimación* del filtro.

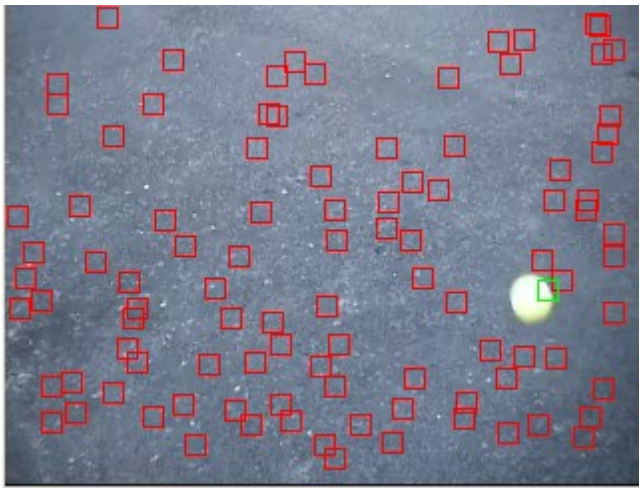


Figura 2.9.Fase de Estimación

- *Selección*: en esta parte del algoritmo, seleccionaremos varias partículas para crear un nuevo conjunto de partículas. Esta fase pertenece a la etapa de *predicción* del Filtro de Partículas. Para llevar a cabo esta fase, vamos a realizas lar

siguientes subetapas:

- *Crear ruleta*: se crea una "ruleta" con la acumulación de pesos calculados en la fase anterior. Para su implementación, generaremos un vector de tantas posiciones como partículas y en cada posición iremos almacenando el peso de la partícula correspondiente y los pesos de todas la partículas anteriores. Estos pesos deberán de estar normalizados, de tal forma que el valor de la último posición deberá de tomar el valor 1. Esto se explica mejor con el siguiente dibujo ilustrativo:

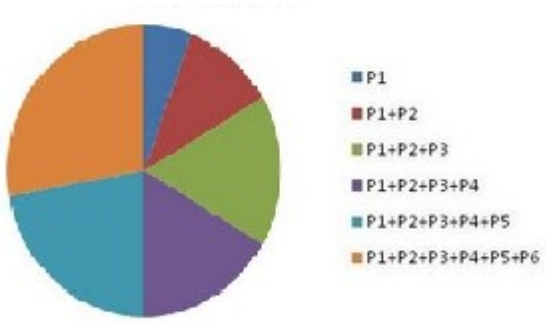


Figura 2.10. Ruleta creada en la fase de selección

- *Tirar ruleta*: "tiramos" la ruleta tantas veces como partículas haya. La elección de las nuevas partículas se hace aleatoriamente, teniendo en cuenta que la partícula de mayor peso tiene más posibilidades de ser escogida.

Figura 2.11.Fase de Selección

- *Difusión*: el vector de la etapa anterior puede tener partículas repetidas que al ser dibujadas se solapan (Figura 2.11). Para evitar este fenómeno, aplicamos la fase de *difusión* que aplica un pequeño desplazamiento de forma aleatoria a cada una de las partículas de dicho vector. Además, tendremos que tener en cuenta que las partículas al aplicarle este desplazamiento no se salgan de la imagen. Esta fase forma parte de la etapa de *predicción* del filtro.

Figura 2.12. Fase de Difusión

2.4. Aplicación del Filtro de Partículas a dos objetos

En este apartado vamos a explicar en qué consiste el Filtro de Partículas implementado para seguir dos objetos.

El objetivo de la implementación de esta posibilidad es poder realizar el seguimiento de dos objetos que aparecen conjuntamente en una secuencia de imágenes 2D utilizando el Filtro de Partículas.

En nuestro caso, para poder realizar el seguimiento del segundo objeto, hemos aplicado un filtro exactamente igual que el aplicado para el primer objeto y cuyo funcionamiento ha sido explicado en el capítulo anterior.

A continuación, mostramos un ejemplo del seguimiento de dos pelotas, una de color rojo y otra de color azul. Para la pelota de color rojo se ha utilizado un tamaño de partícula 10, mientras que para la pelota de color azul hemos utilizado un tamaño de partícula 5. Esta diferencia en el tamaño de las partículas se ha hecho para diferenciar las partículas de cada filtro en la etapa de inicialización.

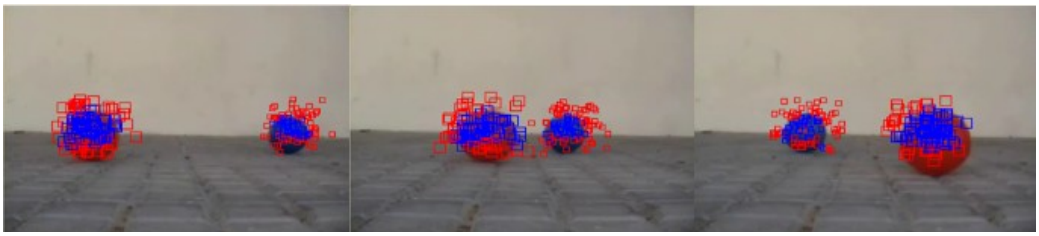


Figura 2.13. Fase de Inicialización de dos Filtros de Partículas

Como era de esperar proceso tarda más tiempo que los casos en los que únicamente aplicamos un filtro de partículas.

Capítulo 3
Implementación de la aplicación

Para la implementación de esta aplicación, hemos utilizado el lenguaje de programación C++ y el paradigma de la programación orientada a objetos(POO). Con esto hemos conseguido una estructura sencilla, fácil de mantener y usar, proporcionándole flexibilidad ante modificaciones y un alto nivel de reusabilidad.

La organización que hemos llevado para el desarrollo de esta aplicación queda reflejada en el siguiente diagrama de bloques:

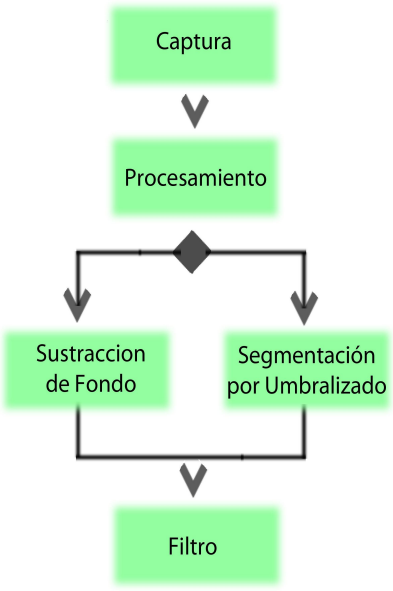


Figura 3.1.Diagrama de Bloques Elaboración de la Aplicación

3.1. Programa Principal

El programa principal de nuestra aplicación está formado por varias partes:

- Captura
- Aplicación de los dos métodos para umbralizar
- Aplicación del primer Filtro de Partículas
- Aplicación del segundo Filtro de Partículas

3.1.1. Captura

En el proceso de captura, hemos incluido la posibilidad de que la aplicación funcione tanto si hacemos captura de la secuencia de imágenes directamente desde la cámara, como si extraemos un vídeo con extensión .avi desde el ordenador.

El código empleado para ello es el siguiente:

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

Figura 3.2. Algoritmo para captura de videos o imágenes

3.1.2. Aplicación de los dos métodos para umbralizar

Como ya hemos dicho, en varias ocasiones, nuestra aplicación responde a dos métodos para extraer información del objeto de interés. Estos dos métodos, son la sustracción de fondo y

la segmentación por umbralizado. Además, como podemos realizar el seguimiento de hasta dos objetos, el método *Umbralizar* lo vamos a aplicar dos veces en el programa.

Para combinar ambos métodos de umbralización lo hemos hecho de la siguiente manera:

```
if(sustraccion)
        P1->SustraccionFondo(fondo,frame_copy,frameUm);
else
{
        P1->Umbralizar(frame_copy,frameUm,xSelec,ySelec,"Umbralizado");
        if(elegido2)
        P2->Umbralizar(frame_copy,frameUm2,xSelec,ySelec,"Umbralizado2");
}
```

Figura 3.3. Algoritmo para aplicación de la Umbralización

Podemos ver, que tenemos dos controles, uno que se llama sustracción, que controla si queremos hacer o no umbralización por sustracción de fondo y otro que se llama *elegido2,* que controla que hemos seleccionado el segundo objeto.

3.1.3. Aplicación del primer Filtro de Partículas

Para aplicar el primer Filtro de Partículas, el algoritmo que vamos a seguir cumple el diagrama de bloques de la figura 2.6:

```
FP1->Actualizacion(frame_copy,frameUm,frame_paint);
init=FP1->Estimacion();
if(!init)
{
        FP1->Inicializar(10);
        FP1->Ponderacion();
        init=FP1->Estimacion();
}
if(init)
{
FP1->Seleccion();
FP1->Difusion(20);
}
```

Figura 3.4. Algoritmo aplicar un Filtro de Partículas

3.1.4. Aplicación del segundo Filtro de Partículas

La aplicación de un segundo Filtro de Partículas, tiene el objetivo de realizar el seguimiento a un segundo objeto.

El algoritmo aplicado para ello, es exactamente el mismo que el de la figura 3.4, solo que para un FP2 en lugar de FP1.

3.2. Clase Procesamiento

La clase *Procesamiento*, como su propio nombre indica, realiza todas las operaciones necesarias para definir las características de la imagen que se utilizan para calcular el peso de las partículas.

Lo que se logra con la clase *Procesamiento*, es transformar la imagen de entrada en una imagen binaria sencilla de analizar.

Los atributos de esta clase definen el píxel sobre el que

pinchemos para la segmentación por umbralizado y son:

- píxel (pixel)
- Control para decirle al programa que solo pinchamos una vez para obtener el píxel (primero)

Esta clase implementa los dos tipos de umbralización disponibles en la aplicación, que como ya explicado en el capítulo2 son segmentación por umbralizado y sustracción de fondo.

3.2.1. Método cvFindColorRGB

La cabecera *cvFindColorRGB* está definida por el siguiente código:

```
void cvFindColorRGB(const IplImage* image, IplImage* imageUm, int B, int G, int R, int BSensibility, int GSensibility, int Rsensibility);
```

Este método sirve para umbralizar una imagen, según el color del píxel seleccionado que se extrae en el método Umbralizar, que explicaremos posteriormente, con una cierta sensibilidad.

El método recorre todos los píxeles de la imagen, viendo el valor de cada una de sus componentes.

Si alguno de los valores de las componentes del píxel seleccionado manualmente son mayores o menores que la diferencia del valor de de la componente y la sensibilidad correspondiente a un un píxel de la imagen, dicho píxel será umbralizado como objeto y tomará valor de intensidad 1, de lo contrario será catalogado como fondo y tomará el valor de intensidad 0.

El algoritmo descrito presenta el siguiente código:

```
{
for (x=0;x<image->height;x++)
      for (y=0;y<image->width;y++)
      {
          pixel=cvGet2D(image,x,y);
          if((pixel.val[0]>(B-BSensibility) && pixel.val[0]<(B+BSensibility))
            &&(pixel.val[1]>(G-GSensibility) && pixel.val[1]<(G+GSensibility))
            &&(pixel.val[2]>(R-RSensibility) && pixel.val[2]<(R+RSensibility)))
            {
                pixel.val[0]=objeto;
                pixel.val[1]=objeto;
                pixel.val[2]=objeto;
                cvSet2D(imageUm,x,y,pixel);
            }
            else
            {
                 pixel.val[0]=fondo;
                 pixel.val[1]=fondo;
                 pixel.val[2]=fondo;
                 cvSet2D(imageUm, x, y, pixel);
            }
      }

  }
```

Figura 3.5.Método cvFindColorRGB

3.2.2. Método Umbralizar

El método *Umbralizar* sirve para obtener el valor del píxel en el que hemos pinchado manualmente, para después aplicar el método *cvFindColorRGB*. Además, para mejorar la umbralización, aplicamos un suavizado a la imagen aplicando un filtrado de mediana.

El algoritmo empleado es el siguiente:

```
{
cvSmooth(image,imageFiltrada,CV_MEDIAN,3,0);
if(primero)
    {
    pixel=cvGet2D(imageFiltrada,y,x);
    primero=false;
    }
 CvFindColorRGB(imageFiltrada,imageUm, pixel.val[0],pixel.val[1],pixel.val[2],5,3,2);
 cvShowImage(ventana,imageUm);
}
```

Figura 3.6.Método Umbralizar

3.2.2. Método Sustracción de Fondo

La cabecera de este método es la siguiente:

void SustraccionFondo(IplImage* fondo,IplImage* image,IplImage* imageUm);

Este método aplica la técnica de la sustracción de fondo para definir el objeto de interés. Para ello, sigue una serie de pasos explicados de forma descriptiva en el capítulo 2 y que son los siguientes:

- Convierte a escala de grises al fondo y a la imagen con la que se esté trabajando
- Se hace el modulo de la resta del fondo y la imagen
- Se umbraliza la imagen diferencia

Además, para mejora la calidad de la umbralización, al igual que en el método *Umbralizar*, hemos aplicado un suavizado aplicando un filtro de mediana, antes de convertir las imágenes.

A continuación, se muestra el código empleado para la implementación de este método:

```
{
cvSmooth(image,imageFiltrada,CV_MEDIAN,3,0);
cvSmooth(fondo,fondoFiltrado,CV_MEDIAN,3,0);
cvCvtColor(imageFiltrada,imageBN,CV_BGR2GRAY);
cvCvtColor(fondoFiltrado,fondoBN,CV_BGR2GRAY);
cvAbsDiff(fondoBN,imageBN,imageDiferencia);
cvThreshold(imageDiferencia,imageUm,50,255,CV_THRESH_BINARY);
cvShowImage("Umbralizado",imageUm);
}
```

Figura 3.7. Método Sustracción de fondo

3.3. Clase Partícula

La clase *Particula* define cada una de las operaciones que se pueden realizar sobre una partícula.

Sus atributos definen una partícula en concreto:

- Coordenada x (x)
- Coordenada y (y)
- Peso (peso)

La clase *Particula* implementa tres métodos, para la gestión de sus atributos en el desarrollo del filtro de partículas que son:

- Método DibujarParticula
- Método CalcularPeso
- Método MoverPartículas

3.3.1. Método DibujarParticula

La cabecera de éste método viene dada a continuación:

```
void Particula::DibujarParticula(bool seleccionada,IplImage* imagePintar)
```

Este método sirve para dibujar las partículas, una vez que tenemos sus coordenadas x,y generadas en una de las fases del Filtro de Partículas (inicialización).

Las partículas que se ubiquen dentro del objetos serán azules, excepto la seleccionada (mayor peso) que será verde y las que se encuentren fuera del objeto se dibujarán en rojo.

Para la realización de las pruebas, hemos suprimido la partícula de mayor peso, para ver el seguimiento del objeto de forma más correcta.

El algoritmo empleado para la implementación de este método es el siguiente:

```
{
CvPoint punto1=cvPoint(x,y);
CvPoint punto2=cvPoint(x+ladoParticula,y+ladoParticula);

if (seleccionada)
        cvRectangle(imagePintar,punto1,punto2,CV_RGB(0,255,0),1,8,0);
else
        if(Peso>0)
                cvRectangle(imagePintar,punto1,punto2,CV_RGB(0,0,255),1,8,0);
        else
                cvRectangle(imagePintar,punto1,punto2,CV_RGB(255,0,0),1,8,0);
}
```

Figura 3.8. Método DibujarParticula

3.3.2. Método CalcularPeso

La cabecera de esta función viene dada por el siguiente código:

```
void Particula::CalcularPeso()
```

Este método se utiliza para calcular el peso de cada partícula. Para ello, se recorre cada una de las partículas que tenemos en la imagen, y a partir de los niveles de intensidad de ésta, se va

calculando el peso de la partícula en cuestión. Si el nivel de intensidad es 1 significa una contribución a la variable peso, si por el contrario es 0 no se le suma nada a dicha variable.

El algoritmo utilizado para esta implementación es el siguiente:

```
Peso=0;
for(i=x;i<x+ladoParticula;i++)
{
        for(j=y;j<y+ladoParticula;j++)
        {
        pixel=cvGet2D(imageUm,j,i);
        if(pixel.val[0]>0)
                Peso++;
        }
}
```

Figura 3.9. Método CalcularPeso

3.3.3. Método MoverParticulas

La cabecera de esta función, viene dada a continuación:

```
void Particula::MoverParticulas(float Aumentarx,float Aumentary)
```

El objetivo de este método es aplicar un pequeño desplazamiento a cada una de las partículas. El propósito de este desplazamiento es que las partículas no se solapen, tras la *fase de selección* del Filtro de Partículas

El algoritmo que aplica este desplazamiento a las partículas es el siguiente:

```
{
x=x+Aumentarx;
y=y+Aumentary;

if(x+ladoParticula>imageUm->width)
        x=imageUm->width-ladoParticula-1;
if(y+ladoParticula>imageUm->height)
        y=imageUm->height-ladoParticula-1;
if(x<0)
        x=0;
if(y<0)
        y=0;
}
```

Figura 3.10. Método MoverParticulas

Además, podemos ver en la imagen anterior, que controlamos que al mover las partículas, éstas no se salgan de los límites de la imagen.

3.4. Clase FiltroParticula

En esta clase se han implementado cada uno de las fases del filtro, es decir, inicialización, ponderación, estimación, selección y difusión, en diferentes métodos cada una.

Los atributos de esta clase definen el número de partículas que utilizamos para el filtro, además de los correspondiente vectores de partículas necesarios para el desarrollo de todas las fases del mismo. A continuación, se reflejan estos atributos:

- Numero de Partículas (NumeroParticulas)
- vector Partículas inicial (particula[])
- vector Partículas para realizar el proceso de selección (particula_aux[])

A continuación, procederemos a la explicación de cada uno

de los métodos.

3.4.1. Método Inicialización

La cabecera de este método es la siguiente:

```
void Inicializar(int ladoParticula);
```

Este método, realiza la fase de inicialización del filtro, que como ya explicamos anteriormente, consiste en la generación aleatoria de partículas, para ello, generamos aleatoriamente con la función rand() valores de coordenadas x y valores de coordenadas y. A continuación, iremos almacenado, estos valores generados en un vector, llamado particulas[i].

El algoritmo empleado para la implementación de este método es el siguiente:

```
for(i=0;i<NumeroParticulas;i++)
      {
         x=rand()%(image->width-ladoParticula);
         y=rand()%(image->height-ladoParticula);
         particulas[i]=new Particula(imageUm,ladoParticula,x,y);
      }
iniciado=true;
```

Figura 3.11. Método Inicialización

3.4.2. Método Ponderación

La cabecera de este método es la siguiente**:**

```
void Ponderacion();
```

Este método se ha creado con el objetivo de que calcule el peso de cada una de las partículas del filtro. Para ello, para cada

una de las partículas llamaremos al método *CalcularPeso* de la clase Particula, explicado en la figura 3.9.

Por lo tanto, el algoritmo empleado en este método sería el siguiente:

```
if(iniciado)
{
for(i=0;i<NumeroParticulas;i++)
    particulas[i]->CalcularPeso();
}
```

Figura 3.12. Método Ponderación

Como podemos ver tenemos un control inicializado de tal forma que si no hemos hecho previamente una inicialización, no pueda calcular el peso de las partículas.

3.4.3. Método Estimación

La cabecera de este método viene dada a continuación:

```
bool Estimacion();
```

El objetivo de este método es ver cuál es la partícula de mayor peso. Para ello, en principio presuponemos que la partícula de mayor peso es la partícula almacenada en particula[0]. A continuación, vamos recorriendo dicho vector, si el peso de particula[i] es menor que el particula[0], seguiremos hasta encontrar una particula[i] cuyo peso sea mayor que particula[0]. En este momento, la partícula de mayor peso será particula[i]. A continuación, se seguirá recorriendo el vector hasta terminarlo, por si hubiese otra particula[i] con mayor peso que la anterior encontrada.

El código empleado para la implementación de este método

es el siguiente:

```
for(i=0;i<NumeroParticulas;i++)
{
        if(particulas[i]->getPeso()>seleccionada->getPeso())
        seleccionada=particulas[i];
        particulas[i]->DibujarParticula(false,imagePintar);
}
```

Figura 3.13. Método de Estimación

3.4.4. Método Selección

La cabecera de este método es la siguiente:

```
void Seleccion();
```

Este método sirve para crear un nuevo conjunto de partículas. Esta compuesto de dos partes muy importantes, con respecto al funcionamiento del Filtro de Partículas.

- Crear Ruleta: para crear la ruleta, tenemos que crear un nuevo vector de partículas de tantas posiciones como partículas tenemos inicialmente. En cada posición del vector iremos sumando el peso de la partícula a la que corresponde con todas las anteriores. Este valor lo normalizaremos, de tal forma que el valor de la ultima posición del vector sea 1.

El algoritmo empleado para la creación del a ruleta es el siguiente:

```
for (i=0;i<NumeroParticulas;i++)
        {
                Peso=Peso+(particulas[i]->getPeso());
                vectorPesoParticulas[i]=Peso;
        }

for (i=0;i<NumeroParticulas;i++)
        vectorPesoParticulas[i]=vectorPesoParticulas[i]/Peso;
```

Figura 3.14. Creación Ruleta Método Selección

- Tirar Ruleta: a continuación, hacemos una copia del vector particula[i] inicial y lo almacenamos en particulas_aux[i]. Una vez que hemos creado el nuevo vector con las acumulaciones de pesos, vamos a generar un numero aleatorio entre 0 y 1 tantas veces como partículas hay. Con este número, veremos a que partícula del vector particulas_aux[i] corresponde, e iremos rellenando el nuevo vector particula[i] con las partículas elegidas. Existe mayor posibilidad de elegir la partícula de mayor peso.

  El algoritmo empleado para tirar la ruleta es el siguiente:

```
for(i=0;i<NumeroParticulas;i++)
{
      numero=(rand()%100)+1;
      numero=numero/100;
      indice=0;
      while(numero>vectorPesoParticulas[indice] && indice<NumeroParticulas)
          indice++;
      if(numero==1)
          numero=NumeroParticulas;
      particulas[i]= new Particula(particulas_aux[indice]);
}
```

Figura 3.15. Tirar Ruleta Método Selección

3.4.5. Método de Difusión

Con el método anterior obtenemos un nuevo conjunto de particula[i], alguna de las cuales puede ser la misma, por lo que al

dibujarla, quedarían solapadas. Para evitar este problema, se aplica el método de *difusión.* Con este método conseguimos, aplicar un pequeño desplazamiento aleatorio a cada una de las partículas.

La cabecera de la función es la siguiente:

```
void Difusion(int Mover);
```

El algoritmo empleado para la implementación de este método es el siguiente:

```
for(i=0;i<NumeroParticulas;i++)
{
        Moverx=rand()%(Mover*2);
        Moverx=Moverx-Mover;
        Movery=rand()%(Mover*2);
        Movery=Movery-Mover;
        particulas[i]->MoverParticulas(Moverx,Movery);
}
```

Figura 3.16.Método de difusión

Capítulo 4
Experimentos y resultados

El objetivo inicial de este proyecto, como ya hemos explicado en más de una ocasión, es el dar una estimación lo más ajustada posible de la posición de un objeto que se mueve en una secuencia de imágenes, aplicando el método del Filtro de Partículas.

En este capítulo se presentarán los resultados obtenidos, al aplicar el programa desarrollado a una serie de pruebas, modificando los parámetros del filtro y de la umbralización.

Para la realización de las pruebas hemos utilizado una cámara digital SONY modelo DCR-HC18E. Para pasar los vídeos a formato AVI se ha utilizado ImTOO 3GP Video Converter.

Estas pruebas han sido realizadas en lugares exteriores, con cambios de iluminación e interiores. Para comprobar la eficiencia del filtro, nos hemos asegurado de que el objeto o los objetos cumplan una serie de requisitos como por ejemplo que no sigan un movimiento plano paralelo, que se produzca su oclusión en algún momento de la secuencia, que aparezca más de un objeto a vez...

En todas las pruebas realizadas, las partículas azules son partículas con peso, mientras que las partículas rojas no tienen peso.

A continuación, procedemos a la exposición de los resultados obtenidos tras la aplicación del programa desarrollado a las pruebas realizadas.

4.1. Pruebas realizadas con la técnica de Sustracción de Fondo

La *sustracción de fondo,* como ya hemos explicado a lo largo de toda la memoria, es una técnica que sirve para detectar objetos móviles sobre fondos estáticos conocidos. Por ello, en la realización de las pruebas para la aplicación de esta técnica, hemos tenido un cuidado especial, para que el fondo de la secuencia de imágenes fuera estático, pues de lo contrario los resultados serían erróneos.

Hemos realizado tres tipos de pruebas para la sustracción de fondo.

- Movimiento no uniforme paralelo de una pelota tenis
- Movimiento uniforme paralelo de un coche
- Movimiento de un tapón en un lugar cerrado

4.4.1. Movimiento no uniforme paralelo de una pelota de tenis

Para esta prueba hemos realizado dos experimentos con el objetivo de ver claramente los problemas que existen a la hora de elegir el umbral, a partir del cual se aplica la binarización de la imagen diferencia (diferencia entre la imagen considerada fondo y cada una de las imágenes de la secuencia).

Para escoger los umbrales, nos hemos basado en la siguiente figura que nos muestra la sensibilidad espectral:

Figura 4.1. Sensibilidad espectral

EXPERIMENTO 1

Para la realización de este experimento hemos optado, por los siguientes parámetros:

- Numero de Partículas: 100
- Tamaño de Partículas:10
- Difusión de Partículas: 20
- Umbral:50

Hemos elegido este umbral, porque el color de la pelota es amarillo y este color esta situado aproximadamente en la mitad del espectro de la sensibilidad de colores (ver Figura 4.1).

El resultado obtenido de la aplicación de la sustracción de fondo sobre una secuencia de imágenes, que definen el movimiento no plano paralelo de una pelota de tenis, sobre un fondo estático conocido y aplicándole los parámetros anteriores son los siguientes:

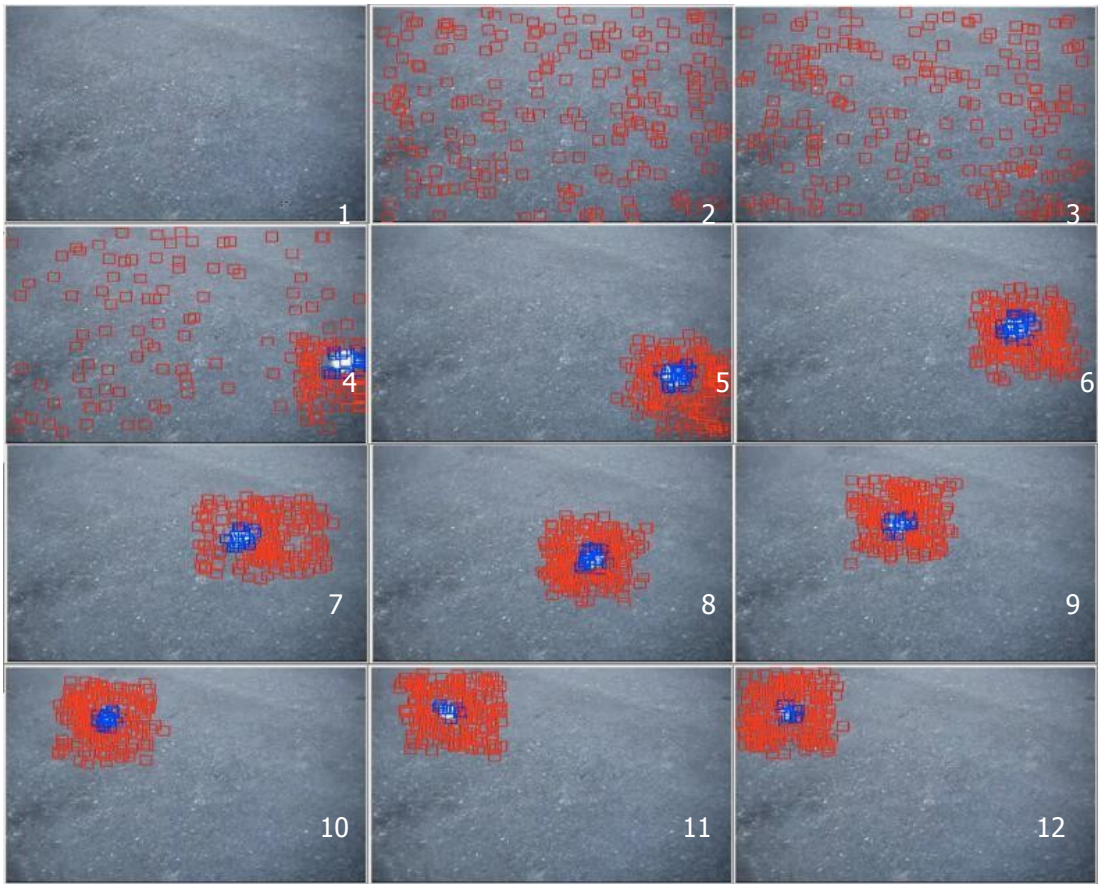


Figura 4.2. Resultado del experimento1 en Pelota de Tenis

Podemos ver en la figura 4.2, que mientras el fondo está invariante, el filtro lanza partículas aleatorias(captura 2 y 3). En cuanto se produce una pequeña modificación con respecto al fondo, las partículas empiezan a concentrarse alrededor del objeto por medio del remuestreo (captura 4).

Una vez que aparece el objeto de interés, las partículas siguen el movimiento de la partícula de mayor peso, para así cubrir todo la pelota una vez que aparece completamente en la imagen (capturas de la 5 a la 12).

También hay que decir, que la elección de partículas en este caso no ha sido muy acertada, pues muchas de ellas quedan ubicadas fuera del objeto.

A continuación, vamos a ver qué problemas surgen al cambiar el valor umbral en esta secuencia de imágenes.

EXPERIMENTO 2

En este experimento, como hemos dicho anteriormente, vamos a aplicar la misma secuencia de imágenes que en el caso que anterior y la misma técnica de umbralización, pero vamos a disminuir el valor umbral, para que la sustracción de fondo sea menos restrictiva, y considere mayor cantidad de píxeles con niveles de intensidad 1.

Por lo tanto, los parámetros que tenemos ahora son los siguientes:

- Numero de Partículas: 100
- Tamaño de Partículas:10
- Difusión de Partículas: 20
- Umbral:20

En este caso, como podemos ver hemos elegido un umbral inferior al del experimento1. Al disminuir el umbral, la binarización va a ser más sensible a colores más oscuros como podemos ver en la figura del espectro de sensibilidad (figura 4.1.). Debido a ello, aparecen errores en el seguimiento de la imagen, ya que no sólo se binariza la pelota, sino que aparecen más píxeles con nivel de intensidad igual a la unidad.

En la siguiente figura vemos los resultados obtenidos, cuando aplicamos la técnica de sustracción de fondo a una secuencia que describe el movimiento no uniforme paralelo de una pelota de tenis en el caso de que el umbral sea muy pequeño:

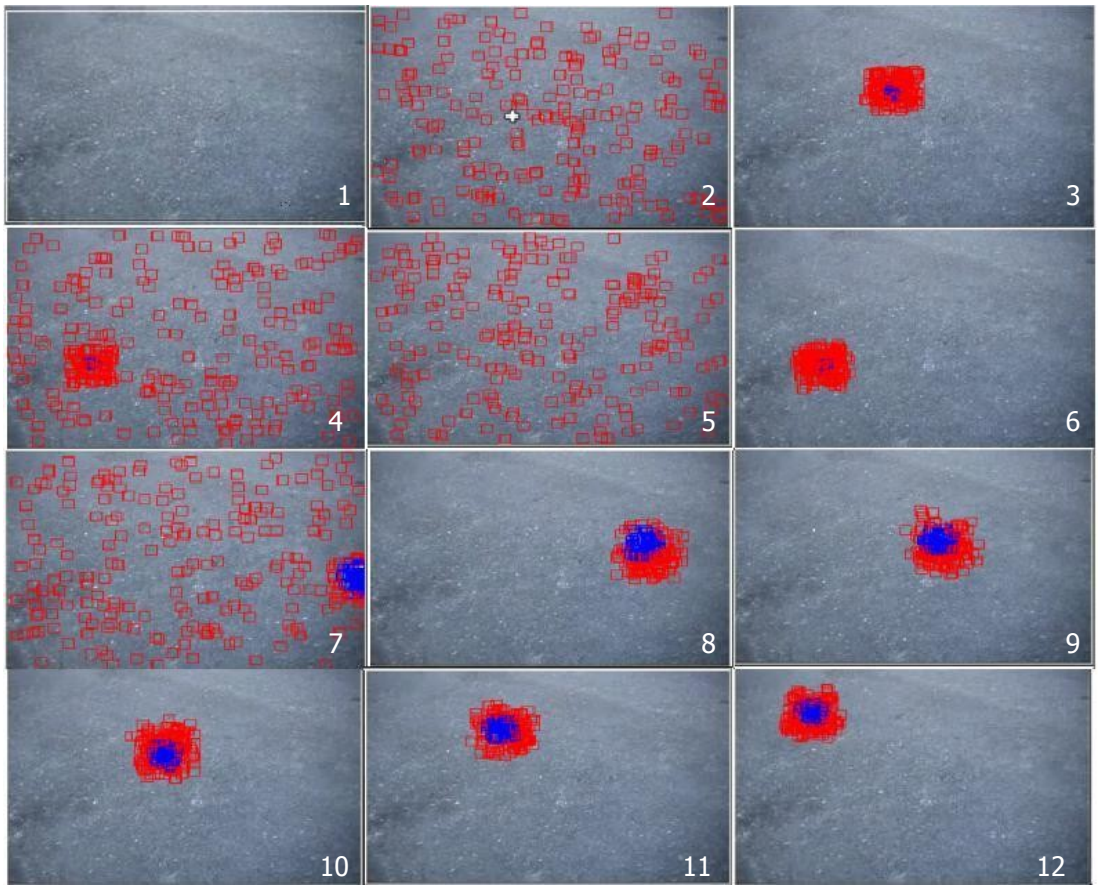


Figura 4.3. Resultado del experimento2 en Pelota de Tenis

Como hemos explicado anteriormente, al modificar el umbral, vemos que los resultados en algunos casos son erróneos (capturas de la 3 a la 6). En estos casos, se detecta objeto de interés, cuando realmente no lo hay.

No obstante, como las partículas se van a concentrar alrededor de la de mayor peso, y ésta una vez que entre el objeto de interés en la secuencia de imágenes (captura7) va a caer en dicho objeto, se realizará un seguimiento correcto de la pelota.

El hecho de que la partícula de mayor peso esté en la pelota cuando ésta entra en la secuencia de imágenes, es debido a que el Filtro de Partículas trabaja por probabilidades. Lo que quiero decir con esto es que aunque en la imagen umbralizada aparezcan más píxeles con nivel de intensidad 1, la partícula de mayor peso tiene más probabilidades de caer en la zona donde mayor acumulación de partículas con nivel de intensidad 1 hay (ver figura 4.4.). Por

ello, en este caso, el seguimiento de la pelota, una vez que aparece en la secuencia de imágenes se realiza correctamente.

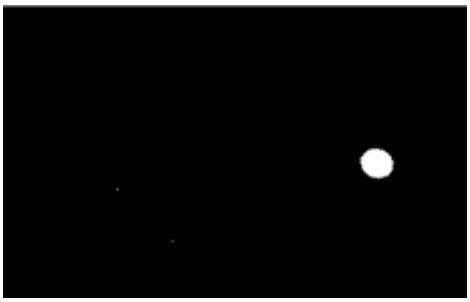


Figura 4.4. Umbralización del experimento2 en Pelota de Tenis

En el caso, de que el valor umbral cambiase de nuevo a un valor todavía más bajo que el anterior, la binarización sería todavía más sensible a colores oscuros, y aparecerían grandes acumulaciones de píxeles con intensidad igual a 1 por lo que no se podría realizar un seguimiento correcto del objeto.

4.4.2. Movimiento uniforme paralelo de un coche

A continuación, vamos a proceder a ver los resultados obtenidos sobre la segunda prueba. Ésta muestra el movimiento uniforme paralelo de un coche sobre un fondo estático no uniforme conocido.

Para esta prueba, hemos realizado dos nuevos experimentos, para ver como influye la velocidad del objeto de interés sobre nuestra aplicación.

EXPERIMENTO 1

En este experimento, vamos a ver los resultados obtenidos al aplicar la técnica de sustracción de fondo a una secuencia de

imágenes que describen un movimiento uniforme paralelo de un coche, para una velocidad media de 5km/h.

Los parámetros empleados en este experimento son los siguientes:

- Numero de Partículas: 100
- Tamaño de Partículas: 10
- Difusión de Partículas: 10
- Umbral: 50

La elección del umbral, la hemos realizado al igual que en los experimentos anteriores, guiándonos por el espectro de sensibilidad de los colores. Como queremos que nuestra aplicación sea sensible a colores azules oscuros le ponemos un umbral de 50.

Además, en el momento de la elección del umbral, probamos un umbral de 35 y el seguimiento del coche también era bueno, pero escogimos la primera opción para poder comparar los resultados con el primer experimento de la prueba realizada con una pelota de tenis. También probamos la posibilidad de que dicho umbral tomase el valor de 30, pero nos daba errores en la binarización, además del coche nos binarizaba las barras oscuras que existen al fondo de la imagen.

Los resultados obtenidos son los siguientes:

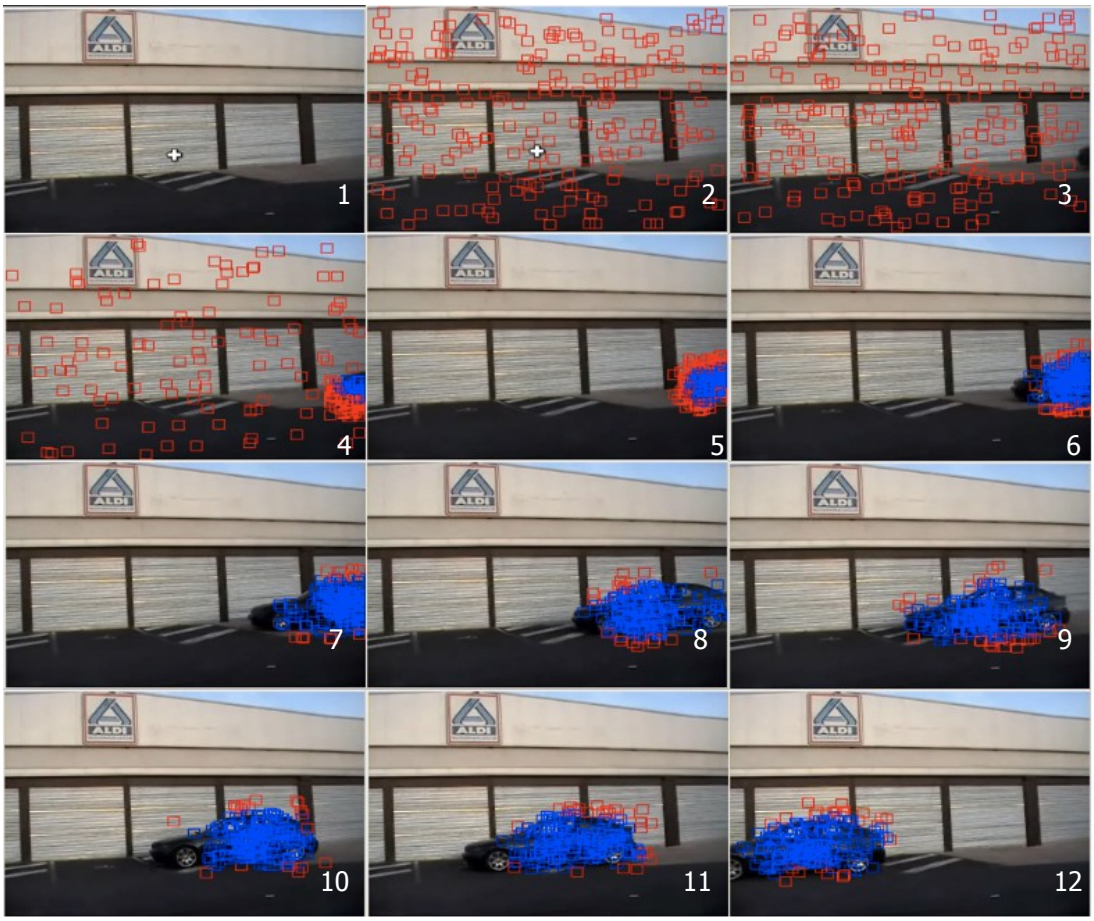


Figura 4.5. Resultados del experimento1 en Coche

En la figura 4.5. podemos ver, que los resultados obtenidos para el seguimiento de un coche que describe una movimiento uniforme son correctos.

Si comparamos este experimento con el realizado para el seguimiento de la pelota de tenis en la prueba anterior, podemos ver que los resultados son muy similares, teniendo en cuenta la relación tamaño-número de partículas con peso.

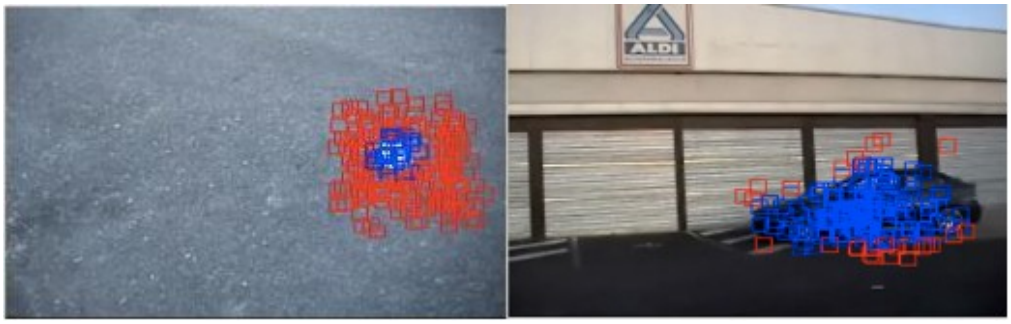


Figura 4.6. Comparación de los experimentos1 de las 2 pruebas exteriores realizadas para sustracción de fondo

Podemos decir que en este caso, existen mayor número de partículas dentro del objeto, por lo que estamos haciendo una buena estimación del objeto de interés. En el caso de la pelota de tenis, teníamos menor número de partículas con peso, debido a el tamaño del objeto era menor. No obstante, la relación tamaño del objeto-número de partículas con peso es buena, ya que en ambos casos se esta haciendo una correcta estimación del objeto de interés.

EXPERIMENTO 2

En este experimento, vamos a usar los mismos parámetros que en el anterior, sólo que ahora la velocidad media del coche va a aumentar a 10km/h, además de que éste va a estar más cerca de la cámara. Los parámetros empleados son:

- Número de Partículas: 100
- Tamaño de Partículas:10
- Difusión de Partículas:20
- Umbral: 50

Los resultados son los siguientes:

Figura 4.7. Resultados del experimento2 en coche

En la figura 4.7. podemos ver el movimiento uniforme de un coche a una velocidad mayor que en la figura 4.5. Al aumentar la velocidad del objeto, las partículas se separan de él, y siguen su movimiento, es decir, muchas de las partículas van detrás del coche, siguiendo su trayectoria.

Además, podemos ver que las partículas no se reparten por todo el objeto, como en el experimento anterior. Esto es debido, a que en este caso, el coche está más cerca de la cámara y la partículas se concentran sobre una zona específica, que no es otro sitio que donde se encuentra la partícula de mayor peso. Esto queda mejor reflejado en al siguiente figura:

Figura 4.8. Resultados ante cambios en la proximidad de la cámara

Por último destacar, la respuesta de la aplicación, cuando se produce la oclusión del objeto de interés en la secuencia (captura 12). Como ya hemos explicado anteriormente, cuando se produce la pérdida del objeto, las partículas vuelve a la fase de inicialización, hasta que se encuentra de nuevo el objeto y se vuelve a realizar su seguimiento.

4.1.3. Movimiento de un tapón. Prueba en interiores

Esta prueba a diferencia que el resto de las desarrolladas para *sustracción de fondo*, se ha realizado en un recinto interior. El objetivo de este experimento es ver la diferencia que existe en los resultados, cuando nuestro programa se aplica en lugares interiores o por el contrario en lugares exteriores.

Se trata de seguir el movimiento de un tapón que se desplaza por una mesa situada en una habitación, cuando la única iluminación existente es la que entra por una ventana. Hemos intentado, que en el momento de la realización de la prueba no existiera un cambio brusco de iluminación, ya que en tal caso, tendríamos que volver a repetir el experimento para cambiar el umbral.

EXPERIMENTO 1

Como hemos dicho anteriormente, el objetivo de este experimento es ver la respuesta de nuestra aplicación, cuando la secuencia de imágenes es tomada en un lugar interior.

En este caso, vamos a elegir un número de partículas menor para evitar la acumulación innecesaria de partículas, como sucedía en el caso de la pelota de tenis.

Por lo tanto, los parámetros utilizados para la realización de este experimento son los siguientes:

- Número de partículas:50
- Tamaño de partículas:10
- Difusión de partículas:20
- Umbral: 30

La elección del valor umbral la hemos estipulado de la misma manera que en el resto de experimentos, mirando el color del objeto y viendo que umbral tiene en la figura 4.1.

Los resultados obtenidos son los siguientes:

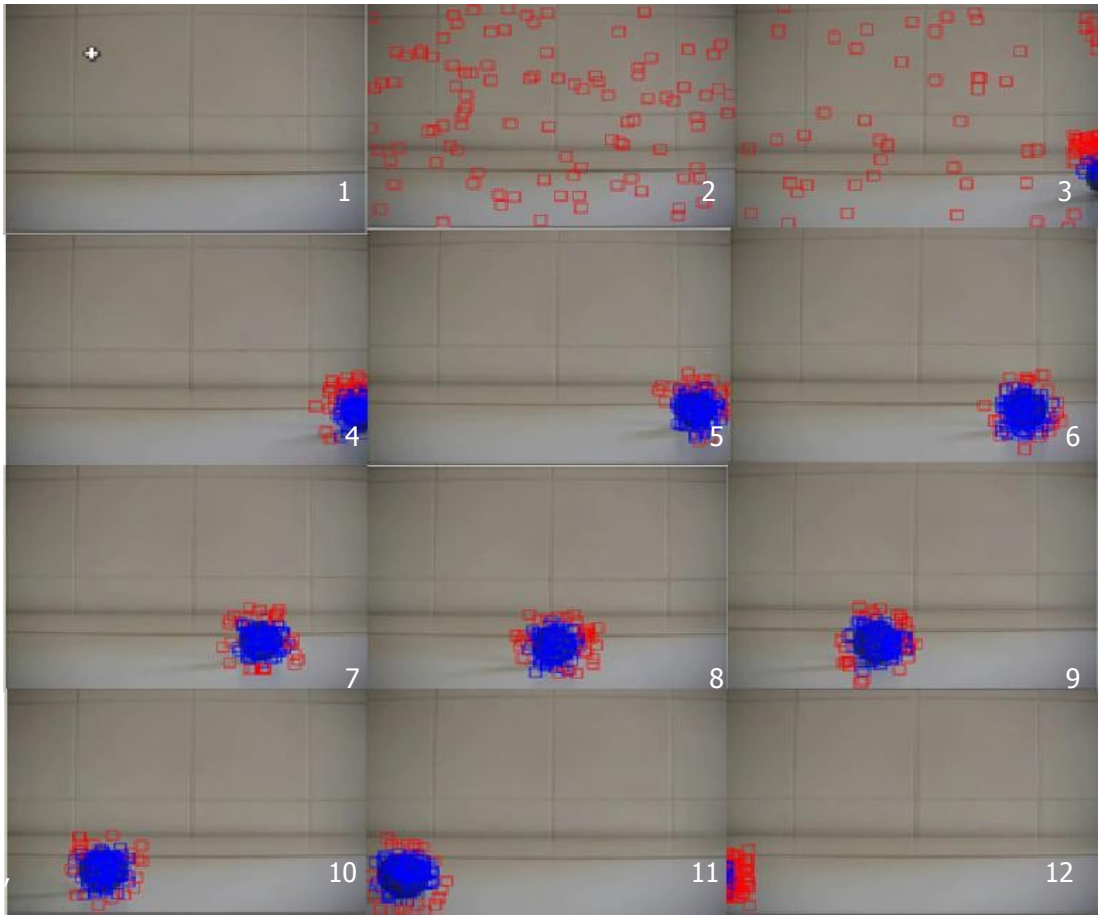


Figura 4.9. Resultados experimento1 Tapón

Como podemos ver en la figura 4.9, el seguimiento del tapón aunque éste no esté en sitio exterior como en el resto de casos es correcto.

La diferencia entre ambos casos es mínima, como podemos ver en la figura 4.10, ya que la umbralización en ambos casos es muy buena. No obstante, si las condiciones de iluminación del caso de realizado en el interior de un habitación, hubiesen sido peores, la umbralización sería peor, y se notarían más los cambios.

A continuación, se muestran dos figuras, la primera compara los dos casos mostrando el seguimiento de los objetos y la segunda compara las imágenes binarizadas.

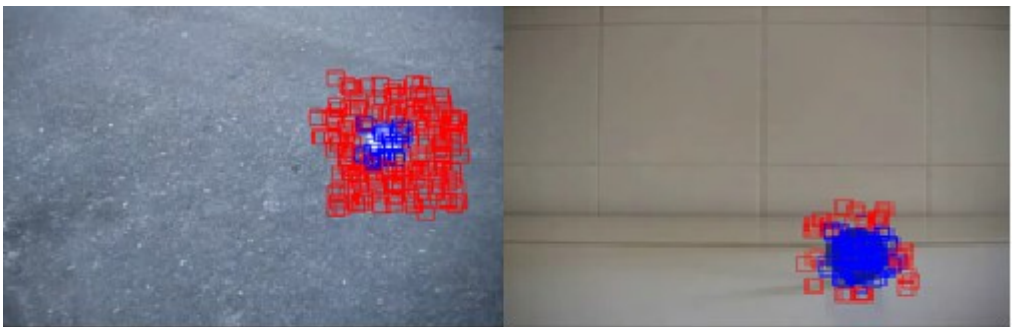


Figura 4.10. Comparación entre resultados obtenidos en lugares interiores y exteriores (seguimiento)

Figura 4.11. Comparación entre resultados obtenidos en lugares interiores y exteriores (binarización)

En el primer caso, como podemos ver en la figura 4.10 hemos empleado mayor número de partículas que en el segundo caso. Es más correcto éste último, ya que en el primer caso se está produciendo una aglomeración innecesaria de partículas.

4.2. Pruebas realizadas con la técnica de Segmentación por Umbralizado

Como ya hemos dicho en varias ocasiones, nuestra aplicación cuanta con dos posibles técnicas para realizar la umbralización de la imagen. En este caso, vamos a realizar pruebas con la técnica de *Segmentación por Umbralizado.*

Nuestro programa, cuando se aplica la técnica de

Segmentación por Umbralizado, puede llegar a realizar el seguimiento de hasta dos objetos.

Para esta técnica, hemos realizado tres tipos de pruebas:

- Movimiento uniforme de un ciclista vestido de rojo
- Movimiento no uniforme de dos pelotas
- Movimiento de un tapón en un lugar cerrado

En ambos casos, al igual que en los experimentos anteriores, las partículas rojas no tienen peso, mientras que las azules si tienen peso.

A continuación, procedemos a mostrar los resultados obtenidos al probar nuestra aplicación en las pruebas realizadas.

4.2.2. Movimiento uniforme de un ciclista vestido de rojo

Esta prueba ha sido realizada con el objetivo de comprobar en el método de segmentación por umbralizado, los problemas que existen con la sensibilidad de los colores. En esta técnica, no se establece un valor umbral, sino las sensibilidades para cada una de las componentes R,G,B.

EXPERIMENTO 1

En este experimento, vamos a comprobar el correcto funcionamiento de la aplicación, utilizando la técnica de segmentación por umbralizado.

Para la realización de este experimento se ha considerado oportuna la utilización de menor número de partículas, debido a

que el objeto a seguir ocupa una porción muy pequeña en la pantalla a causa de la lejanía del objetivo de la cámara (captura 2). La utilización de mayor número de partículas, provocaría una aglomeración innecesaria sobre el objeto, como nos sucedió en la primera prueba realizada para la sustracción de fondo.

A continuación, se detallan los parámetros utilizados:

- Número de partículas: 30
- Número de tamaño: 10
- Difusión de las partículas:20
- Sensibilidad (R,G,B): (80,20,20)

Podemos ver que hemos empleado una sensibilidad mayor para la componente roja que para el resto de componentes. Esto lo hemos hecho, para asegurarnos el mayor número de totalidades de rojos, ya que al realizar la prueba en exteriores, la iluminación puede cambiar y nos estropearía el experimento.

Los resultados obtenidos son los siguientes:

Figura 4.12. Resultados experimento1 Ciclista

Podemos ver en la figura 4.12. que el seguimiento del ciclista utilizando la técnica de la segmentación por umbralizado, también es correcta. Cabe destacar, al igual que en el experimento2 del coche, que debido a la velocidad, muchas de las partículas se quedan detrás del ciclista siguiendo su movimiento, es decir, su trayectoria (captura 12).

EXPERIMENTO 2

A continuación, hemos realizado otro experimento para la misma secuencia de imágenes que en el caso anterior, con el objetivo de mostrar los problemas que existen al elegir la sensibilidad de cada una de las componentes en la técnica de segmentación por umbralizado.

En el primer experimento teníamos mayor sensibilidad para la componente roja, en este caso, vamos a considerar las tres componentes con igual sensibilidad. Además su valor será menor que 80, teniendo por lo tanto menor número de totalidades de rojos.

Los parámetros elegidos son los siguientes:

- Número de Partículas: 30
- Tamaño de partículas: 10
- Difusión de partículas: 20
- Sensibilidad (R,G,B): (60,60,60)

Como podemos ver los valores de sensibilidad elegidos son (60,60,60). Al elegir estos valores, para la componente roja hemos disminuido el número de totalidades, por el contrario, para las componentes verdes y azules hemos ampliado el número de tonalidades, dejando pasar mayor número de colores que pueden ser tomados en la binarización como píxeles de nivel de intensidad 1.

A continuación, vemos la repercusión de esta elección en la misma secuencia de imágenes que en el experimento anterior:

Figura 4.13. Resultados experimento2 Ciclista

Podemos ver, que en este caso hemos utilizado menor número de capturas que en los experimentos anteriores. Esto es debido a que los resultado de este experimento son totalmente erróneos y muy similares unos a otros.

Lo que está sucediendo en este caso, como podemos ver en la figura 4.14, es que al aplicar sensibilidad (60,60,60), estamos binarizando más objetos de la imagen.

Figura 4.14. Umbralización del experimeto2 Ciclista

Como el Filtro de Partículas funciona por probabilidades, como ya hemos explicado en otra ocasión, las partículas se han centrado en la parte del suelo que es más clara, ya que en la

imagen binarizada (figura 4.14) esta zona es la de mayor acumulación de píxeles de nivel de intensidad 1. En el caso de que hubiéramos utilizado otra sensibilidad para la que el suelo no hubiese sido umbralizado, es posible que si se hubiese realizado el seguimiento del ciclista, puesto que es la siguiente región binarizada de mayor tamaño.

4.2.2 Movimiento no uniforme de dos pelotas

Esta prueba la hemos realizado con el objetivo de mostrar la posibilidad que tiene nuestra aplicación de hacer el seguimiento de dos objetos, en el caso de la segmentación por umbralizado.

Para la realización de esta prueba, se han utilizado los siguientes parámetros:

Primer Filtro de Partículas:

- Número de Partículas: 100
- Tamaño de Partículas:10
- Difusión de Partículas:20
- Sensibilidad: (40,10,40)

Segundo Filtro de Partículas:

- Número de Partículas:100
- Tamaño de Partículas:5
- Difusión de Partículas:20
- Sensibilidad: (40,10,40)

Como podemos ver, le hemos dado mayor sensibilidad a las

componentes roja y azul, para tener mayor número de tonalidades de estos colores, ya que las pelotas utilizadas para la realización de la prueba son de estos colores.

Además, hemos empleado el mismo número de partículas en ambos filtros, ya que el tamaño de las pelotas es el mismo. Sin embargo, las partículas del primer filtro son de tamaño mayor a las del segundo filtro, para poder diferenciarlas.

Los resultados obtenidos son los siguientes:

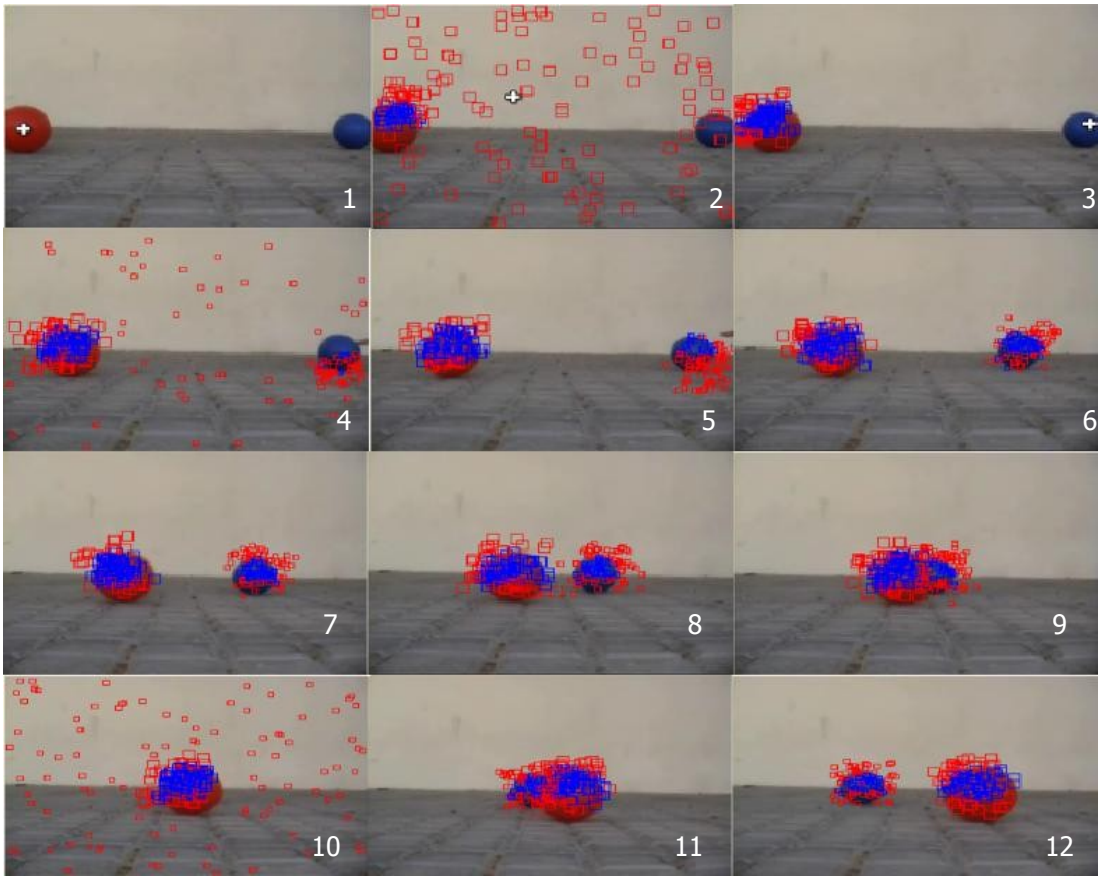


Figura 4.15. Resultados del experimento para dos objetos

Podemos ver que el seguimiento realizado sobre las dos pelotas es correcto.

El funcionamiento de cada uno de los filtros por separado es el mismo que en los experimentos anteriores.

En la figura 4.15 podemos ver en la captura 2 el momento en el que se produce la selección de la primera pelota (rojo), en

primer lugar se produce una inicialización de las partículas (captura2), para después distribuirse sobre la pelota(captura3) por la fase del remuestreo. En la captura 4 se produce la selección de la segunda pelota (azul). El proceso de selección del objeto de interés es el mismo que el explicado para la pelota de color rojo.

En las capturas 6, 7 y 8 de la figura 4.12 se muestra el seguimiento conjunto de ambas pelotas.

Cabe destacar la importancia de las captura 9, ya que es el momento en el que comienza la oclusión de la pelota azul, quedando sus partículas en la fase de inicialización, mientras que el resto de partículas continúan realizando el seguimiento de la pelota roja. En la captura 11 aparece de nuevo la pelota azul, y sus partículas vuelven a encontrar al objeto de interés.

4.3. Comparación de los resultados obtenidos con el Filtro de Kalman

A continuación, se exponen las principales diferencias entre el filtro de Kalman y el filtro de Partículas. Para la extracción de estas diferencias nos hemos apoyado en el proyecto fin carrera "Tranking automático de objetos en secuencia de imágenes usando Filtro de Kalman". En este proyecto, se han realizado las mismas pruebas que en el presente, utilizando los mismos valores umbrales y de sensibilidades, con el objetivo de ver las diferencias que implican la aplicación de estos dos estimadores en el seguimiento de imágenes.

1. Procesado de la imagen

De acuerdo con los resultados obtenidos, al Filtro de Kalman le afectan más los cambios de iluminación, ya que tras la

umbralización de la imagen (en ambos proyecto la umbralización se realiza de la misma manera), tiene que englobar al objeto en una región de interés para calcular el centro de masas, a partir de la cuál realiza la estimación. Dicha región se calculará mal (Figura 4.16), si en la imagen umbralizada aparecen otros objetos distintos del objeto de interés (Figura4.17).

El Filtro de Partículas, como ya hemos explicado en varias ocasiones funciona por probabilidades, y por lo tanto, el hecho de que aparezcan otros objetos umbralizados, no afecta tanto como al Filtro de Kalman. El único caso, en el que el Filtro de Partículas se ve afectado por la umbralización, es cuando aparecen objetos umbralizados de mayor tamaño que el objeto de interés, pues la partícula de mayor peso tendería a estar en esta zona.

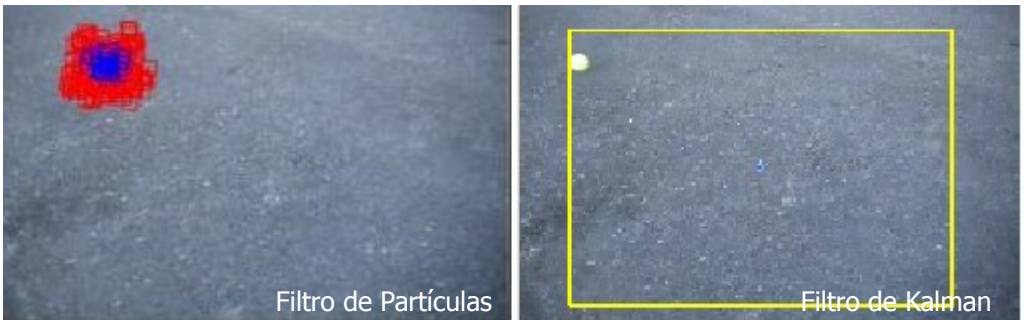


Figura 4.16. Comparación del filtro de Kalman y Partículas con respecto al procesado de la imagen

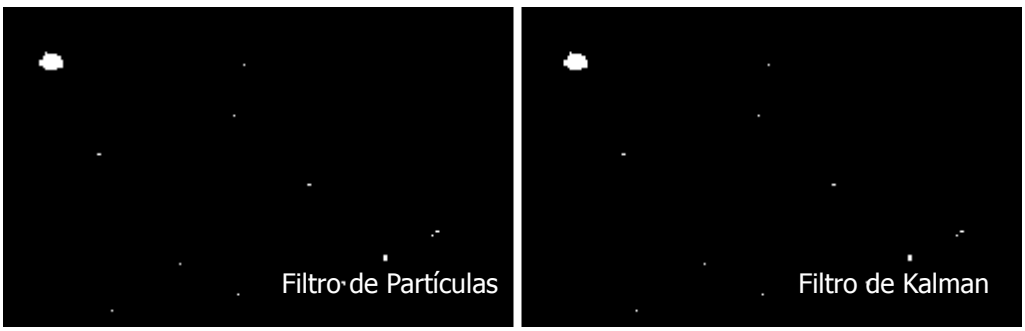


Figura 4.17. Comparación del filtro de Kalman y Partículas con respecto al procesado de la imagen (Umbralización)

2. Velocidad del objeto

El filtro de Partículas es más sensible a la velocidad del objeto que el filtro de Kalman. Como ya hemos dicho en varias ocasiones, cuando el objeto de interés alcanza una cierta velocidad, las partículas se retrasan con respecto al objeto, continuando su seguimiento. En la figura 4.18, podemos ver un ejemplo:

Figura 4.18. Comparación del filtro de Kalman y Partículas con respecto a la velocidad del objeto

2. Tiempos de ejecución de ambos filtros

Con respecto a los tiempos de procesados y fases de los filtros, he llegado a la conclusión de que el Filtro de Kalman es más rápido que el de Partículas. Esto tiene una clara explicación, y es que el Filtro de Kalman únicamente tiene dos fases (predicción y corrección), mientras que el Filtro de Partículas tiene cinco fases (inicialización, ponderación, estimación, selección y difusión).

Con respecto a nuestra implementación, la parte de procesado de la imagen es la que más tardar, siendo en ambos casos una velocidad similar.

3. Funcionamiento

Evidentemente, el Filtro de Kalman difiere con el filtro de

Partículas, en que el primero se basa en la aplicación de unas ecuaciones matemáticas mientras que el segundo se basa en probabilidades. Además, como hemos dicho anteriormente, el Filtro de Partículas tiene más fases que el filtro de Kalman.

CONCLUSIÓN:

A partir de todos los resultados obtenidos, he llegado conclusión de que el Filtro de Kalman estima mejor la posición de del objeto de interés. Esto es debido como hemos dicho en el punto 3 del apartado anterior, a que el filtro de Kalman se basa en la aplicación de unas ecuaciones matemáticas, mientras que el filtro de Partículas se basa en probabilidades.

Capítulo 5
Conclusiones y Trabajo Futuro

En este proyecto se han alcanzo los objetivos propuestos inicialmente. Este capítulo esta dedicado a la exposición de las principales conclusiones extraídas y a la presentación de posibles líneas de trabajo futuras.

5.1 Conclusiones

Con el presente Proyecto Fin de Carrera se ha presentado una implementación del *Filtro de Partículas* aplicado al seguimiento de objetos en secuencias de imágenes 2D. Para ello, se ha aprendido a usar el lenguaje de programación Visual C++, junto con la librería OpenCV, aprendiendo a la vez muchas técnicas para el procesado y el tratamiento de imágenes.

Este filtro se ha implementado mediante una jerarquía de clases utilizando el paradigma de la orientación a objetos, lo que hace que este filtro sea una aplicación sencilla en cuanto a estructura (encapsulamiento y modularidad), flexible frente posibles modificaciones y muy reutilizable para seguimiento visual en diferentes condiciones.

El filtro implementado puede utilizar diferentes modelos de

medidas, por un lado sustracción de fondo y por otro segmentación por umbralizado. Los resultados experimentales en ambos casos han sido muy satisfactorios. Se han realizado en diferentes condiciones que muestran diversos objetos, que difieren en la forma, en el tamaño, en la velocidad...Para todos ellos, se obtienen buenos resultados teniendo en cuenta los valores de sensibilidad y umbrales empleados.

Se ha realizado un estudio comparativo entre las distintas soluciones, llegando a la conclusión de que la diferencia en cuanto al rendimiento es mínima. Además se han comparado los resultados de nuestros experimentos y los obtenidos en el proyecto "Tracking automático de objetos en secuencia de imágenes usando Filtro de Kalman", comprobando que el Filtro de Kalman estima mejor la posición del objeto de interés debido a su carácter matemático, frente al carácter probabilístico del Filtro de Partículas.

Como resultado se tiene una aplicación, que realiza el seguimiento de múltiples objetos y que cuenta con un alto grado de reusabilidad y flexibilidad.

Entre sus posibles aplicaciones, podríamos destacar su uso para el desarrollo de sistemas inteligente de vigilancia o interfaces de usuario avanzadas.

5.1. Trabajos futuros

A continuación, se exponen algunas líneas de trabajo futuras de este proyecto:

1. Mejorar el modelo de medida (procesamiento), ya que este produce grandes limitaciones en el Filtro. Además, también se podría enfocar esta mejora para permitir el seguimiento de bordes, la orientación de los objetos o su posición.

2. Mejorar la implementación del doble Filtro de Partículas, ya que en nuestro caso únicamente se pueden seguir objetos de

diferentes colores.

3. Puede ser interesante para el alumnado conocer de manera intuitiva este tipo de técnicas de seguimiento, que por su complejidad teórica dificultan su asimilación. Por esto, propongo como línea de futuro trabajo la creación de una interfaz gráfica interactiva que facilite el uso y el aprendizaje del Filtro de Partículas en asignaturas de Visión Artificial.

Con la realización de este proyecto fin de carrera, he tenido que aprender nuevos conceptos, no sólo en el ámbito de la programación y de la librería OpenCV, sino también en áreas de visión artificial. Por ello, concluyo diciendo que me ha resultado un Proyecto Fin de Carrera muy interesante, que me ha formado para solventar gran parte de los problemas que me puedan surgir en el futuro profesional.

.

Capítulo 6
Anexo

En este capítulo, vamos a explicar la instalación de OpenCV en Windows, además de las distintas funciones implementadas por mí misma en la aplicación y las funciones utilizadas de OpenCV.

6.1. Instalación de OpenCV en Windows

A continuación, vamos a describir cuales son los pasos necesarios para la instalación de OpenCV en Windows.

Evidentemente, en primer lugar tenemos que descargar e instalar OpenCV. Para ello seguimos los siguientes pasos:

1. Descargar OpenCV en la siguiente dirección: htpp://sourceforge.net/projects/opencvlibrary
2. Ejecutar la instalación de OpenCV

Una vez que hemos instalado OpenCV, tenemos que seguir los siguientes pasos para configurar bajo Microsoft Visual C++ 6.0 un nuevo proyecto de consola que trabaje con las librerías OpenCV.

1. Abrimos Visual Studio y creamos una nueva aplicación:

    - Desde el menú File->New->Projects. En la ventana

abierta escogemos Win32ConsoleApplication

- Nombramos y guardamos el proyecto
- Pulsamos next
- Elegimos un proyecto vacío y finalizamos

2. Añadimos los ficheros al proyecto:

   - File->New->Files
   - En la ventana abierta escogemos categoría VisualC++ y plantillas Archivos C++
   - Le damos un nombre al fichero y guardamos

A continuación, añadimos los #includes necesarios para trabajar con OpenCV:

```
#include "cv.h"
#include "cvaux.h"
#include "highgui.h"
```

3. Configuración de los setting del proyecto
   - Botón derecho sobre el nombre del proyecto en la ventana de explorador de soluciones y pinchamos sobre propiedades
   - En la ventana abierta escogemos C++->General y añadimos en directorios de inclusión:

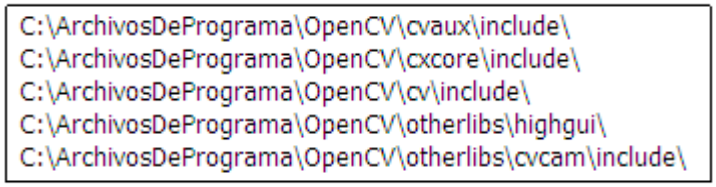


Figura 6.1. Directorios de inclusión

   - En la misma ventana que en el caso anterior, escogemos Vinculador->Entrada y añadimos en

dependencias adicionales

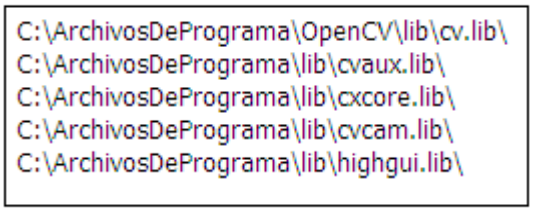


Figura 6.2. Dependencias adicionales

4. Si todo ha ido correctamente, ahora se podría ejecutar cualquier ejemplo de OpenCV o la aplicación explicada en esta memoria.

6.2. Funciones de OpenCV utilizadas en la aplicación

Como ya dijimos en el capítulo de introducción, para la implementación de esta aplicación, hemos utilizado tres módulos de OpenCV:

- cv
- cvaux
- highgui

Todas las funciones que hemos utilizado de OpenCV pertenecen a estos módulos.

A continuación, detallamos cada una de las funciones de OpenCV empleadas en la realización de este proyecto fin de carrera.

- cvNamedWindow

Esta función sirve para crear ventanas en las que

mostraremos las capturas realizadas.

La cabecera de esta función es la siguiente:

```
int cvNamedWindow( const char* name, int flags=CV_WINDOW_AUTOSIZE );
```

name: es el nombre de la ventana que muestra los resultados de nuestras capturas.

flags: este parámetro sirve para indicar el tamaño de la ventana. Con CV_WINDOW_AUTOSIZE se ajusta automáticamente el tamaño de la ventana a la imagen mostrada.

- cvDestroyWindow

Esta función sirve para destruir las ventanas creadas con la función anterior.

Su cabecera es la siguiente:

```
void cvDestroyWindow( const char* name );
```

name: nombre de la ventana que queremos destruir.

- CvShowImage

Esta función, muestra la imagen que se le indique. Su cabecera se muestra a continuación:

```
void cvShowImage( const char* name, const CvArr* image );
```

name: nombre de la ventana en la que queremos mostrar la imagen

image: imagen que queremos mostrar

Si la ventana ha sido creada con CV_WINDOW_AUTOSIZE entonces la imagen mostrada tendrá el tamaño original. Si la ventana ha sido creada con un tamaño específico, la imagen se adaptará a este tamaño.

- cvCreateImage

Esta función crea imágenes. La cabecera es la siguiente:

```
IplImage* cvCreateImage( CvSize size, int depth, int channels );
```

size: tamaño de la imagen que queremos crear.

depth: profundidad en bit de los elementos de la imagen. Puede tomar cualquiera de los siguiente valores: IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16U, IPL_DEPTH_16S, IPL_DEPTH_32S, IPL_DEPTH_32F y IPL_DEPTH_64F;

channels: número de canales que queremos que tenga la imagen.

- cvReleaseImage

El objetivo de esta función es el de liberar la memoria de las cabeceras y los datos de la imagen. Su cabecera es la siguiente:

```
void cvReleaseImage( IplImage** image );
```

image: doble puntero a la cabecera de la función de la imagen que queremos liberar.

- cvLoadImage

Esta función sirve para cargar una imagen desde un archivo. La cabecera de la función es la siguiente:

IplImage*cvLoadImage(const char* filename, int flags=CV_LOAD_IMAGE_COLOR);

filename: nombre del archivo que queremos cargar

flags: especifica el color y la profundidad de la imagen cargada

Esta función soporta archivos de tipo: jpeg, bmp, dib, jpg, png, tiff, exr...

- cvGrabFrame

Esta función graba frames desde de una cámara o de un archivo. Su cabecera se muestra a continuación:

int cvGrabFrame(CvCapture* capture);

capture: estructura del video capturado

El frame grabado se guarda internamente. Este frame no está expuesto y para volver a conseguirlo se usa la función cvRetrieveFrame.

- cvRetrieveFrame

Como hemos dicho anteriormente, cvRetrieveFrame recupera los frames grabados por cvGrameFrame. La cabecera de la función es la siguiente:

IplImage* cvRetrieveFrame(CvCapture* capture);

capture: estructura del video capturado

- cvCopy

Esta función sirve para copiar un array en otro. Su cabecera

se muestra a continuación:

```
void cvCopy( const CvArr* src, CvArr* dst, const CvArr* mask=NULL );
```

src: array de destino
dst: array de fuente
mask: número de elementos que cambia de un array a otro

- cvFlip

Esta función sirve para girar arrays en dirección vertical, horizontal o en ambas direcciones a la vez. La cabecera de la función es la siguiente:

```
void  cvFlip( const CvArr* src, CvArr* dst=NULL, int flip_mode=0);
```

scr: array fuente
dst: array de destino
flip_mode: dependiendo el número que pongamos giraremos el array en un sentido. Si ponemos 0 lo giramos en la dirección del eje de x, si ponemos un valor mayor que 0 lo giramos en dirección y, y si ponemos un valor menos que 0 lo giramos en ambas direcciones a la vez.

- cvCaptureFromCamara

Esta función sirve para asigna e inicializa la estructura cvCapture para leer una secuencia de videos desde la cámara. La cabecera de la función es la siguiente:

```
CvCapture* cvCaptureFromCAM( int index );
```

index: índice de la cámara que se está utilizando. Si sólo hay

una cámara se utiliza -1.

- cvCaptureFromAVI

Esta función al igual que la anterior, asigna e inicializa la estructura cvCapture para leer una secuencia de vídeos desde un archivo .AVI. La cabecera de la función se muestra a continuación:

```
CvCapture* cvCaptureFromAVI( const char* filename );
```

filename: nombre del archivo .AVI

- cvWaitKey

Esta función sirve para introducir pausas. Su cabecera es la siguiente:

```
int cvWaitKey(int delay CV_DEFAULT(0));
```

delay: retardo en milisegundos

- cvSmooth

Esta función sirve para aplicar un filtrado a la imagen. La cabecera de la función es la siguiente:

```
void cvSmooth( const CvArr* src, CvArr* dst,int smoothtype=CV_GAUSSIAN,int param1=3, int param2=0, double param3=0, double param4=0 );
```

src: array de fuente

dst: array de destino

smoothtype: tipo de filtrado. Se puede aplicar diferentes tipos de filtros dados por: CV_BLUR,

CV_GAUSSIAN,CV_MEDIAN,CV_BILATERAL...

- cvCvtColor

Esta función convierte a una imagen de un espacio de color a otro. Su cabecera es la siguiente:

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

scr: array fuente
dst: array destino
code:código de conversión. Los códigos de conversión vienen dados por CV_(espacio de color fuente)2(espacio de color destino). Por ejemplo, si queremos convertir una imagen de RGB a escala de grises, el código sería CV_BGR2GRAY.

- cvAbsDiff

Esta función, calcula el valor absoluto de la diferencia de dos arrays. Su cabecera es la siguiente:

```
void cvAbsDiff( const CvArr* src1, const CvArr* src2, CvArr* dst );
```

src1: el primer array fuente
src2: el segundo array fuente
dst: array de destino. En este array se almacena el valor absoluto de la diferencia de los dos arrays anteriores.

- cvThreshold

Esta función sirve para binarizar un array aplicando un umbral determinado.

void cvThreshold(const CvArr* src, CvArr* dst, double threshold, double max_value, int threshold_type);

src: array fuente

dst:array destino

threshold: valor del umbral aplicado

max_value: máximo valor usado con CV_THRESH_BINARY y CV_THRESH_BINARY_INV

type: tipo de umbralización que queremos aplicar

Existen varios tipos de binarización: CV_THRESH_BINARY, CV_THRESH_BINARY_INV,CV_THRESH_TRUNC,CV_THRESH_TOZERO, CV_THRESH_TOZERO_INV.

- cvRectangle

Esta función sirve para pintar un rectángulo en un imagen. Su cabecera es la siguiente:

void cvRectangle (IplImage* img, CvPoint pt1, CvPoint pt2, int color, int thickness);

img: imagen en la que se quiere dibujar un rectángulo

pt1: es uno de los vértices del rectángulo que queremos dibujar

pt2: es el vértice opuesto del pt1 del rectángulo que queremos dibujar

color: color del rectángulo

thickness: grosor de la línea con la que se dibuja el rectángulo

- cvGet2D

Esta función sirve para obtener el valor que tiene un píxel dela imagen. La cabecera de la función viene dada por:

```
CvScalar cvGet2D (IplImage*image, int x, int y)
```

image: imagen en la que se encuentra el píxel cuyo valor queremos conocer.

x: valor de la coordenada x del píxel

y: valor de la coordenada y del píxel

La función contraria a esta es cvSet2D, que en lugar de obtener el valor de un píxel sirve para ponerlo.

- cvSet2D

Como hemos dicho anteriormente esta función sirve para poner el valor de un píxel en una imagen determinada. La cabecera de función viene dada por:

```
void cvSet2D (IplImage*image, int x, int y)
```

image: imagen en la que queremos insertar el valor de un píxel

x: coordenada x del píxel del que queremos insertar su valor

y: coordenada y del píxel del que queremos inserta su valor

- cvSetMouseCallBack

Esta función sirve para añadir un evento de ratón. Su cabecera es la siguiente:

```
void cvSetMouseCallback( const char* window_name, CvMouseCallback on_mouse, void* param=NULL );
```

window_name: nombre de la ventana en la que vamos a añadir el evento

on_mouse: llamada a la función siempre que se quiera añadir un evento de ratón en la ventana.

Existen varios eventos de ratón:

```
#define CV_EVENT_MOUSEMOVE
#define CV_EVENT_LBUTTONDOWN
#define CV_EVENT_RBUTTONDOWN
#define CV_EVENT_MBUTTONDOWN
#define CV_EVENT_LBUTTONUP
#define CV_EVENT_RBUTTONUP
#define CV_EVENT_MBUTTONUP
#define CV_EVENT_LBUTTONDBLCLK
#define CV_EVENT_RBUTTONDBLCLK
#define CV_EVENT_MBUTTONDBLCLK
#define CV_EVENT_FLAG_LBUTTON
#define CV_EVENT_FLAG_RBUTTON
#define CV_EVENT_FLAG_MBUTTON
#define CV_EVENT_FLAG_CTRLKEY
#define CV_EVENT_FLAG_SHIFTKEY
#define CV_EVENT_FLAG_ALTKEY
```

Esta función la hemos utilizado, para añadir un evento de ratón a nuestro programa. De tal forma que si presionamos el botón izquierdo del ratón haremos segmentación por umbralizado a la imagen capturada, por el contrario si pulsamos el botón izquierdo del ratón y la tecla ctrl, entonces aplicaremos sustracción de fondo.

- cvReleaseCapture

Esta función sirve para liberar las cvCapture realizadas. Su cabecera es la siguiente:

```
void cvReleaseCapture( CvCapture** capture );
```

capture: captura que queremos liberar

6.3. Estructuras de OpenCV utilizadas en la aplicación

A continuación, se detallan las estructuras empleadas de OpenCV en la implementación de nuestra aplicación.

- cvPoint

La definición de esta estructura es la siguiente:

```
typdef struct CvPoint{
int x;
int y;
}
```

Esta estructura sirve para guardar las coordenadas x e y de un punto. Esta estructura en nuestra aplicación la utilizamos en la función cvRectangle para definir cada uno de los vértices del rectángulo que queremos dibujar.

- cvScalar

La definición de esta estructura es la siguiente:

```
typdef struct CvScalar{
double val[4];
}CvScalar;
```

Esta estructura guarda los valores de un píxel para cada componente, además de la componente α de transparencia. Esta estructura la hemos utilizado junto con la función cvGet2D, ya que para ver el valor de un píxel, tenemos que ver el valor de cada una

de sus componentes, almacenadas en una estructura de este tipo.

- cvSize

  La definición de esta estructura es la siguiente:

```
typdef struct CvSize{
 int width;
 int height;
 }
```

Esta estructura almacena la anchura y altura de una imagen. La hemos empleado en diversas ocasiones a la largo de la implementación de la aplicación, ya que en ella estaban recogidas las alturas y las anchuras de las imágenes que utilizábamos continuamente.

- IplImage

Esta estructura fue heredada de la librería de imágenes IPL (Intel Image Processing Library). Su definición es la siguiente:

```
typedef struct _IplImage
{
int  nSize;
int  ID;
int  nChannels;
int  alphaChannel;
int  depth;
char colorModel[4];
char channelSeq[4];
int  dataOrder;
int  origin;
int  align;
int  width;
int  height;
struct _IplROI *roi;
struct _IplImage *maskROI;
void  *imageId;
struct _IplTileInfo *tileInfo;
int  imageSize;
char *imageData;
int  widthStep;
```

```
int  BorderMode[4];
int  BorderConst[4];
char *imageDataOrigin;
}IplImage;
```

Como podemos ver, esta estructura recoge todas las características que tiene una imagen,anchura, altura, número de canales, tamaño, profundidad de en píxeles, región de interés...

Esta estructura ha sido empleada durante toda la implementación.

- cvCapture

Esta estructura almacena las capturas realizadas. Su definición es la siguiente:

```
typedef struct CvCapture CvCapture;
```

6.4. Funciones implementadas

Además de todos los métodos explicados y mostrados en el capítulo 3, hemos implementado otra función. Esta función tiene la siguiente cabecera:

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

Esta función sirve para añadir eventos de ratón al programa. En nuestro caso, la hemos implementado para extraer las coordenadas x e y del píxel en el que pinchamos en el método de segmentación por umbralizado. Si queremos realizar este tipo se umbralización, le he hemos puesto el evento de pulsar el botón izquierdo del ratón. Sin embargo, si queremos realizar la binarización según la técnica de la sustracción de fondo, tenemos

que pulsar la el botón izquierdo del ratón y la tecla crtl.

A continuación, se muestra el código utilizado para la implementación de esta función:

```
if(event==CV_EVENT_LBUTTONDOWN && flags & CV_EVENT_FLAG_CTRLKEY)
{
printf("Has elegido umbralizacion por extraccion de fondo\n");
sustraccion=true;
elegido=true;

}
else if(event==CV_EVENT_LBUTTONDOWN)
{
printf("Has elegido umbralizacion segun el color\n");
xSelec=x;
ySelec=y;
if(elegido==false)
{
        elegido=true;
        printf("COLOR DEL PRIMER OBJETO SELECCIONADO\n");
}
            else
             {
             elegido2=true;
             printf("COLOR DEL SEGUNDO OBJETO SELECCIONADO\n");
             }
```

Figura 6.3. Función on_mouse

Referencias

[1] Ahuactzin, J., E. Talbi, P. Bessiere, y E. Mazer (1992). Using genetic algorithms for robot motion planning. En *European Conference on Artificial Intelligence (ECAI92)*.

[2] C. Hue, J.P. Le Cadre, and P. Perez. Tracking multiple objects with particle ltering. Technical report, Research report IRISA,No1361, Oct 2000

[3] A. Doucet and C. Andrieu. Particle ltering for partially observed Gaussian state space models. Technical Report CUED/FINFENG/R393, Department of Engineering, University of Cambridge CB2 1PZ Cambridge, September 2000.

[4] Y. Bar-Shalom and T. Fortmann. *Tracking and Data Association*,volume 179 of *Mathematics in Science and Engineering*.Academic Press, 1988.

[5] Y. Bar-Shalom and X.R. Li. *Estimation and tracking: principles,techniques, and software*. Artech House, 1993.

[6] N. Bergman. *Recursive Bayesian Estimation: Navigation and Tracking Applications*. Dissertation nr. 579, Linkoping University,Sweden, 1999.

[7] A. Doucet and C. Andrieu. Particle ltering for partially observed Gaussian state space models. Technical Report CUED/FINFENG/TR393, Department of Engineering, University of Cambridge CB2 1PZ Cambridge, September 2000.

[8] I. M. Sóbol. "Métodos de Montecarlo. Lecciones populares de Matemáticas". Editorial Mir (1976).

[9] N. Bergman. Posterior Cramer-Rao bounds for sequential estimation.In A. Doucet, N. de Freitas, and N. Gordon, editors,*Sequential Monte Carlo Methods in Practice*. Springer
[10] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer Verlag, 200
[11] M. Isard and A.Blake, "CONDENSATION: Conditional density propagation for visual tracking", International Journal of Computer Vision, vol. 29, no.1, pp.5–28, 1998.
[12] M. Isard and A.Blake, "ICONDENSATION: Unifying low-level and high-level tracking in a stochastic framework", in Proc. 5th European Conference on Computer Vision, vol.1, pp.893-908, Freiburg, Germany, June 1998
[13] Gonzalo Pajares. Jesús M. De la Cruz. "Visión por Computador. Imágenes digitales y aplicaciones". Ed. Ra-Ma. 2001.
[14] Javier González Jiménez. "Visión por Computador". Ed. Paraninfo. 2000.
[15] José Francisco Vélez. Ana Belén Moreno. Ángel Sánchez. José Luis Esteban. "Visión por Computador". Ed. Universidad Rey Juan Carlos. 2003.
[16] Rafael C. González, Richard E. Woods. "Tratamiento digital de imágenes", Ed. Pratt W.K. 1991.
[17] "Digital Image Processing", Ed.Pratt W.K 1991.
[18] A. Sanchez, J. L. Esteban, J. F. Velez, and A. B. Moreno. Vision por Computador.Dykinson, 2003.