UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

DETECCIÓN Y ELIMINACIÓN DE SOMBRAS Y REFLEJOS EN ENTORNOS DE VIDEO – SEGURIDAD SOBRE PLATAFORMA DE ANÁLISIS DISTRIBUIDO

Proyecto Fin de Carrera

Héctor Cabrejas Fernández

Marzo 2010

DETECCIÓN Y ELIMINACIÓN DE SOMBRAS Y REFLEJOS EN ENTORNOS DE VIDEO – SEGURIDAD SOBRE PLATAFORMA DE ANÁLISIS DISTRIBUIDO

AUTOR: Héctor Cabrejas Fernández
TUTOR: Juan Carlos San Miguel Avedillo
PONENTE: José María Martínez Sánchez

Video Processing and Understanding Lab (VPU-Lab)
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Marzo 2010

PALABRAS CLAVE

Sistema distribuido, video – seguridad, análisis de video, detección de sombras, cámara IP, base de datos multimedia.

RESUMEN

El presente PFC tiene como objetivo contribuir en la mejora de una plataforma de análisis distribuido de secuencias de video – seguridad en dos aspectos: ampliación del sistema existente y detección de sombras en las secuencias de video captadas por el sistema.

Con este propósito, se pretende mejorar cada una de las capas funcionales (adquisición, procesado y gestión de la secuencia de video) de las que se compone el sistema de análisis distribuido disponible en el Grupo de Tratamiento e Interpretación de Video, así como ofrecer unos resultados más precisos durante la etapa de segmentación de objetos en movimiento, gracias a la detección y eliminación de sombras, dado que pueden llegar a alterar la forma del objeto detectado, dificultando sensiblemente una correcta interpretación.

Para ello se ha planteado un estudio de las aproximaciones existentes identificando sus ventajas/desventajas y su posible integración en el sistema de análisis distribuido.

ABSTRACT

The scope of the current PFC has the object of contributing to the enhancement of a video – surveillance distributed analysis platform in two ways: the expansion of the actual system and the shadow detection on the video sequences recorded by the system.

With this intention, we expect to enhance each of the functional layers which the system is composed of (acquisition, processing and video sequence management); as well as offer more accurate results on the movement segmentation phase thanks to the detection and deletion of the shadows, due they can alter the shape of the detected object, hence making the right interpretation slightly harder.

We present a comprehensive survey of moving shadow detection approaches, identifying their advantages / disadvantages, and its possible deployment into the video – surveillance distributed system.

Agradecimientos

En primer lugar, me gustaría agradecer a los profesores que más he admirado durante esta etapa universitaria y que recordaré con más cariño: Chema, Javier, Jesús y Jorge.

Sin duda alguna, agradezco todo el tiempo que me ha dedicado Juan Carlos, mi tutor, siempre que lo he necesitado. Su ayuda ha sido decisiva en la consecución de este PFC.

También se merecen mi agradecimiento Bruno y Gema. Jefes, y sin embargo amigos. Gracias a ellos, estos últimos meses he podido compaginar mi trabajo con la realización de este PFC sin tener que renunciar a ninguno de ellos.

Por supuesto, a todos mis amigos y amigas. Sólo por el hecho de terminar el PFC, hacen que merezca la pena todo este esfuerzo.

Pero sobretodo, se lo quiero dedicar a mi madre. Ella es la que más me ha tenido que soportar estos meses, siempre cariñosa y comprensiva. Después de todo este tiempo, estoy convencido de que ser ingeniero, por fin, es el mejor regalo que puedo darte.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1 MOTIVACIÓN

Actualmente, existe una creciente demanda de sistemas de video – seguridad[1] para su uso en lugares públicos debido al incremento de seguridad solicitado por parte de la sociedad. En este contexto, el uso de sistemas distribuidos se ha convertido en una solución muy popular debido a las ventajas que proporciona para su diseño, instalación y monitorización de grandes áreas (por ejemplo, en lugares públicos).

En este tipo de sistemas distribuidos, tanto su escalabilidad como la comunicación entre los diferentes módulos adquieren un papel esencial con el objetivo de compartir toda la información de las diferentes fuentes disponibles y unidades de análisis. Se deben tener en cuenta por tanto cuestiones como la gestión, almacenamiento, análisis y adquisición de las secuencias de video durante las etapas de diseño e implementación.

El propósito fundamental de dichos sistemas es el de proveer una interpretación automática de la escena basándose en la información adquirida por los sensores. Esta interpretación puede ser utilizada por supervisores humanos para centrar su atención cuando ocurren situaciones peligrosas o extrañas en las secuencias de video capturadas por el sistema. Adicionalmente, el análisis automático puede ser utilizado para indexar el material audiovisual almacenado.

La interpretación automática realizada se basa en la identificación de regiones de interés en la secuencia de video capturada. Este proceso se conoce como detección y segmentación de objetos de interés y es el núcleo fundamental del sistema debido a que las etapas posteriores de análisis (por ejemplo: reconocimiento y seguimiento de objetos, detección de eventos,…) dependen directamente de la eficiencia de esta etapa.

Sin embargo, uno de los principales retos en la detección de objetos consiste en la identificación de aquellas sombras proyectadas por objetos en movimiento, y que pueden llegar a alterar la forma del objeto detectado, dificultando sensiblemente una correcta interpretación.

Las dificultades que presenta una correcta detección de sombras radican en que tanto las sombras como los objetos que las proyectan comparten dos importantes características: ambas modifican la escena y seguirán el mismo comportamiento. No obstante, existen diversos métodos para reducir los efectos negativos que provocan las sombras en movimiento durante la etapa de detección de objetos.

[1] **Video – seguridad** Con este término nos referiremos a aquellos sistemas de seguridad que hacen uso de secuencias de video. También se usa el término video – vigilancia. No obstante, en este texto nos referiremos a dicho concepto usando el término video – seguridad, al considerarlo más amplio. (Del inglés *video – surveillance)*

1.2 OBJETIVOS[2]

El presente PFC tiene como objetivo la contribución en sistemas de análisis de secuencias de video – seguridad en dos aspectos: mejora del sistema distribuido existente en el Grupo de Tratamiento e Interpretación de Vídeo de la Universidad Autónoma de Madrid (VPU-UAM) y detección y eliminación de sombras en las secuencias de video captadas por el sistema.

Para llevar a cabo los objetivos propuestos anteriormente, se proponen los siguientes hitos:

- Estudio de la plataforma de análisis distribuido *DiVA (Distributed Video Analysis)* existente en el Grupo de Tratamiento e Interpretación de Vídeo de la Universidad Autónoma de Madrid (VPU-UAM).

- Diseño, desarrollo e integración en la plataforma *DiVA* de un nuevo módulo capturador de secuencias de video a través de cámaras IP.

- Diseño, desarrollo e integración en la plataforma *DiVA* de un servidor que actúe como base de datos para gestionar toda la información generada por la plataforma *DiVA*.

- Estudio analítico de los algoritmos de detección de sombras en secuencias de vídeo y selección de aproximaciones más representativas. Implementación de algoritmos más representativos.

- Diseño, desarrollo e integración en la plataforma *DiVA* de un componente de análisis dedicado a detectar y eliminar sombras en secuencias de video capturadas.

- Análisis de los resultados obtenidos. Evaluación comparativa de los algoritmos de sombras representativos. Evaluación de las mejoras funcionales introducidas en la plataforma *DiVA*.

[2] **NOTA** Si bien se había planteado en un principio la detección de sombras y brillos en movimiento, finalmente tan sólo se ha realizado un análisis acerca de la etapa de detección de sombras.

1.3 ORGANIZACIÓN DE LA MEMORIA

La memoria consta de los siguientes capítulos:

CAPÍTULO 1 INTRODUCCIÓN

- Objetivos y motivación del proyecto.
- Organización de la memoria.

CAPÍTULO 2 ESTADO DEL ARTE

- Estudio de la evolución y arquitectura de los sistemas de video – seguridad hacia un modelo distribuido.
- Estudio analítico de las técnicas utilizadas en la etapa de detección y eliminación de sombras.

CAPÍTULO 3 SISTEMA *DiVA*

- Descripción del diseño y arquitectura del sistema *DiVA*.
- Mejoras propuestas (objetivos del PFC).

CAPÍTULO 4 MÓDULO CAPTURADOR DE DATOS DE CÁMARAS IP

- Diseño, desarrollo e integración de un servidor de secuencias de video a través de cámaras IP.

CAPÍTULO 5 MÓDULO SERVIDOR DE DATOS

- Diseño, desarrollo e integración de un servidor que actúe como base de datos para gestionar toda la información generada por el sistema.

CAPÍTULO 6 MÓDULO DE DETECCIÓN DE SOMBRAS

- Diseño, desarrollo e integración de un módulo que procese la secuencia de video y detecte aquellas zonas que se correspondan con sombras.

CAPÍTULO 7 PRUEBAS Y RESULTADOS

- Resultados comparativos de los algoritmos de sombras analizados.
- Análisis del comportamiento final del sistema.

CAPÍTULO 8 RESUMEN, CONCLUSIONES Y TRABAJO FUTURO

- Resumen y conclusiones obtenidas tras el desarrollo del sistema. Relación de posibles líneas futuras de mejora del sistema.

2. ESTADO DEL ARTE

2.1 SISTEMAS DISTRIBUIDOS DE VIDEO – SEGURIDAD

2.1.1 INTRODUCCIÓN

Los sistemas de video – seguridad tienen como objetivo monitorizar, tanto en tiempo real como a posteriori (*off – line*), todos aquellos objetos, individuos y eventos de interés en un entorno específico. El propósito fundamental de estos sistemas es el de proveer una interpretación automática de la escena, entender y predecir las acciones e interacciones de los objetos presentes en la escena, basándose en la información adquirida por los sensores del sistema.

El creciente interés en este campo queda patente por las múltiples conferencias impulsadas por IEEE[3] sobre video – seguridad (1; 2; 3) así como ediciones especiales de revistas que se centran específicamente en este ámbito (4; 5; 6).

La evolución tecnológica de los sistemas de video – seguridad comenzó con los sistemas analógicos CCTV[4]. Estos sistemas, denominados de primera generación (7), consisten en un número de cámaras posicionadas en distintas localizaciones y conectadas a monitores, comúnmente situados en una misma sala, y supervisados por uno o varios operadores. En (8), por ejemplo, se estudia la integración de diferentes sistemas CCTV para la monitorización de sistemas de transporte.

Dichos sistemas CCTV utilizan señales analógicas para la distribución de la imagen así como su almacenamiento. Sin embargo, esto provoca la degradación de la imagen y la señal de video se vuelve susceptible al ruido. Además, este sistema tradicional de video – vigilancia activo, se vuelve ineficiente a medida que el número de cámaras excede la capacidad de atención necesaria por los operadores encargados de monitorizarlas.

La evolución tecnológica de los equipos informáticos provocó una mejora en estos sistemas de video – seguridad, aprovechando la ventaja del formato digital en el que se captura la imagen y el incremento de la capacidad de procesado de los equipos. El esfuerzo se centró en la generación de algoritmos de detección y seguimiento de objetos, permitiendo al operador reconocer e identificar más fácilmente aquellos eventos que se producen en las diferentes escenas captadas por las cámaras. Estos sistemas de video – seguridad semiautomáticos son conocidos como sistemas de segunda generación (7).

[3] **IEEE** Instituto de Ingenieros Electricistas y Electrónicos. (Del inglés *Institute of Electrical & Electronics Engineers, Inc.*)

[4] **CCTV** Circuito cerrado de televisión. (Del inglés *Closed Circuit Television*)

Figura 2-1: Sala de control de cámaras de video – seguridad en Bristol.

Más tarde, la demanda creciente de la sociedad por una mayor seguridad guió a la necesidad de una mejora de las actividades y procesos de video – vigilancia en diferentes entornos, de una forma remota, y centrados específicamente en estas áreas:

- Medios de transporte como aeropuertos, entornos marítimos, transportes públicos y seguimiento del tráfico en ciudades y carreteras.

- Vigilancia de comunidades, empresas, edificios importantes e instituciones, hogares, garajes,….

- Análisis y vigilancia remota de agrupaciones de personas para medir la asistencia de público en eventos deportivos u otras actividades como manifestaciones, congregaciones en sitios públicos,…

- Control de procesos industriales, médicos o incluso militares.

Esta denominada tercera generación de sistemas de video – seguridad (7) es referida a aquellos sistemas concebidos para manejar una gran cantidad de cámaras, desplegando los recursos disponibles a lo largo de un área más amplia, incluso con diferentes centros de control compartiendo la información disponible, y tratando de reflejar la naturaleza jerárquica y distribuida del proceso de vigilancia desde un punto de vista humano.

2.1.2 ESQUEMA GENERAL DE UN SISTEMA DE VIDEO – SEGURIDAD

La video – vigilancia, tal y como se puede entender como concepto, trata de detectar, reconocer y seguir ciertos objetos a partir de una secuencia de video dada, y de un modo más genérico, entender y describir el comportamiento de dichos objetos.

El principal propósito de este apartado es el de ofrecer una imagen panorámica del proceso que debe seguir un sistema de video – seguridad en todo su conjunto. A continuación se muestra el esquema general de dicho sistema.

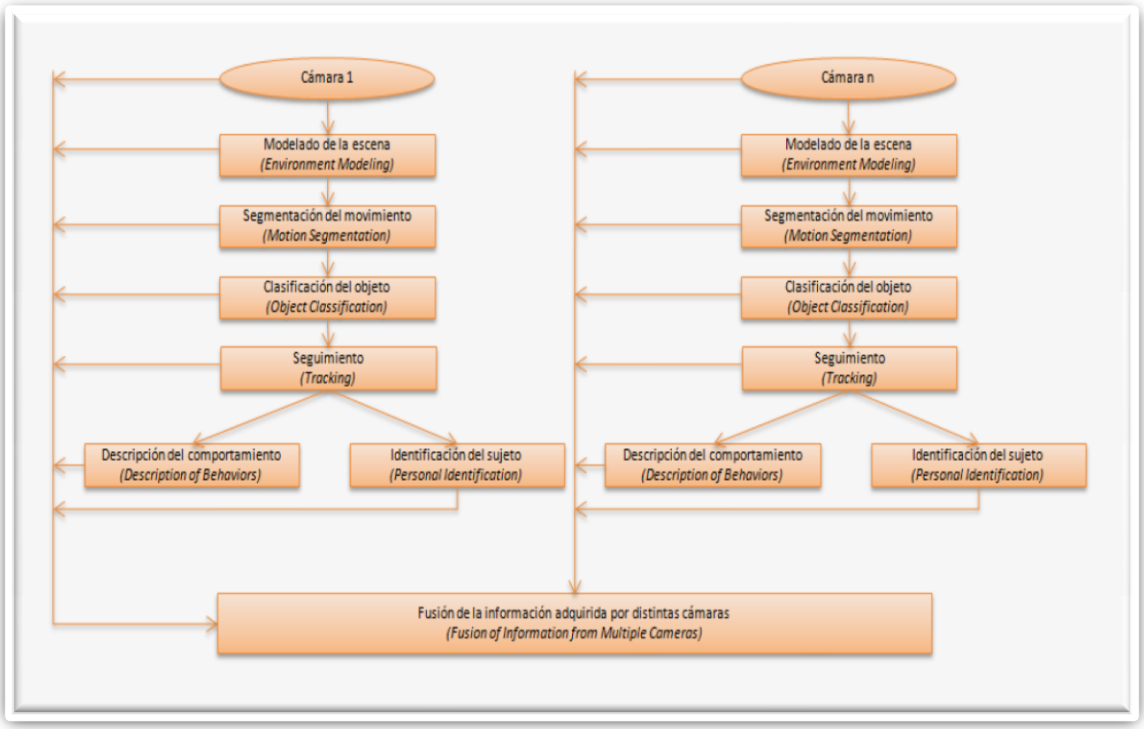

Figura 2-2: Esquema general de un sistema de video – seguridad.

Las principales etapas que deben contemplarse a la hora de diseñar un sistema completo de video – seguridad, utilizando una única cámara deben ser, según (9), las siguientes: detección de movimiento *(Motion Detection)*, seguimiento del objeto *(Object Tracking)*, descripción del comportamiento *(Description of Behaviors)* e identificación del sujeto *(Personal Identification)*.

Por último, se define una última etapa en aquellos sistemas con un amplio número de cámaras: Fusión de la información adquirida por distintas cámaras *(Fusion of Information from multiple cameras)*.

Si bien es cierto que un sistema puede tener todas estas etapas, éstas no son estrictamente obligatorias, pues esta decisión depende de los criterios de diseño y requisitos del sistema, orientados a ofrecer una funcionalidad específica.

- **DETECCIÓN DEL MOVIMIENTO** *(Motion Detection)*

Esta etapa tiene como objetivo la extracción o segmentación de aquellas regiones correspondientes a objetos en movimiento con respecto al resto de la imagen. Etapas posteriores como el seguimiento del objeto y el análisis y descripción del comportamiento dependen completamente de ésta.

- **SEGUIMIENTO DEL OBJETO** *(Object Tracking)*

Esta etapa pretende realizar un seguimiento del objeto en movimiento a lo largo de una secuencia de video. El propósito de ésta radica en relacionar temporalmente la información obtenida en el análisis de cada *frame* y así caracterizar el movimiento del objeto en cuestión. Dicha etapa suele estar interrelacionada con la etapa de detección del movimiento durante todo el proceso.

- **DESCRIPCIÓN DEL COMPORTAMIENTO** *(Description of Behaviors)*

Una vez se ha conseguido detectar el objeto en movimiento, y caracterizar su trayectoria, el siguiente paso es el de comprender el comportamiento de dicho objeto. Basándose en la existencia y definición de modelos de eventos, y buscando similitudes entre éstos y las observaciones, es posible realizar descripciones de alto nivel de acciones o interacciones entre objetos. Uno de los ejemplos más claros y representativos que se realiza en esta etapa es el de detectar la presencia de objetos abandonados.

- **IDENTIFICACIÓN DEL SUJETO** *(Personal Identification)*

Esta etapa tiene como propósito el reconocimiento de los individuos en una determinada escena. Actualmente, existen multitud de técnicas de identificación: biometría, reconocimiento facial, patrones de movimiento,…

- **FUSIÓN DE LA INFORMACIÓN ADQUIRIDA POR DISTINTAS CÁMARAS** *(Fusion of Information from multiple cameras)*

Esta última etapa trata de combinar y compartir la información adquirida por distintas cámaras y favorecer la sinergia proporcionada por la utilización de diferentes fuentes de información. Resulta de especial interés cuando se quiere ampliar el área de vigilancia, además de dar la oportunidad de obtener información adicional del mismo objeto desde otro punto de vista.

2.1.3 EVOLUCIÓN HACIA UN MODELO DISTRIBUIDO

Hasta ahora, se ha analizado el esquema general de un sistema de video – seguridad como tal, en el que un dispositivo se encargaba de procesar la secuencia de video a lo largo de las diferentes etapas.

Sin embargo, los recientes avances tecnológicos, el abaratamiento de los dispositivos de captura, almacenamiento y análisis; y la necesidad de monitorizar grandes áreas han dado lugar al desarrollo de un nuevo tipo de sistemas de video – seguridad distribuidos conocidos como sistemas de tercera generación (7).

El propósito de dichos sistemas es el de facilitar la tarea desarrollada por el operador, centrándose en proveer una interpretación automática de la escena lo más completa posible, distribuyendo los procesos entre diferentes unidades funcionales no especializadas capaces de procesar la secuencia de video y que se comunican entre ellas a través de la red, y aprovechando, en definitiva, las ventajas que nos ofrece un sistema distribuido: escalabilidad y robustez.

Dos requisitos son fundamentales para el diseño de este tipo de sistemas (10):

- Un sistema genérico, que sea fácilmente exportable a otras localizaciones o utilizado con diferentes propósitos.

- Un sistema distribuido y escalable, en el que el alto coste computacional necesario para realizar las tareas de video – seguridad se divida entre varias unidades funcionales.

Los entornos multi – sensores distribuidos a lo largo de un área presentan interesantes oportunidades y retos relacionadas con el entorno distribuido de análisis. Éstas se pueden resumir en las siguientes:

- Diseño y despliegue óptimos de la red de sensores, unidades de almacenamiento, análisis,…

- Comunicación entre los diferentes módulos del sistema. Aspectos como las limitaciones en el ancho de banda o la naturaleza asimétrica de la comunicación (11) deben ser contemplados.

- Almacenamiento de la información extraída por los módulos de análisis (12) así como el planteamiento de nuevas soluciones de comunicación de las señales de video y toda la información extraída en los sistemas distribuidos de video – seguridad. Con este punto de vista, se han propuesto diferentes técnicas de compresión de video (13), técnicas de protocolo y red (14), distribución de las tareas de procesado (15), así como el desarrollo de posibles estándares para la transmisión de la información a través de la red (16; 12).

- Fusión de la información obtenida por diferentes cámaras.

Por ejemplo, (17) presenta un sistema distribuido de video – seguridad diseñado para cubrir, mediante varios módulos autónomos (clústeres), diferentes tramos de tráfico a lo largo de una autopista (Figura 2-3).

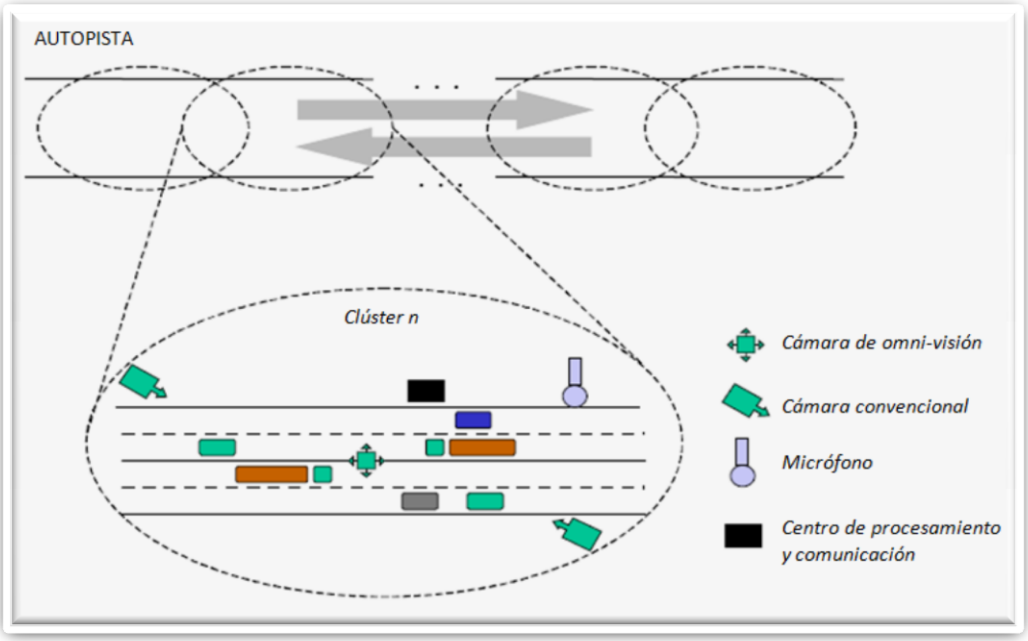

Figura 2-3: Sistema distribuido de video – seguridad en autopistas.

En esta propuesta, cada uno de los centros de procesamiento y comunicación se comporta como un sistema de video – seguridad independiente, siguiendo un esquema similar al expuesto anteriormente. Sin embargo, estos clústeres pueden comunicarse entre ellos, así como con un servidor (o varios) donde se centraliza la información extraída y se permite el análisis del comportamiento a un más alto nivel, tal y como podemos ver en la Figura 2-4.

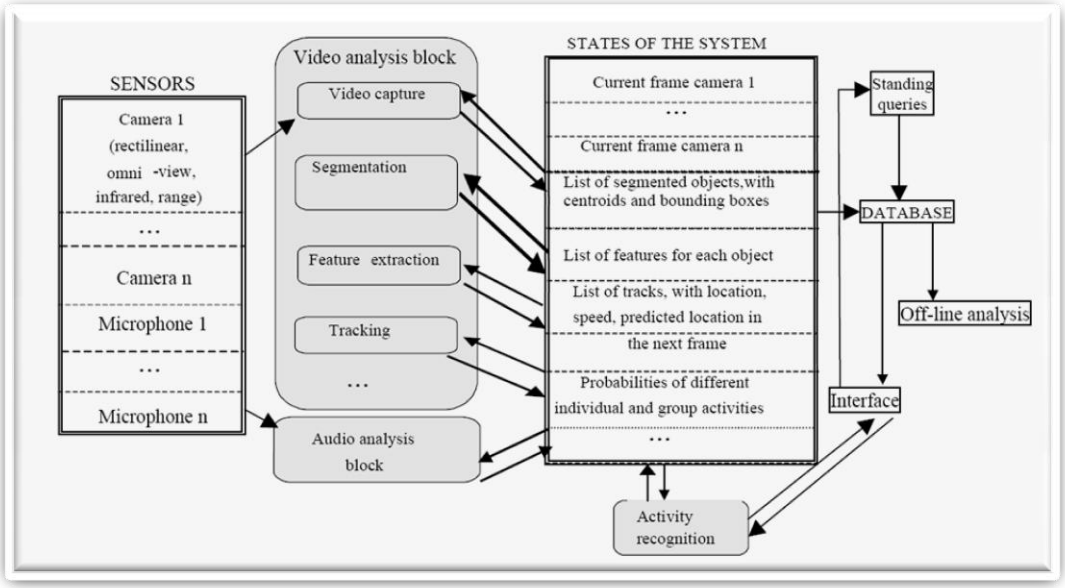

Figura 2-4: Esquema general de un sistema distribuido de video – seguridad.

Sin embargo, estas características dependen en gran medida del diseño del sistema, por lo que no se puede establecer un comportamiento específico, sin analizar también el diseño propuesto para cada uno de los sistemas. Por tanto, y en el ámbito de desarrollo de este PFC, no se profundizará en el estudio de estas investigaciones.

2.2 DETECCIÓN Y ELIMINACIÓN DE SOMBRAS

2.2.1 INTRODUCCIÓN

La detección de objetos de interés es el núcleo principal en el que se basan los sistemas de video – seguridad, debido a que las etapas de análisis de nivel medio (por ejemplo: seguimiento de objetos) o alto (por ejemplo: detección de eventos) dependen directamente de la eficiencia de dicha etapa de detección.

Uno de los principales retos en la detección de objetos consiste en la identificación de aquellas sombras proyectadas por objetos en movimiento, tanto sobre la escena como sobre éste mismo.

Concretamente, el efecto de las sombras puede provocar, en determinadas situaciones, la fusión de varios objetos independientes (*Object merging*), distorsión de la forma del objeto detectado (*Object shape distortion*) o incluso la no detección de objetos (*Object losses*) debido a la proyección de una sombra sobre dichos objetos.

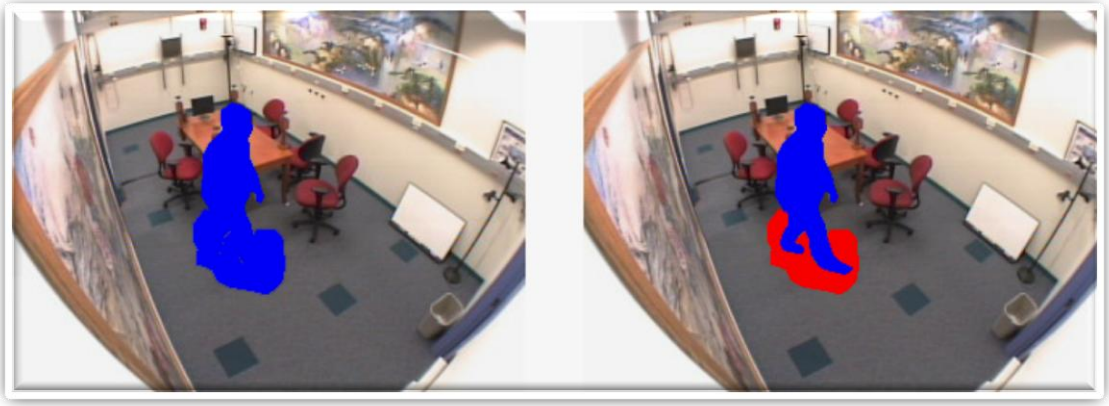

Figura 2-5: Distorsión de la forma del objeto: (i) Sin detección de sombras. (d) Con detección de sombras.

Las dificultades que presenta la etapa de detección de sombras radican en que tanto los objetos como las sombras comparten dos importantes características:

- Los píxeles localizados en una región sombreada también serán clasificados como *Foreground*[5] debido a que existe una diferencia significativa con respecto al *Background*[6].

- Tanto la sombra como el objeto que la proyecta seguirán el mismo comportamiento. Es decir, sus movimientos serán similares.

[5] **Foreground** Sección de la escena donde se encuentra el objeto o persona en movimiento.

[6] **Background** Sección de la escena que permanece estática. También denominado fondo de imagen.

La etapa de detección y eliminación de sombras se realiza normalmente a partir de la información extraída durante la etapa de segmentación de movimiento (ver ANEXO A: , siguiendo el esquema propuesto en la Figura 2-6:

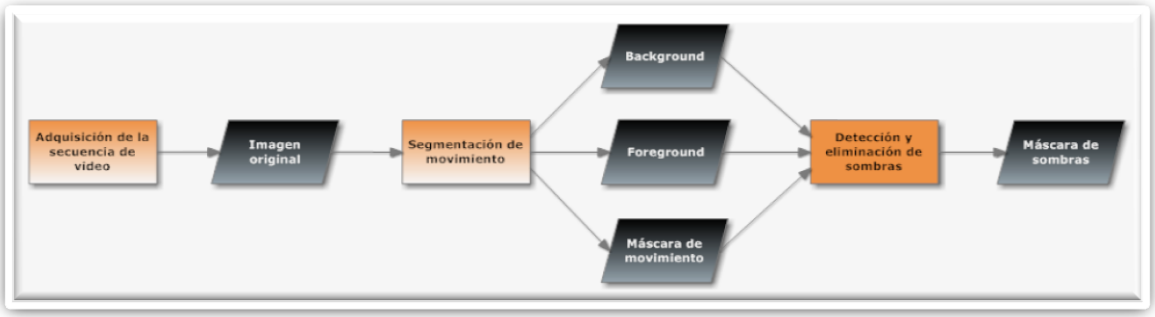

Figura 2-6: Proceso de detección y eliminación de sombras: Diagrama de bloques.

Detectar estas zonas en movimiento provee una información indispensable en la etapa posterior de detección de sombras, ya que tan sólo dichas zonas serán consideradas, con el propósito de mejorar los resultados obtenidos durante la etapa de segmentación de movimiento.

A lo largo de este apartado se plantean las distintas técnicas propuestas para detectar y eliminar sombras en función de las características que éstas presentan.

2.2.2 ANÁLISIS Y MODELADO DE LAS SOMBRAS

Una sombra es una región de oscuridad donde la luz es obstaculizada por un objeto parcial o completamente. Puede proveer información importante acerca de la forma del objeto que la provoca, así como de la orientación de la luz. Sin embargo, es uno de los principales problemas en la detección y segmentación de objetos en movimiento, debido a que no se tiene información de la orientación ni cantidad de luz de la escena.

Figura 2-7: Sombra de la Torre Eiffel a mediodía.

De acuerdo con los modelos *Lambertiano*, *Phong* y dicromático (18), la intensidad de la luz reflejada es la suma de las fuentes de intensidad y la reflectancia de la superficie iluminada.

Por tanto, la respuesta del sensor será la suma de de las respuestas de aquellas fuentes reflectantes así como las superficies iluminadas reflectantes.

$$\boldsymbol{I(\lambda,\vec{x}) = I_{\alpha}(\lambda,\vec{x})\rho(\lambda,\vec{x}) + I_{i}(\lambda,\vec{x})\rho(\lambda,\vec{x})}$$

$$\boldsymbol{C_l(\vec{x}) = \int_{\lambda} I(\lambda,\vec{x})S_C(\lambda)d\lambda = \int_{\lambda} \big(I_{\alpha}(\lambda,\vec{x}) + I_i(\lambda,\vec{x})\big)\rho(\lambda,\vec{x})S_C(\lambda)d\lambda}$$

Donde $\boldsymbol{I(\lambda,\vec{x})}$ es la intensidad de la luz reflejada, $\boldsymbol{\lambda}$ la longitud de onda, $\boldsymbol{\vec{x}}$ el punto que pertenece a la superficie del objeto, $\boldsymbol{I_{\alpha}(\lambda,\vec{x})}$ la intensidad de la superficie iluminada, $\boldsymbol{I_i(\lambda,\vec{x})}$ la intensidad de la fuente de iluminación, $\boldsymbol{\rho(\lambda,\vec{x})}$ el coeficiente de reflexión, $\boldsymbol{C \epsilon R, G, B}$ la respuesta al sensor, $\boldsymbol{C_l(\vec{x})}$ la respuesta iluminada al sensor en cada punto, y $\boldsymbol{S_C(\lambda) \epsilon S_R(\lambda), S_G(\lambda), S_B(\lambda)}$ la sensibilidad espectral del sensor.

Para píxeles situados en una región sombreada, la intensidad de la fuente de iluminación $\boldsymbol{I_i(\lambda,\vec{x})}$ será cero, ya que ésta es obstaculizada. Por tanto, la respuesta al sensor tan sólo se verá determinada por la respuesta de la superficie iluminada reflectante. Así,

$$\boldsymbol{I(\lambda,\vec{x}) = I_{\alpha}(\lambda,\vec{x})\rho(\lambda,\vec{x})}$$

$$\boldsymbol{C_S(\vec{x}) = \int_{\lambda} I_{\alpha}(\lambda,\vec{x})\rho(\lambda,\vec{x})S_C(\lambda)d\lambda}$$

Donde $\boldsymbol{C_S(\vec{x})}$ es la respuesta del sensor en la región sombreada. Suponiendo que la fuente tiene las mismas características espectrales y diferente magnitud de la función de distribución espectral con respecto a la iluminación de la superficie, la intensidad de la fuente de iluminación será proporcional a la iluminación de la superficie.

Esto es:

$$\boldsymbol{I_i(\lambda,\vec{x}) = \beta I_{\alpha}(\lambda,\vec{x})}$$

De manera que:

$$\boldsymbol{C_l(\vec{x}) = (1 + \beta)C_S(\vec{x})}$$

De acuerdo con esta última ecuación, y para cada píxel, la intensidad de color en la región sombreada será más pequeña que bajo iluminación.

2.2.3 CLASIFICACIÓN DE LAS SOMBRAS

Generalmente, las sombras pueden dividirse entre sombras estáticas y sombras dinámicas. No obstante, las sombras estáticas no son detectadas en la etapa de detección de movimiento, ya que son modeladas como parte del *Background* de la escena. Por tanto, y en el ámbito que nos ocupa, tan sólo serán objeto de interés aquellas sombras dinámicas asociadas a objetos tales como vehículos o personas en movimiento.

Según nuestra percepción visual, se puede asumir que las sombras en movimiento deben cumplir una serie de condiciones:

- La sombra es la proyección de un objeto en movimiento y siempre está relacionada con dicho objeto.

- La luminancia (intensidad) de aquellas regiones de la escena cubiertos por una sombra será menor que en aquellas regiones similares de la escena donde no lo esté.

- La sombra proyectada sobre un píxel no modifica su información de color o cromaticidad.

Siguiendo este criterio, y tal y como se contempla en (19), las sombras se pueden clasificar en dos grupos:

- Autosombra *(Self-Shadow)*: Es la parte del objeto que no se encuentra iluminada.

- Sombra proyectada *(Cast Shadow)*: Es la región no iluminada proyectada por el objeto sobre la escena. Este grupo puede, a su vez, subdividirse en: umbra *(Dark Shadow)* y penumbra *(Soft Shadow).*

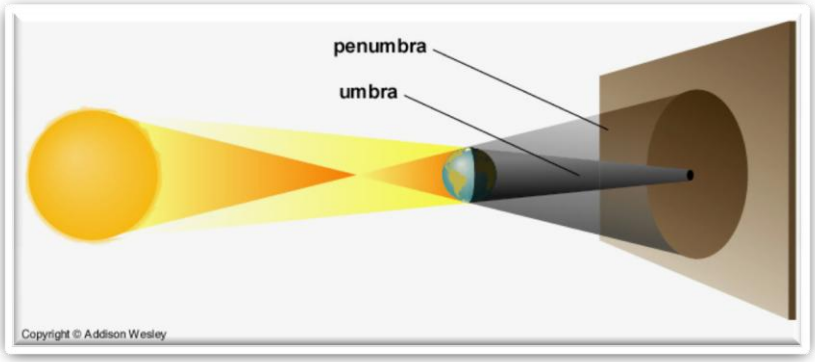

Figura 2-8: Clasificación de las sombras proyectadas: umbra y penumbra.

Si la fuente de luz es fija, y mientras el objeto permanezca en movimiento, tanto la autosombra como la sombra proyectada cambiarán de forma. Debido a que la autosombra es parte del objeto, ésta no debe eliminarse durante la detección de movimiento. Así pues, la mayoría de los algoritmos de detección de sombras se centran en la eliminación de los efectos de las sombras proyectadas (umbra y penumbra).

Debido a las diferentes condiciones de iluminación posibles en una escena, las características espectrales de las regiones sombreadas serán diferentes. Siguiendo este razonamiento (20), se pueden reclasificar aquellas sombras que se pretenden detectar y eliminar en dos grupos: sombras invisibles *(Invisible Shadow)* y sombras visibles *(Visible Shadow).*

Figura 2-9: Reclasificación de sombras: (i) Sombra invisible. (d) Sombra visible.

Si bien según nuestra percepción visual no se puede discernir, normalmente, las sombras invisibles proyectadas en la escena, existen multitud de situaciones iluminadas con intensidades de luz débiles en las que los objetos en la escena proyectan una leve sombra y modifican ligeramente su entorno.

Como veremos más adelante, existen espacios de color que se verán más afectados por este tipo de sombras, mientras que otros no se alterarán debido a la presencia de éstas.

2.2.4 ALGORITMOS DE DETECCIÓN DE SOMBRAS

La mayoría de los algoritmos de detección y eliminación de sombras en movimiento desarrollados en la literatura se caracterizan por basarse en el modelo de sombras descrito en 2.2.2. Sin embargo, existen diferentes propuestas dependiendo del enfoque utilizado.

2.2.4.1 ENFOQUE BASADO EN EL PROCESO DE DECISIÓN

(21) propone una clasificación de algoritmos, tal y como podemos ver en la Figura 2-10, basándose en los procesos de decisión desarrollados, y destacando las diferentes características que se plantean para el análisis de la secuencia.

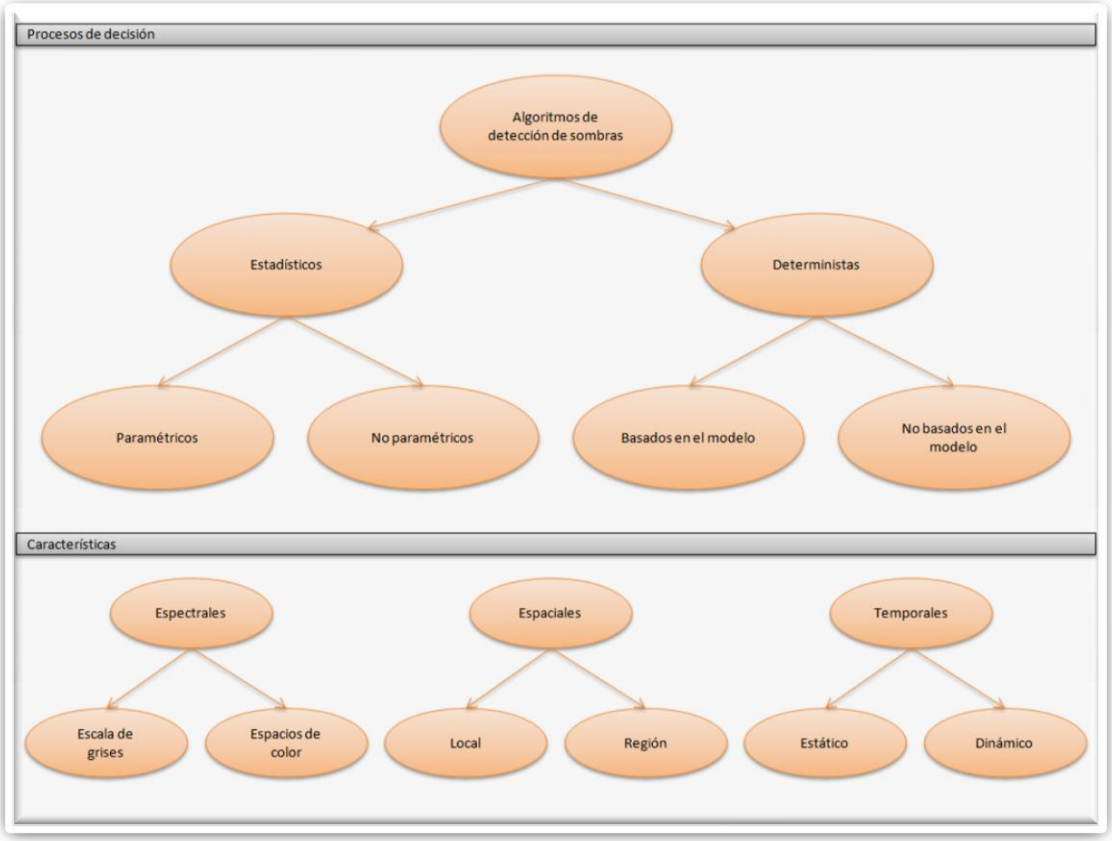

Figura 2-10: Clasificación de los algoritmos de detección de sombras basada en el proceso de decisión.

Se consideran dos procesos de decisión diferentes: métodos determinísticos y estadísticos. Los métodos estadísticos se basan en procesos de decisión binarios (*On/Off*). Adicionalmente, los métodos determinísticos también pueden sub-dividirse, basándose en si la decisión binaria se toma a partir del conocimiento del modelo de la escena o no. Los métodos estadísticos utilizan funciones probabilísticas para tomar estas decisiones, siendo la etapa de selección de parámetros crítica para conseguir un correcto comportamiento en la detección de sombras, y que provoca la escisión de éstos en dos sub-niveles: paramétricos y no paramétricos.

El segundo nivel de esta clasificación propone diferentes características de los algoritmos de detección y eliminación de sombras. Básicamente, estas características pueden ser extraídas de tres dominios: espectral, espacial y temporal.

Prácticamente todos los estudios al respecto se basan en la extracción de características espectrales. No obstante, muchos plantean el uso adicional de técnicas que exploten las características espaciales y temporales para mejorar el comportamiento final de los algoritmos.

A continuación, se describen brevemente aquellas propuestas representativas de tres de las cuatro categorías clasificadas en función del proceso de selección. El método determinístico basado en el modelo no se ha incluido en este estudio, pese a que ofrece indudablemente los mejores resultados, ya que es excesivamente complejo y con un alto coste computacional.

- **MÉTODO ESTADÍSTICO NO PARAMÉTRICO**

El trabajo propuesto por (22) es el ejemplo de método estadístico no paramétrico (*SNP*[7]) que va a analizarse.

En primer lugar, se utilizan los primeros $\boldsymbol{N}$ *frames* para calcular las medias y varianzas de los distintos canales de color para cada píxel (en este caso, en el espacio de color *RGB*). Así, se obtiene $\boldsymbol{E_i} = [\boldsymbol{\mu_{R_i}}, \boldsymbol{\mu_{G_i}}, \boldsymbol{\mu_{B_i}}]$ (vector media), y $\boldsymbol{s_i} = [\boldsymbol{\sigma_{R_i}}, \boldsymbol{\sigma_{G_i}}, \boldsymbol{\sigma_{B_i}}]$ (vector varianza), para un píxel localizado en i.

La distorsión del brillo $\boldsymbol{\alpha_i}$ y la distorsión de la crominancia $\boldsymbol{CD_i}$de la diferencia entre el valor esperado del píxel y su valor en la imagen actual $\boldsymbol{I_i} = [\boldsymbol{I_{R_i}}, \boldsymbol{I_{G_i}}, \boldsymbol{I_{B_i}}]$, se calcula de la siguiente manera:

$$\boldsymbol{\alpha_i} = \frac{\left(\frac{\boldsymbol{I_{R_i}}\boldsymbol{\mu_{R_i}}}{\boldsymbol{\sigma^2_{R_i}}} + \frac{\boldsymbol{I_{G_i}}\boldsymbol{\mu_{G_i}}}{\boldsymbol{\sigma^2_{G_i}}} + \frac{\boldsymbol{I_{B_i}}\boldsymbol{\mu_{B_i}}}{\boldsymbol{\sigma^2_{B_i}}}\right)}{\left(\left[\frac{\boldsymbol{\mu_{R_i}}}{\boldsymbol{\sigma_{R_i}}}\right]^2 + \left[\frac{\boldsymbol{\mu_{G_i}}}{\boldsymbol{\sigma_{G_i}}}\right]^2 + \left[\frac{\boldsymbol{\mu_{B_i}}}{\boldsymbol{\sigma_{B_i}}}\right]^2\right)}$$

$$\boldsymbol{CD_i} = \sqrt{\left(\frac{\boldsymbol{I_{R_i}} - \boldsymbol{\alpha_i}\boldsymbol{\mu_{R_i}}}{\boldsymbol{\sigma_{R_i}}}\right)^2 + \left(\frac{\boldsymbol{I_{G_i}} - \boldsymbol{\alpha_i}\boldsymbol{\mu_{G_i}}}{\boldsymbol{\sigma_{G_i}}}\right)^2 + \left(\frac{\boldsymbol{I_{B_i}} - \boldsymbol{\alpha_i}\boldsymbol{\mu_{B_i}}}{\boldsymbol{\sigma_{B_i}}}\right)^2}$$

Y una vez determinados estos valores y utilizando $\widehat{\boldsymbol{CD_i}}$ y $\widehat{\boldsymbol{\alpha_i}}$, siendo éstos los valores normalizados de $\boldsymbol{CD_i}$ y $\boldsymbol{\alpha_i}$ respectivamente, se clasifica el píxel en una de estas tres categorías:

$$\boldsymbol{C_i} = \begin{cases} Foreground: \widehat{\boldsymbol{CD_i}} > \boldsymbol{\tau_{CD}}\ o\ \widehat{\boldsymbol{\alpha_i}} < \boldsymbol{\tau_{alo}} \\ Background: \widehat{\boldsymbol{\alpha_i}} < \boldsymbol{\tau_{\alpha 1}}\ y\ \widehat{\boldsymbol{\alpha_i}} > \boldsymbol{\tau_{\alpha 2}} \\ Sombra: \widehat{\boldsymbol{\alpha_i}} < 0 \end{cases}$$

Donde $\boldsymbol{\tau_{CD}}$, $\boldsymbol{\tau_{alo}}$, $\boldsymbol{\tau_{\alpha 1}}$y $\boldsymbol{\tau_{\alpha 2}}$ son umbrales de decisión determinados empíricamente.

[7] **SNP** Método estadístico no paramétrico. (Del inglés *Statistical Non-Parametric approach*)

- **MÉTODO ESTADÍSTICO PARAMÉTRICO**

Para analizar el comportamiento de los métodos estadísticos paramétricos (*SP*[8]), se va a estudiar el algoritmo descrito en (23), utilizado en escenas de tráfico.

Este algoritmo utiliza dos fuentes de información: local (basado en la información del píxel a analizar) y espacial (basado en el estudio de la similitud del valor de los píxeles adyacentes en regiones que presentan sombras).

En primer lugar, y al igual que en el método *SNP*, se utilizan los primeros $\boldsymbol{N}$ *frames* para calcular, para cada píxel, las medias $\boldsymbol{E_i^B} = [\boldsymbol{\mu_{R_i}^B}, \boldsymbol{\mu_{G_i}^B}, \boldsymbol{\mu_{B_i}^B}]$ y varianzas $\boldsymbol{s_i^B} = [\boldsymbol{\sigma_{R_i}^B}, \boldsymbol{\sigma_{G_i}^B}, \boldsymbol{\sigma_{B_i}^B}]$, de los distintos canales de color (de nuevo, *RGB*) del *Background*.

A continuación, y asumiendo que $\boldsymbol{v} = [\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}]^{\boldsymbol{T}}$ es el valor de un píxel no sombreado, se realiza una transformación lineal $\overline{\boldsymbol{v}} = \boldsymbol{D}\boldsymbol{v}$ (donde $\boldsymbol{D}$ es una matriz diagonal obtenida de manera empírica, relacionada con la reflectancia del *Background*) para determinar cuál será el valor del píxel cuando se encuentre en una región sombreada.

$$\boldsymbol{D} = \begin{vmatrix} \boldsymbol{d_R} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{d_G} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d_B} \end{vmatrix}$$

Si el *Background* no es plano, o lo que es lo mismo, presenta diferentes texturas, deberán considerarse tantas matrices $\boldsymbol{D}$ como sean necesarias para representar el comportamiento de éstas.

Por tanto, y dadas las medias y varianzas de los distintos canales de color para un píxel en concreto, se puede considerar que los valores de dicho píxel, cuando éste se encuentre en una región sombreada, serán los siguientes:

$$\boldsymbol{\mu_{R_i}^S} = \boldsymbol{\mu_{R_i}^B d_R} \qquad \boldsymbol{\sigma_{R_i}^S} = \boldsymbol{\sigma_{R_i}^B d_R}$$

$$\boldsymbol{\mu_{G_i}^S} = \boldsymbol{\mu_{G_i}^B d_R} \qquad \boldsymbol{\sigma_{G_i}^S} = \boldsymbol{\sigma_{G_i}^B d_R}$$

$$\boldsymbol{\mu_{B_i}^S} = \boldsymbol{\mu_{B_i}^B d_R} \qquad \boldsymbol{\sigma_{B_i}^S} = \boldsymbol{\sigma_{B_i}^B d_R}$$

Finalmente, se estiman las probabilidades del píxel de pertenecer al *Background*, *Foreground*, o de ser clasificados como sombras. Esta etapa se ve mejorada mediante el análisis de los píxeles adyacentes, favoreciendo la correcta clasificación de las diferentes regiones de píxeles en sus grupos correspondientes.

Este algoritmo de detección de sombras basado en el método estadístico paramétrico presenta como principal inconveniente el proceso de estimación de parámetros.

[8] **SP** Método estadístico paramétrico. (Del inglés *Statistical Parametric approach*)

En el propio estudio, proponen realizar una segmentación manual de un cierto número de *frames* para determinar tanto los datos estadísticos necesarios, como los valores de la matriz (o matrices) $\boldsymbol{D}$.

- **MÉTODO DETERMINÍSTICO NO BASADO EN EL MODELO**

Este método, denominado *DNM*[9], se basa en la aplicación de umbrales sobre la reducción de intensidad y cromaticidad para evaluar si un determinado píxel se encuentra en una región sombreada o no.

En (24) se propone la aplicación de este método en el espacio de color *HSV*. El motivo principal por el cual se inclinan por este espacio de color radica en que el comportamiento de dicho espacio se corresponde estrechamente con la percepción humana del color (25), y ofrece una mayor precisión en la detección de sombras, tal y como se verá más adelante.

El proceso de decisión que se sigue para detectar si un determinado píxel se encuentra en una región sombreada $\boldsymbol{S_k(x,y)}$ es el siguiente:

$$\boldsymbol{S_k(x,y)} = \begin{cases} \boldsymbol{1, si\ \alpha \leq \dfrac{I_k^V(x,y)}{B_k^V(x,y)} \leq \beta} \\ \boldsymbol{\wedge \left(I_k^S(x,y) - B_k^S(x,y)\right) < \tau_S} \\ \boldsymbol{\wedge \left|I_k^H(x,y) - B_k^H(x,y)\right| < \tau_H} \\ \\ \boldsymbol{0, resto} \end{cases}$$

Donde $\boldsymbol{I_k(x,y)}$ y $\boldsymbol{B_k(x,y)}$ son los valores del píxel localizados en $\boldsymbol{(x,y)}$ en la imagen actual k-ésima y el *Background* correspondiente en el *frame* k-ésimo, respectivamente; y $\boldsymbol{\alpha, \beta, \tau_S}$ y $\boldsymbol{\tau_H}$ son los umbrales de decisión utilizados.

Otros métodos se centran en la discriminación entre los bordes de las sombras y los bordes o límites de los objetos (26). Sin embargo, en diversas escenas se hace complicado extraer regiones de movimiento conectadas en su totalidad a partir del mapa de bordes resultante, que en ocasiones es irregular. Por otra parte, escenarios más complejos que contengan diversos objetos de pequeño tamaño presentan desventajas para estos modelos.

Los autores apuntan que su algoritmo es robusto cuando los bordes de las sombras están definidos de una manera clara, pero ofrece un peor comportamiento en escenas con sombras superpuestas o difusas con bordes poco definidos. Además, y desde un punto de vista práctico, la complejidad computacional de estos algoritmos debe reducirse para poder ofrecer un análisis en tiempo real, según apuntan ellos mismos.

[9] **DNM** Método determinístico no basado en el modelo.
(Del inglés *Deterministic Non-based Model*)

- **RESULTADOS COMPARATIVOS**

En (21) se da una visión general acerca de la detección de sombras en entornos de video – seguridad siguiendo esta clasificación, y se realiza una evaluación de las distintas propuestas mencionadas.

Para ello, segmentan manualmente distintas secuencias de imágenes y clasifican los píxeles dentro de una de estas tres categorías: *Foreground*, *Background* y sombras. Los resultados pueden contemplarse en la Tabla 2-1 (21):

		SNP	SP	DNM
Escenario 1	η%	81,59%	59,59%	69,72%
	ξ%	63,76%	84,70%	76,93%
	%	**52,02%**	**50,47%**	**53,64%**
Escenario 2	η%	51,20%	46,93%	54,07%
	ξ%	78,92%	91,49%	78,93%
	%	**40,41%**	**42,94%**	**42,68%**
Escenario 3	η%	80,58%	72,43%	82,87%
	ξ%	69,37%	74,08%	86,65%
	%	**55,90%**	**53,66%**	**71,81%**
Escenario 4	η%	84,03%	64,85%	76,26%
	ξ%	92,35%	95,39%	89,87%
	%	**77,60%**	**61,86%**	**68,53%**
Escenario 5	η%	72,82%	76,27%	78,61%
	ξ%	88,90%	90,74%	90,29%
	%	**64,74%**	**69,21%**	**70,98%**
		SNP	SP	DNM
Resultados Evaluación	%	**58,13%**	**55,63%**	**61,53%**

Tabla 2-1: Evaluación de los métodos de detección de sombras SNP, SP y DNM.

$\boldsymbol{\eta}$ se considera la precisión en la detección de sombras y $\boldsymbol{\xi}$ la precisión en la discriminación de sombras:

$$\eta = \frac{TP_S}{TP_S + FN_S}$$

$$\xi = \frac{\overline{TP_F}}{TP_F + FN_F}$$

Donde $\boldsymbol{TP_S}$ se considera el número de píxeles sombreados en movimiento correctamente detectados (*shadow true positives*), $\boldsymbol{FN_S}$ el número de píxeles sombreados en movimiento no detectados (*shadow false negatives*).

$\boldsymbol{TP_F}$ se considera el número de píxeles pertenecientes a un objeto en movimiento correctamente detectados (*foreground true positives*), $\boldsymbol{FN_F}$ el número de píxeles pertenecientes a un objeto en movimiento no detectados (*foreground false negatives*) y $\overline{\boldsymbol{TP_F}}$ el número de píxeles pertenecientes a objetos en movimiento menos el número de píxeles detectados y clasificados como sombras, pero pertenecientes al objeto en movimiento.

Los autores destacan el comportamiento de los métodos *SNP* y *DNM*, que funcionan en diferentes espacios de color, como *RGB* (23) y *HSV* (27). Otros estudios se enfocan en el espacio de color *L*u*v** (28) y en *L*a*b** (29).

2.2.4.2 PROPIEDADES DE LAS SOMBRAS EN DISTINTOS ESPACIOS DE COLOR

En este estudio, y tal como se contempla en (30), se analizan las propiedades de las sombras en diferentes espacios de color y reclasifican los diferentes algoritmos que se han desarrollado hasta ahora centrándose en tres grupos: constancia del color del píxel, constancia del color entre píxeles y consistencia temporal.

Si bien estos algoritmos se van a analizar por separado, cabe destacar que el uso de éstos es complementario y por tanto pueden utilizarse conjuntamente, mejorando los resultados finales, como se verá más adelante.

- **CONSTANCIA DEL COLOR DEL PÍXEL EN DISTINTOS ESPACIOS DE COLOR**

Generalmente, los algoritmos de detección y eliminación de sombras utilizan la información extraída en los espacios de color[10], y no sólo en la luminancia[11] de la escena, basándose en una hipótesis principal: la sombra proyectada sobre un píxel no modifica su información de color o cromaticidad (por ejemplo, la tonalidad – *H* – y saturación – *S* – en el espacio de color *HSV*).

En este apartado, nos centraremos en la decisión acerca de la selección de un espacio de color u otro en función de las características de la escena para obtener unos mejores resultados en la detección y eliminación de sombras en la secuencia de video.

[10] **Espacio de color** Un espacio de color define un modelo de composición del color. Por lo general, un espacio de color lo define una base de N vectores (donde N suele tomar valores de 1 a 4 para los principales espacios) cuya combinación lineal genera todo el espacio de color. Un color, por tanto, se especifica utilizando estos atributos o coordenadas, que representan su posición dentro de un espacio de color en particular.

[11] **Luminancia** La luminancia se define como la densidad angular y superficial de flujo luminoso que incide, atraviesa o emerge de una superficie siguiendo una dirección determinada. En el ámbito que nos ocupa, podemos considerarla como la componente que codifica la información de luminosidad de la imagen, y de una forma no rigurosa, su equivalente psicológico sería el brillo de dicha imagen.

La selección del espacio de color debe realizarse siguiendo un criterio definido, y éste debe ser calculado en un único espacio de color. Sin embargo, muy pocos estudios contemplan la importancia de elegir un espacio de color u otro en función de la escena de la cual queremos extraer información, así como qué espacio de color es más efectivo en cada caso.

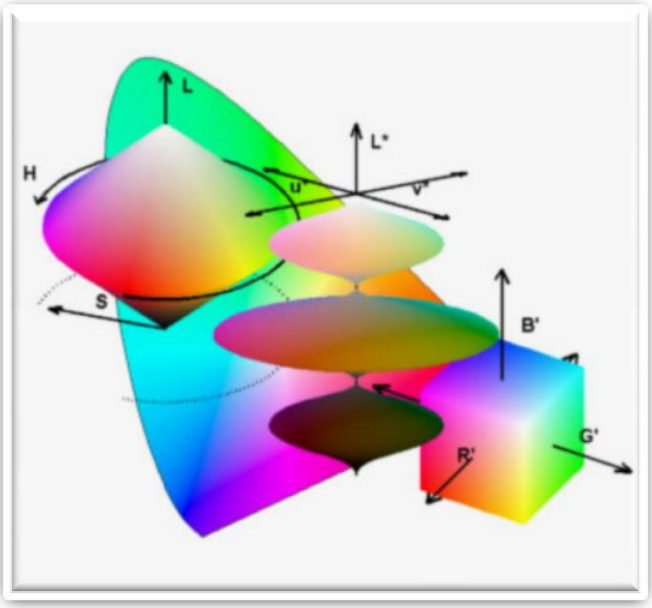

Figura 2-11: Representación de diferentes espacios de color.

En (31), pese a destacar la importancia de la selección de uno u otro espacio de color en función de la escena, finalmente presentan *L*u*v** como el mejor. No obstante, no se muestran resultados concluyentes ni, de una forma clara, el método que se ha seguido.

En (20) se realiza una comparativa del comportamiento de las sombras en diferentes espacios de color. Para ello, supone tres premisas fundamentales:

- Existen diferentes tipos de sombras debido a la diversidad de escenas contempladas, así como las diferentes propiedades de las sombras.

- Las propiedades de las sombras cambian en función del espacio de color con el que se observen, lo que implica diferentes tasas de acierto en la detección de éstas.

- Puede existir un espacio de color óptimo para un tipo de escena concreta, en la que las propiedades de las sombras sean más explotables.

Tomando como referencia estas premisas, se trata de averiguar qué espacio de color se comporta mejor en una serie de escenas donde las propiedades de las sombras son diferentes (ver **¡Error! No se encuentra el origen de la referencia.**). Los

spacios de color analizados son los siguientes: *RGB*, *HSV*, *YCbCr*, *XYZ*, *L*a*b**, *c1c2c3*, *l1l2l3* y *rgb normalizado*.

Como conclusión, y resumido en la Tabla 2-2, *YCbCr*, *L*a*b*, *c1c2c3* y *l1l2l3* son espacios de color donde se pueden explotar mejor las características de la sombra invisible. Por otro lado, los espacios de color *HSV*, *c1c2c3*, *l1l2l3* y *rgb normalizado* serán espacios más aptos para detectar sombras visibles.

	Sombra Visible	Sombra Invisible
RGB	×	×
H	✓	✓
S	✓	×
V	×	×
Y	×	×
Cb	×	✓
Cr	×	✓
XYZ	×	×
*L**	×	×
*a**	×	✓
*b**	×	✓
c1c2c3	✓	×
l1l2l3	✓	✓
rgb normalizado	✓	×

Tabla 2-2: Invariancia de las coordenadas en distintos espacios de color

Una vez clasificada la escena, y basándose en una de las características de las sombras (la luminancia de aquellas regiones de la escena cubiertos por una sombra será menor que en aquellas regiones similares de la escena donde no lo esté), realizan una selección del espacio de color óptimo (ver ANEXO C:).

A continuación, el algoritmo de detección de sombras utilizado es, desde un punto de vista genérico, y utilizado por la mayoría de los autores, el siguiente:

Considerando que:

$$\boldsymbol{C_1 \epsilon \{V, Y, L*, c1, l1, r\}}$$

$$\boldsymbol{C_2 \epsilon \{H, Cb, a*, c2, l2, g\}}$$

$$\boldsymbol{C_3 \epsilon \{S, Cr, b*, c3, l3, b\}}$$

Donde la región de movimiento es detectada en el canal $\boldsymbol{C_1}$, y las diferencias de gradiente se obtienen en los canales $\boldsymbol{C_2}$ y $\boldsymbol{C_3}$ respectivamente.

El píxel cuya diferencia de gradiente sea menor que un umbral definido, será considerado como píxel candidato a ser sombra.

$$M_1 = \begin{cases} 1, \left|I_{C_1}^k(x,y) - B_{C_1}^k(x,y)\right| > D_{th} \\ 0, resto \end{cases}$$

$$M_2 = \begin{cases} 1, \left|G_{C_2}^k(x,y) - G_{BC_2}^k(x,y)\right| > G_{th1} \\ 0, resto \end{cases}$$

$$M_3 = \begin{cases} 1, \left|G_{C_3}^k(x,y) - G_{BC_3}^k(x,y)\right| > G_{th2} \\ 0, resto \end{cases}$$

$$M = M_1 \wedge (M_2 \vee M_3)$$

Donde M_1 es la máscara de la región en movimiento, M_2 y M_3 son las máscaras de sombras, M la máscara final, $I_{C_1}^k$ y $B_{C_1}^k$ es la imagen actual (*frame k-ésimo*) y su *Background* en el canal C_1, $G_{C_2}^k$ y $G_{C_3}^k$ son las funciones gradiente en los canales C_2 y C_3, $G_{BC_2}^k$ y $G_{BC_3}^k$ son las funciones gradiente del *Background* en los canales C_2 y C_3 y D_{th}, G_{th2} y G_{th2} son parámetros previamente definidos. M_1 contiene a los objetos detectados en la etapa de detección de movimiento. M_2 y M_3 aquellas regiones no sombreadas. Por tanto, M contendrá el objeto en movimiento, con las sombras eliminadas.

Según las pruebas realizadas en dicho estudio, sacan las siguientes conclusiones:

- Espacio de color *RGB*: Ya que el espacio de color RGB es sensible a la iluminación, no es un espacio adecuado para detectar y eliminar sombras.

- Espacio de color *HSV*: También es sensible a la iluminación. Sin embargo, y puesto que la luminancia está separada de la crominancia, este espacio se ve menos afectado por las sombras visibles. Por otro lado, y aunque también las sombras invisibles pueden ser eliminadas, muchos de los píxeles de los objetos en movimiento no se incluyen en la máscara final, debido principalmente a que las sombras invisibles tienen unos valores similares a la escena. Desde un punto de vista cuantitativo, el espacio de color *HSV* es adecuado para eliminar sombras visibles.

- Espacio de color *YCbCr*: No es sensible a la iluminación, ya que al igual que el espacio de color *HSV*, separa luminancia y crominancia en distintos canales. Desde un punto de vista cuantitativo, este espacio es apto para eliminar sombras invisibles.

- Espacio de color *L*a*b**: Es el espacio de color más completo y usado convencionalmente para describir todos los colores visibles al ojo humano. De hecho, según nuestra percepción, no podemos detectar diferencias entre la sombra invisible y la escena, tal y como ocurre en este espacio de color. Desde un punto de vista cuantitativo, este espacio de color es apto para eliminar sombras invisibles.

- Espacios de color *l1l2l3* y *c1c2c3*: Pese a que tanto las sombras visibles como las invisibles son eliminadas en este espacio de color, muchos de los píxeles pertenecientes al objeto en movimiento desaparecen, por lo que estos espacios no son los más aptos para la detección y eliminación de sombras.

- Espacio de color *rgb normalizado*: Este espacio de color no es sensible a la iluminación, y desde un punto de vista cuantitativo, la sombra visible puede ser borrada debido a la invariancia a la escala.

Sin embargo, y debido a la naturaleza de las escenas (tanto *indoor* como *outdoor*) ante las que se enfrentan los sistemas de video – seguridad, con unas condiciones de iluminación complejas, no se ha definido en el desarrollo de estos algoritmos un modelo de sombras específico, y por tanto no se puede confirmar en qué situaciones no se cumple esta hipótesis en las cuales la cromaticidad de un píxel sí se ve modificada bajo la proyección de una sombra.

Ésta es la razón por la cual, pese a haber considerado diversos espacios de color capaces de detectar y eliminar sombras, en la práctica no todos son aptos para realizar dicha tarea. Por este motivo, se plantean otros métodos complementarios para obtener un mejor comportamiento.

- **CONSTANCIA DEL COLOR ENTRE PÍXELES**

Estos algoritmos están basados en el estudio de la similitud del valor de los píxeles adyacentes en regiones que presentan sombras. Y más específicamente, en la relación de la reducción de intensidad entre dichas sombras y el *Background*. También se les denomina en la literatura algoritmos basados en la textura del píxel.

Siendo $\boldsymbol{I}(\boldsymbol{x},\boldsymbol{y})$ la intensidad del pixel localizado en $(\boldsymbol{x},\boldsymbol{y})$ en la imagen actual, $\boldsymbol{E}(\boldsymbol{x},\boldsymbol{y})$ la irradiancia proyectada en $(\boldsymbol{x},\boldsymbol{y})$, y $\boldsymbol{\rho}(\boldsymbol{x},\boldsymbol{y})$ el coeficiente de reflexión del mismo punto; y de manera análoga, $\boldsymbol{I}'(\boldsymbol{x},\boldsymbol{y})$, $\boldsymbol{E}'(\boldsymbol{x},\boldsymbol{y})$ y $\boldsymbol{\rho}'(\boldsymbol{x},\boldsymbol{y})$ los parámetros asociados al *Background*.

Asumiendo que píxeles vecinos tendrán la misma irradiancia, y que un píxel sombreado con respecto al *Background* reducirá su irradiancia, pero no su reflectancia, tenemos que:

$$\boldsymbol{E}(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{E}(\boldsymbol{x}+\boldsymbol{1},\boldsymbol{y})$$

$$\boldsymbol{E}'(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{E}'(\boldsymbol{x}+\boldsymbol{1},\boldsymbol{y})$$

$$\boldsymbol{\rho}(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{\rho}'(\boldsymbol{x},\boldsymbol{y})$$

$$\boldsymbol{\rho}(\boldsymbol{x}+\boldsymbol{1},\boldsymbol{y}) = \boldsymbol{\rho}'(\boldsymbol{x}+\boldsymbol{1},\boldsymbol{y})$$

Por tanto, la relación entre la intensidad de un píxel localizado en $(\boldsymbol{x}, \boldsymbol{y})$ con respecto a su píxel vecino localizado en $(\boldsymbol{x}+\mathbf{1}, \boldsymbol{y})$ en la imagen actual podrá calcularse de la siguiente manera:

$$\frac{\boldsymbol{I}(\boldsymbol{x},\boldsymbol{y})}{\boldsymbol{I}(\boldsymbol{x}+\mathbf{1},\boldsymbol{y})} = \frac{\boldsymbol{E}(\boldsymbol{x},\boldsymbol{y})\boldsymbol{\rho}(\boldsymbol{x},\boldsymbol{y})}{\boldsymbol{E}(\boldsymbol{x}+\mathbf{1},\boldsymbol{y})\boldsymbol{\rho}(\boldsymbol{x}+\mathbf{1},\boldsymbol{y})} = \frac{\boldsymbol{\rho}(\boldsymbol{x},\boldsymbol{y})}{\boldsymbol{\rho}(\boldsymbol{x}+\mathbf{1},\boldsymbol{y})} = \frac{\boldsymbol{\rho}'(\boldsymbol{x},\boldsymbol{y})}{\boldsymbol{\rho}'(\boldsymbol{x}+\mathbf{1},\boldsymbol{y})}$$
$$= \frac{\boldsymbol{E}'(\boldsymbol{x},\boldsymbol{y})\boldsymbol{\rho}'(\boldsymbol{x},\boldsymbol{y})}{\boldsymbol{E}'(\boldsymbol{x}+\mathbf{1},\boldsymbol{y})\boldsymbol{\rho}'(\boldsymbol{x}+\mathbf{1},\boldsymbol{y})} = \frac{\boldsymbol{I}'(\boldsymbol{x},\boldsymbol{y})}{\boldsymbol{I}'(\boldsymbol{x}+\mathbf{1},\boldsymbol{y})}$$

Por tanto, si un píxel se encuentra en una región sombreada, entonces la relación de intensidad entre píxeles sombreados vecinos en la imagen actual debe ser la misma que dicha relación en la imagen *Background*. Esta propiedad de la sombra se llama constancia del color entre píxeles o textura del píxel.

Siguiendo este planteamiento, y para acelerar el examen de esta igualdad, se utilizan en (30) dos mapas de relación logarítmicos, $\boldsymbol{d}(\boldsymbol{x},\boldsymbol{y})$ para la imagen actual y $\boldsymbol{d}'(\boldsymbol{x},\boldsymbol{y})$ para el *Background*, y que pueden realizarse mediante la convolución de la imagen logaritmo con la máscara de la derivada de primer orden horizontal (*horizontal first – order derivative mask*).

$$\begin{cases} \boldsymbol{d}(\boldsymbol{x},\boldsymbol{y}) = \ln\dfrac{\boldsymbol{I}(\boldsymbol{x},\boldsymbol{y})}{\boldsymbol{I}(\boldsymbol{x}+\mathbf{1},\boldsymbol{y})} = \ln\boldsymbol{I}(\boldsymbol{x},\boldsymbol{y}) - \ln\boldsymbol{I}(\boldsymbol{x}+\mathbf{1},\boldsymbol{y}) \\ \boldsymbol{d}'(\boldsymbol{x},\boldsymbol{y}) = \ln\dfrac{\boldsymbol{I}'(\boldsymbol{x},\boldsymbol{y})}{\boldsymbol{I}'(\boldsymbol{x}+\mathbf{1},\boldsymbol{y})} = \ln\boldsymbol{I}'(\boldsymbol{x},\boldsymbol{y}) - \ln\boldsymbol{I}'(\boldsymbol{x}+\mathbf{1},\boldsymbol{y}) \end{cases}$$

Por tanto, la comparación entre $\boldsymbol{d}(\boldsymbol{x},\boldsymbol{y})$ y $\boldsymbol{d}'(\boldsymbol{x},\boldsymbol{y})$ puede ser aplicada para determinar si un píxel pertenece o no a una región sombreada. No obstante, este método se encuentra no exento de falsos positivos debido a diversos factores. Debido a esto, y para solventar este problema, se suele realizar un post-proceso basado en la consistencia espacial para detectar y eliminar estos píxeles mal clasificados.

La medida para calcular el error dado en la clasificación del píxel como sobra puede realizarse sumando las diferentes absolutas entre $\boldsymbol{d}$ y $\boldsymbol{d}'$ sobre todos los píxeles en una pequeña región centrada en $(\boldsymbol{x},\boldsymbol{y})$.

Además, y dado que trabajamos con tres canales (por ejemplo: *R*,*G*,*B*), el error calculado $D_i(x,y)$ para cada canal puede ser calculado individualmente, y sumados posteriormente:

$$\boldsymbol{\Psi}(\boldsymbol{x},\boldsymbol{y}) = \sum_{\boldsymbol{i}\,\epsilon\,(\boldsymbol{R},\boldsymbol{G},\boldsymbol{B})} \boldsymbol{D_i}(\boldsymbol{x},\boldsymbol{y}) \qquad \boldsymbol{D}(\boldsymbol{x},\boldsymbol{y}) = \sum_{(\boldsymbol{i},\boldsymbol{j})\epsilon\, \boldsymbol{w}(\boldsymbol{x},\boldsymbol{y})} |\boldsymbol{d}(\boldsymbol{i},\boldsymbol{j}) - \boldsymbol{d}'(\boldsymbol{i},\boldsymbol{j})|$$

Considerando dichos píxeles adyacentes situados en una pequeña región o ventana, y combinando la información que nos ofrecen los distintos canales, la discriminación de sombras es más robusta y estable.

En este estudio, los autores concluyen que la utilización de ambas características mejora el comportamiento en la detección y eliminación de sombras, gracias a la información adicional que ofrecen los píxeles adyacentes tanto de la imagen actual, como del *Background*.

No obstante, y aunque estos algoritmos basados en la consistencia del color entre píxeles son menos dependientes a la iluminación de la escena, se vuelven claramente dependientes de la textura del *Background* ofreciendo a menudo un peor rendimiento en aquellas situaciones en las que tanto el *Background* como el objeto en movimiento poseen texturas similares.

Resaltan además la necesidad de implementar un algoritmo sencillo, con una ventana pequeña (en el caso que nos ocupa: 1x2 píxeles) para conseguir un análisis en tiempo real. Existen, por tanto, otros algoritmos más complejos (32) utilizados en el análisis de la textura del píxel, pero que no son válidos para nuestro propósito.

- **CONSISTENCIA TEMPORAL**

Si los objetos en movimiento se desplazan lentamente a lo largo de la escena, la consistencia temporal entre *frame* y *frame* puede aportar información añadida para determinar si un píxel se encuentra en una región sombreada o no, ya que se puede asumir que un píxel permanecerá sombreado en el siguiente *frame*.

Sin embargo, dicha asunción no siempre será cierta, ya que dicha consistencia temporal depende extremadamente tanto de la velocidad del objeto, como de la tasa de cuadros de la cámara que capte la escena.

En (30) se ayudan de este método para prevenir la incorrecta clasificación de los píxeles debido al ruido, una vez analizadas tanto la consistencia del color en el píxel, como la consistencia del color entre píxeles. Para ello, se basan en la siguiente ecuación:

$$\boldsymbol{\Theta_t(x,y)} = \boldsymbol{\alpha}\frac{\boldsymbol{\sigma_{\Lambda}^2(x,y)\Psi_t(x,y)} + \boldsymbol{\sigma_{\Psi}^2(x,y)\Lambda_t(x,y)}}{\boldsymbol{\sigma_{\Psi}^2(x,y)} + \boldsymbol{\sigma_{\Lambda}^2(x,y)}} + \boldsymbol{(1-\alpha)\Theta_{t-1}(x,y)}$$

Donde $\boldsymbol{\Psi_t}$ y $\boldsymbol{\Lambda_t}$ son las máscaras (en el *frame* actual) que contienen los píxeles sombreados detectados según los métodos de constancia del color entre píxeles y en el propio píxel respectivamente y $\boldsymbol{\alpha}$ una constante que controla la importancia que se le da al *frame* actual con respecto al anterior (mayor en escenas en los que predomina el movimiento rápido de los objetos – vehículos – y menor en aquellas en las que se desplacen más lentamente – humanos –). Además, se contemplan dos nuevos parámetros: $\boldsymbol{\sigma_{\Psi}^2(x,y)}$ y $\boldsymbol{\sigma_{\Lambda}^2(x,y)}$, que se corresponden con las varianzas temporales de ambos métodos para cada píxel.

3. SISTEMA DiVA

3.1 INTRODUCCIÓN

El objetivo de este PFC se centra en dos aspectos: la creación de un sistema distribuido de video – seguridad y la detección y eliminación de sombras en las secuencias de video captadas.

Para la consecución de este proyecto, se parte de un sistema en desarrollo por el Grupo de Tratamiento e Interpretación de Vídeo de la UAM (VPU-UAM), denominado *DiVA*[12] (33). Dicho sistema, implementado en *C++*, tiene como objetivo principal el análisis de diferentes fuentes de video (por ejemplo: cámaras) en un entorno distribuido de manera colaborativa.

A partir de la infraestructura existente, se completará el diseño de los módulos existentes y se propondrán nuevos módulos y algoritmos. Esta tarea de diseño comprende tres puntos de vista:

- Actualmente, el sistema obtiene datos de cámaras *USB*[13], *Firewire 1394*[14] y archivos de vídeo (sin comprimir) en disco. Sin embargo, no existe soporte para cámaras IP, que por su sencillez, coste y escalabilidad se presentan como una alternativa con perspectivas de futuro. Se pretende contribuir en esta área con el diseño e implementación de un nuevo capturador de *frames* a través de cámaras IP.

- Se propone el diseño de un servidor que actúe como base de datos para guardar toda la información de interés generada en las capas de adquisición y procesado de la secuencia de video.

- Por último, se propone diseñar una interfaz para un nuevo módulo de análisis centrado en la detección y eliminación de sombras, implementando algunos de los algoritmos de análisis planteados en el Apartado 2.2.4. Este módulo contribuirá al desarrollo de la capa de análisis y procesado de video del sistema.

---

[12] **DiVA** Sistema de procesado distribuido de video (Del inglés *Distributed Video Analysis),* enmarcado dentro del proyecto *ATI@SHIVA:* Algoritmos de Tratamiento de Imágenes para Sistema Homogéneo e Inteligente de video – vigilancia avanzada. (http://www-vpu.ii.uam.es/~ati-shiva/).

[13] **USB** Estándar multiplataforma para entrada /salida de datos en serie a gran velocidad. (Del inglés *Universal Serial Bus*)

[14] **FIREWIRE 1394** Estándar multiplataforma para entrada /salida de datos en serie a gran velocidad.

3.2 ARQUITECTURA DEL SISTEMA DiVA

Este PFC parte del sistema de procesado distribuido de video *DiVA* cuya arquitectura se compone de cuatro capas funcionalmente independientes. Cada una de éstas tiene una tarea específica, y cooperan entre sí para conseguir el propósito final del sistema.

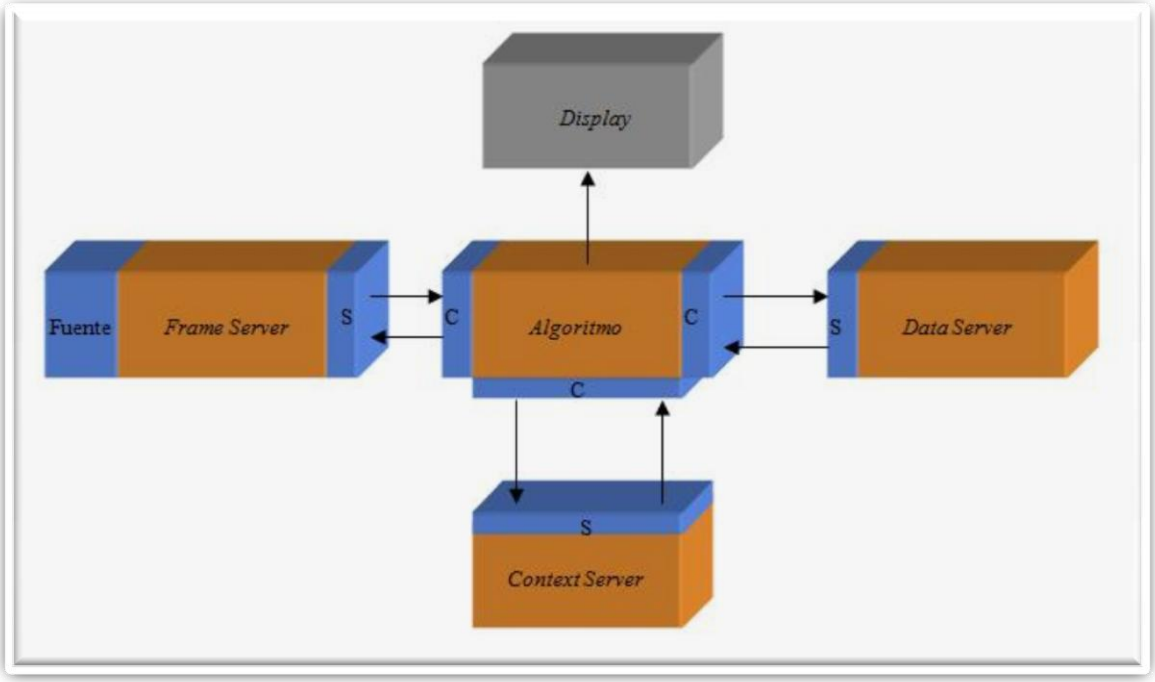

Figura 3-1: Esquema general del sistema *ATI@SHIVA.*

La capa de adquisición de imágenes (en adelante, *FrameServer*) proporciona una conexión entre las fuentes de cuadros (*frames*) y el resto de aplicaciones del sistema a través de su distribución por la red Ethernet del Grupo de Tratamiento e Interpretación de Video de la UAM (VPU-UAM).

El sistema *DiVA* consta además de una capa de procesado de video, denominado en la Figura 3-1 *Algoritmo*. Éste será el componente del sistema responsable de las diferentes tareas de análisis y procesado de video, organizado en módulos. Diferentes módulos pueden ser incluidos en esta capa para analizar la señal de video de manera distribuida.

Por último, cuenta con sendos módulos servidores de contenido (*DataServer y ContextServer*) aún no desarrollados, y que se encargarán de proveer de contenido previamente almacenado a los diferentes módulos de procesado de video.

Las diferentes capas pueden distribuirse de distintas maneras, permitiendo una configuración flexible del sistema final. Por tanto, y dependiendo de los requisitos de la aplicación, los módulos podrán interrelacionarse para ofrecer las funcionalidades requeridas.

3.2.1 CAPA DE ADQUISICIÓN DE VIDEO

Esta capa se encarga de configurar y obtener la información necesaria de las cámaras instaladas en el sistema (o videos en disco) para posteriormente obtener las secuencias de video capturadas por las cámaras y almacenarlas en un *buffer* (para su envío a los diferentes clientes que soliciten dichos datos). Las unidades que proporcionan la secuencia de video al sistema se conocen como *FrameServers*.

Las funciones principales del *FrameServer* son: capturar, almacenar y servir datos para el procesado de los algoritmos dentro del sistema.

3.2.1.1 ARQUITECTURA DEL *FRAMESERVER*

La aplicación *FrameServer* consta de tres sub – módulos:

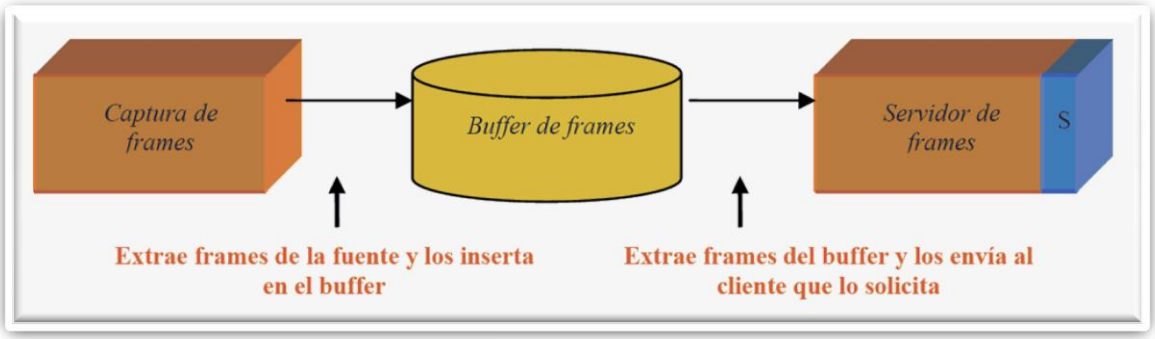

Figura 3-2: Estructura *FrameServer*.

- **CAPTURADOR DE *FRAMES***

Dicho módulo (en adelante, *FrameCapture*) se encarga de ir proporcionando *frames* al *FrameServer*. Para capturar *frames* existen varias aplicaciones en función del tipo de fuente de la que se quieran obtener (videos, distintos tipos de cámara,…).

- **BUFFER DE ALMACENAMIENTO**

Dicho módulo se encarga de almacenar los *frames* proporcionados por el capturador y se los proporcionará al servidor de *frames* cuando éste tenga que servir a un cliente.

- **SERVIDOR DE *FRAMES***

Dicho módulo se encarga de enviar los *frames* solicitados a los distintos clientes del sistema. Para realizar esta tarea, se basa en una arquitectura cliente – servidor. Se usa un modelo multi-hilo que permite servir contenido a varios usuarios a la vez. El protocolo de comunicación utilizado es TCP/IP.

3.2.1.2 INTERFAZ CON EL SISTEMA DiVA

Los datos que proporciona dicho módulo al sistema consisten en *frames* almacenados en el formato de imagen *DiVAImage*, basado a su vez en la clase *IplImage* de la librería de *OpenCV*.

Éste será, además, el formato de imagen utilizado en todo el sistema. Por lo tanto, si el capturador de *frames* obtiene dichas imágenes en cualquier otro formato será necesario realizar previamente una conversión de formato.

```
class DIVAImage
{
public:
        ///Default constructor
        DIVAImage();
        ///Constructor with basic parameters
        DIVAImage(long sizeX, long sizeY, int nChannel ,int Type);
        ///Constructor with a image in disk
        DIVAImage(char *filename,int iscolor=1);
        ///Class's Destructor
        virtual ~DIVAImage();
        ///Method to access to image pixels
        DIVASCALAR getPixel(int x, int y);
        ///Method to set one image pixel to a value
        void setPixel(int x, int y, DIVASCALAR valor);
        ///Set all pixels in the image to a reference value
        void setPixels(DIVASCALAR value);
        ///Set all pixels in the image to a reference matrix
        void setPixels(void *value);
        ///Gives direct access to pixels in the image
        void *getPixels();
        ///Set identification number
        void setId(long id);
        ///Set time stamp
        void setTimeStamp( long timeStamp);
        ///Set origin of pixels data
        void setVAlign(int value);
        ///Get the indentification number
        long getId();
        ///Get the time stamp
        int getTimeStamp();
        ///Get image width in pixels
        int getWidth();
        ///Get image height in pixels
        int getHeight();
        ///Get number of channels
        int getNChannels();
        ///Get type of image data
        int getDepth();
        ///Get image color model (RGB,HSV,...)
        char *getColorModel();
        ///Get origin of pixels data
        int getVAlign();
        ///Get Image data size in bytes
        int getDataSize();
        ///Flips image
        void flip(int flip_mode);
        ///Method to save a image in disk
        int saveImage(char *filename);
        ///Method to load a image from disk
        int loadImage(char *filename,int iscolor=1);
        ///Method to obtain a equal copy of the actual image
        DIVAImage *clone();
        ///Method to set the image pixels with the pixels content for the input image
        int copy(DIVAImage *image);
        ///Converts the image from RGB to Gray
        int RGB2gray();
};
```

Figura 3-3: Estructura pública de la clase *DiVAImage*.

Todas las clases de captura derivan de un interfaz genérico (llamado *DiVACapture*) que proporciona unos prototipos estándar para todas las clases de métodos para capturar *frames*.

```
class DIVACapture:public DIVAThread
{
public:

        ///Constructor
        DIVACapture(int bufferLength=DEFAULT_BUFFERLENGTH);
        ///Destructor
        ~DIVACapture();
        virtual char* getSourceId(){return NULL;};
        int process();
        int start();
        ///Inicialization
        virtual int init();
        /// Get current frame from source
        virtual int getCurrentFrame(DIVAImage* pImage);
        /// Get the next frame from source and advance to it
        virtual int getNextFrame(DIVAImage *nextFrame);
        /// Only capture next frame and no image is returned
        virtual int DIVACapture::captureNextFrame();
        DIVAImage*     getSampleFrame(){return _frame;};
        DIVAImageBuffer*              getImageBuffer(){return _pimageBuffer;};
        int writeInBuffer(DIVAImage* pImage);
        int setBufferLength(int bufferLength){_bufferLength=bufferLength; return
_bufferLength;};
        int setConsumeControl(int consumeControl){_consumeControl =
consumeControl;return _consumeControl;};
        int getConsumeControl(){return _consumeControl;};
};
```

Figura 3-4: Estructura pública de la clase *DiVACapture.*

De los presentes métodos incluídos en la clase *DiVACapture*, aquellos que afectan directamente al comportamiento del *FrameCapture*, y que posteriormente serán implementados en el nuevo módulo de adquisición de secuencias de video a través de cámaras IP, son los siguientes:

- `virtual int init();`

Este método inicializa el capturador de *frames*. Es decir, reserva todos los recursos necesarios para que se produzca la captura de datos.

En el caso de que la inicialización dependa de algunos parámetros iniciales, el método `init()` debe sobreescribirse con los distintos parámetros necesarios para su puesta en funcionamiento `init(param1,param2,…)`.

- `virtual int getCurrentFrame(DiVAImage* pImage);`

Este método devuelve el *frame* actual almacenado en la fuente. Dicho *frame* se pasará a la estructura *DiVAImage* y se copia todo su contenido en el objeto que se pasa como parámetro al método.

Dicho método debe devolver -1 en caso de error (ya sea porque no esté inicializado, no haya contenido que extraer,…) o 0 en caso de éxito.

- `virtual int getNextFrame(DiVAImage *nextFrame);`

Este método devuelve el *frame* actual del origen de datos. Es decir, se captura una nueva imagen, es pasada al formato *DiVAImage* y se guarda en el objeto pasado como parámetro de entrada **`nextFrame`**. La posición actual en los *frames* de la fuente cambia al siguiente *frame*.

Adicionalmente, también debe guardar dicho *frame* en la memoria del capturador (para posibles peticiones de *frame* con **`getCurrentFrame(…)`**).

Dicho método debe retornar un valor -1 en caso de error (ya sea porque no esté inicializado, no haya contenido que extraer,…) o un valor 0 en caso de éxito.

- `virtual int DiVACapture::captureNextFrame();`

Este método debe realizar los mismos pasos que el método **`getNextFrame(DiVAImage *nextFrame)`** sin devolver ningún *frame* al cliente que llamó a la capturadora.

3.2.1.3 MODO DE FUNCIONAMIENTO DEL FRAMESERVER

La secuencia de funcionamiento del *FrameServer* es la siguiente:

- Arranca el *FrameServer* con los parámetros de configuración correspondientes (extraídos del fichero de configuración o por defecto).
  - Se inicializa el *buffer* de *frames*.
- El *FrameServer* lanza un hilo paralelo a su ejecución para capturar los *frames*. Las funciones de este hilo son:
  - Inicializar el capturador de *frames*.
  - Establecer los parámetros de configuración del capturador.
  - Comenzar la captura de *frames*:
    - La captura de *frames* se ejecuta en un bucle infinito que se realiza periódicamente (cada *frame*).
    - La única condición de terminación de la captura es que el *FrameServer* sea parado. En el caso de que una captura no sea posible, ésta se obvia y se continúa con el proceso de obtención de *frames*.
- El *FrameServer* lanza otro hilo para realizar el servicio de distribución de *frames*.

- Se establecen los parámetros básicos para el servicio distribuido de *frames* a toda la red. Posteriormente se pasa al modo de escucha de petición de *frames*.

- Por cada petición recibida, se lanza un hilo que sirve dicha petición. Este hilo busca el *frame* en el *buffer* y lo envía al cliente si el *frame* existe.

♦ Posteriormente, el *FrameServer* se queda bloqueado esperando a terminar. Dicha terminación sólo se produce por intervención directa del usuario y se realiza cuando éste pulsa la tecla ESC.

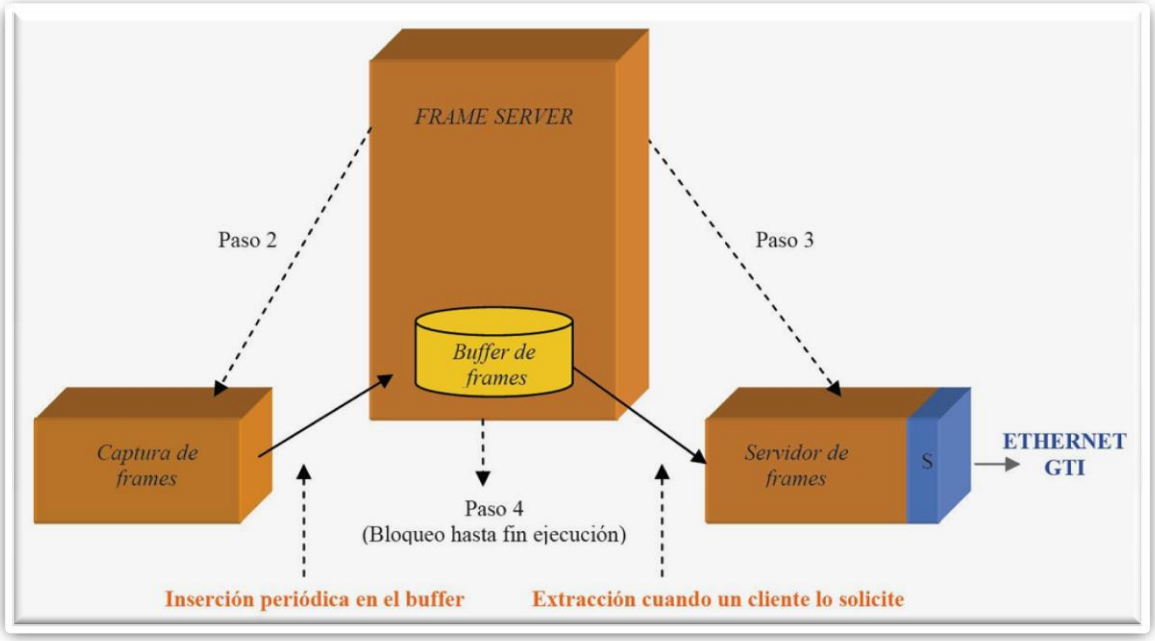

Figura 3-5: Esquema de funcionamiento del *FrameServer.*

3.2.2 CAPA DE ANÁLISIS Y PROCESADO DE VIDEO

Esta capa es la encargada de, a partir de la secuencia de video obtenida por los módulos de adquisición de imágenes o el servidor de datos, realizar las diferentes tareas de análisis y procesado de video, organizados en módulos (en adelante *Algoritmo*).

Para ello, existe una arquitectura genérica definida con las funcionalidades básicas para la captura, análisis y cooperación entre los distintos *Algoritmos* y capas del sistema.

3.2.2.1 ARQUITECTURA GENÉRICA DE LOS ALGORITMOS

Los algoritmos actúan como clientes del módulo de adquisición de imágenes (*FrameServer*) y de los módulos servidores de datos (*DataServer* y *ContextServer*), encargados de proporcionar y almacenar la información disponible y necesaria.

Éstos se ejecutan concurrentemente y de forma asíncrona, organizados en uno o varios procesadores, en función de la estructura general del sistema, caracterizada mediante tres aspectos: aplicación, topología de red y diseño de distribución.

3.2.2.2 INTERFAZ CON EL SISTEMA DiVA

DiVA proporciona una interfaz genérica para el desarrollo de los diferentes módulos de análisis, *DiVAAlgorithm*, que define cómo se interrelacionan los módulos de procesado de video con el sistema.

A continuación se describe la estructura pública de esta clase destacando aquellos métodos de interés:

```
class DIVAAlgorithm:public DIVAThread
{
        public:
        DIVAAlgorithm(char* frameServername, int portnumber,
                        BOOL fileDumping=TRUE,
                        BOOL display=TRUE,
                        char* dataServerName=NULL,
                        char* contentServerName=NULL,
                        BOOL dataServerDumping=FALSE,
                        BOOL contentServerDumping=FALSE,
                        int captureMode=CAPTURE_BUF);
        ~DIVAAlgorithm();
        char* getAlgorithmName(){return _algorithmName;};
        //Initialization
        virtual int init();
        //Execution Control
        int setEnd();
        int process();
        int start();
        int stop();
        //Connections
        int connect();
        int getFrame(DIVAImage** pImage,long idFrame);
        int receiveFrame(DIVAImage** pImage,long idFrame);
        //Data bases
        virtual int receiveData(void** data){return 0;};
        virtual int receiveContentData(void** data){return 0;};
        virtual int releaseData(void* data){return 0;};
        virtual int releaseContentData(void* data){return 0;};
        virtual int dumpData(){return 0;};
        virtual int dumpContentData(){return 0;};
        virtual int dumpResultsInFiles(){return 0;};
        //Image Processing
        virtual int processFrame(DIVAImage* pImage,void* pdata=NULL,void*
pContentData=NULL);
        //Displaying
        virtual int refreshDisplay(){return 0;};
        int             getCaptureMode(){return _captureMode;};
        BOOL    getfileDumping(){return _fileDumping;};
        char*   getdataServerName(){return _dataServerName;};
        char*   getcontentServerName(){return _contentServerName;};
        BOOL    getdataServerDumping(){return _dataServerDumping;};
        BOOL    getcontentServerDumping(){return _contentServerDumping;};
        BOOL    getdisplay(){return _display;};
        Cronometro* getCrono(){return _pCrono;};
        long getLastId(){return _lastId;};
        long setLastId(long lastId){_lastId = lastId;return _lastId;};
        char* getDumpingpathRoot(){return _dumpingpathRoot;};
        void setDisplay(bool display);
};
```

Figura 3-6: Estructura pública de la clase *DiVAAlgorithm*.

- `virtual int init();`

Este método inicializa el algoritmo. Es decir, reserva todos los recursos necesarios para su funcionamiento, establece una conexión con el servidor de contenido (identificado mediante los parámetros **`frameServername`** y **`portnumber`**) y se define el comportamiento del módulo mediante unos parámetros de configuración (por ejemplo: mostrar resultados en pantalla o almacenar resultados en disco, servidor de datos,…).

- `int start();`

Este método comienza el proceso iterativo de análisis de la secuencia de video, adquiriendo la información necesaria del servidor al que se encuentre conectado y procesándola.

Dicho método devuelve -1 en caso de error o un valor 0 en caso de éxito.

- `int stop();`

Este método detiene el proceso de ejecución. Devuelve -1 en caso de error o un valor 0 en caso de éxito.

3.2.2.3 MODO DE FUNCIONAMIENTO DE LOS ALGORITMOS

- **CASO 1:** *Integración de módulos de análisis en una sola clase DiVAAlgorithm.*

  - Se inicializan ambos módulos de procesado de la imagen, con los parámetros de configuración correspondientes y estableciendo además una conexión con el servidor de contenido (ya sea el módulo de adquisición de imágenes o el servidor de datos).

  - Comienza la petición iterativa de *frames* por parte del módulo principal, que procesa la imagen.

  - A continuación, el módulo secundario (en cascada) adquiere la información extraída por el primer módulo. Una vez realizado este paso, devuelve el control al módulo principal.

  - Optativamente, los resultados proporcionados por ambos módulos pueden ser mostrados en pantalla, almacenados en disco o enviados al servidor de datos (o incluso otro módulo), en función de los parámetros de configuración.

  - Posteriormente, dicho módulo continúa su proceso iterativo hasta que dejen de servirse *frames* desde el servidor o se produzca la intervención directa del usuario cuando éste pulse la tecla ESC.

- **CASO 2:** *Integración de módulos de análisis en varias clases DiVAAlgorithm.*

  Es decir, actuando de forma independiente y adquiriendo los datos del módulo de adquisición de imágenes o el módulo servidor de datos.

  - Se inicializa el módulo de procesado de la imagen, con los parámetros de configuración correspondientes y estableciendo una conexión con el servidor de datos.

  - Comienza la petición iterativa de la información necesaria por parte del módulo de procesado de la secuencia de video, procesándola y devolviendo la información extraída.

  - Optativamente, los resultados proporcionados por dicho módulo pueden ser mostrados en pantalla, almacenados en disco o enviados al servidor de datos (o incluso otro módulo), en función de los parámetros de configuración.

  - Posteriormente, dicho módulo continuará su proceso iterativo hasta que dejen de servirse *frames* desde el servidor o se produzca la intervención directa del usuario cuando éste pulse la tecla ESC.

3.2.3 MÓDULO SERVIDOR DE DATOS

Si bien en los apartados anteriores se ha descrito detalladamente la arquitectura de los módulos del sistema a la que debíamos adaptarnos, este módulo solamente se ha definido de manera funcional.

Entre las funcionalidades básicas que dicho módulo servidor de datos, enmarcado dentro de la arquitectura del sistema *DiVA*, debe ofrecer son las siguientes:

- Almacenar y proporcionar *frames* obtenidos desde un módulo de adquisición de imágenes.

- Almacenar y proporcionar máscaras obtenidas desde los distintos módulos de procesado de imágenes.

- Almacenar y proporcionar metadatos obtenidos desde los distintos módulos de procesado de imágenes que generen descripciones de contenido.

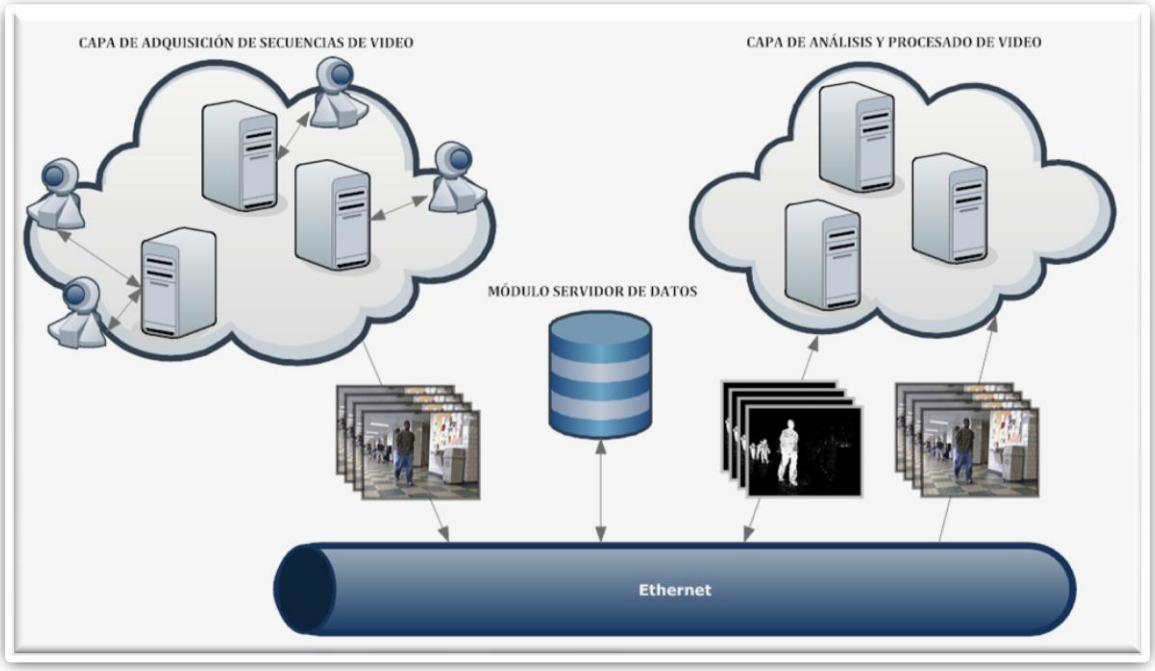

Figura 3-7: Esquema funcional del módulo servidor de datos.

Dado que manejar la ingente cantidad de información multimedia proporcionada por múltiples módulos de adquisición y procesado de imágenes es particularmente crítica, la correcta elección y organización del nivel semántico de información que es extraída y almacenada para una posterior operación es esencial.

3.3 MEJORAS PROPUESTAS

3.3.1 INSERCIÓN DE UNA NUEVA FUENTE DE VIDEO UTILIZANDO CAMARAS IP

Se propone ampliar el sistema *DiVA* con una nueva fuente de datos proveniente de cámaras IP, debido a que éstas aún no se encuentran desarrolladas en el sistema, y dado que por su sencillez, coste y escalabilidad se presentan como una alternativa con perspectivas de futuro. Para ello, se hace necesario diseñar y desarrollar un nuevo *FrameCapture* que establezca la comunicación con la cámara IP (*DiVACaptureIP*) y realizar una posterior integración en el *FrameServer* correspondiente.

Figura 3-8: Estructura *FrameServer*: Detalle *FrameCapture*.

Específicamente, las cámaras utilizadas serán el modelo SONY SNC-RZ50N, que se encuentran situadas en el Hall de la Escuela Politécnica Superior.

Figura 3-9: Hall EPS-UAM: Detalle Cámara IP.

El nuevo capturador de *frames DiVACaptureIP* tiene como propósito configurar las cámaras IP según los parámetros de configuración así como obtener la secuencia de video capturada y adaptarla al formato de imagen específico del sistema, e integrándose con el *buffer* y el servidor de *frames* se genera una nueva aplicación de adquisición de secuencias de video denominado *DiVAFrameServerIP*, que será la que se interrelacione con el resto del sistema *DiVA*.

En primer lugar, será necesario diseñar una interfaz con la que comunicarse con las cámaras IP, definiendo aquellos parámetros necesarios para la conexión y configuración de ésta (ya sea por defecto, o específicos en cada situación):

- Dirección IP y puerto de la cámara.
- Parámetros de autenticación.
- Tamaño de las imágenes.
- Tasa de cuadros por segundo.
- Formato de captura.

Una vez realizado este paso, se desarrollará un método mediante el cual adquirir los *frames* capturados por las cámaras IP, adaptándose el formato de la imagen (*BMP*[15], *JPEG*[16],…) al formato nativo del sistema *DiVA*: *DiVAImage.*

[15] **BMP** Formato de imagen sin compresión, que representa el mapa de bits (o matriz de píxeles) de la imagen (Del inglés *Bit Mapped Picture*).

[16] **JPEG** Formato de imagen con compresión con pérdida (Del inglés *Joint Photographic Experts Group*).

La integración con el sistema será sencilla puesto que el *FrameCapture* se comunica exclusivamente con el *buffer* perteneciente al *FrameServer*, que a su vez entrega el contenido al servidor de *frames*, ya desarrollado previamente en la clase genérica *DiVACapture* por el Grupo de Tratamiento e Interpretación de Video de la UAM (VPU-UAM).

Por tanto, de los diferentes métodos incluídos en la clase *DiVACapture*, los que resultan de especial interés y deberán ser implementados en la nueva clase derivada *(DiVACaptureIP)* son los siguientes:

- `virtual int init();`
- `virtual int getCurrentFrame(DiVAImage* currentImage);`
- `virtual int getNextFrame(DiVAImage *nextFrame);`
- `virtual int DiVACapture::captureNextFrame();`

3.3.2 INSERCIÓN DE UN MÓDULO DE GESTIÓN DE DATOS

Se propone completar el diseño inicial del módulo de gestión de datos (*DataServer*) del sistema *DiVA*, que centraliza toda la información de interés extraída y proporcionada por las capas de adquisición y procesado de video.

Por tanto, en primer lugar se diseñará la estructura de la base de datos con la que adquirir (y servir) el contenido. Posteriormente, se definirán los métodos necesarios para desarrollar un controlador de la base de datos e integrarse así con el sistema propuesto.

Por último, y una vez considerados los diversos formatos de compresión del contenido multimedia, el criterio de selección en cuanto al almacenamiento de la información se basará sencillamente en las características de la aplicación final del sistema de video – vigilancia.

3.3.2.1 ESTRUCTURA DE LA BASE DE DATOS

El diseño de la base de datos debe basarse en dos características del sistema: el formato del contenido que se desea almacenar (y proporcionar) y el diseño de los módulos con los que se va a interrelacionar.

- **BASADO EN EL FORMATO DEL CONTENIDO QUE SE DESEA ALMACENAR**

El sistema *DiVA* permite trabajar con tres tipos de contenido: *frames* obtenidos desde un módulo de adquisición de secuencias de video, máscaras y descriptores de contenidos obtenidos desde los distintos módulos de análisis y procesado de video.

Dichas secuencias de video que proporciona el módulo de adquisición consisten en *frames* almacenados en el formato de imagen *DiVAImage*. Por tanto, se creará una tabla denominada *images* cuyos campos serán, entre otros, aquellos parámetros definidos en dicho formato.

Con el mismo propósito, se crearán dos tablas más: *masks* y *descriptors*. Éstas contendrán las máscaras y descriptores generados en la capa de procesado de video, respectivamente, y estarán relacionados directamente con el *frame* del que se ha extraído la información.

También se creará otra tabla denominada *masktypes*, que contendrá la definición de los diferentes tipos de máscaras soportados por el sistema (*Background*, *Foreground*, máscaras de detección de movimiento, máscaras de detección de sombras,…).

- **BASADO EN EL DISEÑO DE LOS MÓDULOS**

Una vez definido el formato de las tablas donde se almacenará el contenido, queda aún pendiente definir las tablas de los diferentes contenedores que se interrelacionarán con el sistema. Esto es, los diferentes módulos involucrados con el servidor de contenido:

- Módulos de adquisición de secuencias de video:

De momento, se planteará únicamente la integración del módulo servidor de datos con el módulo de adquisición de secuencias de video a través de cámara IP que se pretende desarrollar, si bien posteriormente podrá ampliarse, si es necesario.

En primer lugar, se creará una tabla denominada *frameservers*, que contendrá todos los módulos de adquisición disponibles. Dichos módulos, al conectarse al *DataServer*, quedarán registrados en la tabla *framesessions*, cuyos campos serán aquellos parámetros necesarios para la conexión y configuración de ésta, ya enumerados en el Apartado 3.3.1, además de otros campos de interés.

- Módulos de análisis y procesado de secuencias de video:

Podemos diferenciar, en función del formato del contenido extraído, dos tipos de módulo claramente diferenciados: aquellos que devuelven máscaras y aquellos que devuelven descripciones de contenido (si bien un módulo de procesado de la secuencia de video puede devolver ambos).

Por tanto, se creará una tabla denominada *algorithms*, que contendrá todos los algoritmos disponibles. Éstos, al conectarse al *DataServer*, quedarán registrados en la tabla *algorithmsessions*, identificando la secuencia original sobre la que se ha realizado el procesado de video, además de otros campos de interés.

3.3.2.2 INTERFAZ CON EL SISTEMA DiVA

Este módulo servidor de datos debe ser capaz de gestionar el registro de los distintos *FrameServers* integrados en el sistema, así como la provisión (y posterior adquisición) de las secuencias de video capturadas por dichos módulos, en el formato nativo del sistema, *DiVAImage*. De manera análoga, actúa con los diferentes módulos enmarcados en la capa de procesado de video, gestionando el registro de los distintos *Algoritmos* presentes en el sistema, así como la provisión (y posterior adquisición) de la información extraída por éstos (máscaras y descripciones de contenido).

En este apartado se definirán los principales métodos necesarios que posteriormente serán desarrollados durante la implementación del módulo, definidos en una nueva clase genérica denominada *DiVADataClient*, que actuará como controlador de la base de datos, siendo ésta el nexo entre el servidor de contenido y el sistema *DiVA*.

- ```
  int DiVADataClient::init(char *database, char *host, char *user,
  char *pass);
  ```

Este método inicializa el módulo servidor de contenido. Es decir, reserva todos los recursos necesarios para su funcionamiento, establece una conexión con la base de datos y se define el comportamiento del módulo mediante unos parámetros de configuración.

- ```
  int DiVADataClient::openFrameServerSession(int fsID, int frameRate,
  char *srcIPAddress, int srcPort, int dstPort);
  ```

Este método registra la sesión del *FrameServer* y queda preparado para la recepción de la secuencia de video. Devuelve -1 en caso de error o el identificador de sesión *FrameServer* (*fsSessionID*) en caso de éxito.

- ```
  int DiVADataClient::closeFrameServerSession();
  ```

Este método cierra la sesión del *FrameServer*. Devuelve -1 en caso de error o 0 en caso de éxito.

- ```
  int DiVADataClient::sendImage(DiVAImage *frame);
  ```

Este método envía el *frame* capturado en la capa de adquisición de imágenes, en formato *DiVAImage*, quedando registrado y almacenado en la base de datos para su posterior análisis.

- ```
  int DiVADataClient::receiveImage(int frameID, DiVAImage *frame);
  ```

Este método adquiere el *frame* previamente capturado en la capa de adquisición de secuencias de video, en formato *DiVAImage*, registrado y almacenado en la base de datos.
  ```

- `int DiVADataClient::openAlgorithmSession(char *algName, char *srcIPAddress, int srcPort, char *algXML);`

Este método registra la sesión del *Algoritmo* y queda preparado para la recepción de la información extraída en la capa de procesado de video. Devuelve -1 en caso de error o el identificador de sesión *Algoritmo* (*algSessionID*) en caso de éxito.

- `int DiVADataClient::closeAlgorithmSession();`

Este método cierra la sesión del *Algoritmo*. Devuelve -1 en caso de error o 0 en caso de éxito.

- `int DiVADataClient::sendMask(DiVAImage *mask, int maskTypeID);`

Este método envía la máscara procesada en la capa de procesado de video, en formato *DiVAImage*, quedando registrada y almacenada en la base de datos para su posterior adquisición.

- `int DiVADataClient::receiveMask(int frameID, int maskTypeID, DiVAImage *mask);`

Este método adquiere la máscara previamente procesada en la capa de procesado de video, en formato *DiVAImage*, registrada y almacenada en la base de datos.

- `int DiVADataClient::sendDescriptor(int frameID, int maskTypeID, char *descriptorXML);`

Este método envía el descriptor procesado en la capa de procesado de video, quedando registrado y almacenado en la base de datos para su posterior adquisición.

- `int DiVADataClient::receiveDescriptor(int frameID, int maskTypeID, char *descriptorXML);`

Este método adquiere el descriptor previamente procesado en la capa de procesado de video, registrado y almacenado en la base de datos.

3.3.3 INSERCIÓN DE UN MÓDULO DE DETECCIÓN DE SOMBRAS

Por último, se propone ampliar el sistema *DiVA* añadiendo un módulo de análisis de sombras en los objetos previamente detectados. Dicho módulo tiene un propósito claro: detectar y eliminar las sombras que se encuentren en una secuencia de video.

En primer lugar, se seleccionarán e implementarán algunos de los algoritmos de detección y eliminación de sombras más representativos, previamente descritos en el Apartado 2.2.4. Estos algoritmos serán desarrollados en diferentes clases, denominadas *ShadowSuppressorXXX*, donde *XXX* identifica el algoritmo desarrollado.

Más adelante, se analizará el comportamiento de dichos algoritmos mediante una evaluación sobre distintas secuencias reales, con el propósito de estimar aquellos parámetros necesarios para conseguir un correcto funcionamiento durante el proceso de detección y eliminación de sombras.

Finalmente, se creará una nueva clase que encapsule la comunicación con el sistema *DiVA* y dé soporte a los diferentes algoritmos implementados, denominada *DiVAShadowSuppressor*. Esta clase será una extensión de la clase *DiVAAlgorithm*, ya descrita en el Apartado 3.2.2.2, que define cómo se interrelacionan los módulos de procesado de video con el sistema.

Dada la naturaleza de este módulo, el proceso se realizará basándose en los resultados obtenidos previamente por el algoritmo de estimación del movimiento (segmentación de fondo: *Background*, *Foreground*, y máscara de movimiento), y devolviendo una máscara donde se han eliminado aquellas regiones sombreadas detectadas durante el proceso.

Figura 3-10: De izquierda a derecha y de arriba abajo: imagen original, máscara de movimiento, máscara de sombras y máscara resultante.

Tanto el *Background* como el *Foreground* serán imágenes en color en el formato nativo del sistema *DiVA: DiVAImage*. Las máscaras de movimiento involucradas en el proceso serán imágenes binarias, representando cada píxel de dicha imagen con dos posibles valores: 0 – negro – y 255 – blanco –, también en el formato *DiVAImage*.

Este sistema está definido como un sistema distribuido en el que los distintos módulos de procesamiento se comportan como clientes, y tanto el módulo de adquisición de imágenes como el servidor de datos, como servidores. Sin embargo, y dada la naturaleza de este algoritmo, que se puede entender como un post – proceso de la detección de movimiento, se plantean dos alternativas:

- Diseñar el módulo de forma independiente y adquirir la información del servidor de datos (puesto que el módulo de adquisición de imágenes no provee toda la información necesaria).

- Embeber en el módulo de estimación del movimiento y ejecutar ambos en cascada.

La primera opción se hace necesaria, puesto que el módulo debe ser capaz de obtener información de dicho servidor. La segunda opción es más eficiente ya que se evita saturar la red. Como consecuencia, la implementación del módulo de detección y eliminación de sombras deberá contemplar ambas situaciones.

Por tanto, nuestra nueva clase *DiVAShadowSuppressor* deberá, al menos, contener los siguientes métodos con los que relacionarse con el sistema:

- `virtual int init();`
- `int start();`
- `int stop();`
- `virtual void processFrame(CvArr *bkg, CvArr *frg, CvArr *mask);`
- `virtual CvArr *getShadowMask();`

Además, y puesto que trabajaremos con imágenes en diferentes espacios de color, será necesario contemplar la posibilidad de realizar el desarrollo de uno o varios algoritmos de transformación entre espacios de color, cuando éstos no estén incluidos en la librería *OpenCV*[17].

[17] **OpenCV** Librería libre, multiplataforma, desarrollada en C++, de visión artificial.

4. MÓDULO CAPTURADOR DE DATOS DE CÁMARAS IP

4.1 INTRODUCCIÓN

En este capítulo se detalla el proceso de desarrollo del módulo *DiVACaptureIP* (capturador de *frames*), que realiza la conexión con la cámara IP, y su integración en la aplicación *DiVAFrameServerIP* (servidor de *frames*) como nuevo componente de la capa de adquisición de video del sistema *DiVA*. Para ello, se analizan los siguientes aspectos:

- Detalles técnicos y funcionales de la cámara IP.
- Comunicación con la cámara IP.
- Integración en el sistema *DiVA.*

4.2 DETALLES TÉCNICOS DE LA CÁMARA IP

El modelo de las cámaras que se han utilizado como fuente capturadora de secuencias de video es SONY SNC-RZ50N. Actualmente la infraestructura existente consta de tres cámaras, adquiridas por el Grupo de Tratamiento e Interpretación de Video de la UAM (VPU – UAM), y situadas en el Hall de la Escuela Politécnica Superior (desde la entrada principal a la izquierda, a la derecha y en frente).

Figura 4-1: Cámaras SONY SNC-RZ50N instaladas en la EPS – UAM.

Según las características descritas en el manual de usuario, las características más interesantes de la cámara son las siguientes:

- Monitorización de alta calidad a través de la red.

- Diversos formatos de comprensión de video soportados (*MPEG*, *JPEG*, *H.264*,…).

- Acceso a la información adquirida a través del navegador WEB / API.

- Diversas posibilidades de configuración de la cámara y la secuencia de video adquirida.

Éstas serán las funcionalidades principales que tratarán de implementarse en nuestro desarrollo de un *FrameCapture* a través de IP.

4.3 INTERFAZ CON LA CÁMARA IP

Una vez instalada la cámara en la red Ethernet del Grupo de Tratamiento e Interpretación de Video de la UAM (VPU – UAM) y asignada su dirección IP, se puede acceder a la interfaz WEB desarrollada por Sony a través de un explorador WEB, y escribiendo la dirección IP de ésta en el cuadro URL, se accede a la página de bienvenida de la cámara en concreto, que permite configurar ésta así como mostrar el visor principal.

Sin embargo esta opción, si bien permite familiarizarse con las posibilidades que ofrece la cámara IP, no permite integrar dichas cámaras en el sistema *DiVA*.

Para cumplir nuestro propósito, se utilizará el *SDK*[18] de Sony, denominado Sony Network Camera 4 (*SNC*), compuesto por dos *DLL*[19] (*sncstrm.dll* y *SonyNetworkCamera4.dll*) y una *API* (*SNC.cpp*) desarrollada en *C++.*

Si bien la mayoría de los métodos necesarios para la configuración de la cámara IP y adquisición de la secuencia de video se encuentran definidos en el API existente, es necesaria la configuración de algunos parámetros de interés a través del envío de comandos *CGI*[20]. Dichos comandos se ejecutarán a través del método definido en el *API* proporcionado:

```
SNC::SendCGICommand (SNC_HANDLE sncHandle, char* pMethod, char* pCGI,
char* pQuery, char* pReceiveBuffer, unsigned int receiveBufferSize,
unsigned int* pReceivedDataSize);
```

[18] **SDK** Conjunto de herramientas de desarrollo que permiten la creación de aplicaciones para un sistema concreto. (Del inglés *Software Development Kit*)

[19] **DLL** Biblioteca de enlace dinámico. Término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda de un programa por parte del sistema operativo. (Del inglés *Dinamic-Link Library*)

[20] **CGI** Tecnología que permite transferir datos entre el cliente (navegador WEB) y el programa ejecutado en un servidor WEB. (Del inglés *Common Gateway Interface*)

4.3.1 CONFIGURACIÓN DE LA CÁMARA IP

En este apartado se detallan las dos tareas necesarias para configurar e inicializar la cámara IP: establecer una conexión con ésta, y establecer unos parámetros de configuración de la secuencia de video.

- **ESTABLECIMIENTO DE CONEXIÓN CON LA CÁMARA IP**

El *SDK Sony Network Camera* proporciona unos métodos mediante los cuales es posible conectarse a la cámara IP. En primer lugar, es necesario cargar la librería dinámica y abrir un controlador con el que comunicarse con la cámara. A continuación, se llama al método:

```
SNC::SetNetwork(sncHandle, &netInfo);
```

Donde **sncHandle** identifica al controlador de cámara previamente abierto, y **pNetInfo** es un puntero a una estructura **NETINFO**, definida en el *SDK*, que contiene todos los campos necesarios para el establecimiento de la conexión con la cámara IP: dirección IP de la cámara, puerto de la cámara, dirección IP del proxy, nombre de usuario y contraseña.

```
////////////////////////////////////////////
// Network settings
////////////////////////////////////////////
struct NETINFO
{
        char                    CamIpAddr[256];             // Camera address
        unsigned short          CamPort;                    // Camera port
        bool                    ProxyFunc;                  // Proxy setting true:Use,
false:Not use
        char                    ProxyIpAddr[256];           // Proxy server address
(Available only manual setting□j
        unsigned short          ProxyPort;                  // Proxy server port
(Available only manual setting□j
        bool                    AuthenticationFunc;         // Ahthentication setting
true:use false:not use
        char                    User[17];                   // User ID     (Available at
using ahthenfunc. MAX:16 characters)
        char                    Password[17];               // Password    (Available at
using ahthenfunc. MAX:16 characters)
};
```

Figura 4-2: Estructura NETINFO

Este método devuelve **SNC_OK** si el proceso se realiza correctamente, o **SNC_ERR_XXX** si ocurre algún error.

- **PARÁMETROS DE CONFIGURACIÓN DE LA SECUENCIA DE VIDEO**

A continuación, se hace necesario establecer algunos parámetros de configuración de la secuencia de video:

- Códec de compresión de video (*JPEG*, *MPEG4*, *H.264*).
- Resolución de la imagen.

- Tasa de cuadros (fps[21]).
- Espacio de color (Blanco y negro o color).

Dichos parámetros son enviados a la cámara IP mediante comandos CGI:

```
SNC::SendCGICommand(sncHandle,"POST","/command/camera.cgi",
   pSettingBuffer, pBuffer, BUFFER_SIZE, &sncWrittenLen);
```

Donde **sncHandle** identifica al controlador de la cámara previamente abierto, **pSettingBuffer** es una cadena que indica qué parámetro se quiere modificar y con qué valor, **pBuffer** es un *buffer* donde se almacena el resultado devuelto por dicho método, **BUFFER_SIZE** el tamaño del *buffer*, y **sncWrittenLen** el tamaño del resultado devuelto por el método.

4.3.2 ADQUISICIÓN DE LA SECUENCIA DE VIDEO

Una vez se ha establecido la conexión con la cámara IP y especificado los parámetros de configuración de la secuencia de video, debe habilitarse la opción de *streaming*[22] ofrecida por las cámaras IP, así como la decodificación de dicha secuencia. Después, se define la función iterativa con la que recoger el contenido:

```
SNC::SetCallback(sncHandle,CALLBACK_DEC_VIDEO,frameCallback,NULL);
```

Dicho método establece una función *Callback*[23] de adquisición de *frames*, `frameCallback`, que se ejecuta cada vez que se reciba un *frame* (idealmente a la tasa de cuadros especificada), devolviendo **SNC_OK** si el proceso se realiza correctamente, o **SNC_ERR_XXX** si ocurre algún error.

```
void __stdcall frameCallback(VIDEOINFO *sncVideoCallbackInfo, unsigned
long param)
```

sncVideoCallbackInfo es un puntero a una estructura **VIDEOINFO**, propia del *SDK*, en la cual se almacena toda la información adquirida por la cámara IP.

```
////////////////////////////////////////////
// Get Video/Audio data
////////////////////////////////////////////
struct VIDEOINFO
{
        char                    *pBuf;
        unsigned int    BufLen;
        VIDEOCODECINFO VideoCodecInfo;
        USERDATA                UserData;
};
```

Figura 4-3: Estructura VIDEOINFO

[21] **fps** Tasa de cuadros. (Del inglés *frames per second*)

[22] **Streaming** Transmisión de contenido en tiempo real.

[23] **Callback** Código ejecutable que lleva a cabo iteraciones sobre objetos con propiedades específicas para llevar a cabo operaciones que desde el punto de vista del sistema son equivalentes, pero sobre elementos distintos.

Donde `pBuf` es un puntero al *frame*, `BufLen` el tamaño de dicho *frame*, y `VideoCodecInfo` y `UserData` son estructuras con información adicional acerca de la secuencia de video. Ambas estructuras se muestran a continuación:

```
//////////////////////////////////////////////
// Video codec
//////////////////////////////////////////////
struct VIDEOCODECINFO
{
        VIDEOCODEC              Codec;
        int                             Width;
        int                             Height;
};
```

Figura 4-4: Estructura VIDEOCODECINFO

Donde `Codec` representa la compresión de video utilizada (*JPEG*, *MPEG4* o *H.264*) y `Width` y `Height` representan el ancho y alto del *frame*, respectivamente.

```
//////////////////////////////////////////////
// UserData
//////////////////////////////////////////////
struct USERDATA
{
        char                    CamTim  [32];
        unsigned short FrmRate;
        char                    TimStamp[11];
        char                    CamPos  [17];
        char                    AlmEvent[17];
};
```

Figura 4-5: Estructura USERDATA

Donde `TimStamp` almacena el *TimeStamp* del *frame* en concreto. Los demás parámetros de esta estructura no serán utilizados.

Finalmente, la orden de comienzo de la retransmisión de la secuencia de video viene dada por el siguiente método:

```
SNC::Start(sncHandle);
```

Este método, en el que **`sncHandle`** es el identificador de controlador de la cámara, devolverá **`SNC_OK`** si el proceso se realiza correctamente, o **`SNC_ERR_XXX`** si ocurre algún error.

4.4 IMPLEMENTACIÓN EN EL SISTEMA DiVA

Una vez analizadas por separado las dos funcionalidades básicas del nuevo módulo (configuración de la cámara IP y adquisición de la secuencia de video), se han implementado dichas funcionalidades en el módulo *DiVACaptureIP* utilizando el citado *SDK*.

Posteriormente, la nueva clase desarrollada se integra en la aplicación *DiVAFrameServerIP*, completando la mejora propuesta en la capa de adquisición de video.

4.4.1 DESARROLLO DE LA CLASE DiVACAPTUREIP

A continuación se describen de modo funcional los diferentes métodos de la arquitectura de las capturadoras de video (véase la clase *DiVACapture*, apartado 3.2.1.2) implementados en la nueva clase derivada (*DiVACaptureIP*).

- **MÉTODOS DE INICIALIZACIÓN**

- `virtual int init();`

En el método de inicialización se reservan e inicializan todos los recursos necesarios para la ejecución del módulo *FrameCaptureIP:*

- *Buffer* para la comunicación con la cámara (`pBuffer`).
- Estructura `VIDEOINFO` (`sncVideoInfo`).
- Objeto *DiVAImage*, formato en el que enviaremos cada *frame* (`_frame`).
- Notificador de evento de *Callback* (`frameEvent`).

Una vez realizado este paso, se configura la cámara IP (establecimiento de conexión y parámetros de configuración de la secuencia de video) siguiendo los pasos propuestos en el Apartado 4.3.1. Finalmente, se da la orden de comenzar la retransmisión de la secuencia de video.

- **MÉTODOS DE ADQUISICIÓN DE IMÁGENES**

Dichos métodos concurren en un objetivo común: adquirir y/o servir un *frame* concreto (puede ser el *frame* actual, o el *frame* siguiente) en el formato *DiVAImage* (descrito en el apartado 3.2.1.2).

- `int captureNextFrame();`

Dicho método espera a que se genere un nuevo evento (`frameEvent=true`), en la función `frameCallback`, que almacena el nuevo *frame* en la estructura `sncVideoInfo`.

Una vez generado el evento, dicho método almacena toda la información relevante en el objeto *DiVAImage* `_frame`, y actualiza el notificador de eventos (`frameEvent=false`). Devuelve 0 si todo es correcto, o -1 si ocurre algún error.

- `int getCurrentFrame(DiVAImage* currentImage);`

Dicho método adquiere el *frame* actual ya recogido y almacenado en el objeto *DiVAImage* `_frame`.

Como entrada, toma como parámetro `currentFrame`, un puntero a un objeto *DiVAImage*, donde se almacenará el frame previamente almacenado en `_frame`. Devuelve 0 si todo es correcto, o -1 si ocurre algún error.

- `virtual int getNextFrame(DiVAImage *nextFrame);`

Dicho método hace uso de los dos métodos anteriores. En primer lugar, captura el siguiente *frame* mediante la llamada al método **`captureNextFrame()`**. A continuación, hace la llamada a **`getCurrentFrame(DiVAImage* nextFrame)`**, almacenando el *frame* en el objeto *DiVAImage* **`nextFrame`**. Devuelve 0 si todo es correcto, y -1 si ocurre algún error.

4.4.2 DESARROLLO DE LA APLICACIÓN DiVAFRAMESERVERIP

Una vez desarrollado el módulo *DiVACaptureIP*, es necesario integrarlo en una aplicación *DiVAFrameServerIP*, de manera que pueda actuar como un módulo independiente en la capa de adquisición de secuencias de video.

Para ello, se define una nueva aplicación que, a partir de unos parámetros de configuración, genere y ejecute un servidor de *frames* a través de cámara IP.

En primer lugar, se hace necesario leer el fichero de configuración, denominado **`config-IP.ini`**, donde se definen los parámetros de configuración de la cámara IP previamente mencionados en el apartado 4.3.1, así como los parámetros del servidor (puerto de servicio).

Para leer dicho fichero, se hace uso de la clase *DiVAFileReader*, desarrollada por el Grupo de Tratamiento e Interpretación de Video de la UAM (VPU-UAM). Dicha clase ofrece la posibilidad de adquirir la información de configuración de una manera rápida y sencilla, extrayendo las duplas parámetro – valor, separados por el signo "=", y almacenándolo en variables para su posterior utilización.

En caso de no encontrarse disponible el fichero **`config-IP.ini`**, se ha definido una configuración por defecto, incluida en la aplicación.

```
//Module Initialization
        DiVAServer *pServer = NULL;
        DiVACaptureIP* pcapturadora = new DiVACaptureIP(camIpAddr, camPort, proxyFunc,
authenticationFunc, user, pass, resolutionX, resolutionY, fps, color);
        pcapturadora->start();
        DiVAImageBuffer* pImageBuffer = pcapturadora->getImageBuffer();
        pServer = new DiVAServer(pImageBuffer,port);
        nRetCode = pServer->start();
```

Figura 4-6: DiVAFrameServerIP: Secuencia de funcionamiento

Una vez se dispone de los parámetros de configuración, la secuencia de funcionamiento de la aplicación será la siguiente:

- Generación de un nuevo objeto **`pcapturadora`**, de la clase *DiVACaptureIP*.
- Comienzo del proceso de adquisición de *frames de la cámara IP*.
- Generación de un nuevo objeto **`pImageBuffer`**, de la clase *DiVAImageBuffer*.
- Generación de un nuevo objeto **`pServer`**, de la clase *DiVAServer*.
- Comienzo del proceso de servicio de *frames* a través del puerto definido.

5. MÓDULO DE GESTIÓN DE DATOS

5.1 INTRODUCCIÓN

En este capítulo se describe la implementación del módulo servidor de datos así como su integración en el sistema *DiVA*, cuyo diseño se planteó en el apartado 3.3.2.

Dicho módulo está compuesto por una base de datos y un conjunto de funciones que permite la interconexión con dicha base de datos (integradas en la clase *DiVADataClient*). El conjunto de ambas conformará el nuevo módulo *DataServer*.

Para la realización de la base de datos se ha hecho uso de la tecnología *MySQL*[24], y ésta se comunica con el sistema *DiVA* a través del API[25] *MySQL++,* cuyo propósito es integrar ambas tecnologías (*C++* y *MySQL*) facilitando la utilización de lenguaje de consultas embebidos en contenedores *STL*[26].

A continuación se describen los siguientes aspectos relacionados en el desarrollo e integración del módulo servidor de datos (*DataServer*):

- Desarrollo de la base de datos para almacenar la información extraída.
- Comunicación con la base de datos (petición / almacenamiento).
- Integración en el sistema *DiVA.*

5.2 DESARROLLO DE LA BASE DE DATOS

En el apartado 3.3.2.1 se definió la estructura de la base de datos basándose en dos características del sistema: el formato del contenido (tipos de máscaras – **`masktypes`** –) que se desea gestionar (imágenes – **`images`** –, máscaras – **`masks`** – y descriptores – **`descriptors`** –) y el tipo de módulos con los que se va a interrelacionar (módulos capturadores de datos – **`frameservers`** – y algoritmos de procesado – **`algorithms`** –).

Adicionalmente, se definió un registro de las sesiones abiertas (módulos en ejecución) en las capas de adquisición y procesado de secuencias de video para facilitar la posterior búsqueda y extracción de toda la información de interés generada: **`framesessions`** y **`algorithmsessions`**.

---

[24] **MySQL** Sistema de gestión de base de datos relacional, multihilo y multiusuario.

[25] **API** Interfaz de programación de aplicaciones. (Del inglés *Application Programming Interface*).

[26] **STL** Recopilación genérica de plantillas y algoritmos para facilitar la implementación de estructuras de datos complejas. (Del inglés *Standard Template Library*).

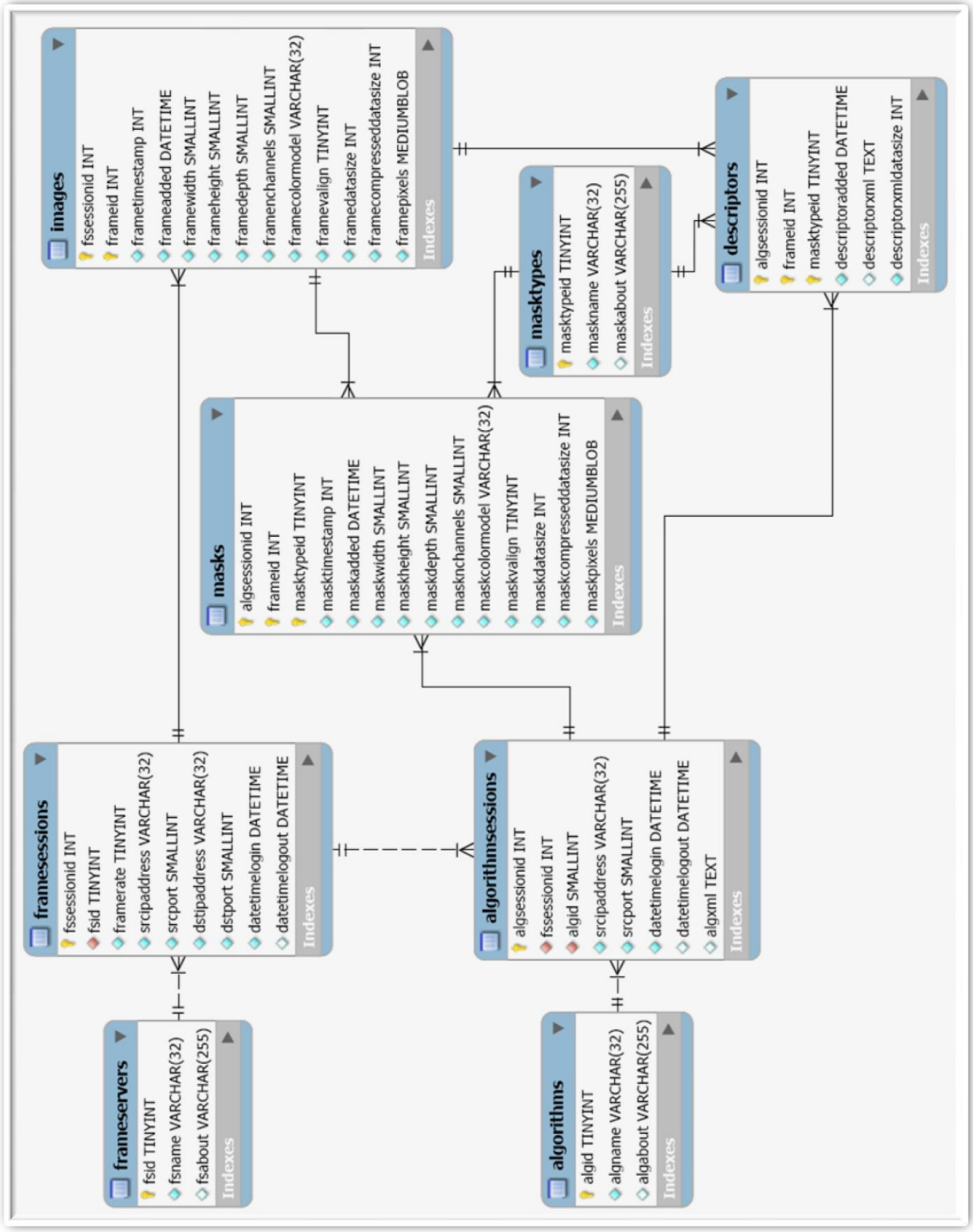

Figura 5-1: Esquema relacional de la base de datos *(Database Scheme)*

En el ANEXO D: se detalla la implementación del diseño propuesto, generando la nueva base de datos utilizando la tecnología *MySQL*, creando las tablas anteriormente nombradas y definiendo las relaciones entre éstas, así como los tipos de datos para cada campo. A continuación se describe de manera funcional, para cada tabla, cada uno de los campos incluidos en el diseño de la base de datos:

- **TABLA frameservers**

Dicha tabla contiene aquellos parámetros identificativos de cada uno de los *FrameServers* disponibles en el sistema.

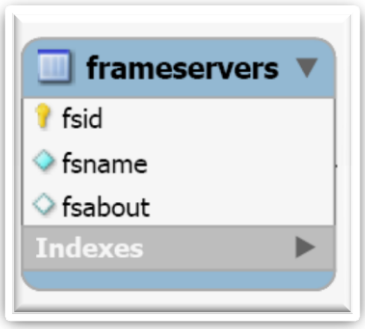

Figura 5-2: Tabla frameservers

Donde *fsid* identifica unívocamente cada uno de los *FrameServers* disponibles, *fsname* es el nombre de éste y *fsabout* su descripción.

- **TABLA framesessions**

Dicha tabla contiene aquellos parámetros relativos a la conexión de un *FrameServer* a la base de datos.

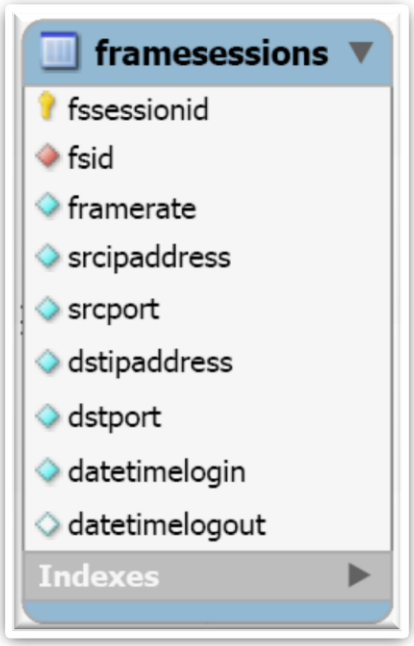

Figura 5-3: Tabla framesessions

Donde *fssessionid* identifica unívocamente la sesión abierta por el *FrameServer*, *fsid* su identificador, *framerate* la tasa de cuadros por segundo a la que sirve el *FrameServer*, *srcipaddress* la dirección IP de la cámara IP, *srcport* el puerto de servicio de la cámara IP, *dstipaddress* la dirección IP del *FrameServer*, *dstport* el puerto de servicio del *FrameServer*, *datetimelogin* la fecha de establecimiento de conexión y *datetimelogout* la fecha de finalización de la sesión.

- **TABLA images**

Dicha tabla contiene aquellos *frames* provisionados en la capa de adquisición de la secuencia de video.

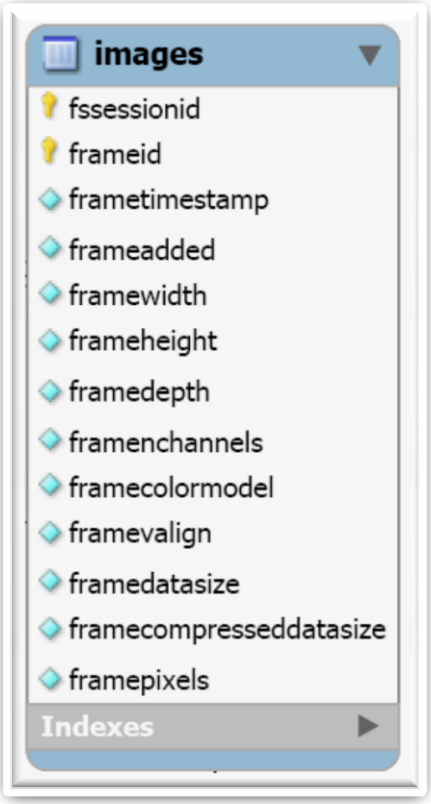

Figura 5-4: Tabla images

Donde *fssessionid* identifica unívocamente la sesión abierta por el *FrameServer*, *frameid* es el identificador de la imagen, *frametimestamp* es su *Timestamp*[27], *frameadded* la fecha en la que se añadió la imagen a la base de datos, *framewidth* el ancho de la imagen (en píxeles), *frameheight* el alto de la imagen (en píxeles), *framedepth* la resolución de la imagen, *framenchannels* el número de canales de la imagen, *framecolormodel* el modelo de color de la imagen, *framevalign* la coordenada de origen de la imagen, *framedatasize* el tamaño (en *bytes*) de la imagen, *framecompresseddatasize* el tamaño comprimido (en *bytes*) de la imagen y *framepixels* el contenido de la imagen.

[27] **Timestamp** Secuencia de caracteres, que denotan la hora y fecha (o alguna de ellas) en la cual ocurrió determinado evento.

- **TABLA algorithms**

Dicha tabla contiene aquellos parámetros identificativos de cada uno de los *Algoritmos* disponibles en el sistema.

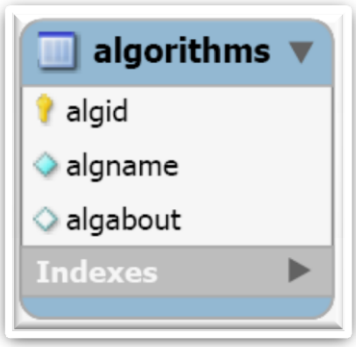

Figura 5-5: Tabla algorithms

Donde *algid* identifica unívocamente cada uno de los *Algoritmos* disponibles, *algname* es el nombre de éste y *algabout* su descripción.

- **TABLA algorithmsessions**

Dicha tabla contiene aquellos parámetros relativos a la conexión de un *Algoritmo* a la base de datos.

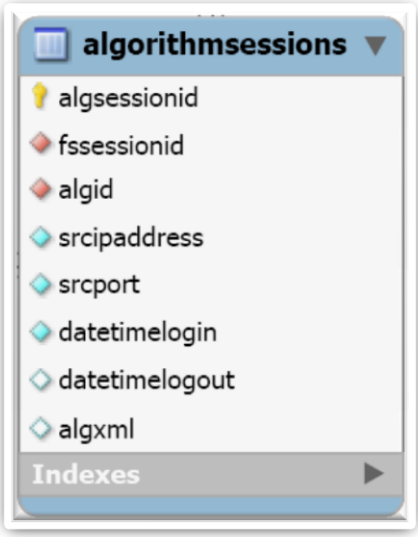

Figura 5-6: Tabla algorithmsessions

Donde *algsessionid* identifica unívocamente la sesión abierta por el *Algoritmo*, *fssessionid* identifica unívocamente la sesión abierta por el *FrameServer, algid* el identificador del *Algoritmo*, *srcipaddress* la dirección IP del *FrameServer*, *srcport* el puerto de servicio del *FrameServer*, *datetimelogin* la fecha de establecimiento de conexión, *datetimelogout* la fecha de finalización de la sesión y *algxml* la descripción que genera (si lo hace) dicho algoritmo.

- **TABLA masktypes**

Dicha tabla contiene los diferentes tipos de máscara generados en la capa de procesado de la secuencia de video.

Figura 5-7: Tabla masktypes

Donde *masktypeid* identifica unívocamente el tipo de máscara generada (*Background*, *Foreground*, máscara de movimiento, máscara de sombras,…), *maskname* es el nombre de éste y *maskabout* su descripción.

- **TABLA masks**

Dicha tabla contiene aquellas máscaras generadas en la capa de procesado de la secuencia de video.

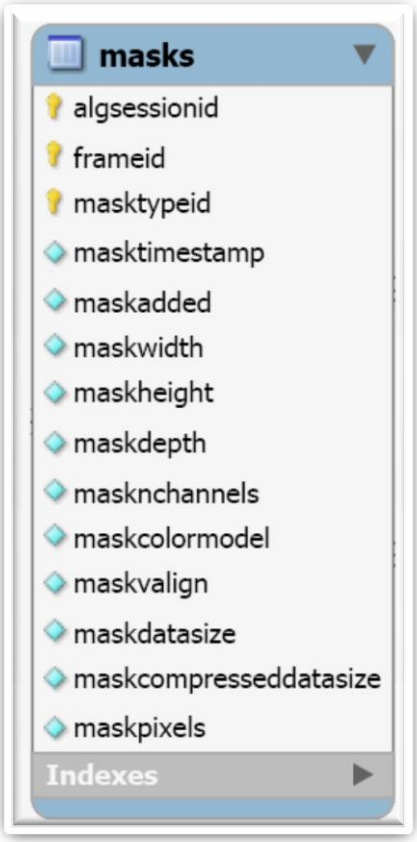

Figura 5-8: Tabla masks

Donde *algsessionid* identifica unívocamente la sesión abierta por el *Algoritmo*, *frameid* es el identificador de la imagen asociada a dicha máscara, *masktypeid* es el tipo de máscara, *masktimestamp* es su *Timestamp*, *maskadded* la fecha en la que se añadió la máscara a la base de datos, *maskwidth* el ancho de la máscara (en píxeles), *maskheight* el alto de la máscara (en píxeles), *maskdepth* la resolución de la máscara, *maskchannels* el número de canales de la máscara, *maskcolormodel* el modelo de color de la máscara, *maskvalign* la coordenada de origen de la máscara, *maskdatasize* el tamaño (en *bytes*) de la máscara, *maskcompresseddatasize* el tamaño comprimido (en *bytes*) de la máscara y *maskpixels* el contenido de la máscara.

- **TABLA descriptors**

Dicha tabla contiene aquellos descriptores generados en la capa de procesado de la secuencia de video.

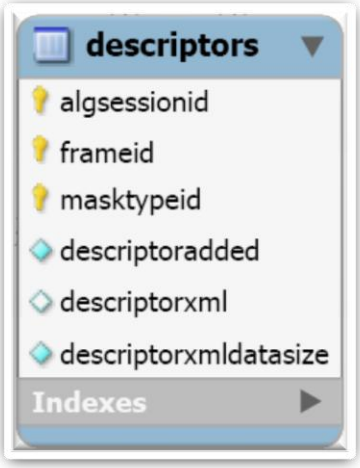

Figura 5-9: Tabla descriptors

Donde *algsessionid* identifica unívocamente la sesión abierta por el *Algoritmo*, *frameid* es el identificador de la imagen asociada a dicho descriptor, *masktypeid* es el tipo de máscara (descriptor) generado, *descriptoradded* la fecha en la que se añadió el descriptor a la base de datos, *descriptorxml* el contenido del descriptor y *descriptorxmldatasize* el tamaño (en bytes) del descriptor.

5.3 INTERFAZ MYSQL CON LA BASE DE DATOS

La comunicación entre la base de datos y el sistema *DiVA* se realiza a través del *API MySQL++,* cuyo propósito es integrar ambas tecnologías (*C++* y *MySQL*) facilitando la utilización de lenguaje de consultas embebidos en contenedores *STL*.

El modo de funcionamiento del *API MySQL++* es similar a la mayoría de *APIs* de acceso a bases de datos. Los pasos a seguir son los siguientes:

- **CREACIÓN DE UNA CONEXIÓN CON LA BASE DE DATOS**

Un objeto `Connection` inicia y controla la conexión con el servidor *MySQL*. Dicho objeto será posteriormente utilizado directa o indirectamente por los demás objetos pertenecientes a dicha API.

```
con = new mysqlpp::Connection(false);
con.connect(database, host, user, pass);
```

Donde `con` será el controlador de la conexión con el servidor, que necesita como parámetros de entrada los siguientes: `database` (nombre de la base de datos), `host` (dirección IP de la base de datos), `user` (nombre de usuario) y `pass` (contraseña).

- **GENERACIÓN Y EJECUCIÓN DE UNA CONSULTA[28]**

Una vez abierta la conexión con el servidor *MySQL*, se crea un nuevo objeto `Query`, referenciado a dicha conexión, y que será el contenedor *STL* para definir y ejecutar consultas sobre la base de datos.

```
query = new mysqlpp::Query(&con, false);
```

El objeto `Query` toma como parámetro de entrada el controlador `con` previamente creado y que vinculará, por tanto, dicho objeto con la base de datos.

Dado que el objeto `Query` pertenece a una clase heredada de `std::stringstream`, el tratamiento será el mismo que para cualquier otro *stream* definido en *C++*:

```
// Consulta
query->reset();
*query << "SELECT * FROM table WHERE field = " << value;
res = query->store();
```

Figura 5-10: Definición y ejecución de una consulta de selección.

Dicho objeto provee, entre otros, diferentes métodos para su ejecución y control de errores, en función de la naturaleza de la consulta, de una manera sencilla y eficaz.

Se utilizará el método `query->store()` para ejecutar aquellas consultas de selección, y `query->execute()` para ejecutar consultas de inserción o modificación (aquellas que no devuelven datos).

[28] **Consulta** Nos referiremos al concepto consulta para definir cualquier clase de interacción con la base de datos. Por tanto, una inserción o modificación se considerará también una consulta a la base de datos. No obstante, se matizará qué tipo de consulta se ha utilizado siempre que sea necesario.

- **ADQUISICIÓN DE LOS RESULTADOS DE LA CONSULTA**

Existen diversos objetos para gestionar la adquisición de datos e información a partir de la ejecución de una consulta o inserción.

Para gestionar consultas de selección, se utilizarán objetos **`Result`**, que incluyen uno o más objetos **`Row`**. Dichos objetos permiten el tratamiento de los resultados como si de una matriz de dos dimensiones se tratara (campos – filas).

```
// Adquisición resultados consulta
res = query->store();
        if (res)
        {
                if (res.empty())
                {
                        return -1;
                }
                mysqlpp::Row row;
                mysqlpp::Row::size_type i;
                for (i = 0; row = res.at(i); ++i)
                {
                        value = row["field"];
                }
        }
```

Figura 5-11: Adquisición de resultados de una consulta de selección.

En aquellas situaciones en las que se pretenda ejecutar consultas que no devuelven datos (inserciones, modificaciones,…), se hará uso de objetos **`ResNSel`**, que tan sólo reportan el estado final de la base de datos una vez ejecutada la consulta.

5.4 IMPLEMENTACIÓN EN EL SISTEMA DiVA

En este apartado se desarrollará, en primer lugar, la nueva clase *DiVADataClient*. Dicha clase aportará los métodos ya expuestos en el capítulo 3.3.2.2 y que permitirán comunicarse con el *DataServer* desde los diferentes módulos que lo requieran, enmarcados en las capas de adquisición y procesado de la secuencia de video (*FrameServers* o *Algoritmos* de análisis, respectivamente).

A continuación se modificará ligeramente el sistema existente con el propósito de aportar aquellas funcionalidades propuestas al sistema *DiVA*: envío de *frames* de *FrameServer* a *DataServer*, envío de máscaras y descriptores desde *Algoritmo* a *DataServer* y envío de *frames*, máscaras y descriptores desde *DataServer* a *Algoritmo*.

5.4.1 DESARROLLO DE LA CLASE DiVADATACLIENT

A continuación se describen de modo funcional los principales métodos implementados en la nueva clase genérica denominada *DiVADataClient*:

- **`int DiVADataClient::init(char *database, char *host, char *user, char *pass);`**

En el método de inicialización se reservan e inicializan todos los recursos necesarios para la conexión y comunicación entre dicha clase y la base de datos:

- Controlador de conexión con la base de datos (**con**).
- Objeto **Query**, con el se definen y ejecutan las consultas (**query**).
- Objeto **Result**, que contiene los resultados de una consulta de selección (**res**).
- Objeto **ResNSel**, que contiene los resultados de estado de consultas de inserción y modificación (**resnsel**).

También se reservan las variables utilizadas para referenciar las sesiones **fsSessionID** y **algSessionID**, así como **fsID** y **algID**, identificadores de módulo.

Los parámetros de entrada de dicho método (**database**, **host**, **user** y **pass**) serán a su vez los parámetros necesarios para abrir una conexión con la base de datos.

- ```
  int DiVADataClient::openFrameServerSession(int fsID, int frameRate,
  char *srcIPAddress, int srcPort, int dstPort);
  ```

Este método realiza una consulta de inserción en la tabla **framesessions**, registrando la sesión del *FrameServer* según los parámetros de entrada de dicho método.

A continuación, adquiere el identificador de sesión *FrameServer* (**fsSessionID**), determinado por la base de datos, mediante una consulta de selección llamando al método **getFrameServerSessionID(fsID)** (donde **fsID** es el identificador del *FrameServer*). Dicho identificador queda referenciado permitiendo la posterior recepción de la secuencia de video. Devuelve -1 en caso de error o el identificador de sesión *FrameServer* (**fsSessionID**) en caso de éxito.

- ```
  int DiVADataClient::closeFrameServerSession();
  ```

Este método realiza una consulta de actualización en la tabla **framesessions**, registrando la finalización de la sesión del *FrameServer* según el identificador de sesión (**fsSessionID**) referenciado. Devuelve -1 en caso de error o 0 en caso de éxito.

- ```
  int DiVADataClient::sendImage(DiVAImage *frame);
  ```

Este método extrae cada uno de los parámetros del objeto *DiVAImage* **frame** y realiza una consulta de inserción en la tabla **images**, según el identificador de sesión (**fsSessionID**) referenciado. Devuelve -1 en caso de error o 0 en caso de éxito.

- ```
  int DiVADataClient::receiveImage(int frameID, DiVAImage *frame);
  ```

Este método realiza una consulta de selección en la tabla **images** de un único *frame* identificado mediante la dupla **fsSessionID** (referenciado) – **frameID** (pasado como parámetro de entrada), adquiriendo todos los datos necesarios para regenerar el objeto *DiVAImage* y almacenándolo en **frame**. Devuelve -1 en caso de error o 0 en caso de éxito.

- `int DiVADataClient::openAlgorithmSession(char *algName, char *srcIPAddress, int srcPort, char *algXML)`

Este método realiza una consulta de inserción en la tabla **`algorithmsessions`**, registrando la sesión del *Algoritmo* según los parámetros de entrada de dicho método.

Mediante una consulta de selección llamando al método **`getFrameServerSessionID(srcIPAddress, srcPort)`** (donde **`srcIPAddress`** es la dirección IP del *FrameServer* y **`srcPort`** el puerto de servicio) se referencia el identificador de sesión *FrameServer* (**`fsSessionID`**).

De manera análoga, mediante una consulta de selección llamando al método **`getAlgorithmID(algname)`** se obtiene el identificador de dicho *Algoritmo* (**`algID`**).

A continuación, adquiere el identificador de sesión *Algoritmo* (**`algSessionID`**), determinado por la base de datos, mediante una consulta de selección llamando al método **`getAlgorithmSessionID(algID)`** (**`algID`** es el identificador del *FrameServer*).

Dicho identificador queda referenciado permitiendo la posterior recepción de la información procesada en el *Algoritmo*. Devuelve -1 en caso de error o el identificador de sesión *Algoritmo* (**`algSessionID`**) en caso de éxito.

- `int DiVADataClient::closeAlgorithmSession();`

Este método realiza una consulta de actualización en la tabla **`algorithmsessions`**, registrando la finalización de la sesión del *Algoritmo* según el identificador de sesión (**`algSessionID`**) referenciado. Devuelve -1 en caso de error o 0 en caso de éxito.

- `int DiVADataClient::sendMask(DiVAImage *mask, int maskTypeID)`

Este método extrae cada uno de los parámetros del objeto *DiVAImage* **`mask`** y realiza una consulta de inserción en la tabla **`masks`**, según el identificador de sesión (**`algSessionID`**) referenciado. Se indica además el tipo de máscara (**`maskTypeID`**) enviado. Devuelve -1 en caso de error o 0 en caso de éxito.

- `int DiVADataClient::receiveMask(int frameID, int maskTypeID, DiVAImage *mask);`

Este método realiza una consulta de selección en la tabla **`masks`** de una única máscara identificada mediante la tripleta **`algSessionID – frameID – masktypeID`**, adquiriendo todos los datos necesarios para regenerar el objeto *DiVAImage* y almacenándolo en **`mask`**. Devuelve -1 en caso de error o 0 en caso de éxito.

- `int DiVADataClient::sendDescriptor(int frameID, char *descriptorXML);`

Este método realiza una consulta de inserción en la tabla **descriptors**, según el identificador de sesión (**algSessionID**) referenciado. Se indica además el tipo de descriptor enviado (**maskTypeID**). Devuelve -1 en caso de error o 0 en caso de éxito.

- ```
  int DiVADataClient::receiveDescriptor(int frameID, int maskTypeID,
  char *descriptorXML);
  ```

Este método realiza una consulta de selección en la tabla **descriptors** de un único descriptor identificado mediante la tripleta **algSessionID – frameID – maskTypeID**, adquiriendo el descriptor y almacenándolo en **descriptorXML**. Devuelve -1 en caso de error o 0 en caso de éxito.

5.4.2 DESARROLLO DE LA APLICACIÓN DATASERVER

Una vez desarrollada la nueva clase *DiVADataClient*, es necesario integrarla en las aplicaciones definidas en las capas de adquisición y procesado de la secuencia de video. A continuación se describen las modificaciones realizadas en el sistema con el objetivo de aportar las funcionalidades propuestas:

- **ENVÍO DE CONTENIDO DESDE FRAMESERVER HACIA DATASERVER**

En primer lugar, se añade un nuevo método a la clase *DiVAServer* (véase descripción funcional de dicha clase en el apartado 3.2.1.1), que inicializa un objeto *DiVADataClient* (**pDataClient**) con el que se realiza la comunicación entre la capa de adquisición de secuencias de video y la base de datos:

- ```
  int initDataServerConnection(void* dbCon);
  ```

Donde **dbCon** es una estructura que contiene todos los parámetros necesarios para el establecimiento de la conexión entre el sistema y la base de datos, previamente mencionados en el apartado 5.3.

Siempre que esté definido el objeto **pDataClient**, y cuando se estime necesario el almacenamiento de la secuencia de video, el *FrameServer* proporcionará la secuencia de video a través del puerto de servicio seleccionado y también enviará dicha secuencia a la base de datos, mediante una llamada al método **pDataClient->sendImage(frame)**.

Dicho proceso se realiza dentro de la clase *DiVAServer*, de forma transparente para el módulo *FrameServer*, al que tan sólo le basta definir la utilización o no de dicha funcionalidad, así como el establecimiento de una sesión.

Para ello, se modifica la aplicación *DiVAFrameServerIP* (desarrollada en el apartado 4.4.2) para que, a partir de unos parámetros de configuración, establezca una conexión con el módulo *DataServer* y envíe la secuencia de video adquirida a la base de datos.

Se hace por tanto necesario leer el fichero de configuración, denominado **config-DataServer.ini**.

```
// Data Client Init
        pServer->initDataServerConnection(&dbCon);
        pServer->pDataClient->openFrameServerSession(  fsID,  framerate,  srcIPAddress,
srcPort, dstPort);
```

Figura 5-12: Envío de contenido desde FrameServer hacia DataServer

- **ENVÍO DE CONTENIDO DESDE ALGORITMO HACIA DATASERVER**

La implementación de esta funcionalidad es muy similar al envío de *frames* desde *FrameServer* hacia *DataServer*, salvo que la clase modificada no es *DiVAServer*, sino que son *DiVAAlgorithm* (que define cómo se interrelacionan los módulos de procesado de video con el sistema. Véase 3.2.2.2) y la propia clase que defina el *Algoritmo* (en el ámbito de desarrollo del PFC, se modificará un algoritmo de detección de movimiento, proporcionado por el Grupo de Tratamiento e Interpretación de Video (VPU – UAM), – **DiVAFDGaussBkgExtractor** – y el nuevo algoritmo de detección de sombras – **DiVAShadowSuppressor** –).

En primer lugar, y según los parámetros de entrada de la clase *DiVAAlgorithm*, se inicializa un objeto *DiVADataClient* (**pDataClient**) con el que se realizará la comunicación entre el sistema y la base de datos.

A continuación, y siempre que esté definido el objeto **pDataClient**, es la propia clase *Algoritmo* quien abre una sesión, y cuando ésta procese la secuencia de video (dentro del método **processFrame()**), enviará la información generada a la base de datos, mediante una llamada a los métodos siguientes, según corresponda:

```
DiVADataClient::sendMask(bkg, BKG)
DiVADataClient::sendMask(frg, FRG)
DiVADataClient::sendMask(mask, MOVEMENT_MASK)
DiVADataClient::sendMask(mask, SHADOW_MASK)
```

- **ENVÍO DE CONTENIDO DESDE DATASERVER HACIA ALGORITMO**

A la hora de implementar esta funcionalidad, se ha optado por diferenciar dos casos concretos: adquisición de contenido previamente capturado por el *FrameServer* (*frames*) y adquisición de contenido previamente procesado por los distintos *Algoritmos*.

La adquisición de *frames* desde el *DataServer* se puede contemplar como una aplicación *FrameServer* cuya fuente de contenido es la base de datos, y por tanto se pueden emplear fácilmente los recursos existentes en el sistema *DiVA* para la realización de una nueva aplicación.

Sin embargo, la adquisición de contenido proporcionada por los distintos *Algoritmos*, implica la adquisición de diferentes máscaras y descriptores de manera simultánea y para cada *frame*.

♦ ADQUISICIÓN DE FRAMES (APLICACIÓN DiVAFRAMESERVERDS)

En primer lugar, se desarrolla una nueva aplicación *FrameServer*, denominada *FrameServerDS*, capaz de servir secuencias capturadas en la capa de adquisición de video.

```
//Module Initialization
DiVAServer *pServer = NULL;
DiVACaptureDS* pcapturadora = new DiVACaptureDS((void*)&dbCon,sessionID);
pcapturadora->start();
DiVAImageBuffer* pImageBuffer = pcapturadora->getImageBuffer();
pServer = new DiVAServer(pImageBuffer,port);
```

Figura 5-13: DiVAFrameServerDS. Secuencia de funcionamiento

El esquema de funcionamiento del *FrameServerDS* es similar al funcionamiento del *FrameServerIP*, planteado en el apartado 4.2.3.2.

A continuación se describen de modo funcional los diferentes métodos de la arquitectura de las capturadoras de video implementados en la nueva clase derivada (*DiVACaptureDS*).

- `int DiVACaptureDS::init(void*dbCon, int sessionID)`

En el método de inicialización se reservan e inicializan todos los recursos necesarios para la ejecución del módulo *FrameCaptureDS:*

- Objeto *DiVADataClient* que realice la conexión con la base de datos (`pDataClient`).

- Objeto *DiVAImage*, formato en el que se envía cada *frame* o máscara (`_frame`).

Donde `dbCon` es una estructura que contiene todos aquellos parámetros necesarios para el establecimiento de la conexión con la base de datos y `sessionID` es el identificador de la sesión (`fsSessionID` o `algSessionID`, según corresponda).

Dicho método abre una sesión *FrameServer* a partir de la cual se extrae el contenido previamente almacenado por las capas de adquisición de la secuencia de video. Devuelve 0 si todo es correcto o -1 si ocurre algún error.

- `int DiVACaptureDS::captureNextFrame();`

Dicho método incrementa el identificador de *frame* actual y llama al método `pDataClient->(frameID,frame)` correspondiente a la clase *DiVADataClient*, almacenando toda la información relevante en el objeto *DiVAImage* `_frame`. Devuelve 0 si todo es correcto o -1 si ocurre algún error.

- `int DiVACaptureDS::getCurrentFrame(DiVAImage *image);`

Dicho método adquiere el *frame* actual ya recogido y almacenado en el objeto *DiVAImage* **`_frame`**.

Como entrada, toma como parámetro **`currentFrame`**, un puntero a un objeto *DiVAImage*, donde se almacenará el *frame* previamente almacenado en **`_frame`**. Devuelve 0 si todo es correcto o -1 si ocurre algún error.

- **`virtual int getNextFrame(DiVAImage *nextFrame);`**

Dicho método hace uso de los dos métodos anteriores. En primer lugar, captura el siguiente *frame* mediante la llamada al método **`captureNextFrame()`**. A continuación, hace la llamada a **`getCurrentFrame(DiVAImage* nextFrame)`**, almacenando el *frame* en el objeto *DiVAImage* **`nextFrame`**. Devuelve 0 si todo es correcto o -1 si ocurre algún error.

♦ ADQUISICIÓN DE MÁSCARAS Y DESCRIPCIONES

Para implementar esta funcionalidad, el módulo *Algoritmo* debe conectarse con el módulo *FrameServerDS*, que le proporciona la secuencia de video original capturada previamente en la capa de adquisición. Este proceso es similar para todos los *FrameServers*.

A continuación, y según los parámetros de entrada de la clase *DiVAAlgorithm*, se inicializa un objeto *DiVADataClient* (**`pDataClient`**) con el que se realiza la comunicación entre el sistema y la base de datos.

Finalmente, y siempre que esté definido el objeto **`pDataClient`**, es la propia clase *Algoritmo* quien abre una sesión de procesado disponible para dicha secuencia de video, y solicita la información procesada al *DataServer* a medida que recibe la secuencia original mediante una llamada a los métodos siguientes, según corresponda, y donde **`frameID`** es extraído del *frame* original adquirido de *FrameServerDS*:

```
DiVADataClient::receiveMask(frameID, bkg, BKG)
DiVADataClient::receiveMask(frameID, frg, FRG)
DiVADataClient::receiveMask(frameID, mask, MOVEMENT_MASK)
DiVADataClient::receiveMask(frameID, mask, SHADOW_MASK)
```

6. MÓDULO DE DETECCIÓN DE SOMBRAS

6.1 INTRODUCCIÓN

En este capítulo se detalla el proceso de desarrollo de algoritmos de detección de sombras y su posterior integración, contribuyendo al desarrollo propuesto en la capa de análisis y procesado de video del sistema *DIVA*.

En primer lugar, se ha realizado una selección analítica de los diferentes tipos de algoritmos desarrollados en la literatura, destacando aquellos más representativos, en función de su comportamiento en entornos genéricos, complejidad, así como el diseño del sistema *DiVA* sobre el que se van a implementar dichos algoritmos.

Una vez realizada la comparativa, se han elegido tres métodos para su desarrollo: *DNM* basados en la extracción de características espectrales (en los espacios de color *HSV* e *YCbCr*) y basados en la extracción de características espaciales (*HFOD*[29]).

Finalmente, se describe el proceso de desarrollo de la clase *DiVAShadowSuppressor*, que conformará, junto con los algoritmos desarrollados, el módulo de análisis de sombras del sistema *DiVA*.

6.2 ANÁLISIS DE ALGORITMOS REPRESENTATIVOS SELECCIONADOS

El propósito de este apartado es el de ofrecer una comparativa analítica y posterior selección de los algoritmos de detección de sombras más aptos para su integración en el sistema *DiVA,* desarrollando un módulo detector de sombras de propósito general y ofreciendo un funcionamiento del sistema completo en tiempo real.

La selección de aquellos algoritmos de detección de sombras más representativos se ha realizado de acuerdo al estudio planteado en el apartado 2.2.4, teniendo en cuenta factores como la complejidad y coste computacional de los algoritmos, su comportamiento, y cuestiones relacionadas con el diseño del sistema.

6.2.1 SELECCIÓN DEL PROCESO DE DECISIÓN

A continuación se enumeran los cuatro procesos de decisión, representados en la Figura 2-10, y se describe el porqué han sido seleccionados (o descartados) en la implementación final del sistema *DiVA*:

[29] **HFOD** Máscara de la derivada de primer orden horizontal.
(Del inglés Horizontal First – Order Derivative Mask)

- **MÉTODO ESTADÍSTICO PARAMÉTRICO**

Se descarta este método debido principalmente a la complejidad que normalmente presentan las escenas ya que en la mayoría de éstas no se puede asumir que el *Background* es plano, y en las que tanto el análisis y segmentación de texturas como el proceso de estimación de parámetros es, cuanto menos, muy complicado y costoso computacionalmente.

- **MÉTODO ESTADÍSTICO NO PARAMÉTRICO**

Pese a que este método obtiene unos resultados aceptables, requiere N *frames* para calcular, en cada píxel, las medias y varianzas de los distintos canales de color.

Este aspecto redunda en un peor comportamiento en la detección durante el comienzo de la secuencia, un mayor coste computacional y dado el diseño del módulo de detección de sombras, enmarcado en la capa de análisis y procesado del sistema *DiVA*, que realiza la detección *frame* a *frame*, implica el rediseño de éste. Por estos motivos, se ha decidido descartar este método.

- **MÉTODO DETERMINÍSTICO BASADO EN EL MODELO**

Este método, aunque ofrece indudablemente los mejores resultados, es excesivamente complejo y con un alto coste computacional. Además, el número y la complejidad de los modelos incrementa considerablemente si se plantean diferentes condiciones de iluminación, clases de objetos, entornos,… Por este motivo, se ha decidido descartar este método.

- **MÉTODO DETERMINÍSTICO NO BASADO EN EL MODELO**

Este método procesa únicamente el *frame* actual, y se basa en determinar heurísticamente aquellos umbrales a partir de los cuales evaluar si un determinado píxel se encuentra en una región sombreada o no.

Es además el que mejor comportamiento ofrece según la evaluación cuyos resultados se pueden contemplar en la Tabla 2-1. Por tanto, se ha decidido seleccionar éste frente a los demás métodos.

6.2.2 SELECCIÓN DE DOMINIOS DE EXTRACCIÓN DE CARACTERÍSTICAS

Este estudio se basa fundamentalmente en la extracción de características espectrales, y son éstas en las que se basa el desarrollo de algoritmos de detección de sombras. No obstante, se plantea la posibilidad de un uso adicional de técnicas que exploten las características espaciales y temporales para mejorar el comportamiento final de los algoritmos.

- **DOMINIO ESPACIAL**

La utilización de ambas características (espectrales y espaciales) mejora el comportamiento en la detección y eliminación de sombras, gracias a la información adicional que nos ofrecen los píxeles adyacentes tanto de la imagen actual, como del *Background*. Por tanto, se opta por seleccionar este dominio espacial en el desarrollo del módulo de sombras, si bien se tratará de un algoritmo sencillo para conseguir un análisis en tiempo real.

- **DOMINIO TEMPORAL**

Si bien este método mejora ligeramente los resultados obtenidos a partir de los métodos anteriores, no ofrece una detección eficaz por sí sólo, y puede llegar a comprometer el comportamiento del sistema en tiempo real, debido al alto coste computacional que supone el uso de los tres métodos conjuntamente. Por tanto, se descarta la extracción de características en este dominio.

6.2.3 SELECCIÓN DE DISTINTOS ESPACIOS DE COLOR

Según la clasificación de las sombras realizada en el apartado 2.2.3, éstas pueden ser sombras visibles o invisibles. Por esta razón, se selecciona para su implementación el espacio de color *HSV*, que ofrece un mejor comportamiento en entornos en los que predominan las sombras visibles, como son las secuencias de video capturadas en el Hall de la EPS. Se propone también el desarrollo de un algoritmo de detección de sombras basado en el espacio de color *YCbCr*, más apto en entornos en los que predominan las sombras invisibles.

6.3 IMPLEMENTACIÓN DE ALGORITMOS SELECCIONADOS

6.3.1 ALGORITMO ESPECTRAL EN EL ESPACIO DE COLOR HSV

La implementación de este algoritmo contempla dos etapas diferenciadas. En primer lugar, se hace necesario realizar una conversión de color de *Background* y *Foreground* del espacio *RGB* (espacio de color estándar recogido por los módulos de adquisición de imágenes) al espacio de color *HSV*:

```
// Conversions between Color Spaces
conversion = CV_RGB2HSV;
// [BackGround]
IplImage* hsvBkg = cvCreateImage(cvGetSize(bkg),8,3);
cvCvtColor(bkg, hsvBkg, conversion);
DiVAHsvBkg = pconverter->getDiVAImage(hsvBkg);
// [ForeGround]
IplImage* hsvFrg = cvCreateImage(cvGetSize(frg),8,3);
cvCvtColor(frg ,hsvFrg, conversion);
DiVAHsvFrg = pconverter->getDiVAImage(hsvFrg);
// [Mask]
DiVAMask = pconverter->getDiVAImage(mask);
```

Figura 6-1: Conversión entre espacios de color

Donde **conversion** define la conversión de un espacio de color a otro, **cvCreateImage(…)** y **cvCvtColor(…)** son métodos aportados por la librería *OpenCV* para crear un objeto *Ipl* (formato nativo de *OpenCV* para almacenar imágenes) y realizar una conversión de color, respectivamente. **pconverter** es un objeto perteneciente a la clase *DiVAConverter* que extrae el contenido de una imagen *Ipl* y lo almacena en un objeto *DiVAImage*.

En segundo lugar, se realiza el proceso de decisión comparando *Background* y *Foreground* según los valores del píxel en los tres canales: tonalidad – H –, saturación – S – y brillo – V –. El proceso de decisión es el siguiente:

```
// Shadow Detection
valueMask = DiVAMask->getPixel(i,j);
if (valueMask.val[0] != 0)
{
        // Getting Pixel Data [BackGround and ForeGround]
        valueHsvBkg = DiVAHsvBkg->getPixel(i,j);
        valueHsvFrg = DiVAHsvFrg->getPixel(i,j);
        // Evaluating Shadow Suppression
        // 1. Hue Condition
        dh = abs(valueHsvFrg.val[0] - valueHsvBkg.val[0]);
        if (min(dh, 180-dh) <= tau_h){
                // 2. Saturation Condition
                if (valueHsvFrg.val[1] - valueHsvBkg.val[1] <= tau_s){
                                // 3. Value Condition
                                if (valueHsvFrg.val[2]/valueHsvBkg.val[2] >= alpha &&
valueHsvFrg.val[2]/valueHsvBkg.val[2] <= beta){
                                        valueMask.val[0] = 0;
                                        DiVAMask->setPixel(i,j,valueMask);
                                }
                        }
        }
}
```

Figura 6-2: *ShadowSuppressorHSV*: Proceso de decisión.

La primera condición evalúa la diferencia absoluta entre *Foreground* y *Background* en el canal H (tonalidad), basándose en que la sombra proyectada sobre un píxel no modifica su información de color o cromaticidad. La segunda condición evalúa la diferencia en el canal S (saturación) entre *Foreground* y *Background*, siendo normalmente negativa para aquellos píxeles sombreados. La última condición evalúa la luminancia del píxel (canal V – brillo –), que debe ser menor para aquellos píxeles que se encuentren en regiones sombreadas.

El uso de **beta** (menor que uno) permite evitar la identificación como sombras de aquellos píxeles del *Foreground* con una pequeña variación con respecto al *Background* debido al ruido de la cámara, mientras que **alpha** determina la variación entre aquellos píxeles sombreados y no sombreados. Así, cuanto mayor sea la iluminación de la escena, menor será el valor **alpha** que se debe escoger.

La elección de los umbrales **alpha**, **beta**, **tau_h** y **tau_s** debe estimarse empíricamente, con el propósito de mejorar el comportamiento final del algoritmo, en función de las características de la escena. Dicha estimación se realizará más adelante, en el apartado 7.1.

6.3.2 ALGORITMO ESPECTRAL EN EL ESPACIO DE COLOR YCBCR

La implementación de este algoritmo, al igual que el anterior, contempla de nuevo dos etapas diferenciadas: conversión del espacio de color *RGB* al espacio *YCbCr* y proceso de decisión comparando *Background* y *Foreground* en los tres canales. Al igual que el espacio de color *HSV*, este espacio separa la luminancia (*Y*) de la crominancia (*Cb* y *Cr*). Por tanto, el proceso de decisión de este algoritmo es similar al ya expuesto anteriormente en la implementación del algoritmo *HSV*:

```
// Shadow Detection
valueMask = DiVAMask->getPixel(i,j);
if (valueMask.val[0] != 0)
{
        // Getting Pixel Data [BackGround and ForeGround]
        valueYCbCrBkg = DiVAYCbCrBkg->getPixel(i,j);
        valueYCbCrFrg = DiVAYCbCrFrg->getPixel(i,j);
        // Evaluating Shadow Suppression
        // 1. Cb Condition
        if ((valueYCbCrFrg.val[1] - valueYCbCrBkg.val[1]) <= tau_cb){
                // 2. Cr Condition
                if (valueYCbCrFrg.val[2] - valueYCbCrBkg.val[2] <= tau_cr){
                                // 3. Y Condition
                                if (valueYCbCrFrg.val[0]/valueYCbCrBkg.val[0] >= alpha &&
valueYCbCrFrg.val[0]/valueYCbCrBkg.val[0] <= beta){
                                        valueMask.val[0] = 0;
                                        DiVAMask->setPixel(i,j,valueMask);
                                }
                        }
        }
}
```

Figura 6-3: *ShadowSuppressorYCbCr*: Proceso de decisión.

6.3.3 ALGORITMO ESPACIAL HFOD

```
// Shadow Detection
valueMask = DiVAMask->getPixel(i,j);
if (valueMask.val[0] != 0)
{
        // Getting Pixel Data [BackGround and ForeGround]
        valueGrayBkg = DiVAGrayBkg->getPixel(i,j);
        valueGrayFrg = DiVAGrayFrg->getPixel(i+1,j);

        valueDiff = log(double(valueGrayBkg.val[0])+EPS)-
log(double(valueGrayFrg.val[0])+EPS);
        valueShadowMask= ((valueDiff*valueGrayFrg[0])/(valueGrayBkg.val[0]-
valueDiff*valueGrayBkg[0]+EPS));

        // Evaluating Shadow Suppression
        if (valueShadowMask >= alpha)
        {
                valueMask.val[0] = 0;
                DiVAMask->setPixel(i,j,valueMask);
        }
}
```

Figura 6-4: ShadowSuppressorHFOD: Proceso de decisión.

La implementación de este algoritmo basa su funcionamiento en la textura del píxel. Básicamente, se determina un mapa de relaciones de intensidad que guarda la información de la textura de la región. Basados en esta idea, un píxel podrá ser clasificado como sombra únicamente si su relación de intensidad en la imagen actual es similar a dicha relación en el Background.

6.4 IMPLEMENTACIÓN EN EL SISTEMA DiVA

En este apartado se desarrolla la nueva clase *DiVAShadowSuppressor*. Dicha clase aporta los métodos ya expuestos en el capítulo 3.3.3 y que permiten procesar la secuencia de video, eliminando las sombras en movimiento detectadas.

Adicionalmente, se desarrolla la aplicación de detección y eliminación de sombras, que contempla dos diferentes modos de funcionamiento, basándose en la integración de los módulos de análisis involucrados, en una sola aplicación o en varias, y enmarcados en la capa de procesado de la secuencia de video del sistema *DiVA*.

6.4.1 DESARROLLO DE LA CLASE DiVASHADOWSUPPRESSOR

A continuación se describen de modo funcional los principales métodos implementados en la nueva clase genérica denominada *DiVAShadowSuppressor*:

- **`DiVAShadowSuppressor::DiVAShadowSuppressor()`**

En el constructor de la clase *DiVAShadowSuppressor* se reservan e inicializan todos los recursos involucrados en el procesado de la secuencia de video:

- Objeto *OpenCVConverter*, con el que se realizan las conversiones entre los formatos *DiVAImage*, formato nativo del sistema *DiVA,* e *Ipl*, formato nativo de la librería *OpenCV* (**`pConverter`**).

- Objeto *DiVAImage*, donde se almacena la máscara procesada en la etapa de detección y eliminación de sombras (**`shadowMask`**).

- Objetos *DiVAImage*, donde se almacena, en caso necesario, la información extraída anteriormente durante la detección de objetos en movimiento (**`frg`**, **`bkg`**, **`movementMask`**).

- Objeto *DiVADataClient*, para el establecimiento de la conexión con el *DataServer*, y la adquisición de información extraída por otros módulos de procesado de la secuencia de video. (**`pMovementClient`**).

- **`virtual int DiVAShadowSuppressor::init();`**

Este método llama, en primer lugar, al método **`DiVAAlgorithm::init()`**, que establece una conexión con el *FrameServer*.

A continuación, se crea un objeto *DiVADataClient* que establece una conexión con el módulo *DataServer*, y se comprueba la disponibilidad de información (*Background*, *Foreground* y máscaras de movimiento) relacionada con dicha secuencia de video y extraída por algún algoritmo de detección de movimiento.

Dicha identificación se realiza mediante una llamada al método nativo de la clase *DiVADataClient* `pMovementClient->getFrameServerSession(srcIPAddress, srcPort)`, donde `srcIPAddress` y `srcPort` son los parámetros pasados como parámetros de entrada de la clase *DiVAShadowSuppressor*, y necesarios para el establecimiento de la conexión con el *FrameServer*. Devuelve 0 si todo es correcto o -1 si ocurre algún error.

- `int start();`

Este método llama, en primer lugar, al método `DiVAAlgorithm::start()`, con el propósito de comenzar la retransmisión de la secuencia de video capturada por la aplicación *FrameServer*.

A continuación, se hace una llamada al método nativo de la clase derivada *DiVADataClient* `pMovementClient->openAlgorithmSession( algName, srcIPAddress, srcPort, algXML)`, que registra la sesión *Algoritmo*, quedando preparado para la recepción de la secuencia extraída previamente en la capa de procesado de la secuencia de video. Los parámetros de entrada identifican el *Algoritmo* (nombre del algoritmo, dirección IP y puerto de servicio). Devuelve 0 si todo es correcto o -1 si ocurre algún error.

- `int stop();`

Este método llama, en primer lugar, al método `DiVAAlgorithm::stop()`, que detiene la recepción de la secuencia de video capturada por el *FrameServer*.

A continuación, se hace una llamada al método nativo de la clase derivada *DiVADataClient* `pMovementClient->closeAlgorithmSession()` que registra el cierre de sesión. Devuelve 0 si todo es correcto o -1 si ocurre algún error.

- `int DiVAShadowSuppressor::processFrame(DiVAImage *bkg, DiVAImage *frg, DiVAImage *mask)`

Este método está sobreescrito por el *Algoritmo* de detección y eliminación de sombras específico, que será el método encargado de procesar la secuencia de video, *frame* a *frame*, y almacena la máscara procesada en el objeto *DiVAImage* `shadowMask`. Devuelve 0 si todo es correcto o -1 si ocurre algún error.

- `int DiVAShadowSuppressor::processFrame(DiVAImage* frame,void* pdata,void* pContentData)`

Este método, que toma como parámetro la imagen adquirida, es utilizado cuando la información extraída durante el proceso de detección de movimiento se encuentre almacenada en el *DataServer*, y realiza la petición de *Background*, *Foreground* y máscaras de movimiento.

Para ello, extrae del objeto *DiVAImage* **`frame`** pasado como parámetro de entrada, el parámetro **`frameID`**, a partir del cual identifica unívocamente el *frame* y realiza las peticiones:

```
pMovementClient->receiveMask(frameID, bkg, BKG)
pMovementClient->receiveMask(frameID, frg, FRG)
pMovementClient->receiveMask(frameID, mask, MOVEMENT_MASK)
```

- `virtual CvArr *getShadowMask();`

Este método devuelve la máscara procesada por el algoritmo de detección y eliminación de sombras, en formato *DiVAImage*.

6.4.2 DESARROLLO DE LA APLICACIÓN EN EL SISTEMA DIVA

Dada la naturaleza de este módulo, el proceso se realizará basándose en los resultados obtenidos previamente por el módulo de estimación del movimiento.

Debido al diseño del nuevo componente, propuesto en el apartado anterior 6.4.1, nuestra nueva aplicación deberá ser capaz de adquirir la información extraída previamente de dos maneras claramente diferenciadas: embebiendo el módulo de detección y eliminación de sombras en la aplicación de detección de objetos en movimiento, y accediendo al contenido previamente almacenado en el módulo *DataServer*.

A continuación se describen las funcionalidades necesarias para la implementación de la aplicación, contemplando ambas situaciones:

- **CASO 1: INTEGRACIÓN DE MÓDULOS DE ANÁLISIS EN UNA SOLA APLICACIÓN**

Según este modo de funcionamiento, la integración de los distintos módulos de análisis o *Algoritmos* se realiza en una sola aplicación.

En el ámbito de desarrollo de este PFC, se opta por crear un nuevo objeto **`pShadowSuppressor`** dentro de la clase *DiVAFDGaussBkgExtractor*, algoritmo encargado de realizar el proceso de detección de objetos en movimiento.

Dicha integración se realizará en el siguiente método, a continuación del proceso de detección de objetos en movimiento:

```
DiVAFDGaussBkgExtractor::processFrame(DiVAImage* pImage, void*
pdata,void* pContentData)
```

Así pues, el esquema de funcionamiento de la nueva aplicación se resume funcionalmente en los siguientes puntos:

- Creación de un nuevo objeto *DiVAShadowSuppressor* **`pShadowSuppressor`**.

- Procesado del *frame* actual, a partir de los parámetros de entrada **`bkg`**, **`frg`** y **`mask`**.

- Obtención de la máscara procesada por el algoritmo de detección de sombras.

▪ CASO 2: INTEGRACIÓN DE MÓDULOS DE ANÁLISIS EN VARIAS APLICACIONES

En este caso, se considera el módulo de detección de sombras como una aplicación independiente dentro del sistema *DiVA*, y no relacionada directamente con la aplicación de detección de objetos en movimiento. Por tanto, la comunicación entre ambos módulos con el propósito de compartir la información extraída se realiza a través de un tercero: *DataServer*.

Así pues, el esquema de funcionamiento de la nueva aplicación se resume funcionalmente en los siguientes puntos:

- Creación de un nuevo objeto *DiVAShadowSuppressor* **`pShadowSuppressor`**.

- Inicialización de **`pShadowSuppressor`** (conexión con el *FrameServer* para la adquisición de la secuencia de video original y con el *DataServer* para la adquisición de la información extraída por el algoritmo de detección de objetos).

- Comienzo de la transmisión de la secuencia original.

- Adquisición de la información extraída y almacenada en el *DataServer* frame a frame:

```
pShadowSuppressorprocessFrame(DiVAImage* frame,void* pdata,void*
pContentData)
```

- Llamada al método de procesado de la imagen:

```
pShadowSuppressor->processFrame(DiVAImage *bkg, DiVAImage *frg,
DiVAImage *mask)
```

- Detención de la transmisión de la secuencia original.

7. PRUEBAS Y RESULTADOS

En este capítulo se presentan, en primer lugar, los resultados experimentales de la evaluación realizada sobre los distintos algoritmos de detección de sombras implementados, describiendo la metodología que se ha seguido.

Posteriormente, se realiza un análisis del comportamiento final del sistema, una vez añadidos los módulos desarrollados en las diferentes capas de las que se compone el sistema *DiVA* (adquisición, procesado y gestión de la secuencia de video).

7.1 RESULTADOS COMPARATIVOS DE LOS ALGORITMOS DE DETECCIÓN DE SOMBRAS IMPLEMENTADOS

En este apartado se presentan los resultados experimentales realizados durante la evaluación de los distintos algoritmos de detección de sombras implementados. Dichos algoritmos se han evaluado sobre una selección de secuencias disponibles en los *datasets* de *ATON* (http://cvrr.ucsd.edu/aton/shadow) y *AVSS 2007* (http://www.avss2007.org), en base a aquellas propiedades representativas durante la etapa de detección de sombras. La descripción de dichas secuencias se encuentra resumida en la Tabla 7-1.

Dataset	Secuencia	Tipo	Duración	Intensidad de la sombra	Tamaño de la sombra	Nivel de ruido
ATON	*Highway I*	Outdoor	1074	Media	Grande	Medio
ATON	*Intel. Room*	Indoor	900	Baja	Medio	Alto
AVSS 2007	*AB_Easy*	Indoor	2000	Baja	Pequeño	Bajo

Tabla 7-1: Descripción de las propiedades de las secuencias seleccionadas

La evaluación se ha realizado según los siguientes parámetros de medida de comportamiento: precisión en la detección de sombras (η) y precisión en la discriminación de sombras (ξ), ya descritos en el apartado 2.2.4.1.

$$\eta = \frac{TP_S}{TP_S + FN_S} \qquad \xi = \frac{\overline{TP_F}}{TP_F + FN_F}$$

Para evaluar y comparar el comportamiento de los algoritmos implementados, se han utilizado secuencias anotadas (34) por el Grupo de Tratamiento e Interpretación de Video (VPU-UAM), en las cuales se ha definido previamente qué píxeles se corresponden a objetos en movimiento, y qué píxeles se corresponden con regiones sombreadas (*Ground Truth*). Esto permite evaluar el comportamiento de los algoritmos de detección de sombras sin tener en cuenta los errores en la detección provocados durante la etapa de segmentación de movimiento.

A continuación se describe la metodología utilizada para la realización de la evaluación y posteriormente se resumen los resultados obtenidos para cada una de las secuencias evaluadas.

7.1.1 METODOLOGÍA DE LA EVALUACIÓN

Para la realización de esta evaluación se ha utilizado la secuencia original, el Background, así como el *Ground Truth*, o secuencia anotada, para determinar qué píxeles se corresponden con objetos en movimiento, y cuáles se corresponden con sombras en movimiento.

Figura 7-1: Escenario 1 - Highway I (*frame* 145): (i) imagen original, (c) Background, (d) Ground Truth

La metodología de la evaluación de los algoritmos ha consistido en realizar una estimación de los diferentes umbrales de decisión necesarios para cada uno de los canales del espacio de color correspondiente. Ambos algoritmos espectrales (*HSV* e *YCbCr*) presentan cuatro umbrales de decisión cada uno: dos para el canal de luminancia y dos para los canales de cromaticidad. En el caso del algoritmo espacial *HFOD*, tan sólo se ha considerado un umbral de decisión.

En primer lugar, se presenta un histograma de los valores del píxel detectados en movimiento (separando sombras de objetos) para cada uno de los canales del espacio de color. En la figura Figura 7-2 se presenta el histograma de los valores del píxel detectados en movimiento en el espacio de color HSV (azul: sombra, rojo: objeto)

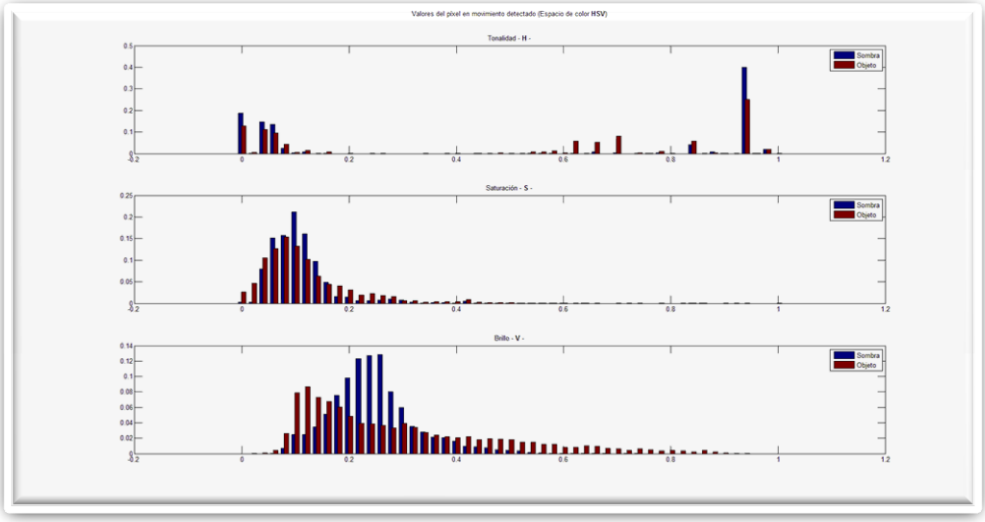

Figura 7-2: Valores del píxel detectado en movimiento: Histograma (Secuencia 1 - HSV)

Atendiendo a las condiciones de decisión del algoritmo de detección, se ha extraído posteriormente la información necesaria para la correcta estimación de los umbrales en cada uno de los canales del espacio de color:

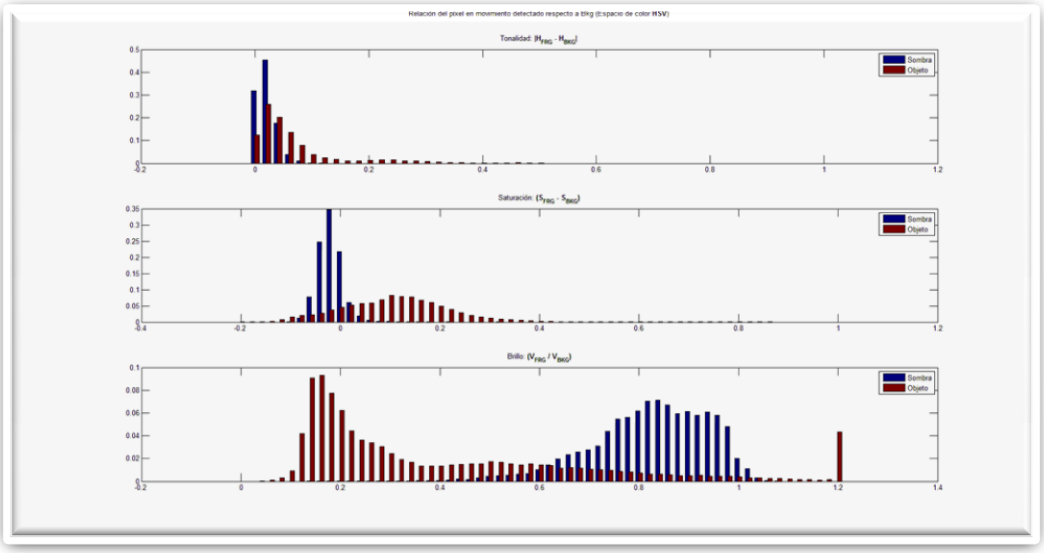

Figura 7-3: Relación del píxel en movimiento detectado respecto al Background: Histograma (Secuencia 3 - HSV)

En la Figura 7-3 se presenta un histograma de la relación del píxel en movimiento detectado respecto al Background en cada uno de los canales del espacio de color (azul: sombra, rojo: objeto). Dicha información acentúa las diferencias entre los objetos y las sombras detectadas, y sirve como referencia para la obtención de una primera estimación de los umbrales de decisión.

A continuación, se ha realizado un barrido de cada uno de los umbrales de decisión involucrados en el proceso de detección, canal por canal, para determinar el comportamiento del algoritmo, evaluándolo en función de las medidas de comportamiento propuestas: precisión en la detección de sombras ($\boldsymbol{\eta}$) y precisión en la discriminación de sombras ($\boldsymbol{\xi}$)**.**

La Figura 7-4 representa estas medidas en función de las variaciones del valor del umbral de decisión **`beta`**, donde la gráfica azul representa la precisión en la detección de sombras, y la verde la precisión en la discriminación de sombras.

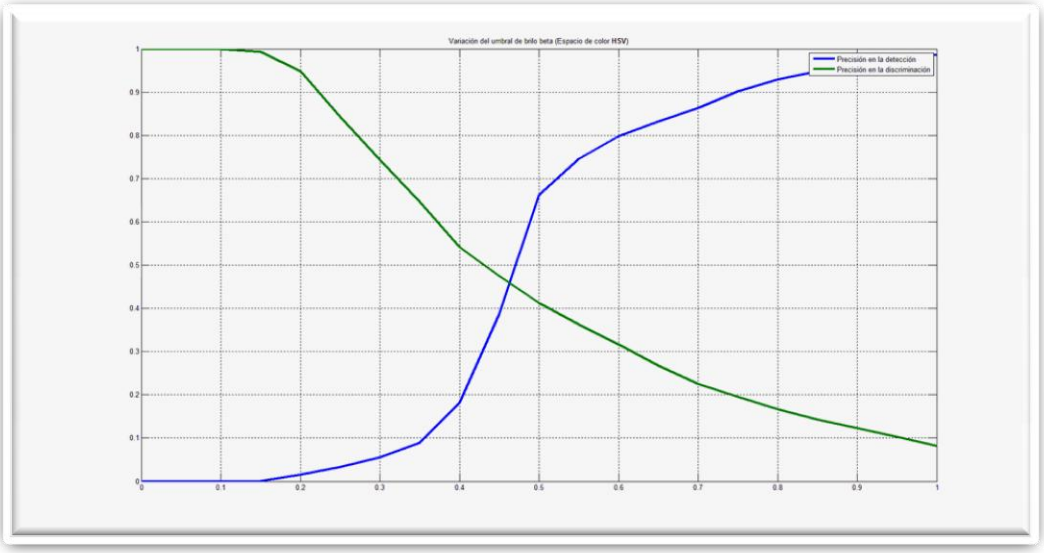

Figura 7-4: Variación del umbral de brillo beta (Secuencia 1 – HSV)

Finalmente, y una vez determinados los umbrales de forma independiente, se realiza de nuevo la evaluación, esta vez contemplando los tres canales de manera simultánea, y mediante un reajuste canal por canal, se maximiza la precisión, tanto en la detección, como en la discriminación de sombras.

7.1.2 EVALUACIÓN DE LOS ALGORITMOS

A continuación se resumen los resultados de la evaluación realizada de los distintos algoritmos, sobre las diferentes secuencias:

		HSV	YCbCr	HFOD
Escenario 1: Highway I	η%	81,96%	67,38%	70,02%
	ξ%	65,77%	65,84%	71,34%
	%	**53,91%**	**44,36%**	**49,95%**
Escenario 2: Intel. Room	η%	74,63%	76,56%	95,76%
	ξ%	86,58%	84,37%	44,09%
	%	**64,61%**	**64,59%**	**42,22%**
Escenario 3: AB_Easy	η%	80,62%	81,03%	98,71%
	ξ%	96,49%	96,07%	81,09%
	%	**77,79%**	**77,85%**	**80,04%**
		HSV	YCbCr	HFOD
Resultados Evaluación	%	**65,44%**	**62,27%**	**57,41%**

Tabla 7-2: Resultados comparativos (en %) de los algoritmos

El primer escenario (*Highway I*) es un escenario *outdoor* en el que predominan las sombras visibles, motivo principal por el cual el algoritmo *HSV* es capaz de obtener una buena precisión en la detección de sombras.

La evaluación del algoritmo *YCbCr* arroja unos resultados esperados: la precisión en la detección de sombras es peor, dado que este algoritmo es más apto para detectar sombras invisibles.

En ambos casos, y tal y como se puede apreciar en la Figura 7-2, dicha escena presenta muy poca información cromática, siendo los canales de luminancia, los únicos con diferencias significativas entre los objetos y las sombras detectadas, dificultando sensiblemente el proceso de discriminación de sombras, tanto con el algoritmo *HSV* como con el algoritmo *YCbCr*.

Por su parte, el algoritmo espacial *HFOD* obtiene unos resultados aceptables en la evaluación de esta secuencia, si bien no mejoran el comportamiento del algoritmo *HSV*.

El segundo escenario (*Intel. Room*), pese a tratarse de una escena *indoor* en la que predominan las sombras invisibles, posee dos características por las cuales tanto *HSV* como *YCbCr* presentan un buen comportamiento en la detección y discriminación de sombras, con unos resultados similares: la escena presenta abundante información cromática y la buena iluminación contrarresta los efectos negativos introducidos por el alto nivel de ruido.

El algoritmo *HFOD* presenta en este escenario el peor comportamiento de la evaluación. Aún ofreciendo una muy alta precisión en la detección de sombras, el alto nivel de ruido de la secuencia repercute en una peor precisión en la discriminación de sombras.

Por último, la tercera escena (*AB_Easy*) es la que presenta unos mejores resultados en la evaluación de los distintos algoritmos, si bien es cierto que es la que presenta una mayor diferencia entre las personas y objetos en movimiento y el *Background*.

Cabe destacar, que pese a ser el algoritmo espacial *HFOD* el que ofrece un ligeramente mejor comportamiento en la tercera escena, éste se ve reducido debido a la presencia de objetos de tamaño reducido, que dificultan el proceso de discriminación de las sombras en la escena.

En la Figura 7-5 se muestran las máscaras resultantes de los diferentes algoritmos de detección de sombras:

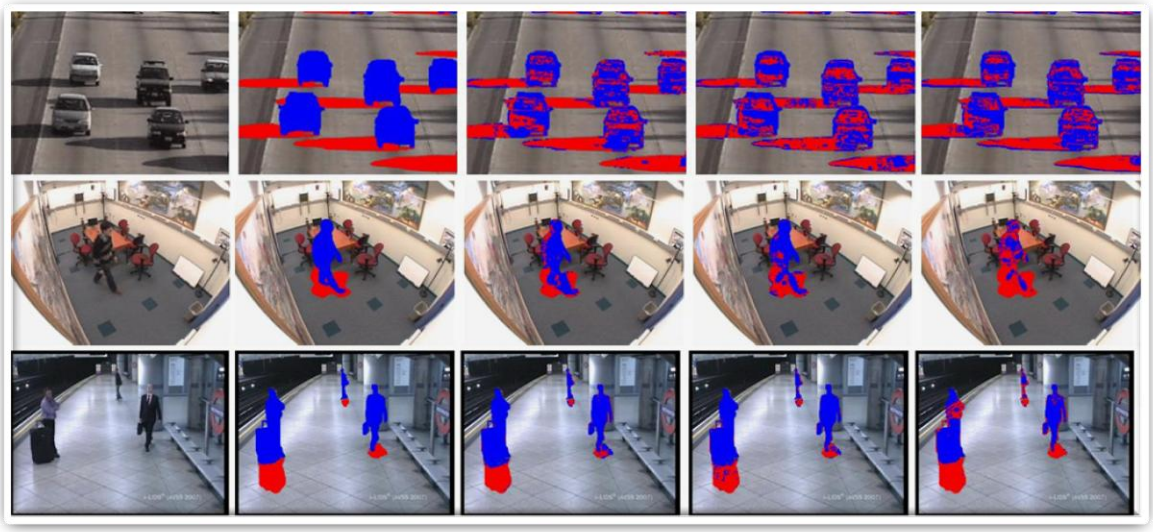

Figura 7-5: Resultados comparativos (de izquierda a derecha): imagen original, Ground Truth, HSV, YCbCr y HFOD

Como conclusión, de los algoritmos implementados, el algoritmo espectral en el espacio de color *HSV* es el que mejor comportamiento ofrece desde un punto de vista genérico, siendo capaz de eliminar correctamente tanto sombras invisibles como sombras visibles. Por tanto, este es el espacio de color adecuado para su explotación en escenarios de propósito general.

7.2 ANÁLISIS DEL COMPORTAMIENTO FINAL DEL SISTEMA

En este apartado se realiza un análisis del comportamiento final del sistema, una vez añadidos los módulos desarrollados en las diferentes capas de las que se compone el sistema *DiVA* (adquisición, procesado y gestión de la secuencia de video).

Para ello, se han evaluado cada uno de los componentes desde un punto de vista cualitativo, y siempre que ha sido posible, también desde un punto de vista cuantitativo. Dicho sistema se ha ejecutado en un ordenador Pentium IV con una frecuencia de procesador de 3 GHz y 1 GB de RAM.

- **MÓDULO CAPTURADOR DE DATOS A TRAVÉS DE CÁMARAS IP**

Dicho componente permite la adquisición de la secuencia de video proporcionada por cada una de las cámaras IP situadas en el Hall de la EPS, proveyendo dicho contenido a través de la red Ethernet del Grupo de Tratamiento e Interpretación de Video de la UAM (VPU-UAM).

Para secuencias de video con una resolución de 320x240 píxeles, la tasa de cuadros obtenida puede ser de hasta 25 *fps*, independientemente del códec de compresión de video utilizado.

Sin embargo, para secuencias de video con una resolución de 640x480 píxeles, la tasa de cuadros se reduce a 16 *fps* utilizando los *códecs MPEG4* y *H.264*. En el caso del códec de compresión *JPEG*, la tasa se reduce incluso a 8 fps. No obstante, mediante la modificación del parámetro *JPEG Quality* (de 10 a 8), se permite aumentar la tasa de cuadros hasta los 25 *fps* máximos que puede llegar ofrecer la cámara IP.

Dichos resultados quedan resumidos en la siguiente tabla:

		Resolución	
		320x240	640x480
CÓDEC DE COMPRESIÓN	MPEG4	*Hasta 25 fps*	*Hasta 16 fps*
	H.264	*Hasta 25 fps*	*Hasta 16 fps*
	JPEG (Q=10)	*Hasta 25 fps*	*Hasta 8 fps*
	JPEG (Q=9)	-	*Hasta 16 fps*
	JPEG (Q=8)	-	*Hasta 25 fps*

Figura 7-6: Resolución máxima en función del códec de compresión.

Por tanto es el códec de compresión *JPEG (Q=8)* el códec que se ha definido por defecto permitiendo unas mayores tasas de cuadros, dado que además la pérdida de calidad de la secuencia de video transmitida es prácticamente imperceptible.

También se ha evaluado el número máximo de cámaras conectadas desde un mismo ordenador (*JPEG Q=8* a 640x480 píxeles), arrojando los siguientes datos:

Nº de cámaras conectadas	Frame Rate servido
1 cámara	*25 fps*
2 cámaras	*25 fps*
3 cámaras	*17 fps*
4 cámaras	*12 fps*

Figura 7-7: *Frame Rate* servido

Si bien es cierto que hay que contemplar esta limitación, normalmente no va a ser un problema, dado que cada ordenador (o unidad funcional) no soportará más de dos cámaras conectadas a la vez.

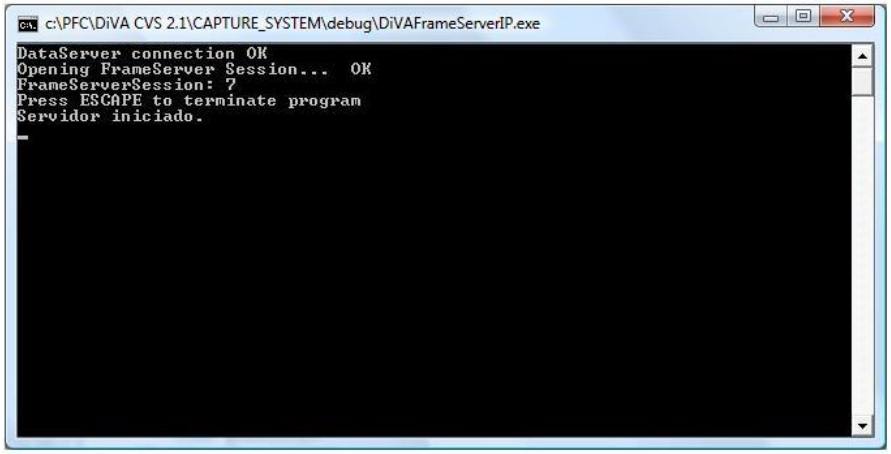

Figura 7-8: Ejecución de la aplicación *DiVAFrameServerIP*.

En definitiva, la integración de este componente en el sistema *DiVA* ofrece un comportamiento correcto, cumpliendo su propósito y en tiempo real. Sin embargo, cabe destacar que la secuencia de video captada presenta un alto nivel de ruido, lo que dificultará la posterior etapa de análisis y procesado de la secuencia de video.

- **MÓDULO DE GESTIÓN DE DATOS**

Se ha diseñado y desarrollado un prototipo que da soporte a las diferentes aplicaciones en las capas de adquisición y procesado de la secuencia, ofreciendo unas funcionalidades básicas al sistema *DiVA* (almacenamiento y adquisición de información extraída).

La evaluación de este componente es favorable, dado que cumple su cometido correctamente. Sin embargo, presenta como principal inconveniente una limitada interacción con el sistema *DiVA*, dificultando sensiblemente la tarea de búsqueda y selección de secuencias, y mermando las posibilidades que puede llegar a ofrecer este módulo.

Además, y dado el volumen de proceso del sistema, no se puede obtener un comportamiento en tiempo real almacenando toda la información extraída, si bien es cierto que es solucionable mediante la compresión de la información transmitida.

- **MÓDULO DE DETECCIÓN DE SOMBRAS**

En este apartado se ha evaluado cualitativamente el comportamiento del módulo de detección de sombras, que procesa la secuencia de video captada en la capa de adquisición por las cámaras IP, una vez se ha segmentado el movimiento.

Debido al diseño del componente, descrito en el apartado 6.4.2, existen dos modos de funcionamiento o casos de aplicación: integración de los módulos de análisis en una sola aplicación o en varias.

Dado que el primer caso es la situación más crítica, con un mayor consumo de recursos, se ha optado por analizar esta situación, embebiendo el módulo de detección y eliminación de sombras en la aplicación de segmentación de movimiento, ejecutando ambos módulos en cascada.

Las secuencias analizadas proceden de las cámaras IP. Sin embargo, y dado que la evaluación contempla diferentes situaciones, dichas secuencias han sido almacenadas previamente en el módulo de gestión de datos para su posterior procesado.

En primer lugar, se ejecuta la aplicación de segmentación de movimiento de manera individual, donde se puede observar el comportamiento del proceso. En la Figura 7-10 se muestran las ventanas de ejecución de dicha aplicación.

Como se puede apreciar, la etapa de detección de movimiento es buena. Sin embargo, la presencia del ruido en la secuencia de video captada repercute en una reducción en la precisión de detección de movimiento.

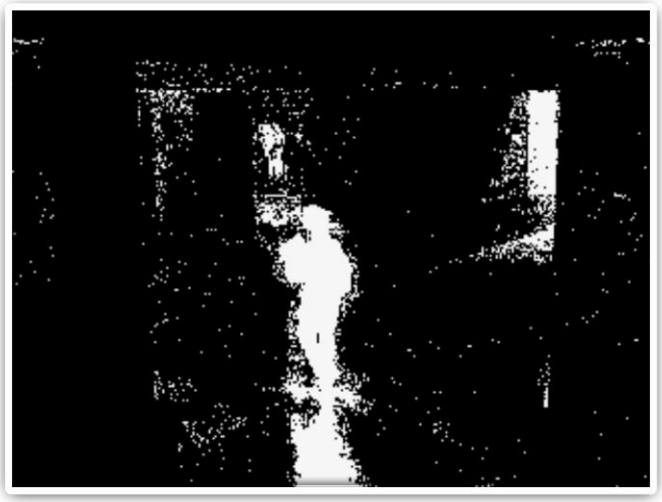

Figura 7-9: Máscara de segmentación de movimiento.

En la evaluación del módulo de detección y eliminación de sombras, se estimará, de manera cualitativa, la calidad de la detección. El motivo por el cual se ha decidido evaluar así la secuencia procesada se basa en la no disponibilidad de secuencias anotadas.

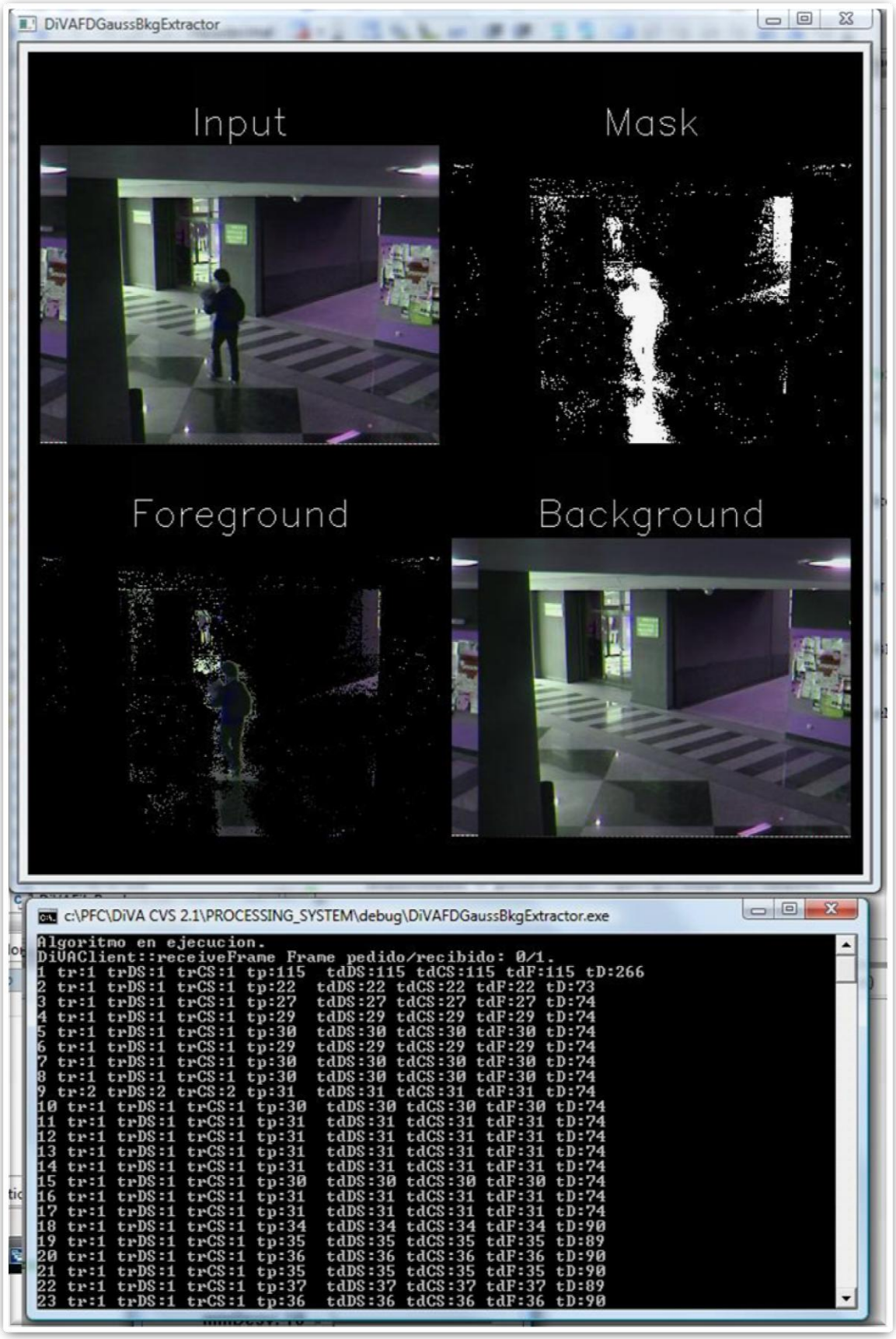

Figura 7-10: Ejecución de la aplicación de segmentación de movimiento.

Posteriormente, se procede a integrar el módulo de detección de sombras en la aplicación. Una vez realizada la estimación de los umbrales de decisión, los resultados obtenidos apuntan a una sensible mejora en la detección de objetos, tal y como se puede observar en la Figura 7-11:

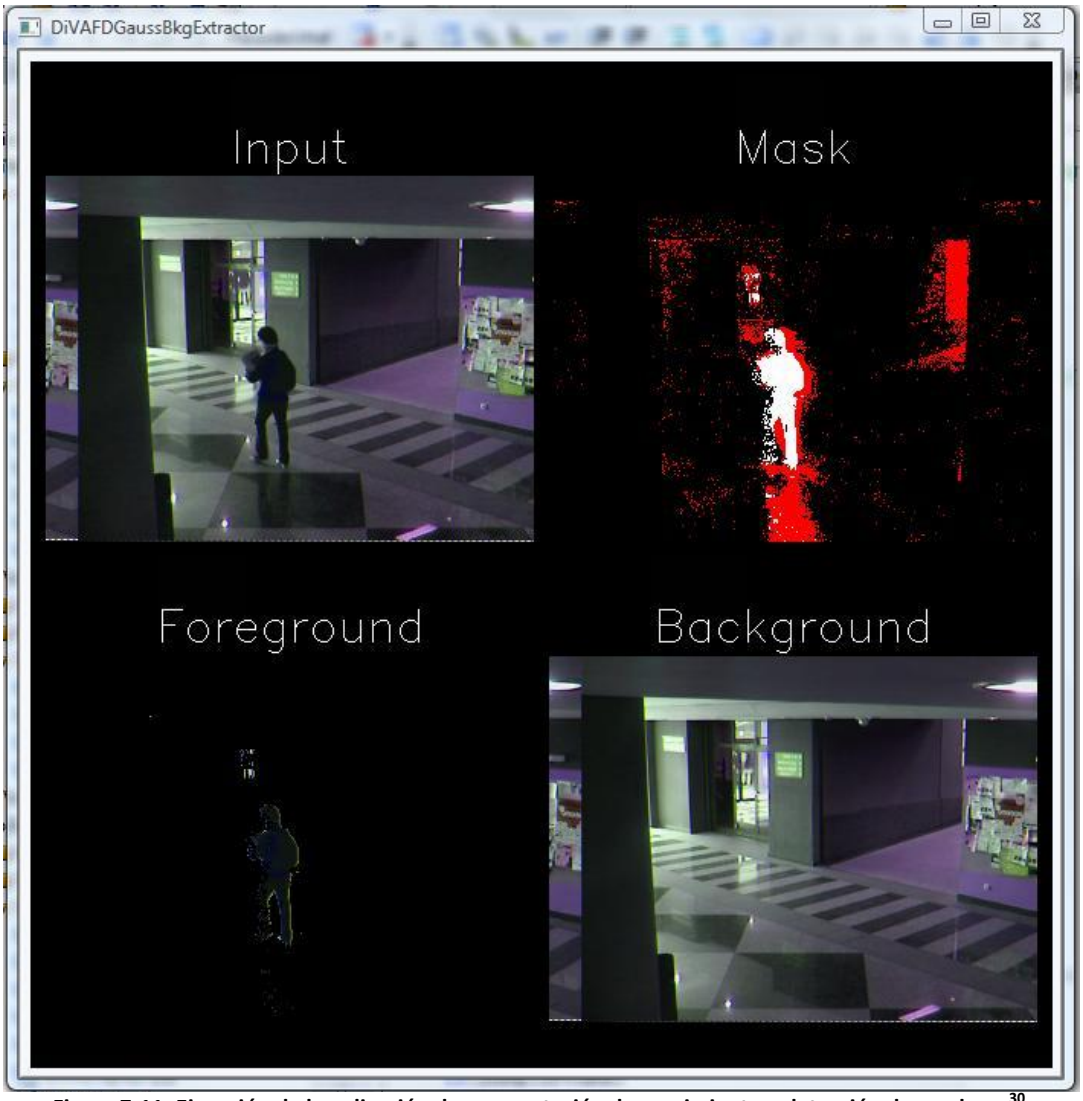

Figura 7-11: Ejecución de la aplicación de segmentación de movimiento + detección de sombras.[30]

Los resultados obtenidos son alentadores. Se ofrece una elevada precisión, tanto en la detección como en la discriminación de sombras, detectando y eliminando tanto el ruido presente en la secuencia como las sombras correctamente.

Por otro lado, el comportamiento de la aplicación final ofrece un análisis y procesado de la secuencia de video en tiempo real, ofreciendo una tasa de cuadros de hasta 16 *fps* en secuencias con una resolución de 640x480 píxeles, y 22 *fps* en secuencias con una resolución de 320x240 píxeles.

[30] **NOTA** Se han marcado en rojo aquellos píxeles detectados y eliminados durante la etapa de detección de sombras para poder discernir claramente los resultados ofrecidos en cada una de las etapas. No obstante, la aplicación final muestra únicamente la máscara resultante (en blanco y negro).

8. RESUMEN, CONCLUSIONES Y TRABAJO FUTURO

8.1 RESUMEN

Inicialmente, se contempló contribuir en la mejora de la plataforma de análisis distribuido *DiVA* en dos aspectos: ampliación del sistema existente y detección y eliminación de sombras en las secuencias de video captadas por el sistema. En general, las mejoras desarrolladas cumplen su propósito de manera satisfactoria.

En primer lugar, se ha ampliado la funcionalidad de la capa de adquisición de secuencias de video del sistema *DiVA* con una nueva fuente de datos proveniente de cámaras IP, ofreciendo un comportamiento en tiempo real a una tasa de cuadros de hasta 25 *fps* y permitiendo adquirir para su posterior procesado, de una manera sencilla, la información captada por las cámaras situadas en el Hall de la Escuela Politécnica Superior de la UAM.

En segundo lugar, se ha diseñado y desarrollado un módulo de gestión de datos generados por la plataforma *DiVA*. La principal aportación de este nuevo módulo no radica tanto en su aplicación final como componente del sistema, sino en demostrar su viabilidad ofreciendo una visión completa a lo largo de las etapas de diseño, desarrollo e integración de un nuevo componente mediante la utilización de dos tecnologías diferentes (*C++* y *MySQL*).

Si bien el módulo de gestión ofrece unas funcionalidades básicas, posibilitando la comunicación entre las diferentes aplicaciones que lo requieran, y centralizando toda la información de interés extraída y proporcionada por las capas de adquisición y procesado de la secuencia de video, presenta como principal inconveniente una limitada interacción con el usuario final, dificultando sensiblemente la tarea de búsqueda y selección de secuencias.

En tercer lugar, se ha presentado un estudio analítico de los algoritmos de detección de sombras a partir de la cual se ha realizado una selección y evaluación de las aproximaciones más representativas. La principal conclusión a la que se ha llegado es que en sistemas de propósito general, en los que el conocimiento de la escena es escaso, el método determinístico no basado en el modelo, (y en concreto, en el espacio de color *HSV*), asegura los mejores resultados.

Finalmente, se ha ampliado la funcionalidad de la capa de procesado de secuencias de video del sistema *DiVA* con un nuevo componente de análisis y detección de sombras. Dicho módulo mejora los resultados provistos durante la etapa de segmentación de movimiento, sin comprometer el funcionamiento en tiempo real.

8.2 CONCLUSIONES

Una vez realizado el diseño, desarrollo e implementación de los diferentes módulos propuestos, se han sacado unas conclusiones en relación a la funcionalidad de cada una de las mejoras.

La utilización de cámaras IP como fuente de adquisición de datos representa una mejora significativa en plataformas distribuidas de video – seguridad dada su sencillez de integración, así como su coste y escalabilidad, idónea en entornos distribuidos.

En cuanto a la inclusión de un módulo de gestión de datos, se hace necesario dada las funcionalidades que puede llegar a ofrecer, centralizando toda la información extraída y mejorando la distribución y comunicación entre procesos. Sin embargo, y en el ámbito de desarrollo de este PFC, tan sólo se han definido algunos aspectos y funcionalidades sencillas que redundan en una aportación básica al sistema, sin proporcionar la sinergia propuesta.

Existen además otros aspectos a considerar como la selección y compresión de la información previamente procesada y transmitida, con el objeto de reducir la carga de red soportada y aumentar el rendimiento final de los sistemas de video – seguridad, siempre tomando en consideración los diferentes aspectos de diseño y arquitectura propios de cada sistema.

En relación a la etapa de detección y eliminación de sombras en movimiento, ésta aporta una sensible mejora durante la etapa de segmentación de objetos en movimiento, reduciendo considerablemente los píxeles mal clasificados, debido tanto a las sombras, como al ruido presente en la secuencia.

Sin embargo, no existe una implementación válida y universal para todas las escenas. Existen diferentes algoritmos en función del propósito de cada uno de ellos. Algoritmos más precisos en la detección y eliminación de sombras requieren un alto coste computacional y, en la mayoría de los casos, exigen un alto conocimiento de la escena.

No obstante, existen otras aproximaciones más sencillas (desarrolladas en el sistema), que si bien ofrecen una precisión menor, su comportamiento en entornos genéricos ofrece una ventaja añadida en la segmentación de objetos en movimiento, facilitando, en parte, el propósito fundamental de los sistemas distribuidos de video – seguridad, que es el de proveer una interpretación automática de la escena.

En definitiva, el sistema desarrollado pone de manifiesto las posibilidades que ofrecen las tecnologías empleadas para crear sistemas de video – seguridad distribuidos.

8.3 TRABAJO FUTURO

Este PFC presenta unas mejoras desarrolladas, las cuales determinan la viabilidad funcional del sistema en su conjunto. Debido al desarrollo modular del sistema, cualquiera de los componentes puede ser susceptible de mejora, de manera independiente.

Si bien se ha diseñado y desarrollado un prototipo que dé soporte al sistema *DiVA*, existen diversas cuestiones que no se han tenido en cuenta. Por tanto, existen diversas cuestiones de un interés especial que podrían considerarse líneas de trabajo futuro, en relación a los siguientes componentes desarrollados:

- **MÓDULO DE GESTIÓN DE DATOS**

• Interfaz con funcionalidades avanzadas:

Se sugiere la realización de una interfaz más completa, permitiendo al usuario final observar y adquirir toda la información relacionada de una manera más sencilla e intuitiva, además de ofrecer otro tipo de información relacionada con algunos de los siguientes aspectos: configuración del sistema, parámetros de decisión óptimos para cada una de las escenas… y en definitiva, proveyendo funcionalidades avanzadas que puedan mejorar el rendimiento del sistema en su conjunto.

• Comunicación entre el módulo de gestión de datos y el sistema *DiVA*:

Se sugiere el planteamiento de una nueva solución de transmisión de la información extraída entre el módulo de gestión de datos y el sistema *DiVA*: diferentes técnicas de compresión y transmisión de video, implantación de técnicas de protocolo y red, son algunas de las cuestiones cuyo propósito es el de mejorar la seguridad en la transmisión así como reducir la carga en la red.

- **MÓDULO DE DETECCIÓN DE SOMBRAS**

• Criterio adaptativo de selección de espacio de color óptimo:

Se sugiere el desarrollo de una aplicación capaz de decidir, dinámicamente, el espacio de color óptimo para una correcta detección de sombras. Se propone además realizar un estimador de parámetros de decisión basándose en la información de la escena (iluminación, cromaticidad, texturas,…).

• Fusión de las etapas de segmentación de movimiento y detección de sombras:

Se sugiere el desarrollo de una aplicación capaz de realizar la segmentación de movimiento y detección de sombras de una manera conjunta, reduciendo el tiempo de ejecución, y permitiendo por tanto un análisis más preciso, mediante la extracción de un mayor número de características espectrales, espaciales y/o temporales.

REFERENCIAS

1. *IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS2010).*
2. *British Machine Vision Conference (BMVC2009).*
3. *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS2009).*
4. *http://www.masaumnet.com/specialissues/mjbas/bscs_mcma/index.html.*
5. *http://iris.usc.edu/Information/2010/avs-4-10-call.pdf.*
6. *http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=4608844.*
7. *M. Valera and S.A. Velastin: 'Intelligent distributed surveillance systems: a review' IEE Proc.-Vis. Image Signal Process., Vol. 152, No. 2, April 2005.*
8. *Nwagboso, C.: 'User focused surveillance systems integration for intelligent transport systems', in Regazzoni, C.S., Fabri, G., and Vernazza, G. (Eds.): 'Advanced Video-based Surveillance Systems' (Kluwer Academic Publishers, Boston, 1998), Chapter 1.1.*
9. *W.M.Hu, T.Tan, et al. 'A survey on visual surveillance of object motion and behaviors'. IEEE Transactions on Systems, Man, and Cybernetics - PART C: Applications and Reviews, 34(3):334-351, 2004.*
10. *Xiaojing, Y., Zehang, S., Yaakov, V., and George, B.: 'A distributed visual surveillance system'. Department of Computer Scince, University of Nevada, Reno, 2003.*
11. *Regazzoni, C.S., Ramesh, V., and Forest, G.L.: 'Special issue on video communications, processing, and understanding for third generation surveillance systems', Proc. IEEE, 2001, 89, (10), pp. 1355-1365.*
12. *MaKris, D., Ellis, T., and Black, J.: 'Bridging the gaps between cameras'. Int. Conf. Multimedia and Expo, Taiwan, June 2004.*
13. *Soldatini, F., Mähönen, P., Saaranen, M., and Regazzoni, C.S.: 'Network management within an architecture for distributed hierarchical digital surveillance systems' (Kluwer Academic Publishers, Boston, 2000), pp. 143-157.*
14. *Ye, H,. Walsh, G.C., and Bushnell, L.G.: 'Real-time mixed-traffic wireless networks', IEEE Trans. on Ind. Electron., 2001, 48, (5), pp. 883-890.*
15. *Marcenario, L., Oberti, F., Foresti, G.L., and Regazzoni, C.S.: 'Distributed architectures and logical-task decomposition in Multimedia surveillance systems', Proc. IEEE, 2001, 89, (10), pp. 1419-1438.*
16. *ADVISOR specification documents (internal classification 2001).*
17. *Mohan M. Trivedi, Ivana Mikic, and Greg Kogut: 'Distributed video networks for incident detection and management' Computer Vision and Robotics Research Laboratory, Univ. California, 2000.*
18. *A.M.Tekalp. Digital Video Processing. Prentice Hall Inc. 1995.*
19. *A. Prati et al., 'Detecting moving shadow: algorithms and evaluation', IEEE Transactions on Pattern Analysis and Machine Intelligence, 25:918-923, 2003.*
20. *Shan, Y., Yang, F.: 'Color space selection for moving shadow elimination', Fourth International Conference on Image and Graphics, IEEE 2007.*
21. *Prati, A., Mikic, I., Cucchiara, R., Trivedi, M.: 'Comparative Evaluation of Moving Shadow Detection Algorithms', 2001.*
22. *T. Horprasert, D. Harwood, and L.S. Davis: 'A statistical approach for real-time robust background substraction and shadow detection', in Proceedings of IEEE ICCV'99 FRAME-RATE Workshop, 1999.*

23. *Mikic, I., Cosman, P., Kogut, G., and Trivedi, M.: 'Moving shadow and object detection in traffic scenes', in Proceedings of Int'l Conference on Pattern Recognition, Sept. 2000.*
24. *Cucchiara, R., Grana, C., Piccardi, M., and Prati, A.: 'Detecting objects, shadows and ghosts in video streams by exploiting color and motion information', in Proceedings of the IEEE Int'l Conference on Image Analysis and Processing, to appear, 2001.*
25. *Herodotou, N., Plataniotis, K.N., and Venetsanopoulos, A.N.: 'A color segmentation scheme for object-based video coding', in Proceedings of the IEEE Symposium on Advances in Digital Filtering and Signal Processing, 1998, pp.25-29.*
26. *Gevers, T., and Stokman, H.: 'Classifying color edges in video into shadow-geometry, highlight, or material transitions', IEEE Trans on multimedia 5 (2003), 237-243.*
27. *Cucchiara, R., Grana, C., Piccardi, M., Prati, A., and Sirotti, S.: 'Improving shadow suppression in moving object detection with HSV color information', in Proceedings of IEEE Int'l Conference on Intelligent Transportation Systems, Aug. 2001, pp. 334-339.*
28. *Martel-Brisson, N., and Zaccarin, A.: 'Moving cast shadow detection from a Gaussian mixture shadow model', IEEE CVPR 2005, Vol. 20-26, no.2, June 2005, pp. 643-648.*
29. *Rautiainen, M., Ojala, T., and Kauniskangas, H.:' Detecting perceptual color changes from sequential images for scene surveillance, IEICE Transactions on Information and Systems, E84-D, 2001, pp. 1676-1683.*
30. *Yang, M.T., Lo, K.H., Chiang, C.C., and Tai, W.K.: 'Moving cast shadow detection by exploiting multiple cues', IET Image Process., 2008, Vol. 2, pp. 95-104.*
31. *Benedek, Cs., and Szirányi, T.,: 'Color models of shadow detection in video scenes', in Proceedings of VISAPP, Barcelona, Spain, vol. IFP/IA (2007), pp. 225-232.*
32. *Leone, A., Distante, C., and Buccolieri, F.: 'A texture-based approach for shadow detection', pp.371-376, 2005, IEEE Conference on Advanced Video and Signal Based Surveillance.*
33. *San Miguel, J.C., Bescós, J., Martínez, J.M., García, A.: 'DiVA: A distributed video analysis framework applied to video - surveillance systems'.*
34. *Juan C. SanMiguel, José M. Martínez: "Shadow detection in video surveillance by maximizing agreement between independent detectors", ICIP 2009 .*
35. *San Miguel Avedillo, J.C.: 'Transmisión de secuencias de video a tasa binaria muy baja y adaptable basada en generación y transmisión de descripciones', 2006.*
36. *Tadashi Nakanishi and Kenichiro Ishiim: 'Automatic vehicle image extraction based on spatio-temporal image analysis', NTT Human Interface Laboratories 2003 Japan.*
37. *Farin, D. de With, P.H.N. Effelsberg, W.: 'Robust background estimation for complex video sequences'. Image Processing, 2003. ICIP 2003. Proceedings.*
38. *Lipton, A.J., Fujikoshi, H., and Pati, R.S.: 'Moving target classification and tracking from real-time video', in Proc. IEEE WOrkshop Applications of Computer Vision, 1998, pp. 8-14.*
39. *Meyer, D., Denzler, J., and Niemann, H.: 'Model based extraction of articulated objects in image sequences for gait analysis', in Proc. IEEE Int. Conf. Image Processing, 1998, pp. 78-81.*

GLOSARIO

API

Interfaz de programación de aplicaciones. (Del inglés *Application Programming Interface*)

Background

Sección de la escena que permanece estática. También denominado fondo de imagen.

BLOB

Tipo de datos *MySQL* utilizado para almacenar archivos. (Del inglés *Binary Large Object*)

BMP

Formato de imagen sin compresión, que representa el mapa de bits (o matriz de píxeles) de la imagen (Del inglés *Bit Mapped Picture*).

C++

Lenguaje de programación orientado a objetos.

Callback

Código ejecutable que lleva a cabo iteraciones sobre objetos con propiedades específicas para llevar a cabo operaciones que desde el punto de vista del sistema son equivalentes, pero sobre elementos distintos.

CCTV

Circuito cerrado de televisión (Del inglés *Closed Circuit Television*)

CGI

Tecnología que permite transferir datos entre el cliente (navegador WEB) y el programa ejecutado en un servidor WEB. (Del inglés *Common Gateway Interface*)

DiVA

Sistema de procesado distribuido de video (Del inglés *Distributed Video Analysis*), enmarcado dentro del proyecto ATI@SHIVA: Algoritmos de Tratamiento de Imágenes para Sistema Homogéneo e Inteligente de video – vigilancia avanzada. (http://www-vpu.ii.uam.es/~ati-shiva/).

DLL

Biblioteca de enlace dinámico. Término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda de un programa por parte del sistema operativo. (Del inglés *Dinamic-Link Library*)

DNM

Método determinístico no basado en el modelo (Del inglés *Deterministic Non-based Model*)

FIREWIRE 1394

Estándar multiplataforma para entrada /salida de datos en serie a gran velocidad.

Foreground

Sección de la escena donde se encuentra el objeto o persona en movimiento.

fps

Tasa de cuadros. (Del inglés *frames per second*)

Ground Truth

Secuencias anotadas, en las cuales se ha definido previamente qué píxeles se corresponden a objetos en movimiento, y qué píxeles se corresponden con regiones sombreadas.

H.264

También denominado MPEG-4 parte 10 es una norma que define un códec de vídeo de alta compresión.

HFOD

Máscara de la derivada de primer orden horizontal. (*Del inglés Horizontal First – Order Derivative Mask*)

IEEE

Instituto de Ingenieros Electricistas y Electrónicos (*Del inglés Institute of Electrical & Electronics Engineers, Inc.)*

JPEG

Formato de imagen con compresión con pérdida (Del inglés *Joint Photographic Experts Group*).

MPEG - 4

MPEG-4 es el nombre de un grupo de estándares de codificación de audio y video así como su tecnología relacionada.

MySQL

Sistema de gestión de base de datos relacional, multihilo y multiusuario.

OpenCV

Librería libre, multiplataforma, desarrollada en C++, de visión artificial.

PCA

Análisis de los componentes principales (Del inglés *Principal Component Analysis*).

SDK

Conjunto de herramientas de desarrollo que permiten la creación de aplicaciones para un sistema concreto. (Del inglés *Software Development Kit*)

SNP

Método estadístico no paramétrico (Del inglés *Statistical Non-Parametric approach*)

SP

Método estadístico paramétrico (Del inglés *Statistical Parametric approach*)

STL

Recopilación genérica de plantillas y algoritmos para facilitar la implementación de estructuras de datos complejas. (Del inglés *Standard Template Library*)

Streaming

Transmisión de contenido en tiempo real.

Timestamp

Secuencia de caracteres, que denotan la hora y fecha (o alguna de ellas) en la cual ocurrió determinado evento.

USB

Estándar multiplataforma para entrada /salida de datos en serie a gran velocidad. (Del inglés *Universal Serial Bus*)

VPU-UAM

Grupo de Tratamiento e Interpretación de Video de la UAM.

ANEXO A: DETECCIÓN DEL MOVIMIENTO

La extracción o segmentación de movimiento en una secuencia de imágenes tiene como objetivo detectar aquellas regiones correspondientes a diferentes objetos en movimiento.

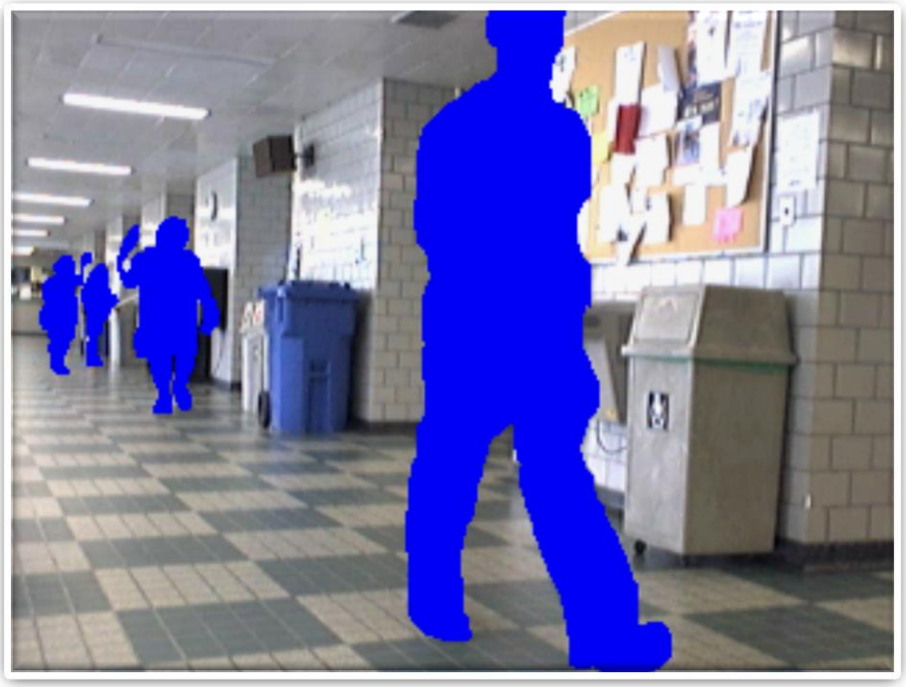

Figura A-0-1: Máscara de segmentación de movimiento

Actualmente, la mayoría de los métodos de segmentación de movimiento están basados en la información espacial y/o temporal de la secuencia. Algunos de los métodos más comúnmente utilizados se resumen a continuación, y se destacan uno o varios estudios característicos de cada uno de éstos:

- **SUSTRACCIÓN DE BACKGROUND**

Este método es quizás el más utilizado en la segmentación de movimiento, utilizado especialmente en aquellas situaciones en las que el *Background* es persistente o, en cierto modo, estático. Su funcionamiento se basa en detectar aquellas regiones en movimiento de la imagen tomando la diferencia, píxel a píxel, entre la imagen adquirida y el *Background* utilizado como referencia.

Dicho método parte con la ventaja de su simplicidad, pero como contraprestación, la correcta generación, adquisición y actualización de un modelo de *Background* se vuelve crítica en los sistemas de video – seguridad. No se podrá realizar una detección de movimiento eficiente si no se implementa un buen algoritmo de generación de *Background* que modele la escena correctamente.

El principal problema radica en la adquisición automática y posterior actualización de la imagen de fondo *(Background)* a partir de una secuencia de video adquirida por la cámara.

Existen diversos factores que dificultan este proceso:

- Cambios de iluminación (tanto graduales como repentinos).
- Cambios de movimiento (vibraciones de la cámara, movimiento de objetos con alta frecuencia,…).
- Cambios en la geometría del fondo (por ejemplo: coches recién aparcados).

Por tanto, la sustracción de *Background* se vuelve inevitablemente dependiente de un buen modelado del escenario para reducir la influencia de estos cambios. Para intentar solucionar estos problemas se han desarrollado una gran cantidad de algoritmos y así luego poder realizar una correcta estimación de movimiento en la secuencia analizada. En (35) quedan recogidas algunas de las diversas técnicas más utilizadas, que se exponen a continuación:

- Método *Running Gaussian Average*:
  - Se intenta ajustar una distribución *gaussiana* $(\boldsymbol{\mu}, \boldsymbol{\sigma})$ sobre el histograma, obteniendo así la función de densidad de probabilidad del *Background*.

- Método *Mixture of Gaussians:*
  - Basándose en el método anterior, se considera un ajuste a una mezcla de *gaussianas* ponderadas $(\boldsymbol{\mu_i}, \boldsymbol{\sigma_i}, \boldsymbol{w_i})$. Con este método se consigue hacer frente a distribuciones multimodales de *Backgrounds*.

- Método *Kernel Density Estimators:*
  - La función de densidad de probabilidad del *Background* se obtiene a partir del histograma de los *n* últimos valores.
  - Si $\boldsymbol{fdp}(\boldsymbol{x}) > \boldsymbol{T_h}$, entonces el píxel es clasificado como *Background*.
  - Este método implica selectividad[31] y grandes requerimientos de memoria *(n*size(frame))*.

- Método *Mean Shift Based Estimation*:
  - El método, iterativo, detecta los modos de una distribución multimodal.
  - Para ello usa un gradiente ascendente y la matriz de covarianzas.

[31] **Selectividad** Para cada nuevo *frame*, los píxeles son clasificados como *Background* o primer plano (*Foreground*). Si son clasificados como *Foreground*, entonces no se toman en cuenta para la actualización del *Background*.

- Método *EigenBackgrounds:*

  - Mediante el análisis de los componentes principales (PCA[32]) se reduce la dimensionalidad de un espacio.
  - Este análisis se aplica a una secuencia de *n frames* para computar los *autobackgrounds (eigenbackgrounds)*. Dichos *frames* son computados una sola vez utilizando conceptos como matriz de covarianzas, autovalores *(eigenvalues)*, autovectores *(eigenvectors)*,…

En la siguiente tabla se resumen las diferentes técnicas expuestas:

Método	Tiempo de ejecución	Requerimiento en memoria
Running Gaussian Average	Rápido	Bajo
Mixture of Gaussians	Intermedio	Intermedio
Kernel Density Estimators	Intermedio	Alto
Mean Shift Estándar	Lento	Alto
Mean Shift Optimizado	Intermedio	Alto
Eigenbackgrounds	Intermedio	Intermedio

Tabla A- 1; Comparación de métodos de generación y actualización de Background.

Hay que destacar que la precisión con la que se obtienen los resultados es muy dependiente de la secuencia de imágenes que se esté analizando así como el ruido introducido por la cámara. No obstante, en (36; 37) se realizan implementaciones de estos métodos en diversos sistemas, y los que ofrecen un mejor comportamiento son los métodos *Kernel Density Estimators* y *Mean Shift* Optimizado.

- **DIFERENCIA TEMPORAL**

La diferencia temporal hace uso de las diferencias respecto a un píxel durante el transcurso de dos o tres *frames* consecutivos en aquella secuencia de imágenes a partir de la cual se desean extraer las zonas en movimiento.

En (38), se presenta una solución que implementa este método, y que se basa en los cambios de nivel de ruido de las secuencias captadas por la cámara.

Se distinguen dos tipos de variaciones en la secuencia: las debidas al efecto del ruido y las debidas al movimiento de los objetos. Para distinguir y clasificar cada variación, se aplica un umbral de medida de similitud entre *frames* consecutivos. A dicha medida de similitud se le suma el efecto del ruido, caracterizado por su varianza.

Este método, muy adaptativo en entornos cambiantes, es, sin embargo, menos eficaz extrayendo todos aquellos píxeles relevantes (como por ejemplo: dejando zonas aisladas en el interior de regiones en movimiento).

[32] **PCA** Análisis de los componentes principales (Del inglés *Principal Component Analysis).*

Para corregir esta desventaja, incluye un post-proceso durante la etapa de detección de movimiento en el que, mediante el análisis de las distintas secciones en movimiento detectadas, éstas son reagrupadas en función de las características de las regiones extraídas anteriormente.

Merece la pena destacar el énfasis realizado acerca de la actualización dinámica de los parámetros así como la varianza caracterizada del ruido, necesario para que el sistema responda coherentemente a las variaciones de las condiciones de la escena, como pueden ser variaciones de iluminación o cambios de temperatura. Para ello, la mejor estrategia consiste en actualizar éstos en aquellos momentos en los que no se detecte ningún (o poco) movimiento en la escena.

- **DETECCIÓN DE CONTORNOS**

En el artículo (36) se presenta un sistema para identificar vehículos en secuencias de imágenes en las que las cámaras recogen la información en distintos tramos de autopistas.

La detección de movimiento se lleva a cabo aplicando la Transformada *Hough*, un algoritmo estadístico orientado a detectar contornos con formas geométricas simples. La detección de rectas se basa en la ecuación de la recta en el plano. Por cada punto de gradiente de la imagen se aumenta el valor de un punto en un espacio equivalente de parámetros de una recta. De este modo, las zonas en el nuevo espacio de alta intensidad se corresponden con rectas presentes en la imagen original, y detectar cualquier objeto que pueda caracterizarse mediante una ecuación.

- **DESPLAZAMIENTO DE VECTORES**

La etapa segmentación de movimiento basada en este método se caracteriza por el uso de vectores de flujo de aquellos objetos en movimiento a lo largo del tiempo para detectar regiones en movimiento a partir de una secuencia de imágenes.

En su trabajo, (39) calcula el desplazamiento a partir del campo de vectores correspondiente a la secuencia de imágenes y obtiene, mediante un algoritmo de seguimiento, la extracción de objetos en movimiento.

Este método puede utilizarse además para detectar objetos en movimiento incluso con una cámara en movimiento. Sin embargo, es computacionalmente complejo y muy sensible al ruido, por lo que no puede ser utilizado en secuencias de imágenes en tiempo real sin *hardware* especializado.

ANEXO B: INVARIANCIA DE LOS DISTINTOS ESPACIOS DE COLOR

En el caso de las sombras visibles, el valor del píxel será varias veces menor que en otras regiones iluminadas de la escena. Y de acuerdo con el modelo de sombra expuesto en el apartado 2.2.2, ante un incremento (o decremento) de la intensidad de la luz, varios canales de los distintos espacios de color mantendrán sus características espectrales. Es decir, serán invariantes a la escala *(Scaling Invariant)*:

$$\begin{aligned}
\boldsymbol{H}(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &= \boldsymbol{H}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{S}(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &= \boldsymbol{S}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{c1}(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &= \boldsymbol{c1}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{c2}(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &= \boldsymbol{c2}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{c3}(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &= \boldsymbol{c3}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{l1}(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &= \boldsymbol{l1}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{l2}(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &= \boldsymbol{l2}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{l3}(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &= \boldsymbol{l3}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{r}(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &= \boldsymbol{r}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{g}(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &= \boldsymbol{g}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{b}(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &= \boldsymbol{b}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B})
\end{aligned}$$

Así pues, se puede asumir que la sombra visible podrá ser eliminada en estos canales, ya que se mantendrán las características espectrales de la escena, y por tanto no existirá diferencia entre aquellas regiones, estén sombreadas o iluminadas.

De manera análoga, y para sombras invisibles, los valores del píxel serán ligeramente menores con respecto al resto de escena del *Background*. Y de nuevo, de acuerdo con el modelo de sombra definido, algunos canales serán también invariantes al desplazamiento *(Shifting Invariant)*:

$$\begin{aligned}
\boldsymbol{H}(\boldsymbol{R} + \boldsymbol{\Delta I}, \boldsymbol{R} + \boldsymbol{\Delta I}, \boldsymbol{R} + \boldsymbol{\Delta I}) &= \boldsymbol{H}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{Cb}(\boldsymbol{R} + \boldsymbol{\Delta I}, \boldsymbol{R} + \boldsymbol{\Delta I}, \boldsymbol{R} + \boldsymbol{\Delta I}) &= \boldsymbol{Cb}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{Cr}(\boldsymbol{R} + \boldsymbol{\Delta I}, \boldsymbol{R} + \boldsymbol{\Delta I}, \boldsymbol{R} + \boldsymbol{\Delta I}) &= \boldsymbol{Cr}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{l1}(\boldsymbol{R} + \boldsymbol{\Delta I}, \boldsymbol{R} + \boldsymbol{\Delta I}, \boldsymbol{R} + \boldsymbol{\Delta I}) &= \boldsymbol{l1}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{l2}(\boldsymbol{R} + \boldsymbol{\Delta I}, \boldsymbol{R} + \boldsymbol{\Delta I}, \boldsymbol{R} + \boldsymbol{\Delta I}) &= \boldsymbol{l2}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{l3}(\boldsymbol{R} + \boldsymbol{\Delta I}, \boldsymbol{R} + \boldsymbol{\Delta I}, \boldsymbol{R} + \boldsymbol{\Delta I}) &= \boldsymbol{l3}(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B})
\end{aligned}$$

Así pues, se puede asumir que la sombra invisible podrá ser eliminada en estos canales, ya que se mantendrán las características espectrales de la escena, y por tanto no existirá diferencia entre aquellas regiones, estén sombreadas o iluminadas.

Analizando las características de la sombra en el espacio de color *L*a*b*, se aprecia claramente que no es invariante a la escala. Esto es,

$$\begin{aligned}
\boldsymbol{L}^*(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &\neq \boldsymbol{L}^*(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{a}^*(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &\neq \boldsymbol{a}^*(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{b}^*(\boldsymbol{\alpha R}, \boldsymbol{\alpha G}, \boldsymbol{\alpha B}) &\neq \boldsymbol{b}^*(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B})
\end{aligned}$$

Por tanto, este espacio no es adecuado para detectar sombras visibles. Sin embargo, y según (20), dicho espacio de color está caracterizado como la compresión no lineal de las coordenadas del espacio de color *XYZ*, $f(q)$. Y para $q \leq 0.008856$, $f(q) = 7.787q + 0.1379$. De esta manera: *X*, *Y* y *Z* deberán satisfacer la siguiente inecuación:

$$\boldsymbol{X \leq 2.25828, Y \leq 2.25828, Z \leq 2.25628}$$

Así pues, para intensidades lumínicas débiles, dicha inecuación será resoluble, y por tanto la sombra invisible podrá ser detectada y eliminada en los canales *a*b*.*

Como conclusión, y asumiendo reflexión dicromática e iluminación blanca, la tonalidad *H*, saturación *S*, canales *c1c2c3*, *l1l2l3*, y *rgb normalizado*, serán invariantes a la iluminación. La tonalidad *H*, saturación *S*, *c1c2c3*, *l1l2l3* y *rgb normalizado* serán invariantes a la escala, y la tonalidad *H*, *Cb*, *Cr*, *a**, *b** y *l1l2l3* serán invariantes al desplazamiento.

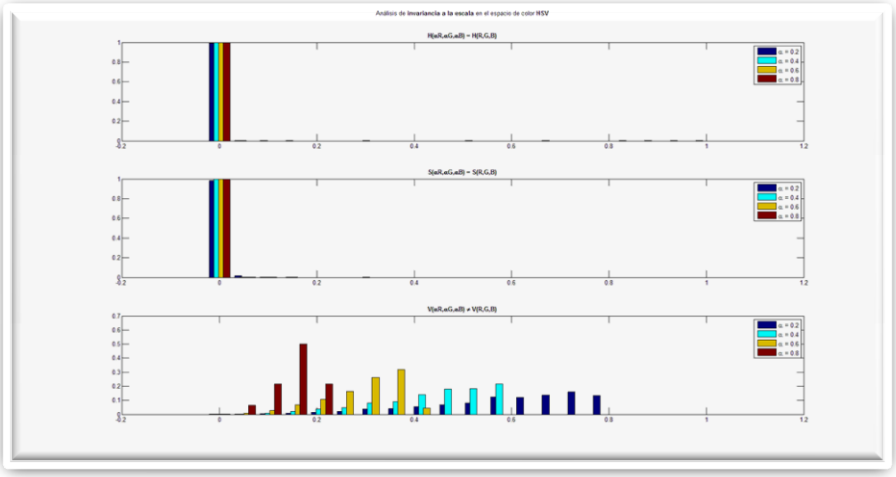

Figura A- 1: Análisis de invariancia a la escala en el espacio de color HSV

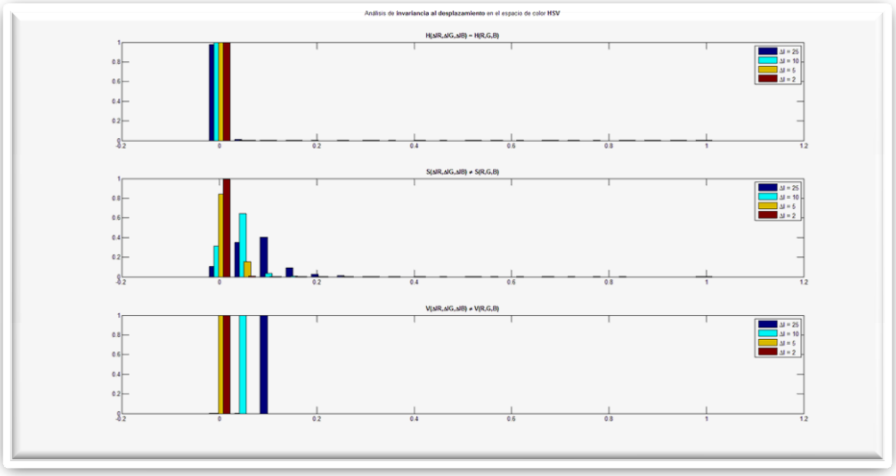

Figura A- 2: Análisis de invariancia al desplazamiento en el espacio de color HSV

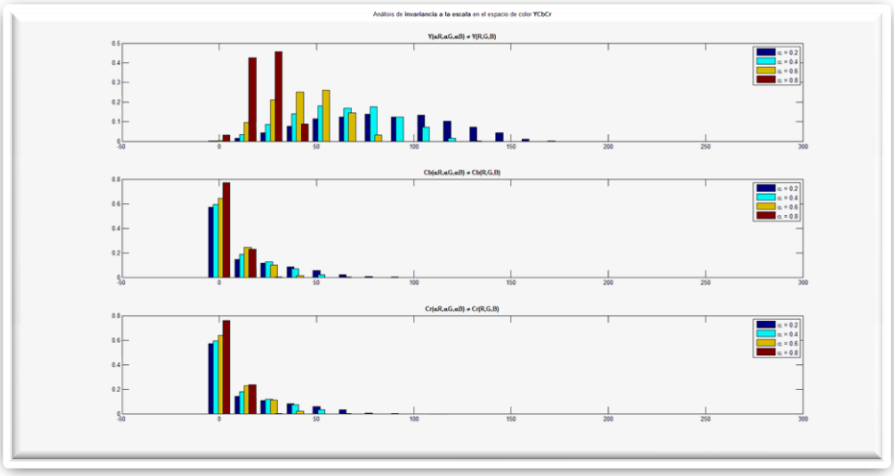

Figura A- 3: Análisis de invariancia a la escala en el espacio de color YCbCr

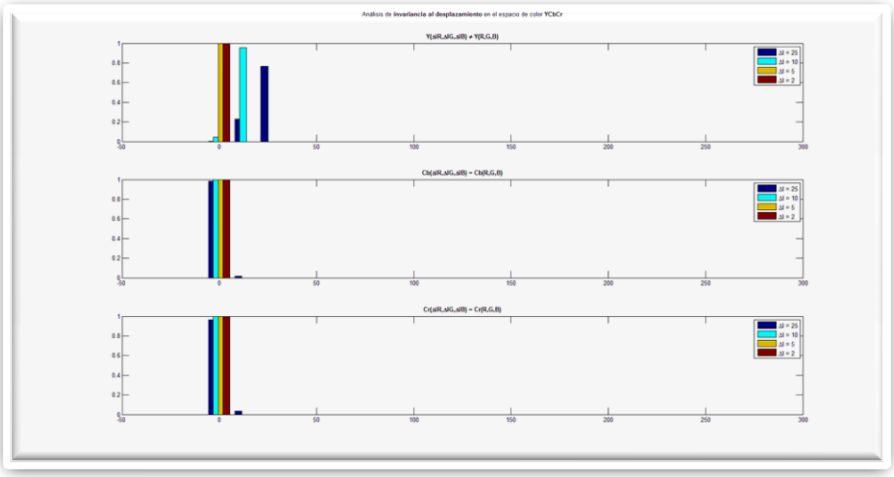

Figura A- 4: Análisis de invariancia al desplazamiento en el espacio de color YCbCr

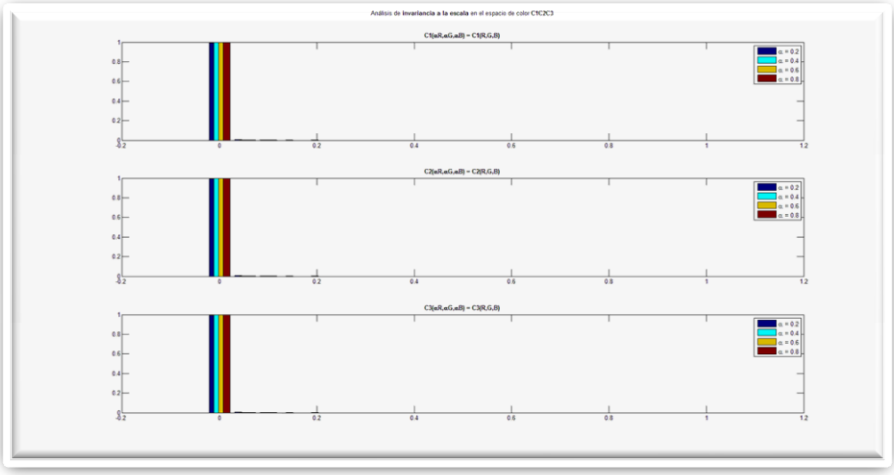

Figura A- 5: Análisis de invariancia a la escala en el espacio de color C1C2C3

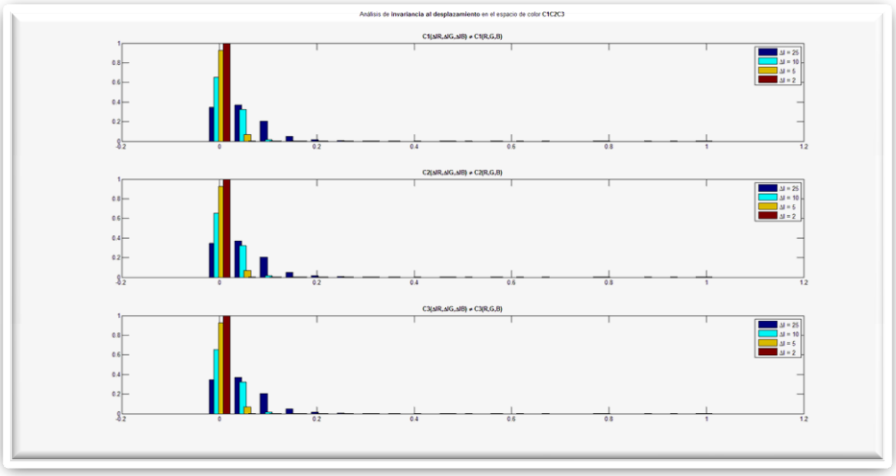

Figura A- 6: Análisis de invariancia al desplazamiento en el espacio de color C1C2C3

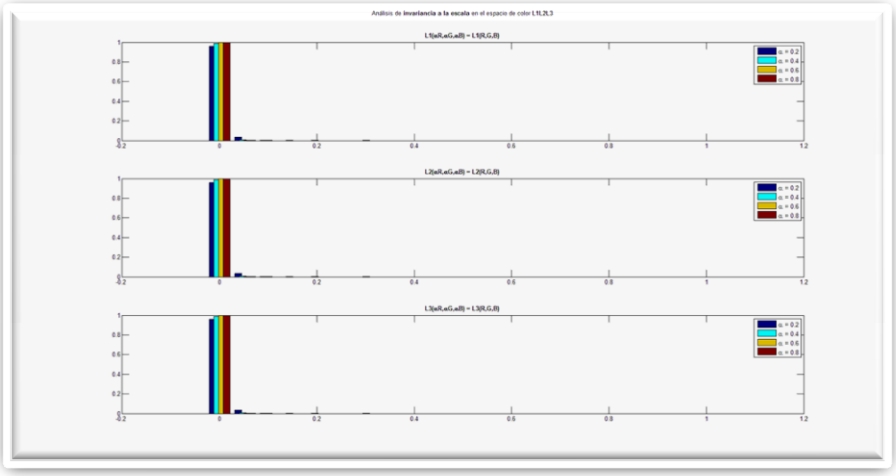

Figura A- 7: Análisis de invariancia a la escala en el espacio de color L1L2L3

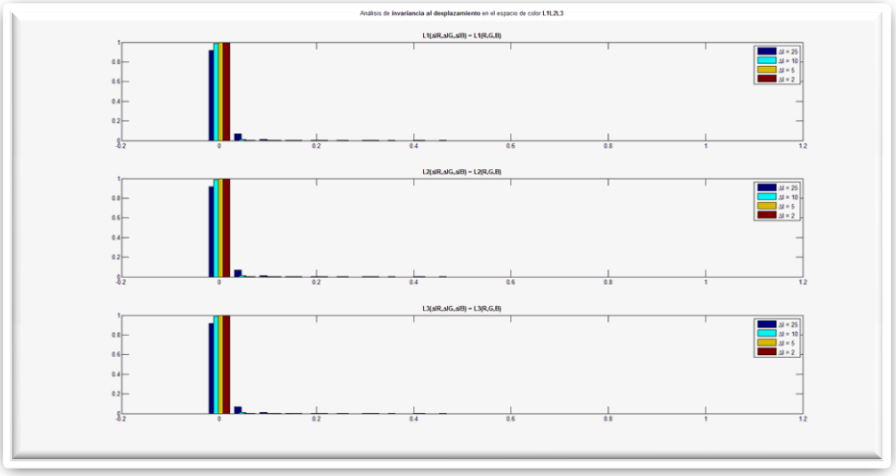

Figura A- 8: Análisis de invariancia al desplazamiento en el espacio de color L1L2L3

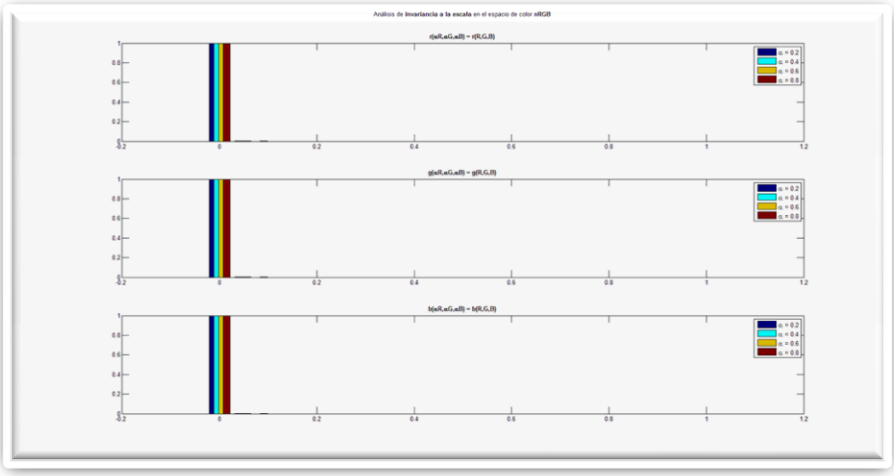

Figura A- 9: Análisis de invariancia a la escala en el espacio de color nRGB

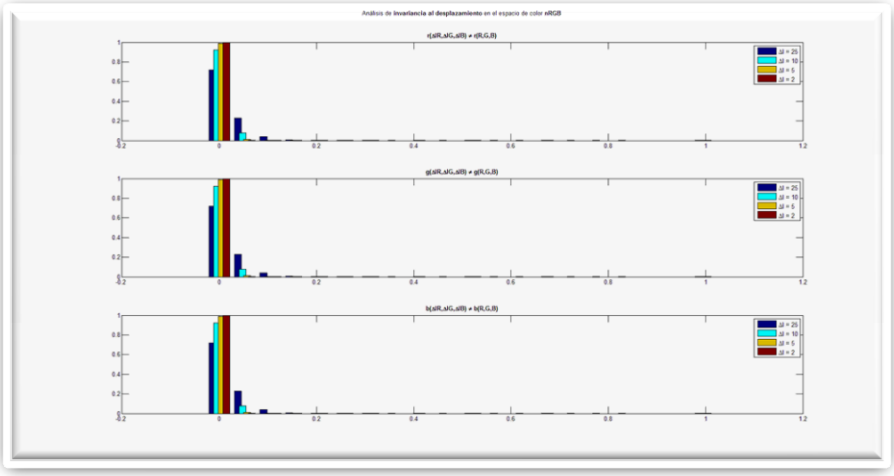

Figura A- 10: Análisis de invariancia al desplazamiento en el espacio de color nRGB

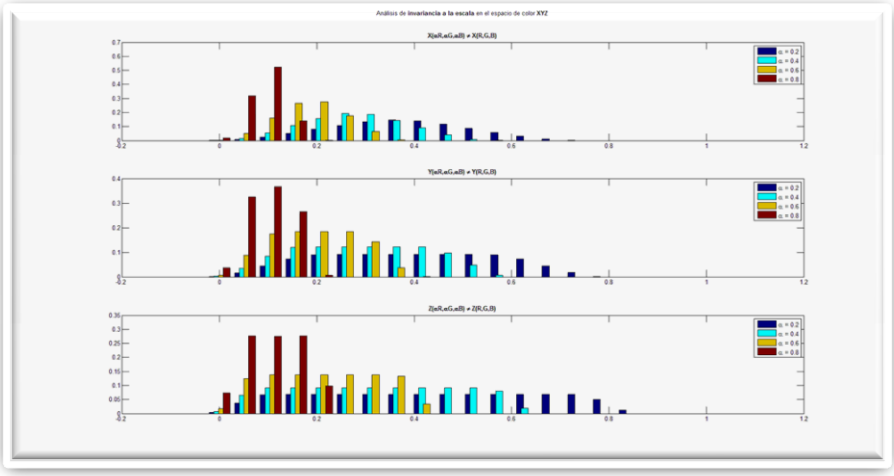

Figura A- 11; Análisis de invariancia a la escala en el espacio de color XYZ

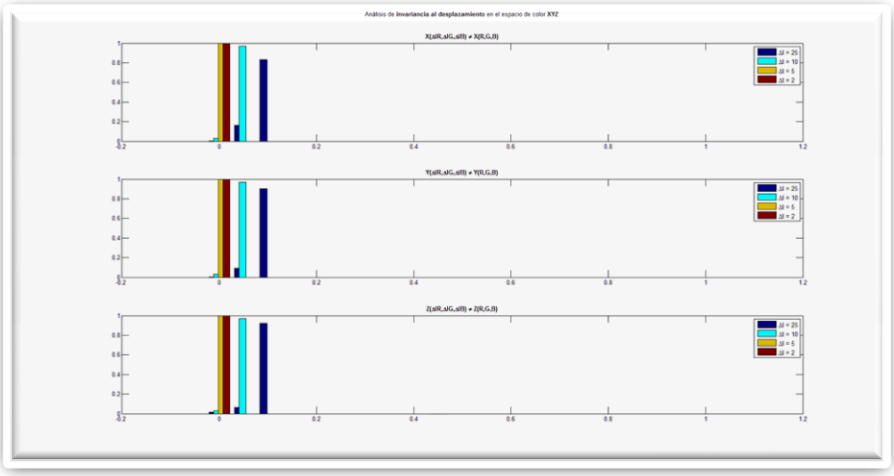

Figura A- 12: Análisis de invariancia al desplazamiento en el espacio de color XYZ

ANEXO C: CRITERIO DE SELECCIÓN DE ESPACIO DE COLOR

En (30) se propone utilizar el espacio de color *RGB*, debido a que es el más común en las aplicaciones de video, para seleccionar el espacio de color óptimo.

Considerando que la diferencia lumínica entre aquellas regiones donde se encuentren sombras invisibles y el *Background* será menor que la diferencia entre las regiones de sombra visible y *Background*, se define un simple algoritmo de decisión y selección del espacio de color:

$$\boldsymbol{D_s(x,y)} = \begin{cases} \mathbf{1}\ , & \boldsymbol{I_o(x,y) - B(x,y) \geq Th_s} \\ \mathbf{0}\ , & \boldsymbol{resto} \end{cases}$$

$$\boldsymbol{D_l(x,y)} = \begin{cases} \mathbf{1}\ , & \boldsymbol{I_o(x,y) - B(x,y) \geq Th_l} \\ \mathbf{0}\ , & \boldsymbol{resto} \end{cases}$$

$$\boldsymbol{\xi} = \frac{\sum \boldsymbol{D_l}}{\sum \boldsymbol{D_s}}$$

Donde $\boldsymbol{Th_s}$ y $\boldsymbol{Th_l}$ son umbrales de decisión y $\boldsymbol{Th_s} < \boldsymbol{Th_l}, \boldsymbol{I_o}$ es la luminancia de la imagen actual y **B** es la luminancia del *Background*.

Para aquellas regiones donde se encuentre sombra invisible, ésta será detectada por el umbral $\boldsymbol{Th_s}$, pero no por el umbral $\boldsymbol{Th_l}$. O lo que es lo mismo: $\boldsymbol{D_s = 1, D_l = 0}$. De manera análoga, y para sombras visibles, éstas serán detectadas por ambos umbrales $\boldsymbol{Th_s}$, $\boldsymbol{Th_l}$ y por tanto $\boldsymbol{D_s = 1, D_l = 1}$.

De acuerdo con el estudio, y según sus experimentos, cuando $\boldsymbol{\xi \leq 0.95}$, se considera que la sombra en dicha escena se trata de sombra invisible, y por tanto se utilizarán los espacios de color *YCbCr*, *L*a*b*, *c1c2c3* y *l1l2l3* para detectar y eliminar las sombras. Y cuando $\boldsymbol{\xi \geq 0.95}$, se asumirá la presencia de una escena en la que predominan las sombras visibles. En este caso, utilizan los espacios de color *HSV*, *c1c2c3*, *l1l2l3* y *rgb normalizado*.

ANEXO D: DISEÑO DE LA BASE DE DATOS

En este anexo se detalla la implementación del diseño propuesto, generando la nueva base de datos utilizando la tecnología *MySQL*, creando las tablas anteriormente nombradas y definiendo las relaciones entre éstas, así como los tipos de datos para cada campo.

- **TABLA frameservers**

Dicha tabla contiene aquellos parámetros identificativos de cada uno de los *FrameServers* disponibles en el sistema.

```
-- Tipos de frameservers
--
CREATE TABLE frameservers (
fsid    TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
fsname  VARCHAR(32) NOT NULL UNIQUE,
fsabout VARCHAR(255),

FULLTEXT    (fsname),
PRIMARY KEY (fsid)
)ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Figura A-0-2: Estructura MySQL de la tabla frameservers

Donde **`TINYINT UNSIGNED`** es un tipo de datos **`INTEGER`** (de 0 a 255), y **`VARCHAR()`** es una cadena de caracteres variable.

La clave primaria de dicha tabla es **`fsid`**, una variable autoincremental que identificará unívocamente a cada fuente de video (capa de adquisición) en concreto.

En el ámbito de desarrollo del sistema propuesto en el presente PFC, incluiremos como datos iniciales las tres cámaras IP que disponemos:

```
INSERT INTO frameservers VALUES (NULL, "CamaraIP_Hall_E", "Cámara
situada en el Hall de la primera planta (Este)");
INSERT INTO frameservers VALUES (NULL, "CamaraIP_Hall_N", "Cámara
situada en el Hall de la primera planta (Norte)");
INSERT INTO frameservers VALUES (NULL, "CamaraIP_Hall_W", "Cámara
situada en el Hall de la primera planta (Oeste)");
```

Por tanto, cada aplicación *FrameServer* debe conocer su **`fsid`** para poder identificarse correctamente en la base de datos.

- **TABLA framesessions**

Dicha tabla contiene aquellos parámetros relativos a la conexión de un *FrameServer* a la base de datos.

```
-- Control de acceso de frameservers e información relevante acerca de
--
CREATE TABLE framesessions (
fssessionid    INT UNSIGNED NOT NULL AUTO_INCREMENT,
fsid           TINYINT UNSIGNED NOT NULL,
framerate      TINYINT UNSIGNED NOT NULL,
srcipaddress   VARCHAR(32) NOT NULL,
```

```
srcport         SMALLINT UNSIGNED NOT NULL,
dstipaddress    VARCHAR(32) NOT NULL,
dstport         SMALLINT UNSIGNED NOT NULL,
datetimelogin   DATETIME NOT NULL,
datetimelogout DATETIME,

FOREIGN KEY     (fsid) REFERENCES frameservers (fsid),
PRIMARY KEY     (fssessionid)
)ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Figura A-0-3: Estructura MySQL de la tabla framesessions

Donde **`INT UNSIGNED`** es un tipo de datos **`INTEGER`** (de 0 a 429.4967.295), **`SMALLINT UNSIGNED`** es un tipo de datos **`INTEGER`** (de 0 a 65.535) y **`DATETIME`** es el formato de almacenamiento de fecha y hora.

El campo **`fsid`** es una clave foránea que hace referencia, en la tabla **`frameservers`**, a un único *FrameServer*. La clave primaria de dicha tabla es **`fssessionid`**, una variable autoincremental que identificará unívocamente a una sesión en concreto.

- **TABLA images**

Dicha tabla contiene aquellos *frames* provisionados en la capa de adquisición de la secuencia de video.

```
-- Imagenes originales almacenadas en la base de datos
--
CREATE TABLE images (
fssessionid     INT UNSIGNED NOT NULL,
frameid         INT UNSIGNED NOT NULL,
frametimestamp INT UNSIGNED NOT NULL,
frameadded      DATETIME NOT NULL,
framewidth      SMALLINT UNSIGNED NOT NULL,
frameheight     SMALLINT UNSIGNED NOT NULL,
framedepth      SMALLINT UNSIGNED NOT NULL,
framenchannels SMALLINT UNSIGNED NOT NULL,
framecolormodel VARCHAR(32) NOT NULL,
framevalign     TINYINT UNSIGNED NOT NULL,
framedatasize   INTEGER UNSIGNED NOT NULL,
framecompresseddatasize INTEGER UNSIGNED NOT NULL,
framepixels     MEDIUMBLOB NOT NULL,

FOREIGN KEY     (fssessionid) REFERENCES framesessions (fssessionid),
PRIMARY KEY     (fssessionid, frameid)
)ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Figura A-0-4: Estructura MySQL de la tabla images

Donde **`MEDIUMBLOB`** es un tipo de datos **`BLOB`**[33] con un tamaño variable de hasta 16.777.215 caracteres (utilizando el set de caracteres **`utf8`**, unos 50 MB). Éste es el tipo de datos utilizado para almacenar las imágenes adquiridas.

El campo **`fssessionid`** es una clave foránea que hace referencia, en la tabla **`framesessions`**, a una única sesión (y ésta a su vez identifica a un único *FrameServer*).

La clave primaria de dicha tabla es la dupla (**`fssessionid, frameid`**), que identifica unívocamente a un *frame* correspondiente a la secuencia de video abierta con el identificador **`fssessionid`**.

[33] **BLOB** Tipo de datos MySQL utilizado para almacenar archivos. (Del inglés *Binary Large Object*)

- **TABLA algorithms**

Dicha tabla contiene aquellos parámetros identificativos de cada uno de los *Algoritmos* disponibles en el sistema.

```
-- Tipos de algoritmos
--
CREATE TABLE algorithms (
algid          TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
algname VARCHAR(32) NOT NULL UNIQUE,
algabout       VARCHAR(255),

FULLTEXT       (algname),
PRIMARY KEY    (algid)
)ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Figura A-0-5: Estructura MySQL de la tabla algorithms

La clave primaria de dicha tabla es **`algid`**, una variable autoincremental que identificará unívocamente a un algoritmo en concreto.

En el ámbito de desarrollo del sistema propuesto en el presente PFC, incluiremos como datos iniciales los algoritmos de detección de objetos en movimiento así como los de detección de sombras:

```
INSERT INTO algorithms VALUES (NULL, "DiVAFDGaussBkgExtractor",
"Algoritmo de detección de objetos en movimiento");
INSERT INTO algorithms VALUES (NULL, "DiVAShadowSuppressor",
"Algoritmo de detección y eliminación de sombras");
```

Por tanto, cada algoritmo debe conocer su **`algid`** (o su nombre) para poder identificarse correctamente en la base de datos.

- **TABLA algorithmsessions**

Dicha tabla contiene aquellos parámetros relativos a la conexión de un *algoritmo* a la base de datos.

```
-- Control de acceso de los algoritmos e informacion relevante
--
CREATE TABLE algorithmsessions (
algsessionid   INT UNSIGNED NOT NULL AUTO_INCREMENT,
fssessionid    INT UNSIGNED NOT NULL,
algid          SMALLINT UNSIGNED NOT NULL,
srcipaddress   VARCHAR(32) NOT NULL,
srcport        SMALLINT UNSIGNED NOT NULL,
datetimelogin  DATETIME NOT NULL,
datetimelogout DATETIME,
algxml         TEXT,

FULLTEXT       (algxml),
FOREIGN KEY    (algid) REFERENCES algorithms (algid),
FOREIGN KEY    (fssessionid) REFERENCES framesessions (fssessionid),
PRIMARY KEY    (algsessionid)
)ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Figura A-0-6: Estructura MySQL de la tabla algorithmsessions

Donde **`TEXT`** es un tipo de datos que permite almacenar una cadena de caracteres con un tamaño de hasta 65.535 caracteres.

El campo **algid** es una clave foránea que hace referencia, en la tabla **algorithms**, a un único *algoritmo*. El campo **fssessionid** es otra clave foránea que hace referencia, en la tabla **framesessions**, a una única sesión. De esta manera, se podrá identificar qué algoritmo se corresponde con qué secuencia de video captada.

La clave primaria de dicha tabla es **algsessionid**, una variable autoincremental que identificará unívocamente a una sesión iniciada por un algoritmo en concreto.

- **TABLA masktypes**

Dicha tabla contiene los diferentes tipos de máscara generados en la capa de procesado de la secuencia de video.

```
-- Tipos de máscaras
--
CREATE TABLE masktypes (
masktypeid     TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
maskname       VARCHAR(32) NOT NULL UNIQUE,
maskabout      VARCHAR(255),

FULLTEXT       (maskname),
PRIMARY KEY    (masktypeid)
)ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Figura A-0-7: Estructura MySQL de la tabla masktypes

La clave primaria de dicha tabla es **masktypeid**, una variable autoincremental que identificará unívocamente a un tipo de máscara en concreto.

En el ámbito de desarrollo del sistema propuesto en el presente PFC, se incluirán como datos iniciales los siguientes:

```
INSERT INTO masktypes VALUES (NULL, "BKG", "Background generado
durante la detección de objetos en movimiento");
INSERT INTO masktypes VALUES (NULL, "FRG", "Foreground generado
durante la detección de objetos en movimiento");
INSERT INTO masktypes VALUES (NULL, "Movement_Mask", "Máscara de
detección de movimiento generada");
INSERT INTO masktypes VALUES (NULL, "Shadow_Mask", "Máscara de
detección de sombras en movimiento generada");
```

- **TABLA masks**

Dicha tabla contiene aquellas máscaras generadas en la capa de procesado de la secuencia de video.

```
-- Máscaras almacenadas en la base de datos
--
CREATE TABLE masks (
algsessionid   INT UNSIGNED NOT NULL,
frameid        INT UNSIGNED NOT NULL,
masktypeid     TINYINT UNSIGNED NOT NULL,
masktimestamp  INT UNSIGNED NOT NULL,
maskadded      DATETIME NOT NULL,
maskwidth      SMALLINT UNSIGNED NOT NULL,
maskheight     SMALLINT UNSIGNED NOT NULL,
maskdepth      SMALLINT UNSIGNED NOT NULL,
```

```
masknchannels  SMALLINT UNSIGNED NOT NULL,
maskcolormodel VARCHAR(32) NOT NULL,
maskvalign     TINYINT UNSIGNED NOT NULL,
maskdatasize   INT UNSIGNED NOT NULL,
maskcompresseddatasize INT UNSIGNED NOT NULL,
maskpixels     MEDIUMBLOB NOT NULL,

FOREIGN KEY    (algsessionid) REFERENCES algorithmsessions (algsessionid),
FOREIGN KEY    (frameid) REFERENCES images (frameid),
FOREIGN KEY    (masktypeid) REFERENCES masktypes (masktypeid),
PRIMARY KEY    (algsessionid, frameid, masktypeid)
)ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Figura A-0-8: Estructura MySQL de la tabla masks

El campo **algsessionid** es una clave foránea que hace referencia, en la tabla **algorithmsessions**, a una única sesión (y ésta a su vez identifica a un único *algoritmo*). El campo **frameid** es otra clave foránea que hace referencia, en la tabla **images**, a un *frame* en concreto. El campo **masktypeid** también es una clave foránea que hace referencia, en la tabla **masktypes**, a los tipos de máscara / descriptores.

La clave primaria de dicha tabla es la tripleta (**algsessionid**, **frameid**, **masktypeid**), que identifica unívocamente al tipo de máscara / descriptor relacionada con el *frame* correspondiente a la secuencia de video, y generada durante la sesión de procesado abierta con el identificador **algsessionid**.

- **TABLA descriptors**

Dicha tabla contiene aquellos descriptores generados en la capa de procesado de la secuencia de video.

```
-- Descriptores almacenados en la base de datos
--
CREATE TABLE descriptors (
algsessionid   INT UNSIGNED NOT NULL,
frameid        INTEGER UNSIGNED NOT NULL,
masktypeid     TINYINT UNSIGNED NOT NULL,
Descriptoradded DATETIME NOT NULL,
descriptorxml  TEXT,
descriptorxmldatasize INT UNSIGNED NOT NULL,

FULLTEXT       (descriptorxml),
FOREIGN KEY    (algsessionid) REFERENCES algorithmsessions (algsessionid),
FOREIGN KEY    (frameid) REFERENCES images (frameid),
FOREIGN KEY    (masktypeid) REFERENCES masktypes (masktypeid),
PRIMARY KEY    (algsessionid, frameid, masktypeid)
)ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Figura A-0-9: Estructura MySQL de la tabla descriptors

El campo **algsessionid** es una clave foránea que hace referencia, en la tabla **algorithmsessions**, a una única sesión (y ésta a su vez identifica a un único *algoritmo*). El campo **frameid** es otra clave foránea que hace referencia, en la tabla **images**, a un *frame* en concreto. El campo **masktypeid** también es una clave foránea que hace referencia, en la tabla **masktypes**, a los tipos de máscara / descriptores.

La clave primaria de dicha tabla es la tripleta (**algsessionid**, **frameid**, **masktypeid**), que identifica unívocamente al tipo de máscara / descriptor relacionada con el *frame* correspondiente a la secuencia de video, y generada durante la sesión de procesado abierta con el identificador **algsessionid**.

PRESUPUESTO

- **EJECUCIÓN MATERIAL**

Compra de ordenador personal (Software incluido).....2.000 €
Alquiler de impresora láser durante 6 meses.....50 €
Material de oficina.....150 €
Total de ejecución material.....2.200 €

- **GASTOS GENERALES**

16 % sobre Ejecución Material.....352 €

- **BENEFICIO INDUSTRIAL**

6 % sobre Ejecución Material.....132 €

- **HONORARIOS PROYECTO**

640 horas a 15 € / hora.....9600 €

- **MATERIAL FUNGIBLE**

Gastos de impresión.....60 €
Encuadernación.....200 €

- **SUBTOTAL DEL PRESUPUESTO**

Subtotal Presupuesto.....12060 €

- **I.V.A. APLICABLE**

16% Subtotal Presupuesto.....1929.6 €

- **TOTAL PRESUPUESTO**

Total Presupuesto.....13989,6 €

Madrid, marzo de 2010

El Ingeniero Jefe de Proyecto

Fdo.: Héctor Cabrejas Fernández
Ingeniero Superior de Telecomunicación

PLIEGO DE CONDICIONES

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un sistema distribuido de video – seguridad. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

CONDICIONES GENERALES

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad “Presupuesto de Ejecución de Contrata” y anteriormente llamado ”Presupuesto de Ejecución Material” que hoy designa otro concepto.

CONDICIONES PARTICULARES

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.